國立臺灣師範大學資訊工程研究所

博士論文

指導教授：林順喜 博士

較高維度演繹競局問題最佳演算法之設計與分析

The Design and Analysis of Optimal Algorithms for Deductive Games with Higher Dimensions

研究生：黃立德 撰

中華民國九十八年七月

摘要

隨著眾多領域中最佳化問題的逐步探索，發現許多重要的問題都能被轉換成演繹競局問題(deductive game)的模型，例如編碼理論(coding theory)、電路測試(circuit testing)、密碼系統破解(differential cryptanalysis)、附加條件搜尋(additive search problem)等問題。換言之，在演繹競局問題上的研究將使其他相關領域問題的求解露出希望曙光，因此發展有效解決演繹競局問題的方法變得不容遲緩。

在過去數十年間，有許多針對演繹競局問題的研究產生。Mastermind 與 AB game(或稱為 Bulls and Cow)是最有名的兩種演繹競局問題，知名的電腦科學家 Donald E. Knuth 在 1976 年於論文中介紹此二者並針對 Mastermind 做相關研究。在本論文中，我們提出一系列理論剪裁(theoretical-pruning)的最佳化方法與數學證明來解決這兩種問題。

在運用這些新方法到欲解決的問題後，我們得到下列新的成果：

(1) 我們提出一個適用於各種演繹競局問題的 admissible heuristic。同時，我們根據此 admissible heuristic，提出一個更有效率的演算法來解決 Mastermind，最後亦得到 Mastermind 在平均狀況下的最佳策略。

(2) 針對 AB game，我們提出一個更精緻的剪裁演算法(pruning algorithm)來處理它。很幸運地，最後我們得到 AB game 在平均狀況下的最佳策略且其平均猜測次數為 5.213。

(3) 我們針對在最差狀況下 3×*n* AB games 的最佳策略做理論性的分析。最後我們成功地導出一個計算最差狀況下的最佳猜測次數之公式。

(4) 我們研究一個 AB game 的變型，稱為容許一次錯誤回應之 AB game。最終我們求得其最佳猜測次數為 8。

關鍵字：AB game、分支界定法、演繹競局問題、競局樹、Mastermind、最佳策略、搜尋演算法、理論剪裁、錯誤回應。

誌謝

轉眼間即將結束博士班的學習生涯，在這四年的時光中，我很感謝指導教授林順喜博士的諄諄教誨與論文指導，使我對學術研究有更直接、更深刻的體認，過程中亦獲益匪淺。除了研究上的指導，學生對老師治學與做事嚴謹的態度及原則深有體會，亦為學生個人做事所效法之準則。

感謝口試委員許舜欽教授、葉耀明教授、吳毅成教授、徐讚昇教授、趙坤茂教授與陳善泰教授、對學生研究成果的肯定，並提出諸多寶貴的指正與建議，使整個研究與論文品質能更趨於完善。另外也感謝陳世旺教授、黃文吉教授與陳柏琳教授於論文計畫口試提出的建議與教導，使得論文整體架構能更加完整。另外要特別謝謝陳善泰教授在研究初始階段的指導與幫忙，其對研究執著認真的態度，是學生研究路途上的標竿。

感謝我們研究群中所有學長與學弟在研究上的建議與幫忙，尤其特別感謝士傑學長與雲青在研究上提供的諸多見解與協助。最後也感謝聖群、俊廷、信翰、傑淳與昱臣在口試事務上的幫忙，使口試能順利完滿結束。

最重要的就是感謝我的家人在背後默默的支持，使我可以專心的從事研究，並順利完成這份論文。尤其感謝太太麗華在精神上的鼓勵支持與實際上的協助，還有我們家小寶貝彧宸天真笑容的陪伴，此為督促我持續努力的最佳動力。

其他尚有許多人在無形中對此研究提供諸多協助。在此一併感謝。最後懷著感恩的心結束博士班生涯，期許自己未來在職場上能更加努力。

The Design and Analysis of Optimal Algorithms for Deductive Games with Higher Dimensions

A dissertation proposed

by

Li-Te Huang

to

the Department of Computer Science

and Information Engineering

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

in the subject of

Computer Science

National Taiwan Normal University

Taipei, Taiwan, R.O.C.

2009

Abstract

With the increasing exploration of optimization problems in numerous fields, many critical issues, such as coding theory, circuit testing, differential cryptanalysis, and additive search problem, can be modeled as deductive games. In other words, the research of these games has led to the hope that the fruitful solutions of problems in related areas may be obtained. Thus, it becomes urgent to develop efficient mechanisms for deductive games.

Over the last few decades, considerable concern has arisen in solving a number of deductive games. Mastermind and AB game (or “Bulls and Cows”), which were introduced by the famous scientist, Donald E. Knuth, in 1976, are the most well-known ones. In this dissertation, we aim to present a series of theoretical-pruning optimization approaches and mathematical proofs to solve both of the two.

As a result of applying these novel methods, the following new results have been obtained.

(1) An admissible heuristic for deductive games is presented. Meanwhile, a more efficient algorithm based on it is introduced to solve Mastermind and an alternative optimal strategy in the expected case is gained eventually.

(2) A refined pruning algorithm is demonstrated to address AB game. Fortunately, an optimal strategy for AB game in the expected case is acquired finally and its expected number of queries is 5.213.

(3) Analyses of playing 3×*n* AB games in the worst case optimally are conducted. Furthermore, a worthwhile formula for calculating the optimal numbers of queries in the worst case is derived successfully.

(4) A variation of AB game, AB game with an unreliable response, is surveyed. Finally, an exact bound of the number of queries for the game is achieved and its value is 8.

Keywords: AB game, branch-and-bound, deductive game, game tree, Mastermind, optimal strategy, search algorithm, theoretical pruning, unreliable response.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Deductive Games

Deductive games are zero-sum games of imperfect information. Two opponents are involved in deductive games. One opponent serves as a *codemaker*, who thinks of a secret code in mind, and the other is a *codebreaker*, who has to acquire the code by making queries iteratively. Each query is a guess for a possible secret code. After a query is made in each ply, the codemaker will give a response. The goal of the codebreaker is to identify the code in the fewest queries in accordance with previous information. The game proceeds in turn until the secret code is eventually obtained by the codebreaker. The original versions of deductive games, *Mastermind* and *AB game* (or "*Bulls and Cows*"), were first introduced by the famous scientist, Donald E. Knuth, in 1976 [45]. Detailed descriptions and categories of deductive games will be introduced in the follow-up paragraphs.

1.1.1 Discussed Categories of Deductive Games

Generally speaking, an $m \times n$ deductive game means that each possible secret code in the game is composed of m digits while every digit has n possibilities (symbols). Without loss of generality, the set of these n symbols is defined as $S = \{0, 1, 2, ..., n - 1\}$. Suppose that the codemaker has a secret code $c = c_1 c_2 \dots c_m$ in mind and the codebreaker makes a query $g = g_1 g_2 \ldots g_m$, where $c_i, g_j \in S, \forall i, j$. Then, the codemaker will give a response [*x*, *y*], where *x* and *y* are defined as follows.

- $x = \left|\{i : c_i = g_i\}\right|, \forall i = 1, \ldots, m$. Thus, *x* means the number of symbols which appear in both *c* and *g* and meanwhile, every symbol occupies the same position in both *c* and *g*.
- $y = \sum_{j=0}^{n} \min(p_j, q_j) - x$, where $p_j = \left|\{i : c_i = j\}\right|$ and $q_j = \left|\{i : g_i = j\}\right|$. In other words, *y* represents the number of symbols which occur in both *c* and *g* but the positions of these symbols in *c* and *g* do not match.

Note that for convenience, [*x*, *y*] is called *x*A*y*B as well. In this notation, the corresponding part can be omitted if *x* or *y* equal to 0. For instance, we can say 1A1B instead of [1, 1] while [0, 1] is also called 0A1B or 1B simply. A deductive game has ended if the codebreaker figures out the secret code, i.e., a response [*m*, 0] is received by the codebreaker. Besides the above definitions, there is one additional characteristic to distinguish two families of deductive games. That is whether repeated symbols are allowed in each secret code or not. One of the two is the family of Mastermind, in which repeated symbols are permitted in a secret code. The other is the family of AB game, in which all symbols within a code are distinct. The following subsection will offer additional introductions to the two families of deductive games and one of their variants.

1.1.1.1 The Family of Mastermind

In this kind of deductive games, a symbol may appear several times within a secret code. The most popular version of Mastermind is 4×6 Mastermind, which is well-known around the world since its appearance in 1972. A secret code in it consists of 4 digits with 6 possible symbols, e.g., 0, 1, …, 5. This is a topic that will be first investigated in the study. In order to simplify its name, 4×6 Mastermind is simply called Mastermind in the later discussion if we do not stress its dimension. Figure 1 shows the screenshot of 4×6 Mastermind, which was captured from [23].

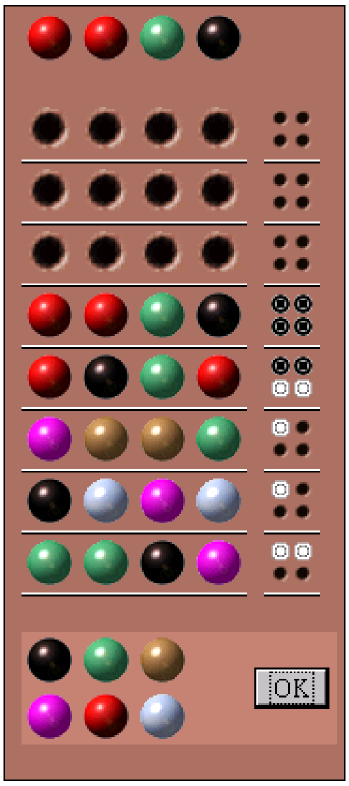
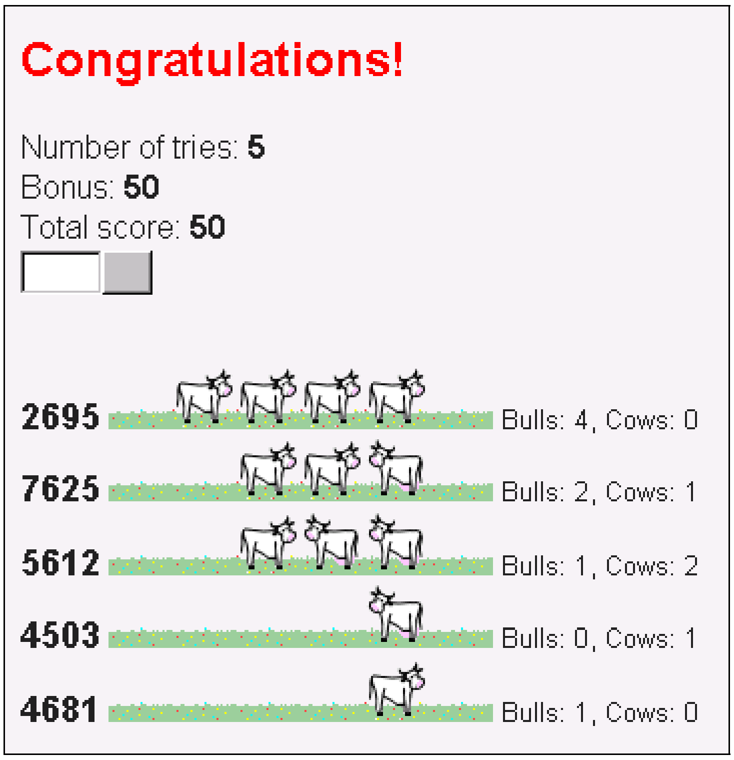


Figure 1. The screenshot of 4×6 Mastermind

1.1.1.2 The Family of AB game

The kind of deductive games is an ancient game that may date back a century or more and Mastermind also resembles it. The family of AB game is innately the same as that of Mastermind except the distinct symbols in a code. 4×10 AB game is the most common version and widespread in Asia and England. A secret code in it is composed of 4 digits while there are 10 possible symbols, i.e. 0, 1, …, 9, in each digit. In this study, we focus on 4×10 AB game and a generalized version, 3×*n* AB games, and for the sake of simplicity, AB game is usually referred to as 4×10 AB game if the dimension is not mentioned. Figure 2 exhibits the screenshot of 4×10 AB game, which was captured from [57].



Congratulations!

Number of tries: **5**
Bonus: **50**
Total score: **50**

```
2695  Bulls: 4, Cows: 0
7625  Bulls: 2, Cows: 1
5612  Bulls: 1, Cows: 2
4503  Bulls: 0, Cows: 1
4681  Bulls: 1, Cows: 0
```

Figure 2. The screenshot of 4×10 AB game

1.1.1.3 Deductive Games with Unreliable Responses

In normal deductive games, the codemaker will always give a correct response when the codebreaker makes a query. In order to fit in with the area of fault tolerance, a variant model of deductive games, called deductive games with unreliable responses, was first demonstrated by Huang *et al.* [38]. In other words, it is the same as the original one but the codemaker is allowed to offer incorrect responses at most *e* times, where the value of *e* is greater than zero. In [38], 4×6 Mastermind with an unreliable response has been solved completely. In this dissertation, a harder problem, 4×10 AB game with an unreliable response, will be considered and likewise, every code in it has 4 digits with 10 possible symbols. We call it AB game with an unreliable response for short as well.

1.1.2 Search Space of Discussed Deductive Games

Before the addressed deductive games are discussed, solid analyses of search space for these problems are necessary. Assume that an $m \times n$ deductive game is taken into account. The numbers of all valid responses given by the codemaker and all possible queries the codebreaker can make are offered here.

- Note that there exists $1+2+3+\cdots+(m+1)=(m+1)(m+2)/2$ combinations of the values of x and y for m digits but the response $[m-1, 1]$ is impossible. Therefore, there are at most $\frac{(m+1)(m+2)}{2}-1=\frac{m(m+3)}{2}$ legal responses. In other words, the codebreaker may receive one of these responses which are $[m, 0]$, $[m-1, 0]$, $[m-2, 2]$, $[m-2, 1]$, $[m-2, 0]$, ..., $[m-i, i]$, ..., $[m-i, 0]$, ..., $[0, m]$, ..., $[0, 0]$.
- All possible guesses the codebreaker can query are same as all valid secret codes the codemaker can choose. Obviously, there are n^m secret codes in the family of Mastermind and $n!/(n-m)!$ codes in the family of AB game. Thus, so are their numbers of all possible queries.

Table 1 summarizes the search space of every deductive games discussed in this study with the use of above formulas. Note that the number of pessimistic queries for these games means the worst-case number of queries required for the codebreaker. In the column "# of pessimistic queries", each value from top to down is referenced from [45], [18], Chapter 4, and Chapter 5 of this study respectively.

Table 1. The search space of discussed deductive games

deductive games	# of valid secret codes	# of legal responses	Pessimistic # of queries	Search space
4×6 Mastermind	$6^4=1296$	14	5	$(1296\times14)^5\approx10^{21}$
4×10 AB game	$10!/(10-4)!=5040$	14	7	$(5040\times14)^7\approx10^{34}$
3×n AB game	$n!/(n-3)!=n^3-3n^2+2n$	9	$\lfloor(n+1)/3\rfloor+3$	$(9n^3-27n^2+18n)^{\lfloor(n+1)/3\rfloor+3}$
4×10 AB game with an unreliable response	$10!/(10-4)!=5040$	14	8	$(5040\times14)^8\approx10^{39}$

1.2 The Classification of Proposed Algorithms

In order to investigate the above games, several kinds of algorithms have been proposed. We therefore give a comprehensive introduction to their classification and major properties.

1.2.1 Computer-aided Proof

A computer-aided proof (or called computer-assisted proof, computational method) is a paradigm of proofs, which has been partially or fully generated by computer. Most computer-aided proofs are implemented with numerous case-by-case exhaustion for desired problems. Sometimes, some theorems seem concise in nature whereas their mathematical proofs rely on heavy analyses of different configurations [69]. Thus, the computing power of computers is necessary to do an exhaustive verification.

In fact, not only the use of computers can make the analyses of complicated algorithms fun but also the results may not be gained in a reasonable time without the assistance of computers [65][66]. Historically, there were many significant results proven by this approach such as the four-color theorem [4][5], the Kepler conjecture [32], Connect-Four [1][2], Connect-Five [3], checkers [63] and so on.

1.2.2 Branch-and-bound Algorithm

The branch-and-bound algorithm was first demonstrated by Land and Doig in 1960 [49] and its appearance is common in the modern textbooks as well [52]. Another similar algorithm is named as A^* search [62] and previous study reveals that the two types of algorithms are essentially identical and they only differ at the interpretation level [48]. Thus, the two terms will be alternatively used below according to the concept we intend to express.

In general, the branch-and-bound algorithm is a general search algorithm for finding optimal solutions of various optimization problems. The key idea is that if a branch is encountered in the search process, the algorithm decides whether the branch should be cut or not in accordance with the value of the admissible heuristic (or called bound function), which represents a lower bound to the goal.

Good admissible heuristics of a certain problem are usually hard to discover, but are just the core of a branch-and-bound algorithm. Hence, they play significant roles in this kind of methods. On the other hand, admissible heuristics are worth discovering because they also have desirable properties in various search algorithms [56].

1.2.3 Approximate algorithm

Approximate algorithms are developed to solve optimization problems in practice. They sacrifice the guarantee of finding optimal solutions for the sake of getting feasible solutions in a significantly reduced amount of time. Approximate methods are usually distinguished between constructive methods and local search methods. The former ones generate solutions from scratch by adding components (or called moves) until a solution is complete. On the other hand, the latter ones start from some initial solution and iteratively try to replace the current solution by a better one. However, both methods may easily be trapped into local optima.

To escape from local optima, a new kind of approximate algorithms has emerged in the past three decades. These algorithms try to combine basic heuristic methods in higher-level framework aimed at efficiently and effectively exploring a search space. Examples of these algorithms based on local search methods are genetic algorithms [36], simulated annealing [44], tabu search [26][27], ant colony optimization [22], and iterated greedy [60]. On the other hand, examples of algorithms based on constructive methods are iterative sampling [33], HBSS [13], sampling and clustering [19], selective-sampling simulation [10], adaptive sampling [41][61], GRASP [24][55], block search [35], and Monte-Carlo Tree Search [20]. The main difference between these algorithms is the mechanisms used to guide the tree search.

1.2.4 Theoretical pruning

Given a huge search space of a problem, the forward pruning is a common scheme if a search algorithm is adopted to handle this problem. It is able to prune some useless branches in the search to speed up the work. Recently, some new forward-pruning mechanisms are presented such as null move [21], multi-cut [11], and AEL pruning [34]. Although these new approaches can acquire better results in less computation time, they still fail to guarantee the optimal outcomes. Therefore, the mechanism of the theoretical pruning, whose key idea is to conduct a forward pruning based on optimal analyses, is suggested in the research. Since this kind of pruning certified by optimal analyses omits the expansion of some branches, it can not only accelerate the speed of searching but also ensure the acquisition of best results.

Table 2 lists the classification of all algorithms appearing in the study and each algorithm is marked with the corresponding classification. The column "Position" shows in which chapter each algorithm is presented.

Table 2. The classification of proposed algorithms

Algorithms	Computer-aided proof	Branch and bound	Mathematical proof	Approximate	Theoretical pruning	Position
DBB	√	√			√	Chapter 2
RBB	√	√			√	Chapter 3
SR			√			Chapter 4
TPOA				√		Chapter 5
PPV	√	√			√	Chapter 5

1.3 Preliminaries of Related Work

Mastermind and AB game, whose dimensions are 4×6 and 4×10 respectively, are widely known throughout the world. The former is popular in America while the later, which is called "Bulls and Cows" in some places as well, is widespread in England and Asia. AB game is an ancient game and Mastermind, which resembles AB game, was invented in 1970. They were first stressed by the notable scientist, Knuth [45]. A strategy of Mastermind for minimizing the number of queries was also proposed by him and has achieved the optimal result in the worst case, where the maximum number of queries needed is 5. Meanwhile, its number of queries in the expected case is 4.478. Plenty of studies on finding better strategies of Mastermind in the expected case arose from then on. Irving [39], Norvig [54], and Neuwirth [53] enhanced the results, in which the bounds of expected numbers of queries are 4.369, 4.47, and 4.364, respectively. Flood [25], Ko and Teng [46] thus defined general notations for m×n deductive games and proposed some improved strategies. Eventually, Koyama and Lai [47] introduced an optimal strategy in the expected case for it in 1993 while the expected number of queries is about 4.34. Rosu [59] also proposed a faster algorithm and obtained the optimal strategy as well. A thorough introduction to Mastermind and a new heuristic approach were demonstrated by Barteld [6].

Chen *et al.* [16] demonstrated 2×n Mastermind and solved it completely with the graph-partition approach and Goddard [28] also obtained the same results for this problem independently. On the other hand, there is another variation called static Mastermind, where the codebreaker has to make all queries at once and has to uniquely decide the secret code after receiving all answers. Greenwell [31] derived some results of the game for small cases and provided some upper bounds of the game in a few cases. Afterwards, Goddard [29] completely solve static mastermind for at most 3 digits, and for some cases of 4 digits. Huang *et al*. [38] also presented a variation, called Mastermind with an unreliable response, and obtained an optimal strategy for it. In 2006 the Mastermind Satisfiability Problem has been shown to be NP-complete [70]. Jäger and Peczarski [40] investigated the generalized Mastermind and used the computer aided methods and mathematical proofs to decide the optimal number of queries in the worst case for $3\times n$ Mastermind and to derive the lower and upper bounds of the numbers of queries for $4\times n$ Mastermind, $m\times 2$ Mastermind, and $m\times n$ Mastermind. Goodrich [30] studied the algorithmic complexity of Mastermind with single-dimensional responses, which means that there is only one number (the value of x) in each response.

Much more efficient meta-heuristic algorithms, which produced comparable results with less running time in various dimensions of Mastermind, were investigated by Bernier *et al.* [9], Bento *et al.* [7], Kalisker and Camens [43], Singley [68], Ugurdag *et al.* [73], and Berghman *et al.* [8]. Although these methods are often efficient and effective, they are not able to attain the optimal strategy of Mastermind. Chen *et al.* [17] described a systematic method to address 4×6 Mastermind and it can achieve a near-optimal result in the expected case.

There are some scientists that emphasized the efficiency of acquiring the good results, such as Shapiro [67], Swaszek [71], Rosu [59], Temporel and Kovacs [72]. However, the qualities of strategies they discovered may usually be incomparable with those of other carefully considered approaches due to quick selections of queries.

Compared to 4×6 Mastermind, there is less research on 4×10 AB game because of its huge search space although 4×10 AB game has longer history. Chen *et al.* [15] introduced $2\times n$ AB game, and found the optimal number of queries in both the worst and the expected cases. Moreover, Chen *et al.* [18] first proved the exact number of queries in the worst case to identify a secret code for 4×10 AB game and showed that the number is 7.

Merelo *et al.* [51] indicated that many critical issues, such as coding theory, circuit testing, differential cryptanalysis, and additive search problem, can be modeled as deductive games. In other words, the research of these games has led to the hope that the fruitful solutions of problems in related areas may be obtained.

1.4 Research History of Deductive Games

Table 3 has concluded with a series of significant progressive and conclusive results of deductive games since Knuth [45] stressed two famous deductive games, Mastermind and AB game, in 1976. Progressive results mean that the research of the handled problem has acquired better results but it may be refined again in the future. Conclusive results represent that a complete conclusion (often refers to as an optimal strategy) is obtained via the stressed problem.

The field “Problem” is the game that paper dealt with. If it writes “Several dimensions of Mastermind”, then there are several versions of Mastermind surveyed in that paper. We can observe that many variations of deductive games are included as well. Moreover, the field “Case” indicates which condition the addressed problem is considered. The terms, “Worst” and “Expected”, mean that the problem is taken into account in the worst case and in the expected case. Note that “NP-C” is filled in the field if the game was proven to be an NP-Complete problem in that study while “Fixed” is used in static Mastermind and indicates a fixed number of queries is required. The field “Author” shows the scholars who conducted this research.

Because of space restrictions, we omit each citation of the corresponding paper in Table 3, readers can reference the previous subsection for more information. Furthermore, it deserves to be mentioned that our contributions to the area of deductive games are also highlighted with gray backgrounds in Table 3.

Table 3. Significant research of deductive games

Year	Conclusive results			Progressive results		
	Problem	**Case**	**Author**	**Problem**	**Case**	**Author**
1976	4×6 Mastermind	Worst	Knuth [45]			
1978				4×6 Mastermind	Expected	Irving [39]
1982				4×6 Mastermind	Expected	Neuwirth [53]
1983				4×4, 4×5 Mastermind	Expected	Shapiro [67]
1984				4×6 Mastermind	Expected	Norvig [54]
1986				Several dimensions of Mastermind	Expected	Ko and Teng [46]
1988				Several dimensions of Mastermind	Expected	Flood [25]
1993	4×6 Mastermind	Expected	Koyama and Lai [47]			
1996				Several dimensions of Mastermind	Expected	Bernier *et al*. [9]
1999	4×6 Mastermind	Expected	Rosu [59]	Several dimensions of Mastermind	Expected	Bento *et al*. [7]
				4×6 Mastermind	Expected	Swaszek [71]
2000				4×6 static Mastermind	Fixed	Greenwell [31]
2003	3×*n* static Mastermind	Fixed	Goddard [29]	4×*n* static Mastermind	Fixed	Goddard [29]
				Several dimensions of Mastermind	Expected	Temporel and Kovacs [72]
				Several dimensions of Mastermind	Expected	Kalisker and Camens [43]
2004	2×*n* AB game	Worst, expected	Chen *et al*. [15]			
	2×*n* Mastermind	Worst, expected	Chen *et al*. [16], Goddard [28]			
2005				4×6 Mastermind	Expected	Barteld [6]
				Several dimensions of Mastermind	Expected	Singley [68]
2006	Mastermind Satisfiability Problem	NP-C	Stuckman and Zhang [70]	Several dimensions of Mastermind	Expected	Ugurdag [73]
	4×6 Mastermind with an unreliable response	Worst	Huang *et al*. [38]	4×6, 5×8 Mastermind	Expected	Merelo *et al*. [51]
2007	4×10 AB game	Worst	Chen *et al*. [18]	4×6 Mastermind	Expected	Chen *et al*. [17]
	4×6 Mastermind	Expected	Huang *et al.*			
2009	3×*n* Mastermind	Worst	Jäger and Peczarski [40]	4×*n*, *m*×2, and *m*×*n* Mastermind	Worst	Jäger and Peczarski [40]
	Mastermind with black-peg results	NP-C	Goodrich [30]	Several dimensions of Mastermind	Expected	Berghman *et al*. [8]
	4×10 AB game	Expected	Huang *et al.*			
	3×*n* AB game	Worst	Huang and Lin			
	4×10 AB game with an unreliable response	Worst	Huang and Lin			

1.5 Terminologies of Deductive Games

There are two issues for optimizing deductive-game problems. One is to minimize the queries made by the codebreaker in the worst case, and the other is to minimize that in the expected case. ***An optimal strategy in the worst case*** is a strategy which minimizes the maximum number of queries needed by the codebreaker for any secret code chosen by the codemaker. ***An optimal strategy in the expected case*** is a strategy which minimizes the expected number of queries required with considerations of all possible codes. Note that a uniform distribution over all the codes the codemaker may choose is assumed.

An alternative aspect of viewing the optimization for strategies of deductive games as a game-tree search is adopted in this study. In order to formulate the problem precisely, some general definitions used in the entire study are listed as follows while other specific terms are defined in each chapter individually, if necessary.

Definition 1. A secret code is *eligible* if it is compatible with all queries and the corresponding responses given so far.

Definition 2. A set, which contains some eligible codes, is referred to as a *state*.

Definition 3. For an $m \times n$ deductive game, a state with only one eligible code, which has also been queried by the codebreaker now, is defined as a *final state*. That is to say that the secret code has been identified and the game is over.

Definition 4. If finding an optimal strategy for a deductive game is regarded as a game-tree search, then each *internal node* of the game tree indicates a state while every *leaf* represents a final state.

Definition 5. The *external path length* (or called *EPL* for short) is the sum of the depth of all leaves of the game tree.

Definition 6. The number of queries needed by the codebreaker in the expected case (also called the expected number of queries) is L/k, where L is the external path length of the game tree formed by the codebreaker's strategy and k is the number of all possible codes in the game.

Definition 7. A *strategy* discussed in the study refers to one of the options that the codebreaker can choose. Each strategy has its corresponding game tree. Trivially, the codebreaker has a lot of possible strategies.

Definition 8. An *optimal strategy in the expected case* is the strategy which has the minimum expected number of queries. In other words, the external path length of the game tree should be minimized.

Definition 9. An *optimal strategy in the worst case* is the strategy which has the minimum pessimistic number of queries. Hence, the height of the game tree should be minimized.

Definition 10. An *equivalence transformation* is defined as a composition of a permutation on the set of symbols and a permutation on the set of digits. Thus, a query g_1 is said to be *equivalent* to another query g_2 if there exists an equivalence transformation t such that $g_2 = t(g_1)$. This concept is presented by Neuwirth [53].

Definition 11. Suppose the codebreaker has made i-1 queries, named as $g_1, g_2, \ldots, g_{i-1}$, then two codes u_1 and u_2 at the i-th query are called *strategy equivalent* if $\langle g_1, g_2, ..., g_{i-1}, u_2 \rangle = t(\langle g_1, g_2, ..., g_{i-1}, u_1 \rangle)$. In other words, we can only take u_1 as a representative for computing an optimal strategy if u_1, u_2, …, and u_j are strategy equivalent.

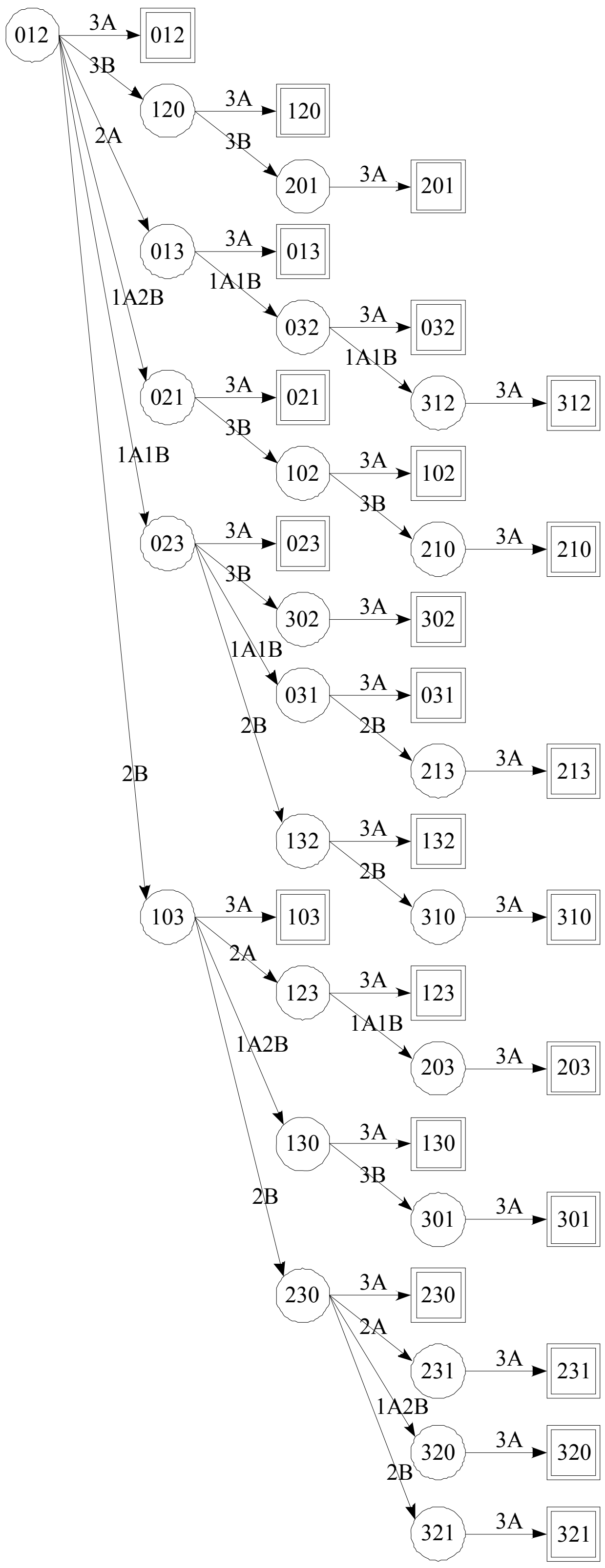


Figure 3. A strategy for 3×4 AB game

Figure 3, which is used for illustrating the above terminologies, is a codebreaker's strategy for 3×4 AB game. In the game, the codemaker comes up with a secret code consisting of 3 digits out of 4 symbols, i.e., 0, 1, 2, and 3. A response, which is one of [3, 0], [2, 0], [1, 2], [1, 1], [0, 3], and [0, 2], is received by the codebreaker in each ply. Consequently, the codebreaker entails investigating the code with making use of those responses. Each circle appearing in Figure 3 represents a state and the number in it is a query made by the codebreaker at that moment while this state is encountered. Every double-lined square means a leaf of the game tree or a final state. The text above each arrow means the response offered by the codemaker. Note that the same notations will be adopted in the following discussions.

Some phenomena are able to be verified easily from Figure 3. First, there are totally 24 possible secret codes as the game starts and thus, these 24 codes are eligible at that moment. Meanwhile, the set yielded by the 24 codes is the state at the beginning. It is also obvious that the 24 leaves in the tree imply final states. Moreover, Figure 3 exhibits that the external path length is $1\times1 + 2\times5 + 3\times9 + 4\times9 = 74$ and the expected number of queries required by the codebreaker is equal to $74/24 \approx 3.083$ as well. Meanwhile, the pessimistic number of queries is 4 since the height of the game tree is 4.

1.6 Organization of the Dissertation

This research proposes a series of theoretical-pruning optimization algorithms and mathematical proofs for deductive games and therefore, the following studies are composed of five major parts. In Chapter 2, a complete search algorithm, *depth-first backtracking algorithm with branch-and-bound pruning*, is introduced to address Mastermind. Meanwhile, an admissible heuristic, which can be applied to various deductive games, is presented as well. Chapter 3 demonstrates a *refined* *branch-and-bound algorithm with speed-up techniques* for AB game in the expected case. Three useful techniques for accelerating the speed of the search algorithm are brought up. In Chapter 4, 3×*n* AB games is investigated and a sophisticated method, called *structural reduction*, is developed to explain the worst situation in this game. Chapter 5 presents a variation of AB game, AB game with an unreliable response. An important theorem for deductive games is proven and two algorithms based on it, which are *two-phase optimization algorithm with theoretical pruning* and *pigeonhole-principle-based verification algorithm with theoretical pruning*, are proposed. Fortunately, an exact bound of the number of queries needed for the problem is achieved because the upper and lower bounds resulting from the two methods are equal. Chapter 6 concludes with remarkable results in our study and some future work. Moreover, two appendixes, which contain the detailed information on some proofs, are attached at the end of the dissertation.

Chapter 2

Depth-First Backtracking Algorithm with Branch-and-Bound Pruning

An optimal strategy in the expected case for Mastermind has already been proposed by Koyama and Lai [47] in 1993 by using an exhaustive search but that study took too much time to search the strategy. Therefore, a more efficient algorithm, called *depth-first backtracking algorithm with branch-and-bound pruning* or abbreviated to DBB, is developed for solving Mastermind in this chapter. Compared to other heuristic methods, DBB can guarantee to yield the optimal tactic if the search procedure finishes. Moreover, an admissible heuristic, which can be applied to various deductive games, is presented as well. Section 2.1 gives an intuitive concept of our proposed approach. Section 2.2 introduces our depth-first backtracking algorithm with branch-and-bound pruning for Mastermind. In Section 2.3, some experimental results are discussed. Section 2.4 summarizes our concluding remarks in the chapter and a critical issue is mentioned for future research.

2.1 Introduction

Mastermind, whose dimension is 4×6, is a two-player game and both of two players involved are the codemaker and the codebreaker. Suppose that the set of the six symbols, which may appear in secret codes, is $S = \{0, 1, 2, 3, 4, 5\}$. Thus, there are $6^4 = 1296$ valid secret codes in Mastermind. Meanwhile, there are also 14 legal responses, which are [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], and [0, 0]. The other definitions and properties are described in Chapter 1 and so, they are omitted here.

A complete algorithm with a novel pruning technique, named as a *depth-first backtracking algorithm with branch-and-bound pruning* (DBB), is proposed to solve the problem. The idea of our scheme is similar to the admissible heuristic in the A^* search. The A^* search is a tree search algorithm which finds a best path from a given initial state to a given goal with the lowest cost. The algorithm will terminate if a best solution is found. However, a complete search is conceptually required for our problem. Hence, DBB will search the full game tree and prune the unnecessary queries by using an admissible heuristic. The following sections will demonstrate the sophisticated algorithm and its power of searching.

2.2 The Depth-first Backtracking Algorithm with Branch-and-Bound Pruning

A large number of real-world problems can be modeled as optimization problems or games. A search algorithm is therefore a general approach for them. Unfortunately, most of these problems are NP-hard or PSPACE. In other words, it has to take exponential time to search for an optimal solution. Thus, there are plenty of pruning techniques published in the literature such as A^* search [62], branch-and-bound pruning [52], and so on.

Previous pruning approaches are appropriate for optimization problems since their goal is to find a best solution in the search space. So, the search ends when it is found. A complete search is theoretically required to our problem because of the considerations of the optimal strategy in the expected case. Hence, traditional pruning approaches may not easily be applied to our problem directly.

A novel pruning technique based on the admissible heuristic in the A^* search is proposed to solve the problem. In Section 2.2.1, the framework of our depth-first backtracking algorithm with branch-and-bound pruning (DBB) is introduced. Section 2.2.2 illustrates the detailed operations of our scheme.

2.2.1 The Framework of DBB

The idea of our scheme is similar to the admissible heuristic in the A^* search. The A^* search is a tree (graph) search algorithm which finds a best path from a given initial state to a given goal with the lowest cost. The algorithm will terminate if a best solution is found. However, a complete search is conceptually required for our problem. Hence, DBB will search the full game tree and prune the unnecessary queries by using an admissible heuristic. Notice that a solution described here means a strategy for the codebreaker to identify a secret code with respect to our problem.

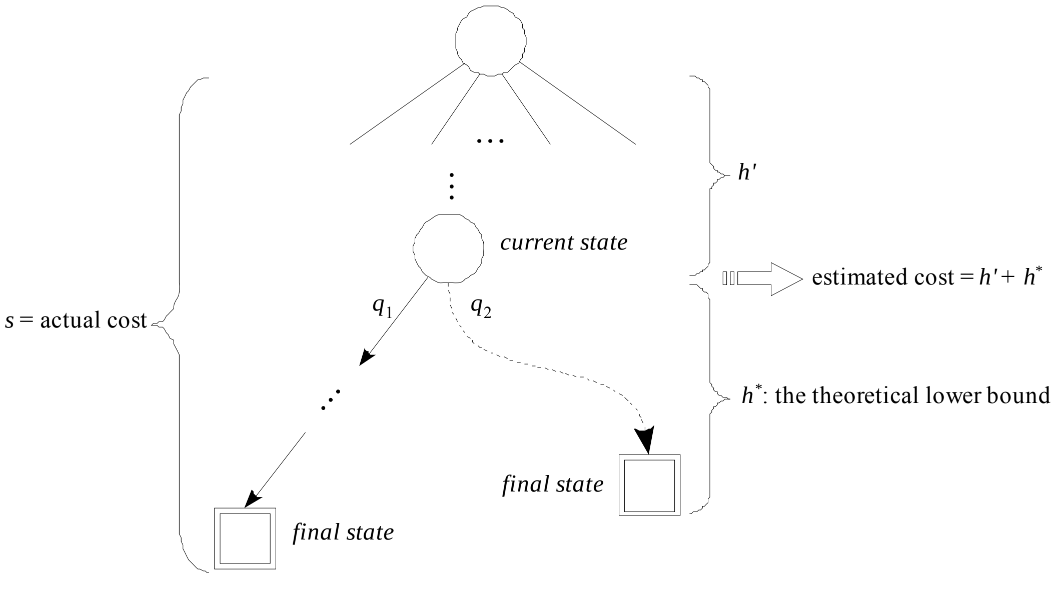


Figure 4. The scenario of branch-and-bound pruning

Figure 4 shows a scenario of DBB. Suppose that h' is the cost from the root to the current state and h^* is the cost from the current state to the final state. Then, h^* is called *admissible* if it never overestimates the cost to reach the final state. In other words, the actual cost is less than or equal to $h' + h^*$. It can also be viewed as a theoretical lower bound for the problem we deal with.

Our scheme traverses the game tree in depth-first fashion until a final state is reached. It then gets an actual cost s which is initially assigned to be the current-best solution. Note that the actual cost s results from the query q_1 in its traversed path. Afterwards, it soon backtracks to its parent, e.g., the *current state*, and picks one of the other queries, e.g., the query q_2, and uses an admissible heuristic to estimate the cost h^* of q_2. The search continues if s is larger than $h' + h^*$. Otherwise, a cut happens because s is less than or equal to $h' + h^*$. In other words, there is no need to expand the branch of q_2 and the correctness of the algorithm is still maintained. This continues in a similar manner until the full game tree is searched.

```
DBB (state $v$)
if (a final state is reached) then return the current-best solution $s$;   // Final state indicates the leaf of the game tree.
Expand $v$;
for (each branch $q$ of $v$)                                                // Each $q$ is a branch of $v$.
    $h^*$ = ESTIMATE($q$);                                                  // ESTIMATE is an admissible heuristic of predicting the cost from $q$ to a final state.
    if ($h' + h^* < s$) then                                                // $h'$ is the actual cost from the start state to $v$.
        DBB (the states resulting from $q$);                                // Search recursively from the states resulting from $q$.
    else
        Cut the branch $q$;                                                 // A cut happens if $h' + h^* \geq s$.
```

Figure 5. The depth-first backtracking algorithm with branch-and-bound pruning

A rough sketch of the entire algorithm is exhibited in Figure 5. It is especially important to notice that DBB always maintains a current-best solution s during the search. Hence, DBB goes through the downward direction at first until a final state is reached. It therefore gets a current-best solution (s is updated). Then, DBB backtracks and starts to estimate h^* in each of the other queries. Unnecessary branches of the queries will never be expanded. Note that it updates s constantly when final states are encountered. So, DBB will finally obtain an optimal solution when the full game tree has been traversed completely.

2.2.2 DBB for Mastermind in the Expected Case

In this section, we will deal with Mastermind in the expected case. First, the pruning technique applied to Mastermind is introduced in Section 2.2.2.1. Second, the admissible heuristic we used is designed and explained carefully in the follow-up section. Eventually, an optimal strategy is found as a result of applying DBB to this problem.

2.2.2.1 DBB for Mastermind

According to the analyses in Table 1, the search space for Mastermind is $(1296\times14)^5 \approx 10^{21}$. Therefore, it takes much time to find an optimal strategy by searching the game tree completely. A pruning technique adopted by DBB is used to save a lot of time instead of making an exhaustive search. Figure 6 shows the game tree of Mastermind by applying DBB. The circles in the Figure 6 mean the states which are the sets of eligible secret codes while the diamonds are the possible queries the codebreaker can choose (1296 valid queries in each ply). In the game tree, the 14 branches produced by the codemaker's responses should be traversed completely and the 1296 branches expanded by the codebreaker may be pruned by the admissible heuristic since we are aiming at finding an optimal strategy for the codebreaker. Let's consider the situation exhibited in Figure 6. The traversal to the subtrees of q_1 (in bold style) is just finished and q_2 is now taken into account. An estimated value h^* is obtained with the use of the admissible function. The subtrees below q_2 do not have to be expanded if the result of expanding q_1 is better than h^*. This is the key idea of DBB and the search can thus be completed in a more reasonable time. Note that the correctness of DBB is preserved because of the admissible heuristic.

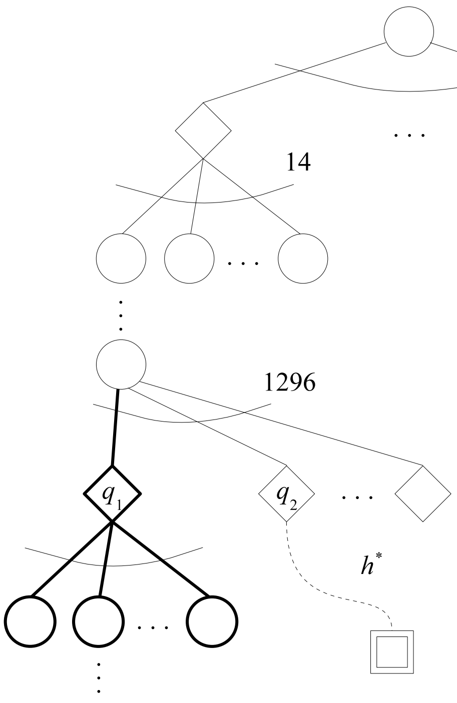


Figure 6. The game tree of Mastermind by applying DBB

2.2.2.2 The Admissible Heuristic for Deductive Games

Now the most critical issue is how to design an admissible heuristic function to estimate the theoretical lower bound h^*. Note that minimizing the number of queries in the expected case is the same as minimizing the external path length of the game tree. So, the concept of volumes introduced by Huang *et al.* [38] is involved to get the theoretical maximum bounds for the 14 classes (responses). In order to make sense, the term, "response", is replaced by "class" here. We know that different queries in a certain ply result in distinct distributions of the eligible codes in 14 classes. The distribution discussed here is the sizes of 14 classes resulting from a certain query. Thus, the volume of a class [*x*, *y*] is defined as the maximum value of the numbers of the eligible codes when the codebreaker makes all the valid queries in one ply and the codemaker responses with [*x*, *y*]. In the beginning, at the root of Figure 6, there is a total of 1296 secret codes. While the first query is considered, there are 5 nonequivalent queries in 1296 possible codes, i.e., “0000”, “0001”, “0011”, “0012”, and “0123” for the codebreaker [47]. If the codebreaker queries “0000” and the codemaker gives the response [1, 0], then we can derive that there are 500 possible secret codes. Similarly, if the codebreaker queries “0001”, “0011”, “0012”, or “0123”, and the codemaker gives the response [1, 0], then we can derive that there are 317, 256, 182, and 108 possible secret codes, respectively. So, the volume of the class [1, 0] is set to be 500, the maximum value of these numbers: 500, 317, 256, 182, and 108. With the use of Get_volume function (see Huang *et al.* [38]) based on the above idea, the volumes of the 14 classes (responses) are obtained as in Table 4.

Table 4. The volumes of 14 classes calculated by Get_volume function

class	[4, 0]	[2, 2]	[1, 3]	[0, 4]	[3, 0]	[2, 1]	[1, 2]	[0, 3]	[2, 0]	[1, 1]	[0, 1]	[0, 2]	[1, 0]	[0, 0]
volume	1	6	8	9	20	48	132	136	150	252	308	312	500	625

The same principle of the extended pigeonhole principle presented by Chen *et al.* [18] is therefore employed to estimate the lower bounds of the queries needed. However, there are major differences between the problem in the previous study (Chen *et al.* [18]) and this problem we consider now. Only the worst case among the 14 classes is considered for the codemaker in that paper. The so-called “worst case” denotes the response (class) which will result in the maximum number of queries required by the codebreaker. But each class should be taken into account for our problem.

The heuristic function here has to calculate the “theoretical optimal” number of queries in the expected case for a certain query (or called the lower bound of a certain query) for the codebreaker. Suppose that the lower bound of a query q is assessed by the codebreaker. The query q results in 14 classes. It will assume that there exists an optimal strategy such that all of the eligible codes in each class may be divided evenly in the following queries. The rated value calculated by this heuristic for a state (one of the 14 classes) is the external path length (EPL) of the subtree that is yielded by the theoretical optimal strategy we imagine. So, the actual expected number of queries is thus larger than or equal to the estimated value. Trivially, the heuristic is admissible because a theoretical optimal strategy is assumed to rate the EPL of the subtree of each class formed by *q*. Moreover, it can be applied to any deductive games by adjusting the number of legal classes (the number of legal responses the codemaker can give) and its volume of each legal class since any other specific knowledge do not have to be considered. In other words, the lower bound of a query *q* by utilizing this heuristic is equal to the summation of each EPL with respect to 14 classes plus the size of the state, which is the original state before *q* is made.

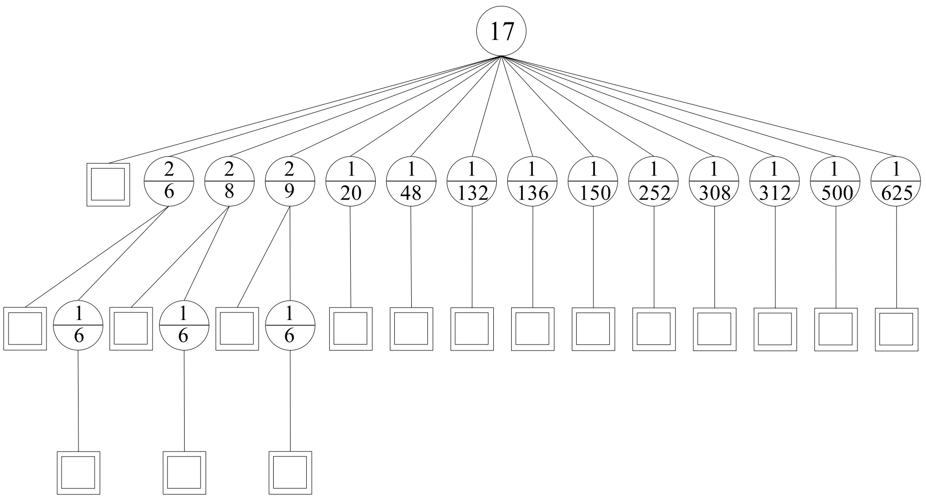


Figure 7. An example of the calculation of the admissible heuristic for Mastermind

A simple example to illustrate the calculation of the EPL regarding some class (state) yielded by *q* is shown in Figure 7 with the use of the proposed admissible heuristic. Given a state with a size of 17, as shown in Figure 5, we imagine that the theoretical optimal strategy will divide the 17 eligible codes into 14 classes evenly without exceeding the corresponding volumes. The number in the lower half of the circle is the volume of each class and the number in the upper half is the number of eligible codes in it. Since there is 1 leaf at level 1, 13 leaves at level 2, and 3 leaves at level 3, it is obvious that the external path length of the tree is $1\times1 + 2\times13 + 3\times3 = 36$ in the ideal situation. Thus, the external path length of the example must be smaller than or equal to the actual expected number of queries. It is therefore easy to see that the heuristic is admissible because it never overestimates the expected number of queries.

2.3 Experimental Results

In order to analyze the performance of the proposed DBB, we demonstrate the results of the original version of Mastermind (4×6 Mastermind) and another version of Mastermind, which is 3×5 Mastermind. 3×5 Mastermind has smaller search space in the case of 3 digits with 5 possible symbols. That is to say that it has $5^3 = 125$ possible secret codes totally. Note that the equivalent properties proposed by Neuwirth [53] are able to reduce the search space. For example, "0000" is equivalent to "1111" at the first query because the numbers, 1 and 2, are both not used before. With the considerations of the properties, there are five nonequivalent queries at the first query, which are "0000", "0001", "0011", "0012", and "0123". The branching factor in the first ply changes from 14×1296 to 14×5 eventually. This technique has also been implemented in our programs in order to speed up the search.

Besides the comparison between 3×5 Mastermind and 4×6 Mastermind, we also investigate the effect of the traversing order during the search. In other words, we have to decide which query is promising when several queries are encountered after the current state is visited. To deal with this issue, we estimate the lower bounds of the child states by making use of the admissible heuristic before they are expanded. We sort their lower bounds and traverse these queries order-by-order in accordance with their values. The smaller the value is, the earlier the traversal is. All experiments were run on a dedicated PC with an AMD Opteron 252 processor. The experimental results are exhibited in Table 5.

Table 5. The experimental results of two versions of Mastermind

	3×5 Mastermind	4×6 Mastermind
DFS	> 10 hr.	> 10 days
DBB	38.68 sec.	43.76 hr.
DBB (promising query)	11.21 sec.	9.5 hr.
External path length	451	5625

DFS is the abbreviation of depth-first search while the term, “promising query”, means that DBB expands the queries in nondecreasing order according to the values of lower bounds. We can see that DBB is able to obtain the optimal strategies for the two versions and their corresponding external path length is 451 and 5625, respectively. This means that the expected number of queries is about 4.34 (≈5625/1296) for Mastermind if we apply the optimal strategy in the expected case. The results also show that DBB with the considerations of promising queries has the best performance. Without the judgement of promising queries, DBB will traverse a lot of useless queries. That is to say that most queries will be cut if DBB expands queries in the correct order.

From the experimental results, DFS has very poor performance doubtlessly since it is certainly an exhaustive search. Hence, DFS can not search the full game tree in a reasonable time and the total number of the states it has to expand is still unknown. On the other hand, DBB is significantly superior to and is over 25 times faster than DFS. Totally, there are 137834651 states expanded by DBB. The results also reveal that the larger the search space is, the more important the pruning technique is.

2.4 Chapter Conclusion

Previously, an exhaustive search was applied to find the optimal strategy for Mastermind. But it may not be adopted in other harder problems or games because of its huge search time. In this chapter, a more efficient *depth-first backtracking algorithm with branch-and-bound pruning* (DBB) for Mastermind in the expected case is introduced, and an alternative optimal strategy is obtained eventually. Moreover, an admissible heuristic, which can be applied to various deductive games, is presented as well. From the experimental results, the effect of expanding promising queries during the search is significant to the performance of DBB. Meanwhile, DBB is significantly superior to and is over 25 times faster than the traditional search algorithm. How to design a more precise admissible heuristic is yet another critical issue. Furthermore, it may be interesting to compare our method with other search algorithms or other heuristics mentioned in the previous studies with the consideration of the qualities of solutions and the search time.

Chapter 3

Refined Branch-and-Bound Algorithm with Speed-up Techniques

Another famous deductive game is AB game, which is popular in Asia and England. However, to date, there have been no optimal expected-case strategies for AB game in formal literature since its appearance. Since the complexity of these deductive games grows at an exponential rate with higher dimensions, DBB can not be directly applied to efficiently solve AB game in the expected case.

In this chapter, a *refined branch-and-bound algorithm with speed-up techniques*, which is abbreviated to RBB, is demonstrated for AB game in the expected case. This algorithm is based on DBB and three useful techniques such as the incremental update of the lower bounds, the hashing technique, and the reduction of equivalent queries are invented to integrate with it. Therefore, RBB will lead to the hope that the optimal tactic of AB game in the expected case is attained. Section 3.1 reviews our handled problem and compares the search space between Mastermind and AB game. Section 3.2 introduces a refined branch-and-bound algorithm while new techniques and significant improvements are demonstrated here as well. In Section 3.3, some experimental results and discussions are given. Section 3.4 summarizes the remarkable results in this chapter.

3.1 Introduction

AB game, which is also called “Bulls and Cows” in England, is another popular deductive game around the world for decades as well. Its dimension is 4×10 and there are also two opponents involved in this game, which are called the codemaker and the codebreaker respectively. There are ten symbols appearing in possible secret codes of AB game, e.g., 0, 1, 2, …, and 9. Note that the repeated symbols are not allowed in a single secret code. Thus, there are 10!/(10-4)! = 5040 valid secret codes in AB game. Meanwhile, the 14 legal responses of AB game, which are [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], and [0, 0], are the same as those of Mastermind. The accurate definitions are exhibited in Chapter 1 and therefore, these descriptions are omitted here.

The search space, which means all possible strategies the codemaker and the codebreaker can adopt, for 4×6 Mastermind and 4×10 AB game is compared in the following equation:

$$\frac{(5040\times 14)^7}{(1296\times 14)^5} > 10^{12}$$

Notice that the upper part of the equation is the search space for 4×10 AB game while the lower one is that for 4×6 Mastermind. Clearly, the search space for 4×10 AB game is far larger than that for 4×6 Mastermind. Moreover, the search space represents the required time to discover an optimal strategy for the codebreaker since the expected number of queries is considered. Hence, it is clear that the difficulty of solving AB game is much harder than that of solving Mastermind.

To the best of our knowledge, the optimal strategy of 4×10 AB game for the codebreaker has never been discovered and meanwhile, its corresponding expected number of queries has not been determined yet due to its difficulty. In Chapter 2, a fruitful pruning framework, DBB, relied upon the admissible heuristic in the A^* search was proposed to solve 4×6 Mastermind. However, it is not capable of solving 4×10 AB game right away since it has much huger search space than 4×6 Mastermind. In this chapter, our goal aims at finding an optimal strategy of 4×10 AB game for the codebreaker to minimize the expected number of queries.

3.2 A Refined Branch-and-Bound Algorithm with Speed-up Techniques

A full search is theoretically conducted to our problem so as to consider the optimal tactic in the expected case. Because DBB can not solve the concerned problem directly, a refined approach based on it is demonstrated. Furthermore, the idea of DBB will be introduced briefly to make this chapter self-contained.

3.2.1 The Fundamental Framework in Terms of Branch-and-Bound Pruning

Although DBB proposed in Chapter 2 can not explore the game tree directly within a reasonable time, it remains a vital basis for us. Therefore, a brief introduction to DBB is still given here.

DBB and the A^* search act in a similar way. The A^* search is regarded as a tree (graph) search algorithm which looks for a path from an initial state to a final goal with the lowest cost. It will terminate if a best solution is obtained. However, a full search is necessarily engaged in dealing with our problem because we need to calculate the value of the external path length of the game tree. Hence, DBB will carry out a search of the whole game tree and prune the useless states by taking advantage of an admissible heuristic. Notice that a solution described here denotes a strategy for the codebreaker to identify a secret code with respect to our problem.

Let $h^{'}$ denote the cost from the root to the current state and h^{*} be an estimated cost from the current state to a final state. Then, h^{*} is called *admissible* if it never overrates the cost to reach the final state. In other words, the actual cost is less than or equal to $h^{'} + h^{*}$. It can also be viewed as a theoretical lower bound for the problem we cope with.

DBB first traverses the game tree in depth-first fashion until a final state is reached. It then gets an actual cost s which is initially assigned to be the current-best solution. Note that the actual cost s results from the query q_1 in its traversed path. Afterwards, it soon backtracks to the *current state*, and picks one of the other queries, e.g., the query q_2, and uses an admissible heuristic to estimate the cost h^{*} of q_2. The search continues if s is larger than $h^{'} + h^{*}$. Otherwise, a cut happens because s is less than or equal to $h^{'} + h^{*}$. This continues in a similar manner until the full game tree is searched. Figure 4 shows roughly the scenario and Figure 5 exhibits this algorithm. The current state is what we consider presently. An admissible heuristic will be used to estimate its cost h^{*} and thus, $h^{'} + h^{*}$ is compared with the actual cost s to determine whether it should be cut or not.

In accordance with the analyses in Table 1, the search space for AB game is $(5040\times14)^7 \approx 10^{34}$. Figure 8 shows the game tree of AB game by applying DBB directly. The circles in Figure 8 mean the states which are the sets of eligible secret codes while the diamonds are the valid queries the codebreaker can choose (5040 queries in each ply). In the game tree, the 14 branches yielded by the codemaker's responses should be traversed completely and the 5040 branches expanded by the codebreaker may be pruned by the admissible heuristic since we are aiming at finding an optimal strategy for the codebreaker. Let's take the situation exhibited in Figure 8 into account. The search to the subtrees of q_1 (in bold style) is just finished and q_2 is now considered. An estimated value h^* is obtained by using the admissible function. The subtrees below q_2 do not have to be expanded if the result of expanding q_1 is better than h^*.

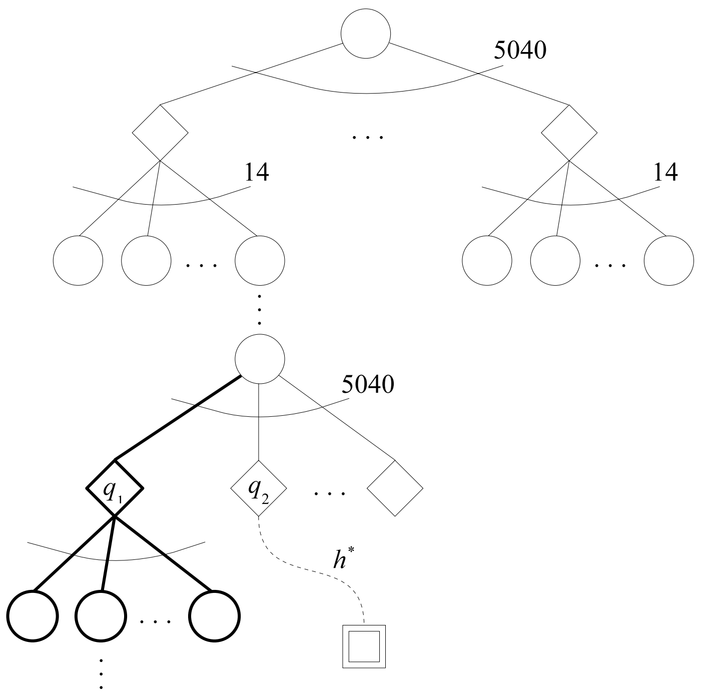


Figure 8. The game tree of AB game by applying DBB directly

The admissible heuristic presented in Section 2.2.2.2 with slight modifications of the volume of each legal class is utilized to estimate the lower bounds of the numbers of queries. Likewise, different queries in a certain ply result in different distributions of the eligible codes in the 14 responses. Similarly, the volume of a response [*x*, *y*] is also defined as the maximum value of the numbers of the eligible codes when the codemaker responses with [*x*, *y*]. The first query made by the codebreaker has only one choice here because all of the queries are equivalent at the first query. As a result, g = “0123” is selected as the representative for the first query. The numbers of eligible codes of each class after g is made form these volumes are concluded in Table 6. From the analyses in Section 2.2.2.2, the actual expected number of queries is thus larger than or equal to the value of estimations. An example to illustrate the calculation of the EPL about some class (state) is shown in Figure 9. Notice that the only difference between Figure 7 and Figure 9 is their volumes.

Providing a state with a size of 17, as shown in Figure 9, we imagine that the theoretical optimal strategy will distribute the 17 codes into 14 responses evenly without exceeding the corresponding volumes and so does the optimal strategy in each of the following levels of the game tree. The number in the lower half of the circle is the volume of each response and the number in the upper half is the number of secret codes in it.

Since there is 1 leaf at level 1, 13 leaves at level 2, and 3 leaves at level 3, it is obvious that the external path length of the tree is 1×1 + 2×13 + 3×3 = 36. Thus, the actual external path length of a state with a size of 17 must be larger than or equal to 36. The heuristic is therefore admissible because it never overrates the expected number of queries.

Table 6. The volumes of 14 classes in AB game

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]
volume	1	24	6	72	180	8	216	720	480	9	264	1260	1440	360

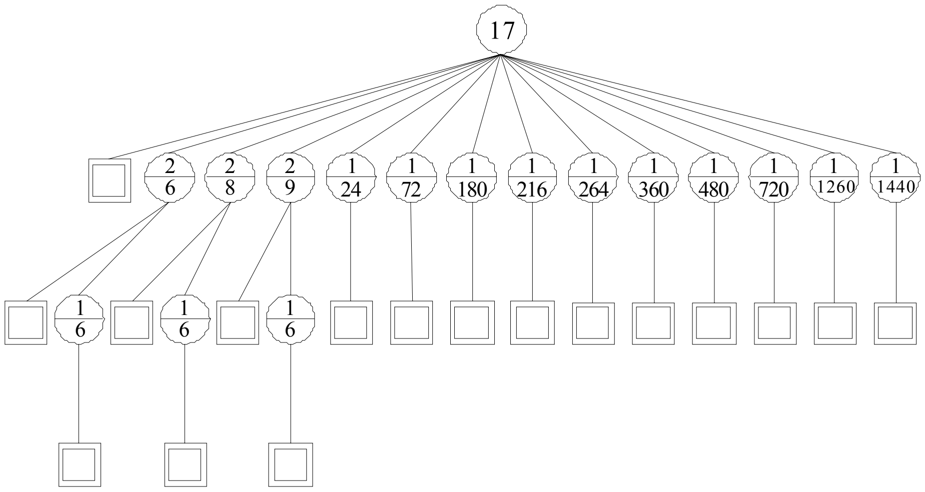


Figure 9. An example of the calculation of the admissible heuristic for AB game

3.2.2 The State-of-the-Art Techniques

The fundamental framework has been reviewed in Section 3.2.1. It has been proven dramatically that the algorithm is highly suitable for addressing deductive games. However, it is not enough to handle AB game in the expected case. Some attributions of the game are observed seriously so that three critical challenges are summarized as follows.

- How to increase the precision of the lower bound?
- How to avoid expanding the redundant states?
- How to prune the equivalent queries?

An optimal strategy will be discovered providing that these challenges are able to be coped with totally. Fortunately, a *refined branch-and-bound algorithm with speed-up techniques* (RBB) is designed and three useful techniques contained in it are introduced and discussed among the follow-up contents.

3.2.2.1 Technique 1: Incremental Updates of the Lower Bounds

During the gaming process, there will be generally 5040 queries for the codebreaker in each ply. When a new state is met, a current best solution s is acquired after DBB undertakes a search to one of the 5040 branches. Thus, DBB has to check other queries and two possible cases are going to take place. One case is that the rated lower bound of the query is less than s, and then the search into it occurs. The other case is that the search will be omitted according to branch-and-bound pruning because s outperforms this rated lower bound. Obviously, this mechanic of the process comes up with a new idea naturally. The percentage of the cutoffs is going to increase markedly if the estimated lower bounds become higher by calculating it more accurately. Concrete steps are offered below.

Suppose that the current best solution s is provided by the query g. There is another query called g^* that we analyze now and moreover, s^* refers to the lower bound which has been rated by the admissible heuristic H at the beginning. Assume that s^* is less than s. It is clear that the subtree yielded by g^* has to be explored in accordance with our proposed manner. However, we come up with an idea to update the lower bound incrementally during the exploring process of g^* so as to stop searching as soon as possible providing that s^* becomes equal to or larger than s. In the detailed considerations, g^* divides the current state into 14 classes (responses) so that H is able to rate its external path length (EPL) with the 14 classes. Hence, s^* is summed with the 14 rated numbers. When every class has been traversed, a real EPL of this class is available as well. Once a real cost of exploring the class has been acquired, an update to s^* happens immediately. Furthermore, s^* grows gradually as we explore these classes one by one.

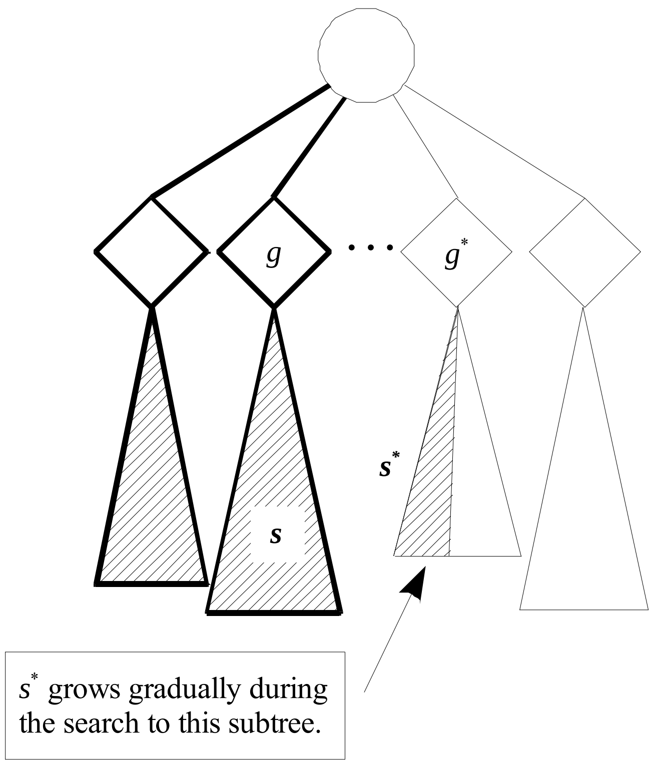


Figure 10. A situation that depicts the exploring process

When an update happens, s competes with the up-to-date s^* at the same time. The exploring process of g^* stops if s^* is equal to or larger than s. Otherwise, it keeps on working until the subtree formed by g^* is searched entirely. And the follow-up actions are performed with the use of DBB as usual. A situation that depicts the searching process is shown in Figure 10. Meanwhile, the bold lines and shaded areas highlight whatever has already been searched and s^* is the latest lower bound until now.

3.2.2.2 Technique 2: Earlier Terminations

It is trivial that the game is over if there exists only one choice for the codebreaker and he has just figured it out. It is also clear that the searching process should be terminated if we are aware of the external path length (EPL) of some states precisely. Accordingly, a critical issue for obtaining the exact EPL of some states has arisen. It is highly difficult to know the exact EPL without conducting a search when the state is larger. In this case, there is a chance to get it more early only if the state is smaller enough. In order to cope with this, two types of pruning methods are proposed to achieve the goal of earlier terminations if the size of a state is below 12.

■ Theoretical pruning

If the size of a state is 2, it is easy to notice that the game tree in Figure 11 is optimal and its EPL is therefore 3.

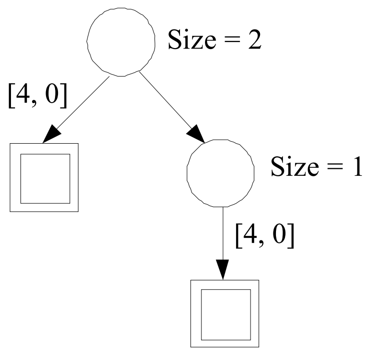


Figure 11. An optimal strategy for a state with a size of 2

On the other hand, the size of a state is 3 is then taken into account. We notice that two situations occur. One is that the tactic for this state chooses one of these three eligible codes as the next query. This will result in a [4, 0] class appearing in its game tree. The left portion, i.e. (a) and (b), of Figure 12, in which there exist two kinds of possible trees, indicates the phenomenon. The other situation is also offered in the right portion, i.e. (c), (d), and (e), of Figure 12, where there are three possibilities in addition. The right part implies that the codebreaker chooses one query from all possible codes except the three ones in this state. Note that the scenario of (e) describes that the size of the state still remains 3 after the query in this ply is taken, where EPL' is the external path length of the following state. In other words, there is no use making this query but to increase its EPL by 3 in addition.

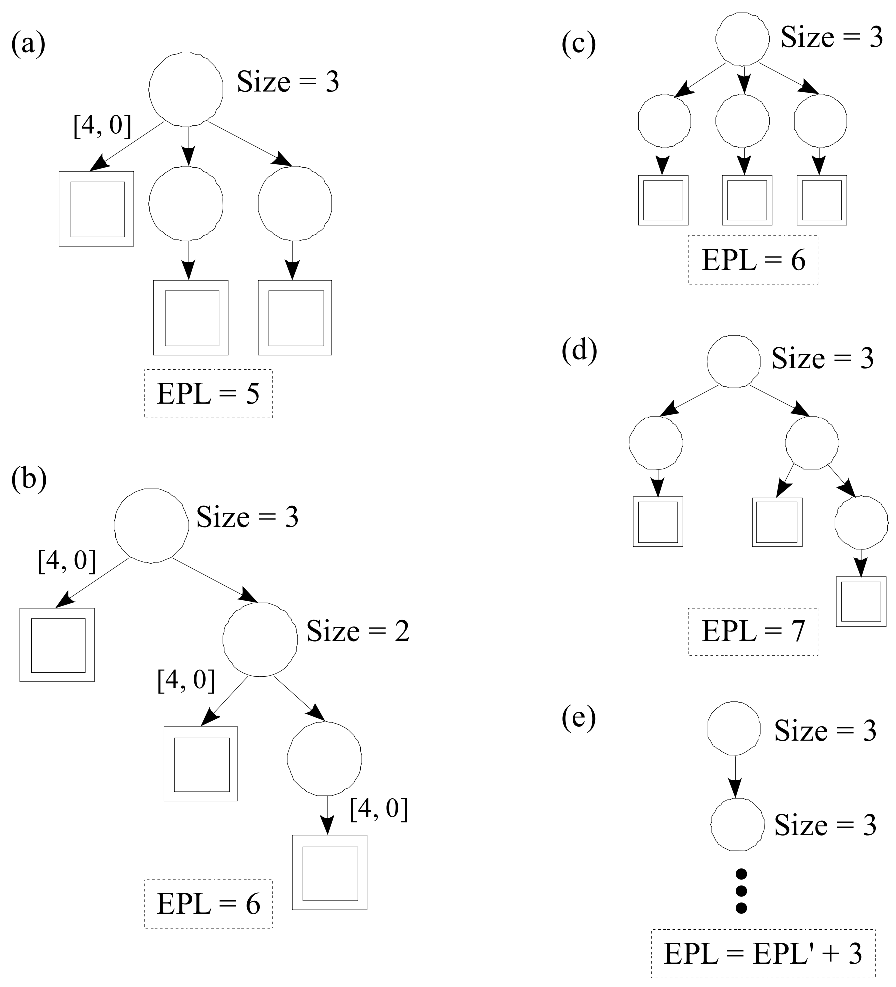


Figure 12. All possible game trees for a state with a size of 3

By perceiving the overall figure, the EPLs for the left trees are 5 and 6 respectively while those of the right ones are 6 and above. It means that an optimal strategy will be generated only by taking the left two trees into account. In this case, the correlations among the three eligible codes should be considered. Assume that the three queries (secret codes) are named as g_1, g_2, and g_3, where their correlations are r_{12}, r_{23}, and r_{31} respectively. The correlation here indicates the response made by the codemaker providing that one of these three queries is his secret code when the codebreaker takes another query from the two residual codes. Note that the optimal EPL for a state with a size of 3 is 6 if r_{12}, r_{23}, and r_{31} are all equal, i.e. the situation of Figure 12(b). Otherwise, the optimal EPL must be 5 as shown in Figure 12(a). With this observation, the optimal EPL can be easily calculated without searching all the 5040 valid queries.

From the above theoretical analyses, we know that the EPL of a state can be easily determined if the state is able to be analyzed. In other words, theoretical pruning of valid queries is feasible if the size of a state is 2 or 3.

- Practical pruning

In accordance with the previous analyses, it seems to be intuitive that DBB will terminate and backtrack earlier if the optimal EPL can be decided as earlier as possible. Furthermore, a crucial property is realized by investigating the game tree when the size is sufficiently small. It reveals that the full game tree is filled with duplicated states with smaller sizes. This discovery comes up with a good idea which is able to reduce the searching time by storing the EPLs of explored states, whose size are between 4 and 12. By utilizing the concept, a hash table is implemented naturally to meet the requirement. The Zobrist hashing approach [74] is adopted as a hash function and a simple replacement method, in which a new record just replaces the value that is already in the corresponding slot, is employed to resolve collisions. Due to the low collisions of the Zobrist hashing method, the simple replacement policy is highly efficient for our problem. Before the use of the Zobrist hashing method, a random number is generated for each possible secret code and represents this corresponding code in the searching process. Suppose that we now have a state with *n* eligible codes, where the value of *n* is between 4 and 12. All corresponding random numbers of the *n* codes are XORed together and the result modulo the size of the hash table is computed to acquire a hash key, which represents the corresponding position for storing this state in the hash table. So, the information of the state and its corresponding optimal EPL are stored in this position after the state has been explored. Once a collision occurs, the new record just replaces the old one that is already in the corresponding position. The EPLs are going to be looked up in the hash table when new states are encountered. Although the hash table is designed in a basic manner, it has contributed substantial performance improvements. The experimental results will clarify the progress in the later discussions. Note that the hash table occupies about 1.6 Gbytes memory because it has 2^{25} entries and each entry contains 13 integers (one for storing the EPL and 12 for keeping the 12 secret codes at most). From an informal test, the performance is better if an entry stores the state whose size is at most 12. Remember that the larger the state which is stored in an entry of the hash table is, the more time our program should take if the program has to decide whether the current state is traversed or not.

Due to the huge number of codes in larger states and the huge amount of memory space for storing the larger states and their EPLs, the states whose sizes go above and beyond 12 will not be held in the hash table. This implies that a normal search is carried out to them.

3.2.2.3 Technique 3: Reductions of Equivalent Queries

The technique introduced in Section 3.2.2.2 focuses on pruning the leaves in the game tree. Moreover, an overall subtree will be cut thoroughly if we can reduce the number of the choices the codebreaker has to concern about at the first few queries. At the first query, only one choice, instead of 5040 secret codes, should be considered because there are no symbols that are used before and the 5040 codes are therefore all equivalent. “0123” is adopted as the first query here. At the second query, only four out of ten symbols are used by the first query and the other six unused symbols (4, 5,.. , and 9) can be treated as equivalent ones. Hence, at the second query, we can only consider $\sum_{i=0}^{4} C(4,i) \times P(4,i) = 209$ nonequivalent codes, where i is the number of symbols used at both the first and the second queries. Furthermore, 20 equivalent sets that come from further categorizing the 209 codes are gained by employing the concept of *equivalence transformation* demonstrated by Neuwirth [53]. An *equivalence transformation* is defined as a composition of a permutation on the set of colors, called *C*, and a permutation on the set of positions, called *P*. For instance, an *equivalence transformation t* is defined as follows:

$$C = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 0 & 3 & 1 & 2 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}, \quad P = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 3 & 1 & 2 \end{pmatrix}.$$

Then, the query, “0132”, is equivalent to “0213” while $t(0132) = 0213$. Furthermore, a crucial property, *strategy equivalent*, which can help us to reduce the search space, is described in Definition 11 of Chapter 1. In other words, we can only take p_1 as a representative for computing an optimal strategy if p_1, p_2, …, p_n are strategy equivalent. The equivalent secret codes at the first three queries can be obtained by using exhaustive search to verify each possible strategy or making use of the algebra package, Nauty, which is a program based on the paper introduced by McKay [50] for computing isomorphism and automorphism groups of graphs.

Table 7, in which each row stands for an equivalent set, lists all of the 209 codes and their categories with the use of the concept of strategy equivalent codes at the second query. Their corresponding equivalence transformations are attached at Appendix A as a proof. The equivalence transformations of strategy equivalent codes at the third query are not listed due to space restrictions. In Table 7, the first code in every row, which is highlighted with bold letters, is chosen as the representative since the codes in the same row are all strategy equivalent.

Table 7. 20 equivalent sets of the 209 codes at the second query

Order	Elements of each set
1	**0123**
2	**0132**, 0213, 0321, 1023, 2103, 3120
3	**4567**
4	**0231**, 0312, 1203, 1320, 2013, 2130, 3021, 3102
5	**1032**, 2301, 3210
6	**1230**, 1302, 2031, 2310, 3012, 3201
7	**0124**, 0143, 0423, 4123
8	**0456**, 4156, 4526, 4563
9	**0145**, 0425, 0453, 4125, 4153, 4523
10	**1456**, 2456, 3456, 4056, 4256, 4356, 4506, 4516, 4536, 4560, 4561, 4562
11	**0134**, 0142, 0243, 0324, 0413, 0421, 1423, 2143, 3124, 4023, 4103, 4120
12	**0245**, 0345, 0415, 0435, 0451, 0452, 1425, 1453, 2145, 2453, 3145, 3425, 4025, 4053, 4105, 4135, 4150, 4152, 4253, 4325, 4503, 4513, 4520, 4521
13	**1045**, 2405, 3450, 4215, 4351, 4532
14	**1245**, 1345, 1405, 1450, 2045, 2415, 2435, 2450, 3045, 3405, 3451, 3452, 4015, 4051, 4205, 4235, 4251, 4315, 4350, 4352, 4502, 4512, 4530, 4531
15	**1435**, 1452, 2345, 2451, 3245, 3415, 4035, 4052, 4250, 4305, 4501, 4510
16	**0214**, 0341, 0432, 1024, 1043, 2104, 2403, 3140, 3420, 4132, 4213, 4321
17	**0234**, 0241, 0314, 0342, 0412, 0431, 1243, 1324, 1403, 1420, 2043, 2134, 2140, 2413, 3024, 3104, 3142, 3421, 4013, 4021, 4102, 4130, 4203, 4320
18	**1034**, 1042, 1432, 2304, 2341, 2401, 3214, 3240, 3410, 4032, 4210, 4301
19	**1234**, 1240, 1304, 1342, 1402, 1430, 2034, 2041, 2314, 2340, 2410, 2431, 3014, 3042, 3204, 3241, 3401, 3412, 4012, 4031, 4201, 4230, 4302, 4310
20	**1204**, 1340, 2014, 2430, 3041, 3402, 4231, 4312

In spite of declining the branching factors of the game tree at the first three queries, there are still numerous choices in the following plies. A similar mean is thereby going to be provided so as to reduce the possibilities at the fourth query. All of the unused symbols during the first three queries are treated as the same one. In short, those choices, in which a few digits contain the same used symbols in the same digits and the rest digits are composed of other unused symbols, are equivalent definitely.

For example, providing that the first three queries are “0123”, “1045”, and “1758” respectively, both “38**69**” and “38**96**” will belong to the same equivalent set because 6 and 9 are not used in the previous queries. With this simple concept, the number of the fourth query will be declined extremely in average and the concrete results are going to be presented in the next section as well.

3.3 Experimental Results and Discussions

This section presents two parts of outcomes which result from the proposed techniques. The first part is to exhibit the individual effects according to each of the three techniques. The second one will make a comparison between DBB and RBB by applying them separately to Mastermind, which has a much smaller search space. This may indicate how efficient our new method is. Finally, AB game is drawn on it and we thereby attain the success in finding the optimal tactic in the expected case. Notice that all the experiments are run on a DELL Precision 7400 Workstation equipped with a Quad-Core Intel Xeon X5450 CPU and 8 Gbytes RAM. Only a single core is utilized at a time due to the sequential programs regardless of a multi-core CPU.

3.3.1 The Effects of the Three Useful Techniques

Table 8 shows the effects with the incremental updates of the lower bounds. Note that

Technique 2 and Technique 3 are integrated into the programs either with or without Technique 1 in this experiment in order to save an enormous amount of running time. The left column in Table 8 indicates the selected states with some proper sizes after making the first query for AB game.

For example, $C_{[3, 0]}$ refers to the state after the codemaker replies [3, 0] and $|C_{[3, 0]}|$ is consequently the number of its eligible codes. The right two columns present the running time required to traverse each subtree of the corresponding state either with or without the use of Technique 1. It is obvious that the speedup of Technique 1 is by a factor of about 5 for some larger states, e.g. $C_{[1, 1]}$, $C_{[1, 0]}$ and $C_{[0, 2]}$. It also shows that the larger the size of the state is, the higher the speedup is.

Table 8. The running time of exploring some states after making 1st query

Size of the state	**Without Technique 1 (Seconds)**	**With Technique 1 (Seconds)**
$|C_{[2, 1]}| = 72$	20	11
$|C_{[2, 0]}| = 180$	121	41
$|C_{[1, 2]}| = 216$	357	120
$|C_{[1, 1]}| = 720$	47645	8386
$|C_{[1, 0]}| = 480$	3001	518
$|C_{[0, 3]}| = 264$	1361	317
$|C_{[0, 2]}| = 1260$	1616865	148592

An evaluation to Technique 2 is depicted in Table 9 and Table 10. The statistics of counting up the numbers of the descendant states with a size of 3 for some states are conducted within Table 9. For instance, there are totally 418161 descendant states that contain 3 eligible codes while a search to $C_{[1, 2]}$ has been undertaken. The result represents the numbers of the cases that Technique 2 can be applied, so the optimal tactic for these states will be gained quickly and easily. This means we can save much time because we do not examine all the 5040 choices in the next ply. On the other hand, the hash table demonstrated in Technique 2 may sometimes be viewed as a cache and detailed information to its performance is thus listed in Table 10.

Table 9. The numbers of the descendant states with a size of 3

States	# of the descendant states with a size of 3
$C_{[3,0]}$	4540
$C_{[2,1]}$	28474
$C_{[2,0]}$	12042
$C_{[1,2]}$	418161
$C_{[1,0]}$	91826
$C_{[0,3]}$	1627148

Note that the corresponding random number for each secret code is 64 bit and the size of the hash table is 2^{25} in our implementation. The state, $C_{[1,2]}$, is taken to serve as an example for the illustration. It means that 366621 states whose sizes vary from 4 to 12 are able to be looked up in the table directly as 27375 ones are not available in it and they have to be explored thoroughly and then be inserted into the table. The hit rate, which is 366621 divided by (366621+27375), is thereby about 0.931. The hash table occupies about 1.6 Gbytes memory in our design and receives a considerable performance promotion.

Table 10. The numbers of hits and misses by using the hash table

States	# of hits	# of misses	Hit rate
$C_{[3,0]}$	3434	502	0.872
$C_{[2,1]}$	11863	1815	0.867
$C_{[2,0]}$	20350	8160	0.714
$C_{[1,2]}$	366621	27375	0.931
$C_{[1,0]}$	163347	106346	0.606
$C_{[0,3]}$	1698126	65811	0.963

The results shown in Table 11 provide the assessments of Technique 3. Recall that the numbers of the choices taken by the codebreaker in the first two plies are 1 and 20 respectively. Table 11 lists the numbers of the third choice and the average numbers of the fourth choice the codebreaker can make and meanwhile, the numbers for the first two choices are offered in it as well.

Table 11. The numbers of choices at the first four queries

1st query	2nd query	# of choices at the 3rd query	Average # of choices at the 4th query
0123	0123	20	852.85
0123	0124	107	1296.66
0123	0132	67	809.37
0123	0134	270	1255.17
0123	0145	295	1993.36
0123	0214	270	1255.17
0123	0231	75	790.99
0123	0234	501	1234.25
0123	0245	1045	1959.62
0123	0456	363	3020.61
0123	1032	39	807.97
0123	1034	270	1255.17
0123	1045	295	1993.36
0123	1204	175	1246.02
0123	1230	59	783.29
0123	1234	501	1234.25
0123	1245	1045	1959.62
0123	1435	541	1957.71
0123	1456	1012	3008.06
0123	4567	180	4162.67

From Table 11, it is easy to realize that the numbers of the third choice vary from 20 to 1045, i.e. 356.50 in average, which occupies 7.07% of the original 5040 choices. Moreover, it also shows that the average number of the fourth choice is 1643.81, which is 32.62% of the original 5040 choices. Therefore, a considerable amount of redundant choices at the first four queries has been removed with the use of Technique 3. In fact, the search space of the entire game tree has roughly become $(1\times14)\cdot(20\times14)\cdot(356.50\times14)\cdot(1643.81\times14)\cdot(5040\times14)^3$, where 356.50 and 1643.81 are the averages of the branching factors at the third and fourth queries respectively.

3.3.2 Performances and Results of RBB for Solving Mastermind and AB Game

In order to examine the performance, a deductive game with smaller dimensions, Mastermind, is first explored with the proposed approach composed of all significant refinements except Technique 3. Technique 3 can not be applied in Mastermind because repeated symbols are allowed in the queries of the game. The program relied upon RBB therefore finished the work with 43 minutes in the experiment and gained the optimal EPL that is 5625. This means that the expected number of queries of the optimal strategy in the expected case for Mastermind is $5625/6^4 \approx 4.34$. On the other hand, it has to take 451 minutes to complete the same work with the use of DBB. Hence, RBB outperforms DBB by 10 times faster only with the first two techniques. Note that DBB explores 137834651 states during the searching process while RBB only expands 31720272 states.

With the success, RBB integrating all the three techniques to find the optimal tactic of AB game in the average case was thereby undertaken. An invaluable result was eventually gained in about 18 days (From Nov. 6, 2008 to Nov. 23, 2008). The optimal strategy for AB game was therefore obtained and its corresponding EPL is 26274. Moreover, a partial strategy is also presented in Appendix B. Now we have the following theorem.

Theorem 1. The expected number of queries of the optimal strategy in the expected case for AB game is $26274/5040 \approx 5.213$.

3.4 Chapter Conclusion

In this chapter, we focus on finding the optimal strategy in the expected case for AB game. An elegant approach, which is named as *refined branch-and-bound algorithm with speed-up techniques* (RBB), essentially based on the incremental update of lower bounds, the hashing technique, and the reduction of equivalent queries is designed to explore its huge search space. In the development of pruning techniques, we also realize that the ratio of pruning is significant if the pruning is based on theoretical analyses. In order to compare RBB with DBB, Mastermind is first addressed by applying these two methods individually. A dramatic improvement is exhibited in the outcomes and RBB outperforms DBB over 10 times faster. Fortunately, an optimal strategy for AB game in the expected case is eventually obtained by utilizing RBB. The corresponding external path length is 26274. In other words, the expected number of queries required by the codebreaker is 26274/5040 ≈ 5.213. Note that Appendix B attached at the end of this dissertation contains the partial optimal strategy for AB game, which is discovered by RBB.

Chapter 4

Structural-reduction Approach

In this chapter, a sophisticated method, called *structural-reduction approach* (SR), which aims at explaining the worst situation in 3×*n* AB games is developed. Section 4.1 introduces our addressed problem and additional definitions that are used in this chapter. Section 4.2 analyzes the optimal strategies for the codebreaker and the devil's strategy for the codemaker. In Section 4.3, a practical example is offered to describe the pessimistic situation of this game. Section 4.4 concludes with our analyses and a worthwhile formula for calculating the optimal numbers of queries required for arbitrary values of *n* is derived and proven finally.

4.1 Introduction

3×*n* AB games means that there are 3 digits in a single secret code and each digit has *n* possibilities (symbols). Suppose that the set of symbols appearing in 3×*n* AB games is $S = \{0, 1, 2, \ldots, n - 1\}$. From the analyses of Chapter 1, the number of all legal responses is 9 and these responses are [3, 0], [2, 0], [1, 2], [1, 1], [1, 0], [0, 3], [0, 2], [0, 1], and [0, 0] respectively. Meanwhile, the number of all possible secret codes equals to $n(n - 1)(n - 2)$ as well. For example, assume that the codemaker chooses c = 215 as a secrete code and the codebreaker makes a query g = 012. Then, the codemaker will offer a response [1, 1].

Before 3×*n* AB games are discussed formally, some additional definitions besides those offered in Chapter 1 have to be explained first in order to describe the analyses precisely. Thus, they are defined as follows.

Definition 12. Let C_1 and C_2 denote two states in the game tree. We say that C_1 is *harder* than C_2 if identifying a secret code in C_1 requires more queries than that in C_2. In other words, the *difficulty* of a state means how many queries the codebreaker requires to identify a secret code.

Definition 13. A strategy of responses taken by the codemaker is called a *devil's strategy* or an *adversary response* if this strategy maximizes the number of queries required by the codebreaker.

Definition 14. Suppose that there are two states, which are C_1 and C_2 respectively. If there exists a one-to-one function *r* such that each secret code in C_1 maps another one in C_2 and preserves the structure of C_1, then we say that C_2 dominates C_1. Furthermore, *r* is called a *structural reduction*. In symbols, we write $C_1 \leq C_2$.

Now, 3×5 AB game is taken into account as an illustrative example. Suppose that the set of five symbols in this simple game is $S = \{0, 1, 2, 3, 4\}$. If the codebreaker makes a query, 012, and the codemaker responses [2, 0] in the first ply, the eligible codes are therefore 013, 014, 032, 042, 312, and 412 after the first ply. The set $C_{[2,0]} = \{013, 014, 032, 042, 312, 412\}$ forms a state. From the result of the later experiment, which conducts an exhaustive search to 3×5 AB game, the number of queries required is maximum if the codemaker implements a devil's strategy to provide the response, [0, 2], at the first response.

On the other hand, $C_{[2,0]}$ and the state, $C_{[1,0]} = \{043, 034, 432, 342, 314, 413\}$, which is produced when the codemaker responses [1, 0] at the first response, are then considered. Notice that the elements in $C_{[2,0]}$ are of the forms, $01b$, $0b2$, or $b12$, where $b \in B = \{3,4\}$. Thus, we define a structural reduction of r as

$$r : \begin{cases} 01b \mapsto 0zb \\ 0b2 \mapsto zb2 \\ b12 \mapsto b1z \end{cases} \text{, where } b \in B \text{ and } z \in B - \{b\}.$$

Figure 1 exhibits the mapping of each code in $C_{[2,0]}$ in detail. Note that the mapped codes in $C_{[1,0]}$ preserve the structures of those in $C_{[2,0]}$. This implies that finding a secret code in $C_{[1,0]}$ is as hard as or harder than that in $C_{[2,0]}$. Intuitively, this is also obvious since there is one more identified symbols in $C_{[2,0]}$ than in $C_{[1,0]}$. Hence, we say that $C_{[1,0]}$ dominates $C_{[2,0]}$. Furthermore, the structural reduction has the property of the transitive relation obviously. That is to say that given three states, C_1, C_2, and C_3, $C_1 \leq C_3$ if $C_1 \leq C_2$ and $C_2 \leq C_3$.

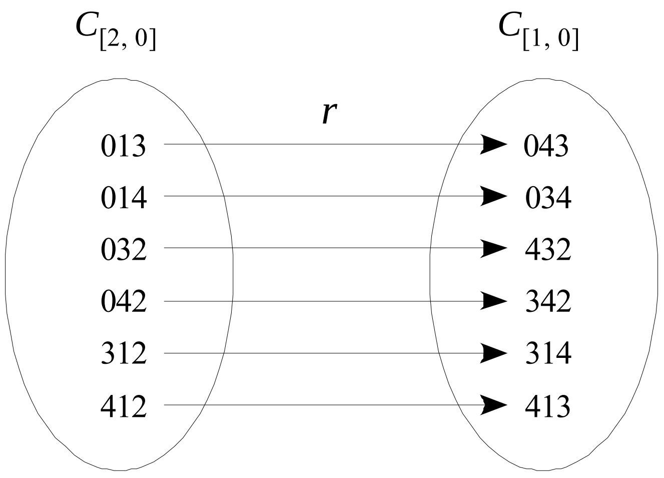


Figure 13. Mapping from codes in $C_{[2,0]}$ to those in $C_{[1,0]}$ for 3×5 AB game

4.2 Optimal Analyses for the Codebreaker and the Codemaker

In this section, we divide the analyses into two parts. The first part discuss a special kind of states C^* that will be considered to determine the best query for the codebreaker when he encounters this kind of states. Then, the discussion in the next part will reveal that the special states that are discussed here just match the attribution of states resulting from the devil’s strategy for the codemaker. Consequently, our conclusions are attained finally.

4.2.1 Analyses of the Optimal Queries for the Codebreaker

Before the formal discussion, a critical concept should be clarified first. Intuitively, the more secret codes a state has, the harder the codebreaker identifies a secrete code in it. However, the rule is not absolutely correct especially when the size of one state is very close to that of the other. Hence, the structural reduction is adopted to determine the difficulties of two states instead of simply comparing their sizes in the following discussion.

Suppose that $S = \{0, 1, 2, ..., n - 1\}$ represents the set of symbols appearing in $3 \times n$ AB games. The set, $B = \{b_0, b_1, ..., b_{h-1}\}$, is a subset of S, where $b_i \in S$ and $|B| = h$, $3 \leq h \leq n - 3$. Moreover, another set, A, is defined as $A = S - B = \{a_0, a_1, ..., a_{n-h-1}\}$, whose cardinality is $(n - h)$.

Assume that there is a special state, called C^*, which consists of the secret codes that are all possible permutations of h symbols in B. In other words, the special state has $h(h - 1)(h - 2)$ secret codes in it. This state may be regarded as a subproblem of a $3 \times n$ AB game, i.e. a $3 \times h$ AB game. Notice that the symbols in A do not appear in the codes of the special state because of the definition of C^*. We can intuitively treat the symbols in A as those eliminated from previous responses made by the codemaker.

Now, imagine a scenario where C^* is encountered for the codebreaker during the process of playing a $3 \times n$ AB game. Since any symbols in S may be used in a query made by the codebreaker for a $3 \times n$ AB game, all possible queries for the codebreaker can be classified into four types according to the numbers of symbols that belong to A and B. Thus, the four types of queries for the codebreaker are listed and discussed as follows. Here we suppose that a_i, a_j, $a_k \in A$ and b_i, b_j, $b_k \in B$.

1. $a_ia_ja_k$

   All symbols of this type of queries belong to A. If the codebreaker makes this kind of queries, all eligible codes are then classified into the substate, $C_{[0,0]}$, trivially. So, the queries of Type 1 are redundant and non-optimal results will be obtained if the codebreaker chooses this kind of queries.

2. $b_ka_ia_j$, $a_ib_ka_j$, and $a_ia_jb_k$

   The queries of Type 2 contain two symbols in A and one symbol in B. This type of queries can be further divided into three kinds of queries such as $b_ka_ia_j$, $a_ib_ka_j$, and $a_ia_jb_k$ in accordance with their positions of symbols. Without loss of generality, $g = b_ka_ia_j$ is taken to conduct the following analyses. The discussions of the other two can be undertaken in a similar way. Three nonempty substates, which are $C_{[1,0]}$, $C_{[0,1]}$, and $C_{[0,0]}$, are produced as the codebreaker makes the query g. Note that their cardinality are $(h-1)(h-2)$, $2(h-1)(h-2)$, and $(h-1)(h-2)(h-3)$ respectively. Now, we can show that $C_{[0,1]} \leq C_{[0,0]}$ and $C_{[1,0]} \leq C_{[0,0]}$ if $h \geq 5$.

   Lemma 1. If the codebreaker encounters the state, C^*, and then makes the query, $g = b_ka_ia_j$, $a_ib_ka_j$, or $a_ia_jb_k$, where a_i, $a_j \in A$ and $b_k \in B$, then $C_{[0,0]}$ dominates $C_{[0,1]}$ and $C_{[1,0]}$ if $h \geq 5$.

   Proof. In order to prove that $C_{[0,1]} \leq C_{[0,0]}$, a structural reduction, r_1, is defined as

$$r_1 : \begin{cases} b_pb_kb_q \mapsto b_pz_1b_q \\ b_pb_qb_k \mapsto b_pb_qz_2 \end{cases} \text{, where } b_p, b_q \in B' = B - \{B_k\} \text{ and } z_1, z_2 \in B' - \{b_p, b_q\}.$$

   From r_1, it reveals that the structures of the secret codes, which are $b_p?b_q$ and $b_pb_q?$, are preserved after mapping. Note that $b_pz_1b_q$ and $b_pb_qz_2$ should be distinct to reserve the property of one-to-one mapping. We can achieve this by assigning the symbols of z_1 and z_2 carefully while mapping is conducted. On the other hand, there should be two symbols left for the assignments of z_1 and z_2 once b_p and b_q have been fixed during the mapping. The proof is therefore correct if $h \geq 5$. The proof of $C_{[0,1]} \leq C_{[0,0]}$ is finished now. Afterwards, another structural reduction, r_2, is defined as

$$r_2 : b_k b_p b_q \mapsto z_1 b_p b_q, \text{ where } b_p, b_q \in B' = B - \{ b_k \} \text{ and } z_1 \in B' - \{ b_p, b_q \}.$$

There should be one symbol left for the assignment of z_1 once b_p and b_q have been assigned. Hence, the proof is right if $h \geq 4$. In other words, $C_{[1,0]} \leq C_{[0,0]}$. From the results of r_1 and r_2, we know that $C_{[0,0]}$ dominates $C_{[0,1]}$ and $C_{[1,0]}$ when $h \geq 5$. This completes the proof of Lemma 1. □

3. $a_ib_jb_k$, $b_ja_ib_k$, and $b_jb_ka_i$

The queries of this type are composed of a symbol in A and two symbols in B. These queries can also be further classified into three kinds of queries, i.e., $a_ib_jb_k$, $b_ja_ib_k$, and $b_jb_ka_i$. Without loss of generality, $g = a_ib_jb_k$ is choosen to undertake the following discussions. Besides, the analyses of $b_ja_ib_k$ and $b_jb_ka_i$ can be derived in a similar way and so, they are omitted here. There are six nonempty substates after the codebreaker makes the query g. They are $C_{[2,0]}$, $C_{[1,1]}$, $C_{[0,2]}$, $C_{[1,0]}$, $C_{[0,1]}$, and $C_{[0,0]}$ respectively. Note that their corresponding cardinality are $(h - 2)$, $2(h - 2)$, $(h - 2)$, $2(h - 2)(h - 3)$, $4(h - 2)(h - 3)$, and $(h - 2)(h - 3)(h - 4)$. Now, we show that $C_{[0,0]}$ dominates the other five substates if $h \geq 8$.

Lemma 2. If the codebreaker encounters C^*, and then makes the query, $g = a_ib_jb_k$, $b_ja_ib_k$, or $b_jb_ka_i$, where $a_i \in A$ and b_j, $b_k \in B$, then $C_{[0,0]}$ dominates $C_{[0,1]}$, $C_{[1,0]}$, $C_{[0,2]}$, $C_{[1,1]}$, and $C_{[2,0]}$ when $h \geq 8$.

Proof. Five structural reductions, called r_3, r_4, r_5, r_6, and r_7, are defined as follows to certify that $C_{[0,1]} \leq C_{[0,0]}$, $C_{[1,0]} \leq C_{[0,0]}$, $C_{[0,2]} \leq C_{[0,1]}$, $C_{[1,1]} \leq C_{[1,0]}$, and $C_{[2,0]} \leq C_{[1,0]}$ respectively.

$$r_3 : \begin{cases} b_j b_p b_q \mapsto z_1 b_p b_q \\ b_p b_q b_j \mapsto b_p b_q z_2 \\ b_k b_p b_q \mapsto z_3 b_p b_q \\ b_p b_k b_q \mapsto b_p z_4 b_q \end{cases} \text{, where } b_p, b_q \in B' = B - \{b_j, b_k\} \text{ and } z_1, z_2, z_3, z_4 \in B' - \{b_p, b_q\}.$$

$$r_4 : \begin{cases} b_p b_j b_q \mapsto b_p z_1 b_q \\ b_p b_q b_k \mapsto b_p b_q z_2 \end{cases} \text{, where } b_p, b_q \in B' = B - \{b_j, b_k\} \text{ and } z_1, z_2 \in B' - \{b_p, b_q\}.$$

$$r_5 : \begin{cases} b_j b_k b_p \mapsto b_j z_1 b_p \\ b_p b_k b_j \mapsto b_p z_2 b_j \\ b_k b_p b_j \mapsto b_k b_p z_3 \end{cases} \text{, where } b_p \in B' = B - \{b_j, b_k\} \text{ and } z_1, z_2, z_3 \in B' - \{b_p\}.$$

$$r_6 : \begin{cases} b_k b_j b_p \mapsto z_1 b_j b_p \\ b_j b_p b_k \mapsto z_2 b_p b_k \end{cases} \text{, where } b_p \in B' = B - \{b_j, b_k\} \text{ and } z_1, z_2 \in B' - \{b_p\}.$$

$$r_7 : b_p b_j b_k \mapsto b_p b_j z_1 \text{, where } b_p \in B' = B - \{b_j, b_k\} \text{ and } z_1 \in B' - \{b_p\}.$$

Note that $z_1b_pb_q$, $b_pb_qz_2$, $z_3b_pb_q$, and $b_pz_4b_q$ in r_3 should be distinct to reserve the one-to-one mapping property. Likewise, $b_pz_1b_q$ and $b_pb_qz_2$ in r_4 should be distinct and $b_jz_1b_p$, $b_pz_2b_j$, and $b_kb_pz_3$ in r_5 should also be distinct while $z_1b_jb_p$ and $z_2b_pb_k$ in r_6 have to be distinct as well. We can attain this with assigning these symbols of z_1, z_2, z_3, and z_4 carefully when mapping is undertaken. In order to meet requirements of the assignments of z_i in r_3, r_4, r_5, r_6, and r_7, the following conditions should be maintained respectively: $h \geq 8$, $h \geq 6$, $h \geq 6$, $h \geq 5$, and $h \geq 4$. Consequently, it is true that $C_{[0,0]}$ dominates $C_{[0,1]}$, $C_{[1,0]}$, $C_{[0,2]}$, $C_{[1,1]}$, and $C_{[2,0]}$ while $h \geq 8$. Hence, the proof of Lemma 2 is completed. □

4. $b_ib_jb_k$

All symbols of this kind of queries belong to B entirely. There are totally nine nonempty substates, which are $C_{[3,0]}$, $C_{[1,2]}$, $C_{[0,3]}$, $C_{[2,0]}$, $C_{[1,1]}$, $C_{[0,2]}$, $C_{[1,0]}$, $C_{[0,1]}$, and $C_{[0,0]}$ respectively, as the codebreaker makes the query, $g = b_ib_jb_k$. Notice that their cardinality are 1, 3, 2, $3(h-3)$, $6(h-3)$, $9(h-3)$, $3(h-3)(h-4)$, $6(h-3)(h-4)$, and $(h-3)(h-4)(h-5)$ respectively. In the following statements, we would certify that $C_{[0,1]} \le C_{[0,0]}$, $C_{[1,0]} \le C_{[0,0]}$, $C_{[0,2]} \le C_{[0,1]}$, $C_{[1,1]} \le C_{[1,0]}$, $C_{[2,0]} \le C_{[1,0]}$, $C_{[0,3]} \le C_{[0,0]}$, $C_{[1,2]} \le C_{[0,0]}$, and $C_{[3,0]} \le C_{[0,0]}$.

Lemma 3. As the codebreaker encounters C^*, and then makes the query, $g = b_ib_jb_k$, where b_i, b_j, $b_k \in B$, then $C_{[0,0]}$ dominates $C_{[0,1]}$, $C_{[1,0]}$, $C_{[0,2]}$, $C_{[1,1]}$, $C_{[2,0]}$, $C_{[0,3]}$, $C_{[1,2]}$, and $C_{[3,0]}$ when $h \ge 11$.

Proof. Since the cardinalities of $C_{[3,0]}$, $C_{[1,2]}$, and $C_{[0,3]}$ are fixed numbers, then $C_{[0,0]}$ trivially dominates $C_{[3,0]}$, $C_{[1,2]}$, and $C_{[0,3]}$ as long as there are at least three symbols in B and thus, the three symbols can be permuted appropriately to map the three substates. On the other hand, five definitions of structural reductions, which are named as r_8, r_9, r_{10}, r_{11}, and r_{12}, are provided as follows to confirm that $C_{[0,1]} \le C_{[0,0]}$, $C_{[1,0]} \le C_{[0,0]}$, $C_{[0,2]} \le C_{[0,1]}$, $C_{[1,1]} \le C_{[1,0]}$, and $C_{[2,0]} \le C_{[1,0]}$ respectively.

$$r_8 : \begin{cases} b_pb_ib_q \mapsto b_pz_1b_q \\ b_pb_qb_i \mapsto b_pb_qz_2 \\ b_jb_pb_q \mapsto z_3b_pb_q \\ b_pb_qb_j \mapsto b_pb_qz_4 \\ b_kb_pb_q \mapsto z_5b_pb_q \\ b_pb_kb_q \mapsto b_pz_6b_q \end{cases} \text{, where } b_p, b_q \in B' = B - \{b_i, b_j, b_k\} \text{ and } z_1, z_2, z_3, z_4, z_5, z_6 \in B' - \{b_p, b_q\}.$$

$$r_9 : \begin{cases} b_ib_pb_q \mapsto z_1b_pb_q \\ b_pb_jb_q \mapsto b_pz_2b_q \\ b_pb_qb_k \mapsto b_pb_qz_3 \end{cases} \text{, where } b_p, b_q \in B' = B - \{b_i, b_j, b_k\} \text{ and } z_1, z_2, z_3 \in B' - \{b_p, b_q\}.$$

$$r_{10} : \begin{cases} b_jb_ib_p \mapsto z_1b_ib_p \\ b_pb_ib_j \mapsto b_pz_1b_j \\ b_jb_pb_i \mapsto z_1b_pb_i \\ b_jb_kb_p \mapsto b_jz_1b_p \\ b_pb_kb_j \mapsto b_pb_kz_1 \\ b_kb_pb_j \mapsto b_kb_pz_1 \\ b_kb_ib_p \mapsto z_2b_ib_p \\ b_kb_pb_i \mapsto b_kb_pz_2 \\ b_pb_kb_i \mapsto b_pz_2b_i \end{cases} \text{, where } b_p \in B' = B - \{b_i, b_j, b_k\} \text{ and } z_1, z_2 \in B' - \{b_p\}.$$

$$r_{11}:\begin{cases} b_ib_pb_j \mapsto b_ib_pz_1 \\ b_ib_kb_p \mapsto b_iz_2b_p \\ b_pb_jb_i \mapsto b_pb_jz_1 \\ b_kb_jb_p \mapsto z_2b_jb_p \\ b_pb_ib_k \mapsto b_pz_1b_k \\ b_jb_pb_k \mapsto z_2b_pb_k \end{cases}, \text{ where } b_p \in B' = B - \{b_i, b_j, b_k\} \text{ and } z_1, z_2 \in B' - \{b_p\}.$$

$$r_{12}:\begin{cases} b_ib_jb_p \mapsto b_iz_1b_p \\ b_ib_pb_k \mapsto z_1b_pb_k \\ b_pb_jb_k \mapsto b_pb_jz_1 \end{cases}, \text{ where } b_p \in B' = B - \{b_i, b_j, b_k\} \text{ and } z_1 \in B' - \{b_p\}.$$

Note that each secret code in each structural reduction, i.e., r_8, r_9, r_{10}, r_{11}, and r_{12}, should be distinct from each other to reserve the one-to-one mapping property. This can be attained by assigning these symbols of z_1, z_2, z_3, z_4, z_5, and z_6 carefully. On the other hand, to satisfy each assignment of z_i in r_8, r_9, r_{10}, r_{11}, and r_{12}, the following constraints have to be kept respectively: $h \geq 11$, $h \geq 8$, $h \geq 6$, $h \geq 6$, and $h \geq 5$. So, it is therefore correct that $C_{[0,0]}$ dominates $C_{[0,1]}$, $C_{[1,0]}$, $C_{[0,2]}$, $C_{[1,1]}$, $C_{[2,0]}$, $C_{[0,3]}$, $C_{[1,2]}$, and $C_{[3,0]}$ when $h \geq 11$. Hence, the proof of Lemma 3 is completed. □

After four kinds of queries for the codebreaker are discussed, only three kinds of queries among them are useful since the first one causes non-optimal results trivially. In order to simplify the notations, let $C^{(2)}$, $C^{(3)}$, and $C^{(4)}$ denote the hardest states caused by queries of Type 2, Type 3, and Type 4 respectively. Hence, the difficulties of these three states have to be determined to choose the best query for the codebreaker. The following lemma therefore describes the phenomena.

Lemma 4. When the codebreaker encounters C^*, the hardest states caused by queries of Type 2, Type 3, and Type 4, i.e. $C^{(2)}$, $C^{(3)}$, and $C^{(4)}$, are produced. Thus, we have $C^{(4)} \leq C^{(3)} \leq C^{(2)}$.

Proof. From the meanings of $C^{(2)}$, $C^{(3)}$, and $C^{(4)}$, it reveals that $C^{(2)}$ is composed of secret codes that are permutations of $(h - 1)$ symbols, and $C^{(3)}$ consists of what are permutations of $(h-2)$ symbols while the codes in $C^{(4)}$ are permutations of $(h-3)$ symbols. Let $S^{(2)}$, $S^{(3)}$, and $S^{(4)}$ denote the sets of symbols appearing in $C^{(2)}$, $C^{(3)}$, and $C^{(4)}$ respectively. Then, let the symbols in $S^{(2)}$, $S^{(3)}$, and $S^{(4)}$ be sorted separately according to the lexicographical order. A mapping is generated naturally if we map each symbol in $S^{(4)}$ to that in $S^{(3)}$ one by one in sorted order. So does the mapping between $S^{(3)}$ and $S^{(2)}$. Obviously, we have $C^{(4)} \leq C^{(3)} \leq C^{(2)}$. This proof is completed entirely. □

Concluding with Lemma 1, Lemma 2, Lemma 3, and Lemma 4, we have the following lemma.

Lemma 5. For a special state, C^*, which also represents a $3\times h$ AB game ($11 \leq h \leq n$), the optimal query for the codebreaker now is $b_ib_jb_k$, where $b_i, b_j, b_k \in B$.

Proof. From Lemma 4, $C^{(4)}$ is the easiest state to identify a secret code compared to $C^{(2)}$ and $C^{(3)}$. The goal of the codebreaker is to minimize the number of queries required and so, the codebreaker has to choose the query which results in $C^{(4)}$ in the worst situation. The optimal query for the codebreaker is therefore $b_ib_jb_k$. □

4.2.2 The Devil's Strategy for the Codemaker

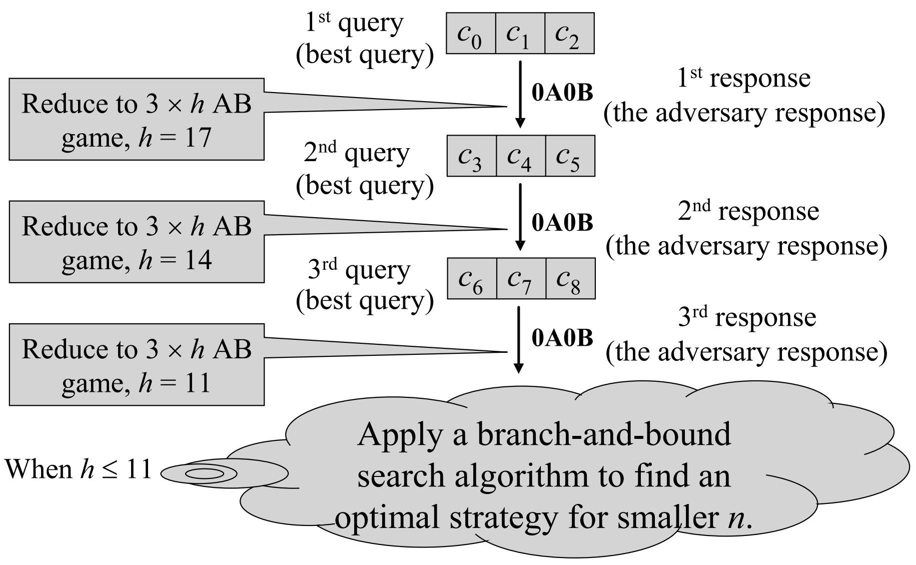
Since the mission of the codebreaker aims to minimize the number of queries to acquire a secret code, the codemaker tries to maximize the number of queries for the codebreaker if he decides to implement a devil's strategy. Hence, the worst case for the codebreaker means that his opponent conducts a devil's strategy (or called a worst response for the codebreaker) in each ply during the gaming process in order to maximize the number of queries. In the follow-up, a lemma is exhibited to demonstrate what is the worst response for the codebreaker if he encounters a $3\times h$ AB game, where $h \leq n$.

Lemma 6. For a $3\times h$ AB game, where $11 \leq h \leq n$, the codebreaker will require a maximum number of queries to get the code while the codemaker answers [0, 0] after the codebreaker's query.

Proof. From Lemma 5, it is obvious that the codebreaker must choose $b_i b_j b_k$ as a query for a 3×h AB game. After the codebreaker makes the optimal query, nine substates will be formed. These substates are $C_{[0,0]}$, $C_{[0,1]}$, $C_{[1,0]}$, $C_{[0,2]}$, $C_{[1,1]}$, $C_{[2,0]}$, $C_{[0,3]}$, $C_{[1,2]}$, and $C_{[3,0]}$ respectively. $C_{[0,0]}$ dominates $C_{[0,1]}$, $C_{[1,0]}$, $C_{[0,2]}$, $C_{[1,1]}$, $C_{[2,0]}$, $C_{[0,3]}$, $C_{[1,2]}$, and $C_{[3,0]}$ in accordance with the result of Lemma 3. In other words, $C_{[0,0]}$ is the hardest substate among the nine ones. Conclusively, the codemaker must response [0, 0] as his worst response and this will result in the worst case for the codebreaker because of the maximum number of queries. The proof is therefore finished. □

4.3 An Illustrative Example of the Pessimistic Situation

In order to clarify the key idea of the pessimistic situation (worst case) of 3×n AB games we have discussed above, a 3×20 AB game, which is a 3×n AB game while n = 20, is taken as an illustrative example. The scenario is shown in Figure 14. Suppose that the set of symbols is $S = \{c_0, c_1, \ldots, c_{19}\}$. In the first ply, the codebreaker makes the first query, $c_0c_1c_2$, and the codemaker offers [0, 0] as the first response which is the worst-case response. Thus, the 3×20 AB game reduces to a 3×h AB game, where h = 17. The similar operations proceed at the second and third queries. After the third query and third response, the original 3×20 AB game reduces to a 3×11 AB game. The minimum number of queries can not be obtained easily with the use of analyses when $h \leq 11$ because of the irregular behavior. Hence, a branch-and-bound search algorithm, which has been proposed in Chapter 2, is applied to find an optimal strategy for smaller h.



1st query (best query)
c_0 c_1 c_2
0A0B
1st response (the adversary response)
Reduce to 3 × h AB game, h = 17
2nd query (best query)
c_3 c_4 c_5
0A0B
2nd response (the adversary response)
Reduce to 3 × h AB game, h = 14
3rd query (best query)
c_6 c_7 c_8
0A0B
3rd response (the adversary response)
Reduce to 3 × h AB game, h = 11
When $h \leq 11$
Apply a branch-and-bound search algorithm to find an optimal strategy for smaller n.

Figure 14. The scenario of the pessimistic situation of a 3×20 AB game

4.4 Chapter Conclusion

From the above discussions, the optimal query for the codebreaker and the adversary response for the codemaker, which refers to the worst case for the codebreaker as well, are eventually obtained with the consideration of the special state C^*. In the follow-up, all results mentioned above will be concluded to derive a theorem.

Theorem 2. For a 3×n AB game, the minimum number of queries for the codebreaker in the worst case is

$$\begin{cases} \lfloor n/3 \rfloor + 3 & \text{, if } 3 \leq n \leq 7 \\ \lfloor (n+1)/3 \rfloor + 3 & \text{, if } n \geq 8. \end{cases}$$

Proof. At the beginning of a 3×n AB game, the n symbols are not used and then all secret codes are all equivalent. As a result, a secret code is chosen randomly as the first query for the codebreaker. Nine substates are therefore produced and [0, 0] is taken as an adversary response according to Lemma 6. Afterwards, $C_{[0,0]}$, which results from the first response, matches the attribution of the special state C^* described in Lemma 5. Thus, Lemma 5 can be applied to this state. We find that the situations mentioned in Lemma 5 and Lemma 6 will appear alternately in the following gaming process. So we have the following recurrence.

$$T(n) = T(n-3) + 1, \text{ when } n > 11.$$

Because of the irregular behavior of a 3×*n* AB game with a smaller value of *n*, its minimum number of queries can be obtained with the use of a branch-and-bound search algorithm, which originates from Chapter 2, when $n \leq 11$. After the use of computer programs written with this approach, the minimum numbers of queries required for the codebreaker in the worst case are obtained in several hours and they are 4, 4, 4, 5, 5, 6, 6, 6, and 7 respectively when *n* = 3, 4, 5, 6, 7, 8, 9, 10, and 11. For example, an optimal strategy for 3×7 AB game is considered with *S* = {0, 1, 2, 3, 4, 5, 6}. If the codemaker takes 165 as a secret code, a gaming process in the worst case will be as follows: 012, [0, 1], 023, [0, 0], 041, [0, 1], 156, [1, 2], 165, [3, 0]. In other words, the codebreaker requires 5 queries to identify 165 while playing the worst-case optimal strategy.

We derive the above recurrence and conclude with the results of smaller values of *n*. Hence, the closed form of the formula is exhibited as follows.

$$\begin{cases} \lfloor n/3 \rfloor + 3 & , \text{if } 3 \leq n \leq 7, \\ \lfloor (n+1)/3 \rfloor + 3 & , \text{if } n \geq 8. \end{cases}$$

This completes the proof. □

Partial results of 3×*n* AB games, $3 \leq n \leq 16$, are summarized in Table 12. As 3×*n* AB games have been solved successfully, a natural generalization is to explore the techniques for *m*×*n* AB games, where $m \geq 4$. This problem remains open.

Table 12. The minimum number of queries for 3×*n* AB games in the worst case

n	3	4	5	6	7	8	9	10	11	12	13	14	15	16
# of queries	4	4	4	5	5	6	6	6	7	7	7	8	8	8

Chapter 5

Optimization Algorithm and Verification Algorithm

This chapter introduces two algorithms, called the *two-phase optimization algorithm* (TPOA) and *pigeonhole-principle-based verification algorithm* (PPV) to investigate the game, AB game with an unreliable response. TPOA was proposed by us in [17] and was proved to be an effective approximate algorithm for deductive games. PPV is modified slightly from the pigeonhole-principle-based fast backtracking algorithm in [37], which was also demonstrated by us. Section 5.1 gives a comprehensive introduction for our problems while some notations are redefined here to match the properties of the handled problem. Section 5.2 provides an introduction to TPOA and its performance. In Section 5.2.3, PPV is illustrated and the verified results are also shown. Section 5.4 contains the summary of our remarks.

5.1 Introduction

In this chapter, a variant of AB game, which is called AB game with an unreliable response, is presented. The game is the same as 4×10 AB game in addition to the concept of fault tolerance added to the variant. In other words, there is an additional rule in the game — the codemaker is allowed to give at most a wrong response. For example, it is a wrong response if the codemaker answers [1, 0] instead of [1, 2] if the codemaker chooses “2134” as a secret code and the codebreaker makes a query “0123”. Furthermore, the termination criterion of the game is modified in order to fit in with the area of fault tolerance. That is, the game is over if there is only one eligible code now. In short, it is not necessary for the codebreaker to figure out the secret code but to acquire it in his mind.

AB game with an unreliable response has ever been studies by us [37]. That results show that the upper bound of the required number of queries in this game is 9 while the lower bound of it is 8. Unfortunately, the two bounds are not the same and then, two more effective algorithms will be exhibited in this chapter to decide the exact bound of it.

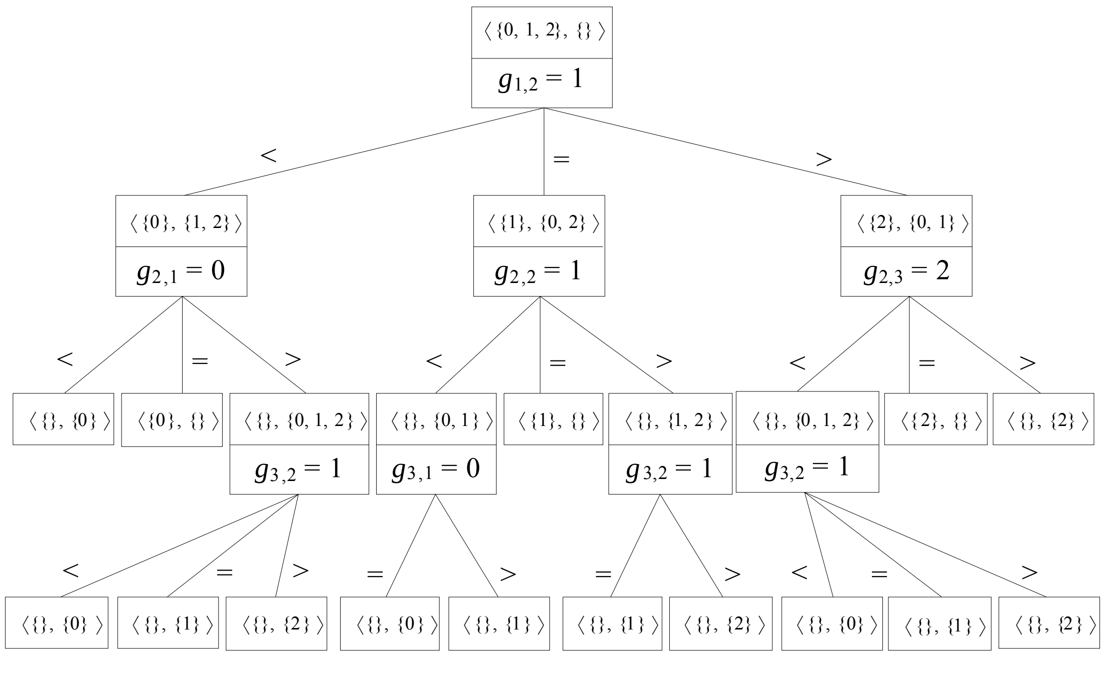


Figure 15. A game tree for the 1×3 game with an unreliable response

In order to clarify the problem and our proposed methods precisely, here we redefine some notations, which may have been defined in Chapter 1, to match the properties of AB game with an unreliable response. Consequently, a simple number guessing game, denoted $1 \times n$ games with an unreliable response, is taken as an illustrative example to explain these new notations. In the $1 \times n$ games with an unreliable response, the codemaker chooses a secret code c, $c = \{0, 1, 2, \cdots, n-1\}$. After each query g made by the codebreaker, the codemaker gives him a response r, $r = \{<, =, >\}$, i.e., they stand for $g < s$, $g = s$, and $g > s$. The codemaker is allowed to give at most a wrong response in this game. The goal of the game is to obtain the secret code by using as few queries as possible. We can represent the gaming process as game-tree search. For instance, a game tree for the 1×3 game with an unreliable response consisting of internal nodes and leaves is shown in Figure 15.

Definition 15. The *state* $\left\langle C_i^{(0)}, C_i^{(1)} \right\rangle$ consists of two sets, which are composed of eligible codes after the codebreaker makes the i-th query. The first set $C_i^{(0)}$ is the set of secret codes which satisfy all previous responses and $C_i^{(1)}$ represents the set of secret codes which satisfy all but one of the previous responses. For example, the root in Figure 15 is $\left\langle \{0,1,2\}, \{\ \} \right\rangle$, which indicates that the elements in $C_0^{(0)}$ are 0, 1, and 2 while $C_0^{(1)}$ is an empty set.

Definition 16. A *weight*, $\left(\left|C_i^{(0)}\right|, \left|C_i^{(1)}\right| \right)$, is a couple of natural numbers. The first number is the size of the set $C_i^{(0)}$ and the second number is the size of the set $C_i^{(1)}$. For instance, the weight of the root in Figure 15 is (3, 0).

Definition 17. The query $g_{i,j}$ made by the codebreaker means that the query is the j-th choice among all valid queries with respect to the current state and

(i−1) queries have been made previously. In Figure 15, "$g_{3,2} = 1$" means that it is the third query and the query is 1.

Definition 18. There are 14 legal responses in AB game. After the codebreaker makes the (i+1)-th query and the (i+1)-th response offered by the codemaker is j, this query will divide each set of the current state $\left\langle C_i^{(0)}, C_i^{(1)} \right\rangle$ into 14 subsets, $\left\langle R_{i+1,j}^{(0)}, R_{i+1,j}^{(1)} \right\rangle, j = 1, 2, \ldots, 14$. In other words, $\bigcup_{j=1}^{14} R_{i+1,j}^{(0)} = C_i^{(0)}$ and $\bigcup_{j=1}^{14} R_{i+1,j}^{(1)} = C_i^{(1)}$.

Definition 19. A *final state* is the state which is $\left\langle C_i^{(0)}, C_i^{(1)} \right\rangle$ and $\left|C_i^{(0)}\right| + \left|C_i^{(1)}\right| = 1$. In other words, only one eligible code remains in the final state and the game is over.

From the above definitions, the accurate relation of the states in each ply can be derived. Suppose that the codemaker offers j as the (i+1)-th response after the (i+1)-th query. The codebreaker has to consider whether the response j is correct or not. Hence, there are two possible cases discussed below.

- If the response is correct, the states we have to consider now are therefore $R_{i+1,j}^{(0)}$ and $R_{i+1,j}^{(1)}$.
- If the response is wrong, we need to think of this state, $\bigcup_{1 \le p \le 14, p \ne j} R_{i+1,p}^{(0)}$.

Before the game starts, we know that $C_0^{(0)}$ is the set that contains all valid secret codes and $C_0^{(1)} = \phi$. From the two discussed cases, we have the following relations.

$$C_{i+1}^{(0)} = R_{i+1,j}^{(0)} ,$$

$$C_{i+1}^{(1)} = R_{i+1,j}^{(1)} \cup \left(\bigcup_{1 \le p \le 14, p \ne j} R_{i+1,p}^{(0)} \right)$$

During the gaming process, the secret codes, which dissatisfy the previous responses just one time, will be moved from $C_i^{(0)}$ to $C_i^{(1)}$. If the secret codes in $C_i^{(1)}$ dissatisfy a response again in the future, we do not have to consider these codes in the following plies.

5.2 Two-Phase Optimization Algorithm

The two-phase optimization algorithm (TPOA) was originally proposed by us to solve Mastermind [17]. It is an approximate algorithm and is able to discover results with higher quality. TPOA can also be thought as a general improver for heuristic strategies. That is, given a heuristic, TPOA has higher chance to obtain results better than those obtained by the heuristic. Moreover, it sometimes can achieve near-optimal results that are difficult to find by the given heuristic.

In this section, we will attempt to apply TPOA to discover the upper bound of the number of queries for AB game with an unreliable response. We first review the properties of TPOA and the hashing collision group that is used in TPOA. Second, a well-designed hashing function and the heuristic of evaluation are provided. Finally, TPOA is utilized to address the game.

5.2.1 The Structure of TPOA

The search tree of TPOA, abbreviated to TPOA tree, is divided into two phases, exploration and exploitation. The objective of exploration phase is to discover promising partial solutions; on the other hand, the exploitation phase is to choose the way that leads each of the partial solution to a "best" complete solution. Two parameters, the *branching factor k* and the *exploration depth d*, are used to decide how large the search space TPOA intends to explore. That is, the parameters determine how many potential (promising) solutions that TPOA will exploit.

We [17] have presented two versions of TPOA, which are TPOA$^+$ (k, d) and TPOA*(k, d), in the previous study. Because a larger search space may be required to get a better upper bound of the game, only TPOA$^+$ (k, d) is adopted to investigate our problem. TPOA$^+$ (k, d) indicates TPOA with a branching factor of k and an exploration depth of d. The TPOA$^+$ (k, d) tree is shown in Figure 16. Given a TPOA tree with an arbitrary height h, after level d the algorithm does a greedy search form that node on. The number of potential solutions exploited in a TPOA$^+$ (k, d) tree will be k^d.

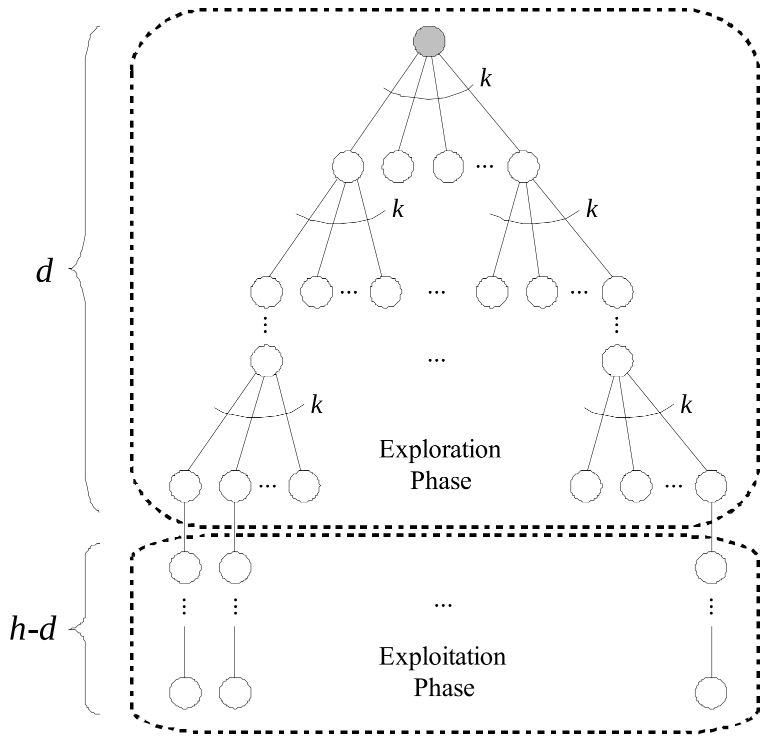


Figure 16. The construction for TPOA$^+$ (k, d) tree

The structure and properties of TPOA are described now. Given parameters (k,d), the sketch of a recursive procedure for TPOA is shown in Figure 17. TPOA can be implemented by a *modified exhaustive depth-first search* on a TPOA tree. The main modification to depth-first search is that at each visited node in the exploration phase (within depth d), we consider only b branches and ignore other branches. In Figure 17, TPOA$^+$ has a fixed b (= k) in the exploration phase, as shown in line 3. In the exploitation phase, TPOA$^+$ has a fixed $b = 1$ in line 4. Therefore, TPOA$^+$(k, d) is able to prune a huge search space to a manageable size k^d as shown in Figure 16. For AB game, since the 14 response nodes at each level should be kept, the search space is reduced to $(14 \times k)^d$.

```
    TPOA(k, d, b, c) {                          // k, d: the given constants
1   l = Current_level();                        // get the current level in the TPOA tree
2   If (c is a complete solution) Then Return c;
3   If (l < d) Then b = k;                      // in the exploration phase
4           Else b = 1;                         // in the exploitation phase
5   For (each move m ∈ M)                       // M: the set of all next potential moves
6           i = Hash(m);                        // classify possible next moves to HCGs by a
7           HCG_i ← HCG_i ∪ {m};                   hash function
8   B = {HCG_j | HCG_j is the top b groups that could obtain promising results};
9   For (each HCG_i ∈ B)                        // B: the set of b selected HCGs
10          c_i = Choose(HCG_i);                // c_i: the selected representative for HCG_i
11          C = C ∪ { c_i };                    // C: the set of b representatives c_i in B
12  S ← ∅;                                      // S: the set of potential solutions from
13  For (each c_i ∈ C)                             descendant nodes
14      s_i = TPOA(k, d, b, c_i);               // recursively b-way search to find the best
15      S ← S ∪ { s_i };                           solution from descendant nodes
16      c = Max_{s_i ∈ S} (eval(s_i));          // select the best solution discovered in S
17      Return c;                               // return c to the parent node.
    }
```

Figure 17. The sketch of TPOA

Given two constants (k, d), the time complexity of TPOA$^+$ (k, d), in terms of number of nodes exploited, is k^d $(h - d)$, where h is the height of the game tree, i.e., the number of queries required in the worst case. This means that no matter how large an instance of problem is given, TPOA can always obtain an approximate result by appropriately selecting the parameters (k, d). Furthermore, depending on the execution time and space allowed, the value of parameters (k, d) can be increased to approach the optimal result. Now, the fundamental components of TPOA are summarized as follows:

- A constructive heuristic for the problem at hand
- A hash function according to the heuristic
- Two parameters (*k*, *d*) to decide how large the search space TPOA intends to explore

5.2.2 Hash Collision Groups

In TPOA, how to select the (most likely) best *b* next potential components is a critical issue. The problem can be effectively and efficiently solved by a clustering approach. TPOA performs clustering using a concept of *hash collision groups* [14], which are abbreviated to HCGs. The next potential components of solutions with similarity are clustered together in an HCG by a given *hash function* to the problem at hand. That is, the potential components with the same hash value will be clustered together. Section 5.2.3 will give detailed examples of how the clustering mechanism works. Properties of HCGs are now described. Figure 18 illustrates the relation between HCGs and equivalent classes in a search space of next potential components. There are several advantages of using HCGs in TPOA. The important properties of HCGs include:

- For two components in the same HCG, they are most likely equivalent. 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 *b* best HCGs.
- Without losing the generality, an arbitrary component can be chosen to represent its HCG.

Therefore, TPOA is able to efficiently and effectively select the *b* “best” representatives among all next potential components. On the other point of view, if an evaluation function is used in TPOA, each HCG can be regarded as a set of the next potential components which have a tie on the return value of the function. Note that most ties are equivalent but equivalent solutions will produce ties.

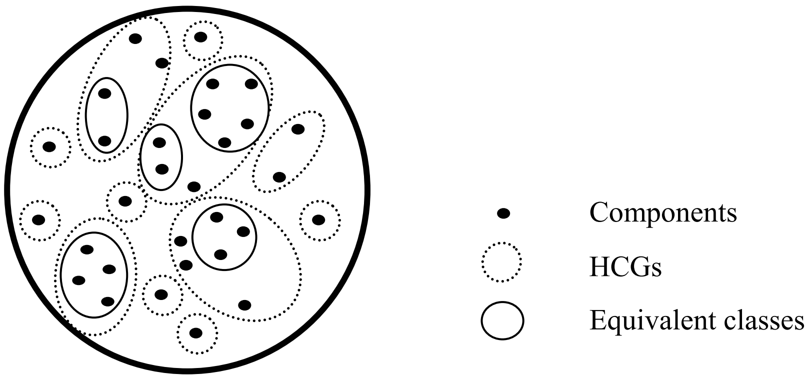


Figure 18. The relation between HCGs and equivalent classes

5.2.3 TPOA for AB game with an Unreliable Response

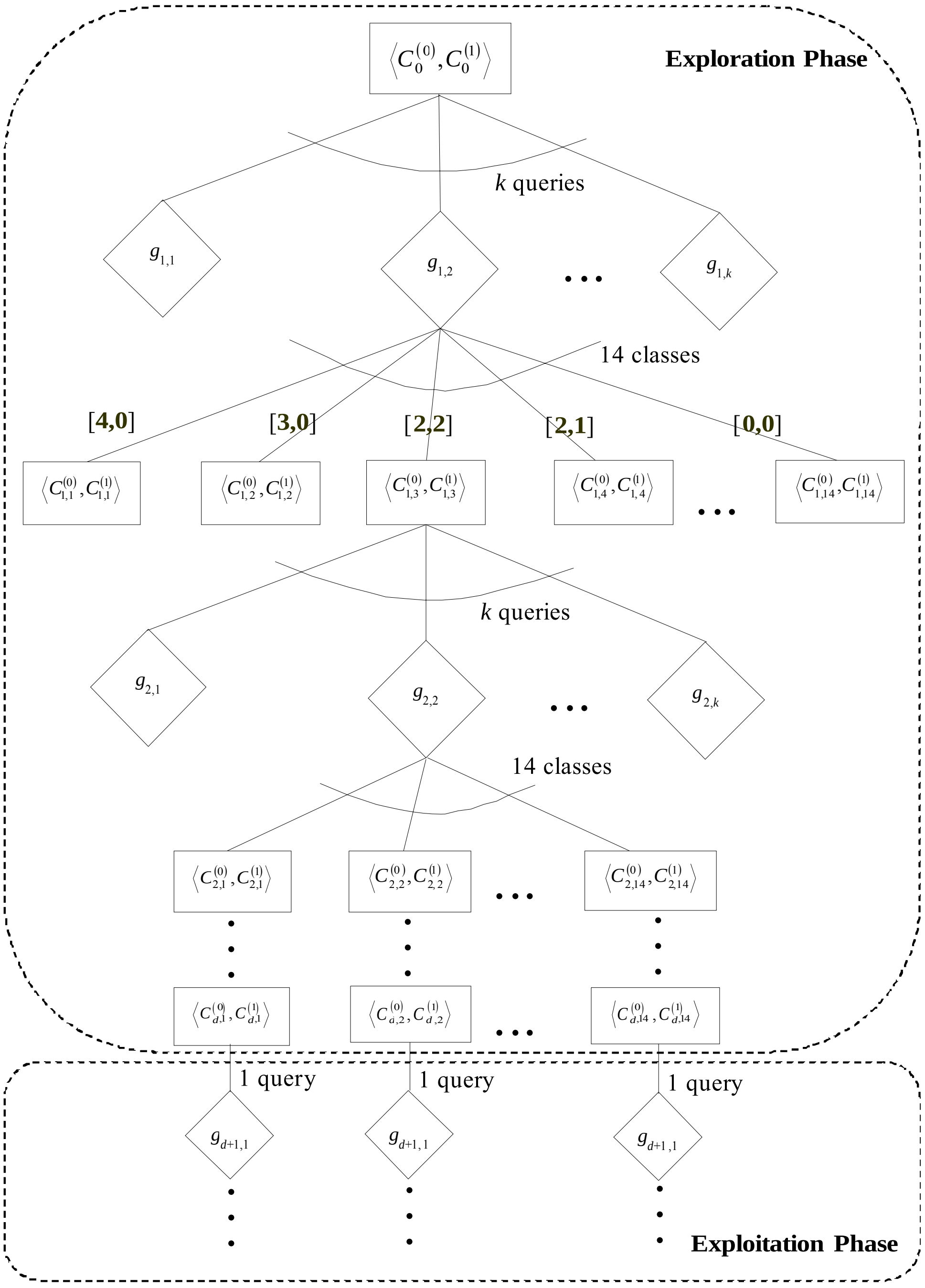
In this section, TPOA will be applied to our problem, AB game with an unreliable response. Figure 19 shows the game tree by applying the TPOA to this problem. Among them, $\left\langle C_{i,j}^{(0)}, C_{i,j}^{(1)} \right\rangle$ is the *j*-th state, i.e., the *j*-th class (response), after the *i*-th query. And $g_{i,j}$ is the *j*-th among the *k* best codes chosen by the TPOA at the *i*-th query.

According to the hashing function, which will be demonstrated in Section 5.2.4, all valid queries are categorized into several HCGs and the representative of each HCG is evaluated in order to select *k* best codes as the explored queries. The designed hashing function and the heuristic of evaluation are described in detail in the next subsection.

In the beginning, the initial state is the root of the game tree in Figure 19, which means that there are totally 5040 queries satisfying all previous responses. Note that while the codebreaker takes the first query into account, TPOA chooses the *k* best codes, $g_{1,1}$, $g_{1,2}$, …, $g_{1,k}$, to conduct this search. After that, there are 14 classes which have to be expanded since the codemaker has 14 legal responses. Then the codebreaker selects *k* best queries to expand the game tree again after the first response is determined. The two steps take turns until the final state is met. At final state, the program backtracks to its parent node and expands other branches continuously.

$C_i^{(0)}$: the set of eligible codes which satisfy all previous responses

$C_i^{(1)}$: the set of eligible codes which satisfy all but one previous responses



$\langle C_0^{(0)}, C_0^{(1)} \rangle$
Exploration Phase
k queries
$g_{1,1}$
$g_{1,2}$
$g_{1,k}$
14 classes
[4,0]
[3,0]
[2,2]
[2,1]
[0,0]
$\langle C_{1,1}^{(0)}, C_{1,1}^{(1)} \rangle$
$\langle C_{1,2}^{(0)}, C_{1,2}^{(1)} \rangle$
$\langle C_{1,3}^{(0)}, C_{1,3}^{(1)} \rangle$
$\langle C_{1,4}^{(0)}, C_{1,4}^{(1)} \rangle$
$\langle C_{1,14}^{(0)}, C_{1,14}^{(1)} \rangle$
k queries
$g_{2,1}$
$g_{2,2}$
$g_{2,k}$
14 classes
$\langle C_{2,1}^{(0)}, C_{2,1}^{(1)} \rangle$
$\langle C_{2,2}^{(0)}, C_{2,2}^{(1)} \rangle$
$\langle C_{2,14}^{(0)}, C_{2,14}^{(1)} \rangle$
$\langle C_{d,1}^{(0)}, C_{d,1}^{(1)} \rangle$
$\langle C_{a,2}^{(0)}, C_{d,2}^{(1)} \rangle$
$\langle C_{d,14}^{(0)}, C_{d,14}^{(1)} \rangle$
1 query
1 query
1 query
$g_{d+1,1}$
$g_{d+1,1}$
$g_{d+1,1}$
Exploitation Phase

Figure 19. The game tree expanded by TPOA

5.2.4 The Hashing Function and the Heuristic of Evaluation

Now, a hashing function is designed carefully and a simple heuristic proposed by Barteld [6] is utilized to cooperate with TPOA. Although the two methods are uncomplicated, they are adequate to solve our problem.

Hashing function for TPOA:

Suppose given a state, $\left\langle C_i^{(0)}, C_i^{(1)} \right\rangle$, let the sizes of the 14 response classes (states), which result from $C_i^{(0)}$, after a query *g* be $S_g^{(0)} = \left\langle R_{i+1,1}^{(0)}, R_{i+1,2}^{(0)}, \ldots, R_{i+1,14}^{(0)} \right\rangle$ while the sizes of the 14 response classes (states) resulting from $C_i^{(1)}$, after a query *g* is $S_g^{(1)} = \left\langle R_{i+1,1}^{(1)}, R_{i+1,2}^{(1)}, \ldots, R_{i+1,14}^{(1)} \right\rangle$. Afterwards, the hash function sorts the original two sequences, $S_g^{(0)}$ and $S_g^{(1)}$, into nonincreasing sequences, $\overline{S}_g^{(0)}$ and $\overline{S}_g^{(1)}$, independently. The hash function is therefore defined as follows:

$$Hash\left(\left\langle S_g^{(0)}, S_g^{(1)} \right\rangle\right) = \left\langle \overline{S}_g^{(0)}, \overline{S}_g^{(1)} \right\rangle,$$

In other words, assume that two queries, *g* and *p*, are considered. If $\overline{S}_g^{(0)} = \overline{S}_p^{(0)}$ and $\overline{S}_g^{(1)} = \overline{S}_p^{(1)}$, then the query *g* and the query *p* are classified into the same HCG.

Remember that we also guarantee the fundamental properties of the designed hashing function that (1) for two components in the same HCG, they are most likely equivalent, and that (2) for two equivalent components, they are definitely in the same HCG. Therefore, we can arbitrarily choose a secret code to represent its HCG, rather than exhaustively explore all secret codes in the HCG, and obtain an approximate result.

Heuristic of evaluation:

In the previous analyses, the height of the game tree has to be minimized so as to obtain the optimal strategy for the game in the worst case. However, it is not intuitive to determine the significance between the number of codes in $C_i^{(0)}$ and that of codes in $C_i^{(1)}$. Hence, a simple and efficient heuristic, called "most-parts heuristic", demonstrated by Barteld [6] is used in TPOA. The most-parts heuristic focuses on the "breadth" the eligible secret codes can be spread. In other words, the more classes the eligible secret codes can occupy after a query, the more favorable this query is.

Because a state in our problem has two sets, e.g., $\left\langle C_i^{(0)}, C_i^{(1)} \right\rangle$, the most-parts heuristic has to sum up the number of the nonzero numbers in $S_g^{(0)}$ and that of nonzero numbers in $S_g^{(1)}$ according to a query *g*. The higher the score is, the better the query is. For example, the query *g* is better than the query *p* if the numbers of parts caused by *g* and *p* are 24 and 18 respectively.

5.2.5 Experiment Results of TPOA

When our program based on TPOA was implemented and tested, we ran it on a dedicated PC equipped with an Intel Core 2 Duo CPU whose frequency is 3.16 GHz. In order to accelerate the running time of TPOA furthermore, another technique is implemented as well. That is, during the searching process, TPOA will terminate as soon as it has found a strategy, in which the minimum number of queries is 8 in the worst case. Thus, this may reduce the necessity to search all the possible pathways in the search space shown in Figure 19, and result in faster finish time.

The results are shown in Table 13. Basically, the larger the values of *k* and *d* are, i.e., the larger the search space is, the fewer the number of queries required for the game is, and the longer the time for running the program is. However, the results in Table 13 do not always seem to show this trend. This is because by using the above speed-up technique, TPOA stops if a strategy with 8 queries required in the worst case is found. In other words, TPOA will stop more quickly if the order of the traversal sequences of the *k* queries in each ply is decided carefully. In our program, the order of the traversal sequences is completely determined by the most-parts heuristic to choose the *k* best queries in each ply. From the results in Table 13, it reveals that the most-parts heuristic is quite outstanding because the running time is shorter when *k* = 7 and *d* = 7.

Table 13. The upper bound derived by our program

k	*d*	The number of queries in the worst case	Running time (Minutes)
1	1	10	3.96
2	6	10	21.60
3	3	10	28.43
5	4	9	319.47
5	5	9	641.47
7	7	8	13.87

Note that the number of queries, whose value is 8, is obtained by our program when *k* = 7 and *d* = 7. This shows that the TPOA can efficiently obtain optimal (or near-optimal) results with a small *k* and *d* (compared to 5040 valid queries). Hence, we have the following Lemma 7 evidently.

Lemma 7. For AB game with an unreliable response, there exists a strategy such that the number of queries required for the codebreaker to obtain the secret code is at most 8.

We can regard Lemma 7 as an upper bound of this problem. In the following section, we demonstrate the pigeonhole-principle-based verification algorithm to prove that the lower bound of the game is also 8.

5.3 Pigeonhole-principle-based Verification Algorithm

In our previous study [37], we have proposed a pigeonhole-principle-based fast backtracking algorithm (PPBFB) to obtain the lower bound of our problem in about 5 days using an AMD Opteron 1.6GHz PC. Here, the concept of PPBFB will be reviewed first and then, the reductions of equivalent queries (Technique 3 in Section 3.2.2.3) are also cooperated with PPBFB to accelerate the speed of the verification. The refined version of PPBFB is called pigeonhole-principle-based verification algorithm (PPV). Finally, the lower bound is also acquired by PPV in only 12.83 minutes using an Intel Core 2 Duo 3.16 GHz PC.

The concept of PPBFB is to conduct an exhaustive worst-first search. It rates the lower bound by making use of the extended pigeonhole principle proposed by us [18] and then backtracks as early as possible to save the search time. The refined version of PPBFB, PPV, is illustrated in Figure 20. The key idea of PPV is to consider the sizes of the two sets in the state when the search proceeds. The rectangles in Figure 20 represent the states. $g_{i,p}$ is the *i*-th possible choice among all secret codes made by the codebreaker at the *p*-th query. $r_{p,\max}$ means the class which results in the most number of queries among 14 classes after the *p*-th query. $q_{\max}$ is the theoretical lower bound which means a fewest number of queries required to reach the final state, i.e., $\langle \{c\}, \phi \rangle$ or $\langle \phi, \{c\} \rangle$, from the current state and *h* is the lower bound we intend to verify.

Theoretically, a search algorithm has to explore all valid 5040 secret codes at each query. However, in fact, PPV only need to explore 1 representative query at the first query, to expand 20 queries at the second query, and to expand 356.50 queries in average at the third query due to the equivalence property. For the codemaker, only the worst case among the 14 classes has to be expanded. The so-called "worst case" denotes the class which will result in the most number of queries needed by the codebreaker.

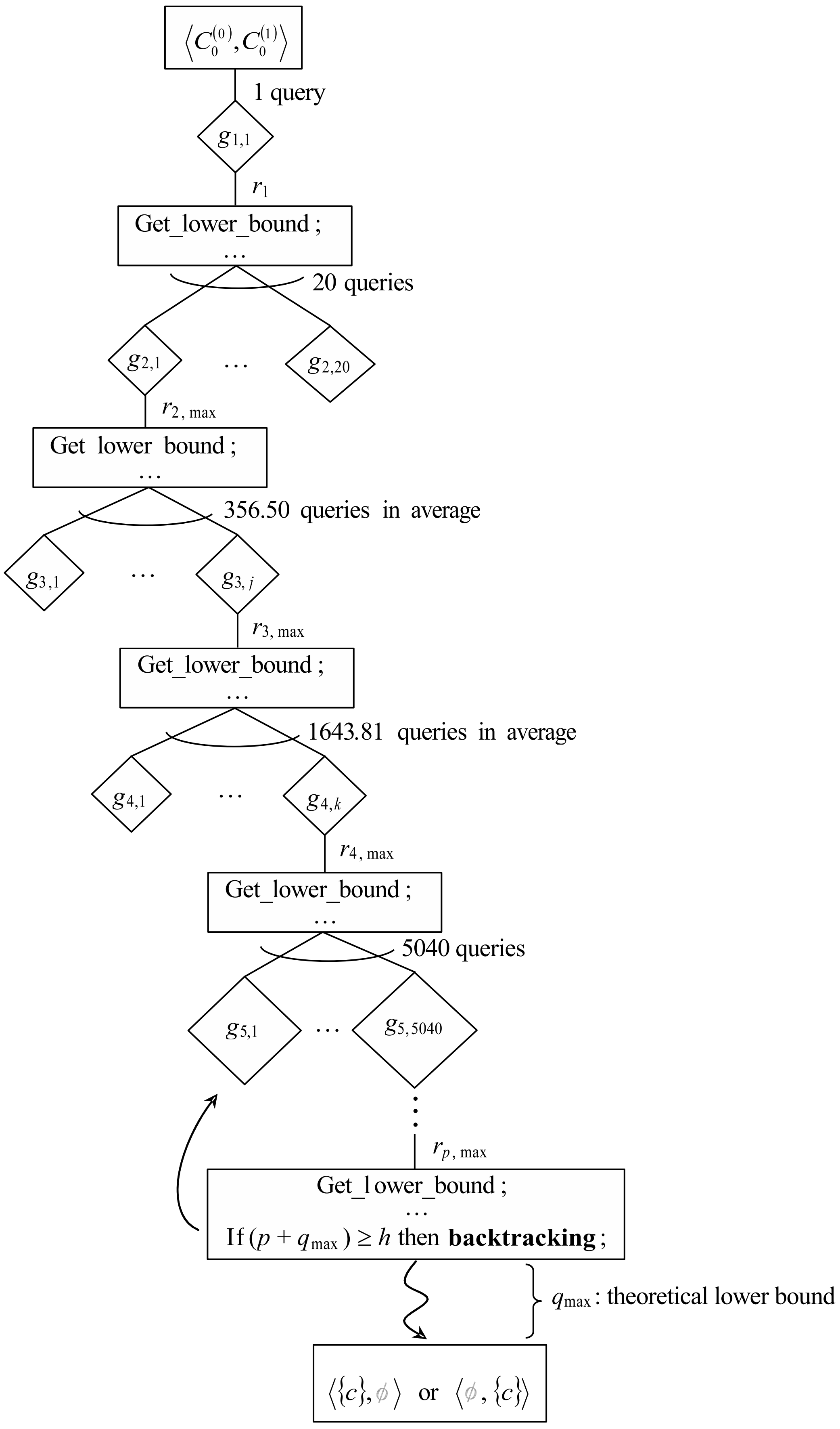


Figure 20. The sketch of the PPV algorithm

The extended pigeonhole principle [18] is employed to estimate the lower bounds of the number of queries needed among 14 classes. The idea of the estimations of lower bounds is similar to that proposed in Chapter 2. In other words, the actual number of queries needed is more than or equal to the most number, $q_{\max}$, of lower bounds among 14 classes. Therefore, our verification program is not necessary to search the whole game tree. It can backtrack to the parent node to expand other branches if the condition holds: $(p + q_{\max}) \geq h$, where we set $h = 8$.

The main idea of the estimation of lower bounds by using the extended pigeonhole principle is that the query made by the codebreaker in each ply may divide the elements of the two sets in the current state evenly. Hence, this ideal strategy can minimize the height of subtree rooted in the current node. That is to say that there exists a “theoretical optimal” strategy for the codebreaker in the following queries such that all the elements of the two sets in each state may be divided evenly. The actual number of queries is thus more than or equal to the value of estimations. Note that we use the function, Get_lower_bound, to rate the lower bounds in Figure 20. The detailed calculation of the lower bounds, the entire algorithms, and other improvements can be found in [37][38]. Hence, the details are omitted here.

After the careful implementation of our program based on PPV, The verification program was run on a dedicated PC equipped with an Intel Core 2 Duo 3.16 GHz CPU to verify the lower bound required for AB game with an unreliable response. If we set the value of h, which indicates the lower bound we want to verify, to 8, our program executed for about 12.83 minutes and the final output is “success!” finally.

In other words, the minimum number of queries is at least 8 in the worst case without respect to any strategies used by the codebreaker. Note that the upper bound of this problem is obtained in Lemma 7 as well. Thus, we have the following theorem which shows that the lower bound as well as the exact bound of the game is 8.

Theorem 3. For AB game with an unreliable response, 8 queries are necessary and sufficient to identify a secret code in the worst case.

5.4 Chapter Conclusion

This chapter utilizes two advanced algorithms to address AB game with an unreliable response. The first one is *two-phase optimization algorithm* (TPOA). With the well-designed hashing function and the simple heuristic of evaluation, the results obtained by TPOA are better than those of the previous work [37]. In other words, TPOA is more effective and efficient. Note that the upper bound of the game is declined from 9 to 8 in this refined approach.

On the other hand, another improvement, *pigeonhole-principle-based verification algorithm* (PPV), is modified from pigeonhole-principle-based fast backtracking algorithm (PPBFB). PPV uses equivalent properties to reduce the branching factors at the first three queries. Although the final outcome is the same as that in [37], the speed of PPV is faster than PPBFB due to the reductions of equivalent queries. Moreover, the lower bound provided by PPV is 8 as well.

Fortunately, we have proved that the upper bound of the game matches the lower bound while its value is 8. Hence, the minimum number of queries for AB game with an unreliable response is 8. Furthermore, it may be interesting to deal with AB game with e unreliable responses, where $e \geq 2$.

Chapter 6

Conclusion and Future Work

In this dissertation, some optimization approaches for deductive games and their variants are taken into account. Section 6.1 concludes with the proposed optimization algorithms and our contributions. Some future work is mentioned in Section 6.2.

6.1 Concluding Remarks

Two advanced algorithms and a reduction technique for deductive games are demonstrated in this study. Moreover, two promising algorithms, which are proposed before, with some modifications are introduced to solve our addressed problem as well. We summarize our main novel contributions:

(1) A more efficient complete algorithm, which is called *depth-first backtracking algorithm with branch-and-bound pruning* (DBB) for Mastermind in the expected case, is introduced to take the place of traditional approaches and meanwhile, an admissible heuristic, which can be applied to various deductive games, is presented as well. From the experiments, DBB is significantly superior to the traditional algorithms and an alternative optimal strategy is also obtained finally.

(2) To date, there have been no optimal expected-case strategies for AB game in formal literature since its appearance. Thus, a *refined branch-and-bound*

algorithm with speed-up techniques (RBB) is demonstrated to deal with this problem. A tactic for playing AB game optimally in the expected case is eventually attained by utilizing RBB and in addition, the corresponding expected number of queries, 26274/5040 ≈ 5.213, is derived.

(3) A sophisticated method, called *structural-reduction approach* (SR), which aims at explaining the pessimistic situation in this game, is presented to investigate $3\times n$ AB games. After careful theoretical analyses, optimal strategies for the codebreaker in the pessimistic situation are discovered. Furthermore, a worthwhile formula for calculating the optimal numbers of queries required for arbitrary values of n is derived and proven successfully.

(4) Two algorithms, which are named as *two-phase optimization algorithm* (TPOA) and *pigeonhole-principle-based verification algorithm* (PPV), are surveyed for solving AB game with an unreliable response. The purpose of TPOA is to discover an upper bound of the required number of queries in this game while PPV aims at identifying a lower bound of it. Fortunately, experimental results show that the upper bound equals the lower bound and then, the exact bound of the number of queries needed, whose value is 8, is achieved.

From the survey of related papers, it reveals that the search space of many games and optimization problems are often so huge that traditional search algorithms are not able to explore it efficiently. Of course, there were plenty of pruning techniques, which were proposed before. However, slight inaccuracy of the measures of these pruning techniques may usually lead to the poor results that are far from the optimum.

In this study, our proposed search algorithms, which are replied upon the admissible heuristics, have contributed success to various deductive games. Note that in general, the admissible heuristics can be regarded as a kind of theoretical pruning techniques since the pruning occurs but does not affect the correctness of search algorithms. In other words, the results of the search algorithms are accurate if the pruning techniques are based on theoretical analyses. Hence, it may be a trend to combine search algorithms with theoretical pruning for solving those complicated problems.

On the other hand, other optimization problems such as coding theory, circuit testing, differential cryptanalysis, and additive search problem may also be solved by taking advantage of our demonstrated methods with modifications in the future. We hope that the research results may assist other scientists with the development of their concerned issues.

6.2 Future Work

There are still some open issues regarding our problem domain. The optimal strategies of deductive games with much higher dimensions, which are called $m \times n$ AB games while $m \geq 4$, are still unknown. It is interesting to investigate them because they may become NP-complete problems or harder problems if the value of m is getting larger constantly. Then, the boundary value of m is significant as well. Besides the original versions of much higher dimensions, other variants of deductive games are also worth studying such as static deductive games or deductive games with multiple unreliable responses. From the progress of research, $3 \times n$ deductive games in the expected case and $4 \times n$ deductive games in the worst case may be solved completely in the near future.

There are other important problems such as the Renyi-Ulam game and the counterfeit coin problem, whose styles are similar to deductive games. In fact, the Renyi-Ulam game has been widely surveyed in the fault-tolerance area and meanwhile, the counterfeit coin problem has been discussed constantly in the information-theory area as well. However, there are still a lot of open issues about the two significant problems. These open questions are likewise worth studying in further detail for discovering their solutions.

Bibliography

[1] Allen, J. (1989), "A Note on the Computer Solution of Connect-Four," *Heuristic Programming in Artificial Intelligence 1: The First Computer Olympiad*, pp. 134–135.

[2] Allis, L. V. (1988), *A knowledge-based approach of Connect-Four—the game is solved: white wins*, Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands.

[3] Allis, L. V. (1994), *Searching for solutions in artificial intelligence*, PhD Dissertation, Universiteit Maastricht, Maastricht, The Netherlands.

[4] Appel, K., and Haken, W. (1977), "Every planar map is four colorable part I: discharging," *Illinois Journal of Mathematics*, Vol. 21, pp. 429–490.

[5] Appel, K., Haken, W., and Koch, J. (1977), "Every planar map is four colorable part II: reducibility," *Illinois Journal of Mathematics*, Vol. 21, pp. 491–567.

[6] Barteld, K. (2005), "Yet another Mastermind strategy," *ICGA Journal*, Vol. 28, No. 1, pp. 13–20.

[7] Bento, L., Pereira, L., and Rosa, A. (1999), "Mastermind by evolutionary algorithms," *Proceedings of the 1999 ACM symposium on Applied computing*, San Antonio, Texas, USA, 28 February-2 March, pp. 307–311.

[8] Berghman, L., Goossensa, D., and Leus, R. (2009), "Efficient solutions for Mastermind using genetic algorithms," *Computers and Operations Research*, Vol. 36, No. 6, pp. 1880–1885.

[9] Bernier, J. L., Herráiz, C. I., Merel, J. J., Olmeda, S., and Prieto, A. (1996), "Solving Mastermind using GAs and simulated annealing: a case of dynamic constraint optimization," *Parallel Problem Solving from Nature* (*PPSN IV*), *Lecture Notes in Computer Science*, Vol. 1141, pp. 554–563.

[10] Billings, D., Papp, D., Peña, L., Schaeffer, J., and Szafron, D. (1999), "Using selective-sampling simulations in poker," *Proceedings of AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*, Stanford, CA, USA, March, pp. 13–18.

[11] Bjornsson, Y., and Marsland, T. A. (2001), "Multi-cut alpha-beta pruning in game-tree search," *Theoretical Computer Science*, Vol. 252, No. 1, pp. 177–196.

[12] Blum, C., and Roli, A. (2003), "Metaheuristics in combinatorial optimization: overview and conceptual comparison," *ACM Computing Surveys*, Vol. 35, No. 3, pp. 268–308.

[13] Bresina, J. L. (1996), "Heuristic-biased stochastic sampling," *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference*, Oregon, Portland, 4-8 August, pp. 271–278.

[14] Chen, S. T. (2004), *On the Study of Optimization Algorithms for Deductive Games and Related Problems*. PhD Dissertation, National Taiwan Normal University, Taipei, Taiwan.

[15] Chen, S. T., Hsu, S. H., and Lin, S. S. (2004), "Optimal algorithms for 2×*n* AB games - a graph-partition approach," *Journal of Information Science and Engineering*, Vol. 20, No. 1, pp. 105–126.

[16] Chen, S. T., Hsu, S. H., and Lin, S. S. (2004), "Optimal algorithms for 2×*n* Mastermind games - a graph-partition approach," *Computer Journal*, Vol. 47, No. 5, pp. 602–611.

[17] Chen, S. T., Lin, S. S., and Huang, L. T. (2007), "A two-phase optimization algorithm for Mastermind," *Computer Journal*, Vol. 50, No. 4, pp. 435–443.

[18] Chen, S. T., Lin, S. S., Huang, L. T., and Hsu, S. H. (2007), "Strategy optimization for deductive games," *European Journal of Operational Research*, Vol. 183, No. 2, pp. 757–766.

[19] Colorni, A., Dorigo, M., Maffioli, F., Maniezzo, V., Righini, G., and Trubian, M. (1996), "Heuristics from nature for hard combinatorial optimization problems," *International Transactions in Operational Research*, Vol. 3, No. 1, pp. 1–21.

[20] Coulom, R. (2006), "Efficient selectivity and backup operators in Monte-Carlo tree search," *Proceedings of the 5th Conference on Computers and Games*, Turin, Italy, 29-31 May, pp. 72–83.

[21] Donninger, C. (1993), "Null move and deep search: selective-search heuristics for obtuse chess programs," *ICCA Journal*, Vol. 16, No. 3, pp. 137–143.

[22] Dorigo, M., and Gambardella, L. M. (1997), "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp. 53–66.

[23] Drakard, K. (1998), "Mastermind: WebGames," Internet: http://www.irt.org/games/js/mind/.

[24] Feo, T. A., and Resende, M. G. C. (1995), "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, Vol. 6, No. 2, pp. 109–133.

[25] Flood, M. M. (1988), "Sequential search strategies with Mastermind variants — Part 1," *Journal of Recreational Mathematics*, Vol. 20, No. 2, pp. 105–126.

[26] Glover, F. (1986), "Future paths for integer programming and links to artificial intelligence," *Computers and Operations Research*, Vol. 13, No. 5, pp. 533–549.

[27] Glover, F. (1990), "Tabu search Part II," *ORSA Journal on Computing*, Vol. 2, No. 1, pp. 4–32.

[28] Goddard, W. (2004), "Mastermind revisited," *Journal of Combinatorial Mathematics and Combinatorial Computing*, Vol. 51, pp. 215–220.

[29] Goddard, W. (2003), "Static Mastermind," *Journal of Combinatorial*

Mathematics and Combinatorial Computing, Vol. 47, pp. 225–236.

[30] Goodrich, M. T. (2009), “On the algorithmic complexity of the Mastermind game with black-peg results,” *Information Processing Letters*, Vol. 109, No. 13, pp. 675–678.

[31] Greenwell, D. (1999-2000), “Mastermind,” *Journal of Recreational Mathematics*, Vol. 30, pp. 191–192.

[32] Hales, T. C. and Ferguson, S. P. (2006), *Discrete and Computational Geometry*, Vol. 36, No. 1.

[33] Harvey, W. D., and Ginsberg, M. L. (1995), “Limited discrepancy search,” *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montréal, Québec, Canada, 20-25 August, pp. 607–613.

[34] Heinz, E. A. (2000), “AEL Pruning,” *ICGA Journal*, Vol. 23, No. 1, pp. 21–32.

[35] Helmstetter, B., and Cazenave, T. (2003), “Searching with analysis of dependencies in a solitaire card game,” van den Herik, H. J. Iida, H. and Heinz, E. A. (eds), *Advances in Computer Games 10*, Kluwer Academic Publishers, Netherlands.

[36] Holland, J. H. (1975), *Adaptation in natural and artificial systems*. University of Michigan Press. Ann Arbor.

[37] Huang, L. T. (2005), *On the study of deductive games with lies*, Master's thesis, National Taiwan Normal University, Taipei, Taiwan.

[38] Huang, L. T., Chen, S. T., and Lin, S. S. (2006), “Exact-bound analyses and optimal strategies for Mastermind with a lie,” *Lecture Notes in Computer Science*, *Advances in Computer Games 11*, Vol. 4250, pp. 195–209.

[39] Irving, R. W. (1978-79), “Towards an optimum Mastermind strategy,” *Journal of Recreational Mathematics*, Vol. 11, No. 2, pp. 81–87.

[40] Jäger, G., and Peczarski, M. (2009), “The number of pessimistic guesses in generalized Mastermind,” *Information Processing Letters*, Vol. 109, No. 12, pp. 635–641.

[41] Juill´e, H., and Pollack, J. B. (1998), “A sampling-based heuristic for tree search applied to grammar induction,” *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, Madison, Wisconsin, USA, 26-30 July, pp. 776–783.

[42] Kabatianski, G., and Lebedev, V. (2000), “The Mastermind game and the rigidity of the Hamming space,” *Proceedings of the 2000 IEEE International Symposium on Information Theory*, Sorrento, Italy, 25-30 June, pp. 375–375.

[43] Kalisker, T., and Camens, D. (2003), “Solving Mastermind using genetic algorithms,” *Genetic and Evolutionary Computation — GECCO 2003*, *Lecture Notes in Computer Science*, Vol. 2724, pp. 1590–1591.

[44] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983), “Optimization by

simulated annealing," *Science*, Vol. 220, No. 4598, pp. 671–680.

[45] Knuth, D. E. (1976), "The computer as Mastermind," *Journal of Recreational Mathematics*, Vol. 9, No. 1, pp. 1–6.

[46] Ko, K. I., and Teng, S. C. (1986), "On the number of queries necessary to identify a permutation," *Journal of Algorithms*, Vol. 7, No. 4, pp. 449–462.

[47] Koyama, K., and Lai, T. W. (1993), "An optimal Mastermind strategy," *Journal of Recreational Mathematics*, Vol. 25, No. 4, pp. 251–256.

[48] Labat, J. M., and Pomerol, J. C. (2003), "Are Branch and Bound and A^* Algorithms Identical," *Journal of Heuristics*, Vol. 9, No. 2, pp. 131–143.

[49] Land, A. H., and Doig, A. G. (1960), "An automatic method of solving discrete programming problems," *Econometrica*, Vol. 28, No. 3, pp. 497–520.

[50] McKay, B. D. (1998), "Isomorph-free exhaustive generation," *Journal of Algorithms*, Vol. 26, No. 2, pp. 306–324.

[51] Merelo-Guervos, J. J., Castillo, P., and Rivas, V. M. (2006), "Finding a needle in a haystack using hints and evolutionary computation: the case of evolutionary MasterMind," *Applied Soft Computing*, Vol. 6, No. 2, pp. 170–179.

[52] Neapolitan, R. and Naimipour, K. (2004), *Foundations of Algorithms Using C++ Pseudocode. 3rd edn*. Jones and Bartlett Publishers.

[53] Neuwirth, E. (1982), "Some strategies for Mastermind," *Mathematical Methods of Operations Research*, Vol. 26, No. 1, pp. 257–278.

[54] Norvig, P. (1984), "Playing Mastermind optimally," *ACM SIGART Bulletin*, No. 90, pp. 33–34.

[55] Pitsoulis, L. S., and Resende, M. G. C. (2002), "Greedy randomized adaptive search procedure," In P. Pardalos, and M. Resende, (eds), *Handbook of Applied Optimization*. Oxford University.

[56] Prieditis, A., and Davis, R. (1995), "Quantitatively relating abstractness to the accuracy of admissible heuristics," *Artificial Intelligence*, Vol. 74, No. 1, pp. 165–175.

[57] PYVA-NET (2000), "Pyva net!," Internet: http://pyva.net/eng/play/bk.html.

[58] Roche, J. R. (1997), "The value of adaptive questions in generalized Mastermind," *Proceedings of the 1997 IEEE International Symposium on Information Theory*, Ulm, Germany, 29 June- 4 July, pp. 135–135.

[59] Rosu, R. (1999), *Mastermind*, Master's thesis, North Carolina State University, Raleigh, North Carolina.

[60] Ruiz, R., and StÄutzle, T. (2007), "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European Journal of Operational Research*, Vol. 177, No. 3, pp. 2033–2049.

[61] Ruml, W. (2001), "Incomplete tree search using adaptive probing," *Proceedings*

of the Seventeenth International Joint Conference on Artificial Intelligence, Seattle, Washington, USA, 4-10 August, pp. 235–241.

[62] Russell, S. and Norvig, P. (2002), *Artificial Intelligence: A Modern Approach. 2nd edn*. Prentice-Hall.

[63] Schaeffer, J., Burch, N., Björnsson, B., Kishimoto, A., Müller, M., Lake, R., Lu, P., and Sutphen, S. (2007), "Checkers is solved," *Science*, Vol. 317, No. 5844, pp. 1518–1522.

[64] Sedgewick, R. (1988), *Algorithms. 2nd edn*. Addison-Wesley.

[65] Seiden, S. (2002), "A manifesto for the computational method," *Theoretical Computer Science*, Vol. 282, No. 2, pp. 381–395.

[66] Seiden, S. (2001), "Can a computer proof be elegant," *ACM SIGACT News*, Vol. 32, No. 1, pp. 111–114.

[67] Shapiro, E. (1983), "Playing Mastermind logically," *ACM SIGART Bulletin*, No. 85, pp. 28–29.

[68] Singley, A. (2005), *Heuristic solution methods for the 1-dimensional and 2-dimensional Mastermind problem*, Master's thesis, University of Florida.

[69] Spencer, J. (1983), "Short theorems with long proofs," *American Mathematical Monthly*, No. 90, pp. 365–366.

[70] Stuckman, J., and Zhang, G. Q. (2006), "Mastermind is NP-complete," *INFOCOMP - Journal of Computer Science*, Vol. 5, No. 2, pp. 25–28.

[71] Swaszek, P. (1999), "The mastermind novice," *Journal of Recreational Mathematics*, No. 30, pp. 193–198.

[72] Temporel, A., and Kovacs, T. (2003), "A heuristic hill climbing algorithm for Mastermind," *UKCI '03, Proceedings of the 2003 UK Workshop on Computational Intelligence*, pp. 189–196.

[73] Ugurdag, H., Sahin, Y., and Baskirt, O. (2006), "Population-based FPGA solution to Mastermind game," *AHS, Proceedings of the first NASA/ESA conference on Adaptive Hardware and Systems*, pp. 237–246.

[74] Zobrist, A. L. (1970), "A new hashing method with applications for game playing," *Technical Report 88*, Department of Computer Science, University of Wisconsin, Madison, USA. Also in *ICGA Journal* (1990), Vol. 13, No. 2, pp. 69–73.

Appendix A. Equivalence Transformations for AB Game at the Second Query

The following equivalence transformations for the second query of AB game transform the 209 codes into their corresponding representatives.

Table 14. Equivalence transformations

Order	**Representative**	**Each query**	**Equivalence transformations**
1	0123	-	-
2	0132	0213	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$
		0321	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		1023	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&0&1&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\2&3&0&1\end{pmatrix}$
		2103	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&0&2&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\1&3&0&2\end{pmatrix}$
		3120	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&0&3&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\1&2&0&3\end{pmatrix}$
3	4567	-	-
4	0231	0312	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&3&2&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}$
		1203	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&1&2&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\3&0&1&2\end{pmatrix}$
		1320	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&1&3&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\2&0&1&3\end{pmatrix}$
		2013	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&2&1&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\3&0&2&1\end{pmatrix}$
		2130	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&2&3&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\1&0&2&3\end{pmatrix}$
		3021	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&3&1&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\2&0&3&1\end{pmatrix}$
		3102	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&3&2&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\1&0&3&2\end{pmatrix}$

5	1032	2301	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		3210	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$
6	1230	1302	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}$
		2031	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&3&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&3&1\end{pmatrix}$
		2310	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		3012	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&2&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&2&1\end{pmatrix}$
		3201	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$
7	0124	0143	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}$
		0423	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&3&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&3&1\end{pmatrix}$
		4123	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&3&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&3&0\end{pmatrix}$
8	0456	4156	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&2&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&2&3\end{pmatrix}$
		4526	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&1&3\end{pmatrix}$
		4563	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&1&2\end{pmatrix}$
9	0145	0425	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		0453	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$
		4125	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&0&3\end{pmatrix}$
		4153	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&0&2\end{pmatrix}$
		4523	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&0&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&0&1\end{pmatrix}$
10	1456	2456	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$

		3456	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&6&4&5&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$
		4056	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&2&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&2&3\end{pmatrix}$
		4256	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&0&3&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&0&3\end{pmatrix}$
		4356	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&0&2&6&4&5&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&0&2\end{pmatrix}$
		4506	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&1&3\end{pmatrix}$
		4516	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&1&0&3&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&1&0&3\end{pmatrix}$
		4536	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&0&1&6&4&5&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&0&1\end{pmatrix}$
		4560	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&1&2\end{pmatrix}$
		4561	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&1&0&2&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&1&0&2\end{pmatrix}$
		4562	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&2&0&1&6&4&5&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&2&0&1\end{pmatrix}$
11	0134	0142	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}$
		0243	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$
		0324	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		0413	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&2&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&2&1\end{pmatrix}$
		0421	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&3&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&3&1\end{pmatrix}$
		1423	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&0&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&0&1\end{pmatrix}$
		2143	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&0&2\end{pmatrix}$
		3124	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&0&3\end{pmatrix}$
		4023	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&1&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&1&0\end{pmatrix}$

		4103	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&2&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&2&0\end{pmatrix}$
		4120	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&3&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&3&0\end{pmatrix}$
12	0245	0345	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&3&2&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}$
		0415	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		0435	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&3&1&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&3&1\end{pmatrix}$
		0451	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$
		0452	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&2&1&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&2&1\end{pmatrix}$
		1425	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&1&3\end{pmatrix}$
		1453	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&1&2\end{pmatrix}$
		2145	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&2&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&2&3\end{pmatrix}$
		2453	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&2&1&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&2&1\end{pmatrix}$
		3145	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&3&2&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&3&2\end{pmatrix}$
		3425	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&3&1&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&3&1\end{pmatrix}$
		4025	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&1&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&1&0&3\end{pmatrix}$
		4053	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&1&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&1&0&2\end{pmatrix}$
		4105	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&0&3\end{pmatrix}$
		4135	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&3&0&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&3&0\end{pmatrix}$
		4150	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&0&2\end{pmatrix}$
		4152	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&2&0&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&2&0\end{pmatrix}$

		4253	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&1&2&0&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&1&2&0\end{pmatrix}$
		4325	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&1&3&0&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&1&3&0\end{pmatrix}$
		4503	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&2&0&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&2&0&1\end{pmatrix}$
		4513	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&2&1&0&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&2&1&0\end{pmatrix}$
		4520	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&0&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&0&1\end{pmatrix}$
		4521	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&1&0&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&1&0\end{pmatrix}$
13	1045	2405	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		3450	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$
		4215	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&0&3\end{pmatrix}$
		4351	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&0&2\end{pmatrix}$
		4532	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&0&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&0&1\end{pmatrix}$
14	1245	1345	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&3&2&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}$
		1405	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&1&3\end{pmatrix}$
		1450	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&1&3\end{pmatrix}$
		2045	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&2&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&2&3\end{pmatrix}$
		2415	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		2435	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&3&1&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&3&1\end{pmatrix}$
		2450	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&2&1&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&2&1\end{pmatrix}$
		3045	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&3&2&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&3&2\end{pmatrix}$

		3405	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&3&1&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&3&1\end{pmatrix}$
		3451	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$
		3452	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&2&1&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&2&1\end{pmatrix}$
		4015	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&1&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&1&0&3\end{pmatrix}$
		4051	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&1&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&1&0&2\end{pmatrix}$
		4205	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&0&3\end{pmatrix}$
		4235	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&3&0&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&3&0\end{pmatrix}$
		4251	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&1&2&0&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&1&2&0\end{pmatrix}$
		4315	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&1&3&0&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&1&3&0\end{pmatrix}$
		4350	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&0&2\end{pmatrix}$
		4352	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&2&0&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&2&0\end{pmatrix}$
		4502	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&2&0&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&2&0&1\end{pmatrix}$
		4512	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&2&1&0&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&2&1&0\end{pmatrix}$
		4530	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&0&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&0&1\end{pmatrix}$
		4531	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&1&0&5&4&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&1&0\end{pmatrix}$
15	1435	1452	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}$
		2345	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		2451	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&1&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&1&0&2\end{pmatrix}$
		3245	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&0&3\end{pmatrix}$

		3415	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&1&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&1&0&3\end{pmatrix}$
		4035	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&2&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&2&3\end{pmatrix}$
		4052	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&3&2\end{pmatrix}$
		4250	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&1&2\end{pmatrix}$
		4305	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&1&3\end{pmatrix}$
		4501	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&3&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&3&1\end{pmatrix}$
		4510	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&2&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&2&1\end{pmatrix}$
		0341	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}$
		0432	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&3&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&3&1\end{pmatrix}$
		1024	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&1&3\end{pmatrix}$
		1043	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&1&2\end{pmatrix}$
		2104	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&2&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&2&3\end{pmatrix}$
16	0214	2403	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&2&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&2&1\end{pmatrix}$
		3140	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&3&2\end{pmatrix}$
		3420	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&3&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&3&1\end{pmatrix}$
		4132	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&3&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&3&0\end{pmatrix}$
		4213	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&1&2&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&1&2&0\end{pmatrix}$
		4321	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&1&3&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&1&3&0\end{pmatrix}$
17	0234	0241	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$

		0314	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		0342	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}$
		0412	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&2&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&2&1\end{pmatrix}$
		0431	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&3&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&3&1\end{pmatrix}$
		1243	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&1&2\end{pmatrix}$
		1324	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&1&3\end{pmatrix}$
		1403	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&2&0&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&2&0&1\end{pmatrix}$
		1420	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&0&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&0&1\end{pmatrix}$
		2043	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&1&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&1&0&2\end{pmatrix}$
		2134	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&2&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&2&3\end{pmatrix}$
		2140	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&0&2\end{pmatrix}$
		2413	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&2&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&2&1\end{pmatrix}$
		3024	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&1&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&1&0&3\end{pmatrix}$
		3104	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&0&3\end{pmatrix}$
		3142	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&3&2\end{pmatrix}$
		3421	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&3&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&3&1\end{pmatrix}$
		4013	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&2&1&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&2&1&0\end{pmatrix}$
		4021	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&1&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&1&0\end{pmatrix}$
		4102	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&2&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&2&0\end{pmatrix}$

		4130	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&3&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&3&0\end{pmatrix}$
		4203	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&1&2&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&1&2&0\end{pmatrix}$
		4320	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&1&3&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&1&3&0\end{pmatrix}$
18	1034	1042	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}$
		1432	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&0&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&0&1\end{pmatrix}$
		2304	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		2341	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&0&2\end{pmatrix}$
		2401	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&3&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&3&1\end{pmatrix}$
		3214	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&0&3\end{pmatrix}$
		3240	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$
		3410	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&2&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&2&1\end{pmatrix}$
		4032	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&1&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&1&0\end{pmatrix}$
		4210	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&3&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&3&0\end{pmatrix}$
		4301	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&2&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&2&0\end{pmatrix}$
19	1234	1240	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&1&2\end{pmatrix}$
		1304	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&1&3\end{pmatrix}$
		1342	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}$
		1402	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&2&0&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&2&0&1\end{pmatrix}$
		1430	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&0&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&0&1\end{pmatrix}$

		2034	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&2&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&2&3\end{pmatrix}$
		2041	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&1&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&1&0&2\end{pmatrix}$
		2314	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		2340	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&0&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&0&2\end{pmatrix}$
		2410	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&0&2&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&0&2&1\end{pmatrix}$
		2431	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&3&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&2&3&1\end{pmatrix}$
		3014	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&1&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&1&0&3\end{pmatrix}$
		3042	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&0&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&0&3&2\end{pmatrix}$
		3204	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&0&3&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&0&3\end{pmatrix}$
		3241	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$
		3401	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&0&3&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&0&3&1\end{pmatrix}$
		3412	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&2&1&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&3&2&1\end{pmatrix}$
		4012	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&2&1&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&2&1&0\end{pmatrix}$
		4031	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&3&1&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&3&1&0\end{pmatrix}$
		4201	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\3&1&2&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\3&1&2&0\end{pmatrix}$
		4230	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&3&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&2&3&0\end{pmatrix}$
		4302	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&2&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\1&3&2&0\end{pmatrix}$
		4310	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\2&1&3&0&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\2&1&3&0\end{pmatrix}$
20	1204	1340	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&1&3&2&4&5&6&7&8&9\end{pmatrix},P=\begin{pmatrix}0&1&2&3\\0&1&3&2\end{pmatrix}$

		2014	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&1&3&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\0&2&1&3\end{pmatrix}$
		2430	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&2&3&1&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\0&2&3&1\end{pmatrix}$
		3041	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&1&2&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\0&3&1&2\end{pmatrix}$
		3402	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\0&3&2&1&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\0&3&2&1\end{pmatrix}$
		4231	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&2&3&0&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\1&2&3&0\end{pmatrix}$
		4312	$C=\begin{pmatrix}0&1&2&3&4&5&6&7&8&9\\1&3&2&0&4&5&6&7&8&9\end{pmatrix}, P=\begin{pmatrix}0&1&2&3\\1&3&2&0\end{pmatrix}$

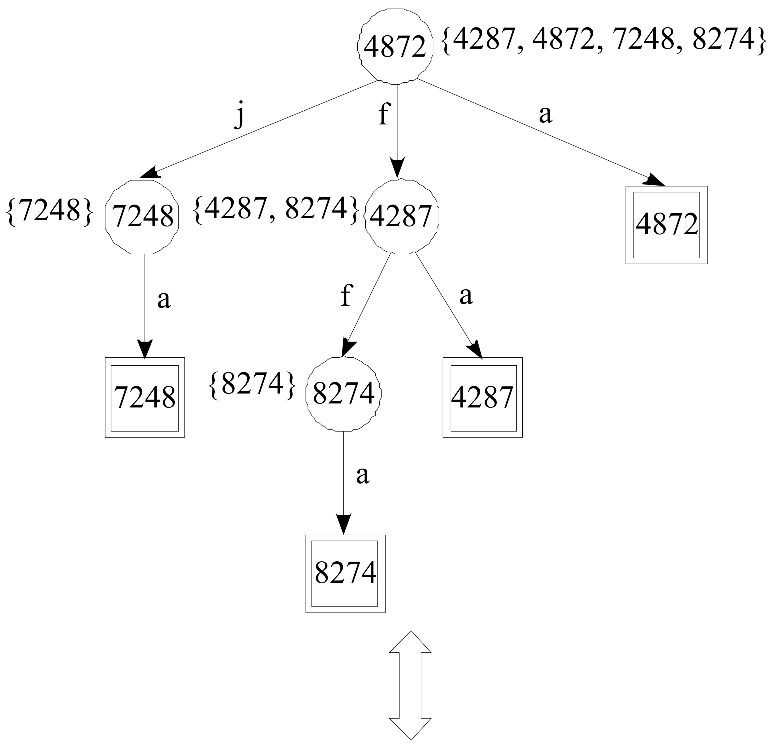
Appendix B. Optimal Strategy for AB Game in the Expected Case

Prior to introducing the optimal strategy of AB game, its representation will be illustrated first. The lower-case alphabets, a, b, c, …, m, n, represent the 14 responses (hints), as shown in Table 15.

Table 15. The mapping between responses and representative letters

Response	**Representative letter**	**Response**	**Representative letter**
[4, 0]	a	[1, 1]	h
[3, 0]	b	[1, 0]	i
[2, 2]	c	[0, 4]	j
[2, 1]	d	[0, 3]	k
[2, 0]	e	[0, 2]	l
[1, 3]	f	[0, 1]	m
[1, 2]	g	[0, 0]	n

Three kinds of tokens will appear in the strategy. The first kind is four-digit Arabic numerals, which means the query made by the codebreaker. The second one is lower-case letters mentioned above, which indicate the responses. The last kind is parentheses. The tokens in parentheses refer to the optimal tactic of the state. In other words, it is an optimal game tree of that state. The tactic is constructed with a recursive form and can be treated as a game tree. For example, suppose that a game tree depicted in Figure 21 is given. Then its corresponding representation will be "4872 (j 7248 (a) f 4287 (f 8274 (a) a) a)". Furthermore, it is easy to reconstruct the game tree from its representation with depth-first ordering.



4872 (j 7248 (a) f 4287 (f 8274 (a) a) a)

Figure 21. The transformation between the game tree and its corresponding representation

The derived optimal strategy of AB game in the average case is shown partially as follows due to space restrictions. In order to clarify the levels, we use an indent structure. We have established a website (http://www.csie.ntnu.edu.tw/~linss/ABgame/optimal_strategy.html) that includes the full text of the optimal strategy.

```
0123 ( n 4567 ( l 5689 ( l 7498 ( j 8974 ( c 9874 ( a )
                                              a )
                                   f 8794 ( f 9748 ( a )
                                              a )
                                   c 7894 ( f 7948 ( a )
                                              j 9478 ( a )
```

```
                  a )
         a )
k 6948 ( l 8795 ( c 7895 ( a )
                  f 9875 ( a )
                  a )
         k 8495 ( l 9876 ( a )
                  j 9854 ( a )
                  h 7896 ( a )
                  e 8796 ( a )
                  a )
         j 8496 ( a )
         h 8975 ( j 9758 ( a )
                  a )
         g 8954 ( j 9845 ( a )
                  f 9458 ( a )
                  e 8976 ( a )
                  a )
         f 6894 ( f 9846 ( a )
                  a )
         e 7958 ( a )
         d 6798 ( l 8945 ( a )
                    a )
                c 6498 ( j 8946 ( a )
                    a )
                b 6978 ( a )
          a )
```

// The full text of the optimal strategy is included at http://www.csie.ntnu.edu.tw/~linss/ABgame/optimal_strategy.html.

Appendix C. Publication List

(a) Referred Papers:

✧ **As a PhD student**

[a1] Huang, L. T., and Lin, S. S. (2009), "Optimal analyses for 3×*n* AB games in the worst case," to appear in Lecture Notes in Computer Science series.

[a2] Huang, L. T., Chen, S. T., Huang, S. J., and Lin, S. S. (2007), "An efficient approach to solve Mastermind optimally," *ICGA Journal*, Vol. 30, No. 3, pp. 143–149.

[a3] Chen, S. T., Lin, S. S., and Huang, L. T. (2007), "A two-phase optimization algorithm for Mastermind," *Computer Journal*, Vol. 50, No. 4, pp. 435–443.

[a4] Chen, S. T., Lin, S. S., Huang, L. T., and Hsu, S. H. (2007), "Strategy optimization for deductive games," *European Journal of Operational Research*, Vol. 183, No. 2, pp. 757–766.

✧ **As a master's student**

[a5] Huang, L. T., Chen, S. T., and Lin, S. S. (2006), "Exact-bound analyses and optimal strategies for Mastermind with a lie," *Lecture Notes in Computer Science*, *Advances in Computer Games 11*, Vol. 4250, pp. 195–209.

[a6] Chen, S. T., Lin, S. S., Huang, L. T., and Wei, C. J. (2004), "Towards the Exact Minimization of BDDs — An Elitism-Based Distributed Evolutionary Algorithm," *Journal of Heuristics: Special Issue on New Advance on Parallel Meta-Heuristics for Complex Problems*, Vol. 10, No. 3, pp. 337–355.

(b) Submitted Paper:

[b1] Huang, L. T., Chen, S. T., and Lin, S. S. (2009), “Optimal Algorithm for AB Game in the Expected Case,” submitted to *IEEE Transactions on Computational Intelligence and AI in Games*.

(c) Conference Papers:

✧ **As a PhD student**

[c1] Huang, L. T., and Lin, S. S. (2009), “Optimal analyses for 3×*n* AB games in the worst case,” *The 12th conference on Advances in Computer Games* (*ACG12*), Pamplona, Spain.

[c2] Huang, L. T., Chen, S. T., Huang, S. J., and Lin, S. S. (2007), “An efficient approach to solve Mastermind optimally,” *COMPUTER GAMES WORKSHOP 2007*, Amsterdam, The Netherlands.

[c3] Chen, S. T., Lin, S. S., Chang, S. W., and Huang, L. T. (2006), “A two-phase search algorithm for the set covering problem”，第十一屆人工智慧與應用研討會，國立高雄應用科技大學，台灣，中華民國。

✧ **As a master’s student**

[c4] Huang, L. T., Chen, S. T., and Lin, S. S. (2005), “Exact-bound analyses and optimal strategies for Mastermind with a lie,” *The 11th Advances in Computer Games Conference* (*ACG11*), Taipei, Taiwan.

(d) Technical Reports:

[d1] 陳善泰、黃立德、張書維、劉耀才、江漢昇、胡淑瓊，2005，國科會研究報告：演繹競局問題最佳化策略及其應用於容錯系統之研究(2/2)，NSC93-2213-E-003-001。

[d2] 陳善泰、黃立德、張書維、劉耀才、江漢昇、胡淑瓊，2004，國科會研究報告：演繹競局問題最佳化策略及其應用於容錯系統之研究(1/2)，NSC92-2213-E-003-006。