第四章 容許 e 次錯誤回應的 Mastermind 和 AB game

在本章中我們將先前所介紹的 k 分支演算法及以鴿籠原理為基礎的快速式回溯驗證演算法推廣並可應用到容許 e 次錯誤回應的 Mastermind 和 AB game 上，其中 $e \geq 1$。在實務上，我們以 $e = 2$ 為例實作程式並得到相關成果。

第一節 一般化的 k 分支演算法

我們所提出的 KWB 演算法很容易地可以推廣到一般化的情況（容許至多 e 個錯誤回應的 Mastermind 與 AB game，其中 $e \geq 1$），僅需針對少部分地方略做修改即可。

在一般的情況下，整個搜尋過程中，在第 i 次猜測後每個候選者序對 $\langle C_i^{(0)}, C_i^{(1)}, \ldots, C_i^{(e)} \rangle$ 內皆有 $e+1$ 個集合，詳見第一章第三節定義 1。除了候選者序對需做修改外，另外在 HCG 中的雜湊函數也需要略作修改，因為必須考慮狀態中所有集合元素個數分佈的平均狀況。若對某個候選者序對 $\langle C_i^{(0)}, C_i^{(1)}, \ldots, C_i^{(e)} \rangle$ 而言，密碼破解者猜測一個密碼 g 後，會將此 $e+1$ 個集合的所有密碼分別分割產生 14 個類別集合，共 14 * (e + 1)群集合。假設 $\langle C_i^{(0)}, C_i^{(1)}, \ldots, C_i^{(e)} \rangle_g$ 表示密碼破解者經過 i 次猜測後目前面臨的候選者序對時，並做第 i + 1 次猜測 g。而 $\langle R_{i+1,1}^{(0)}, R_{i+1,2}^{(0)}, \ldots, R_{i+1,14}^{(0)} \rangle$, $\langle R_{i+1,1}^{(1)}, R_{i+1,2}^{(1)}, \ldots, R_{i+1,14}^{(1)} \rangle$, ..., $\langle R_{i+1,1}^{(e)}, R_{i+1,2}^{(e)}, \ldots, R_{i+1,14}^{(e)} \rangle$ 分別代表 $C_i^{(0)}, C_i^{(1)}, \ldots, C_i^{(e)}$ 被 g 分割後 14 個類別集合所成的序對，所以我們定義一般化下的雜湊函數為

$$Hash\left(\langle C_i^{(0)}, C_i^{(1)}, \ldots, C_i^{(e)} \rangle_g\right) = Str\left(Sort\left(\left|R_{i+1,j}^{(0)}\right|\right), Sort\left(\left|R_{i+1,j}^{(1)}\right|\right), \ldots, Sort\left(\left|R_{i+1,j}^{(e)}\right|\right)\right),$$

其中 j = 1, 2, . . . , 14，而 $Sort\left(\left|R_{i+1,j}^{(0)}\right|\right), Sort\left(\left|R_{i+1,j}^{(1)}\right|\right), \ldots, Sort\left(\left|R_{i+1,j}^{(e)}\right|\right)$ 則分別代表將

$\left(\left|R_{i+1,1}^{(0)}\right|,\left|R_{i+1,2}^{(0)}\right|,\ldots,\left|R_{i+1,14}^{(0)}\right|\right),\left(\left|R_{i+1,1}^{(1)}\right|,\left|R_{i+1,2}^{(1)}\right|,\ldots,\left|R_{i+1,14}^{(1)}\right|\right),\ldots,\left(\left|R_{i+1,1}^{(e)}\right|,\left|R_{i+1,2}^{(e)}\right|,\ldots,\left|R_{i+1,14}^{(e)}\right|\right)$中的 14 個數各以非遞增方式進行排序後形成的新序對，而 *Str* 函數為將此 $e+1$ 個序對中的 $14*(e+1)$個整數轉為字串後依序連接在一起，亦即如果經各自排序後所成的序對為 $\left(Sort\left(\left|R_{i+1,j}^{(0)}\right|\right),Sort\left(\left|R_{i+1,j}^{(1)}\right|\right),\ldots,Sort\left(\left|R_{i+1,j}^{(e)}\right|\right)\right)=\left(n_{g,1},n_{g,2},\ldots,n_{g,14*(e+1)}\right)$，則 $Str\left(Sort\left(\left|R_{i+1,j}^{(0)}\right|\right),Sort\left(\left|R_{i+1,j}^{(1)}\right|\right),\ldots,Sort\left(\left|R_{i+1,j}^{(e)}\right|\right)\right)=Str\left(n_{g,1}\right)Str\left(n_{g,2}\right)\ldots Str\left(n_{g,14*(e+1)}\right)$。此雜湊函數雜湊出來的值亦為一字串，代表索引的代號（*id*）。因此若兩個密碼 g 和 g' 被視為極可能等價，被歸類在同一 HCG 中，意指 $Hash\left(\left\langle C_i^{(0)},C_i^{(1)},\ldots,C_i^{(e)}\right\rangle_g\right)=Hash\left(\left\langle C_i^{(0)},C_i^{(1)},\ldots,C_i^{(e)}\right\rangle_{g'}\right)$。

至於如何選出最好的*k*群HCGs並從中各挑一個代表密碼的作法，相當類似在第二章第二節所描述的作法，亦即使用$d_0, d_1,\ldots, d_e$來調整每個集合的重要性，並使用評估函數*f*的觀念來比較HCGs中每一群的猜測密碼之好壞，唯一不同點在於先前因為$e=1$，所以使用*f*所比較的序對共有 28 欄位，而在一般化的情況下，只需將序對的欄位修改為 $14*(e+1)$個並用相同的觀念比較不同的猜測密碼之好壞即可。圖 13 為KWB演算法在一般情況下的示意圖，其中$\left\langle C_{i,j}^{(0)},C_{i,j}^{(1)},\ldots,C_{i,j}^{(e)}\right\rangle$代表在第*i*次猜測後類別為*j*之候選者序對，$g_{i,j}$為第*i*次猜測時KWB演算法所選出*k*個最好的密碼中的第*j*個。

至於在如何決定$d_0, d_1, \ldots, d_e$參數值的方面，我們提出一個經驗法則的等比數列公式來計算此$e+1$ 個參數值，其公式為

$$d_i=\alpha^{(e-i)},\ i=1,2,\ldots,e$$

此公式代表的意義為錯誤數越少的候選者集合被某個猜測分割後的分佈情況越重要。在實作的程式中，我們令$\alpha=3$，亦即 $d_0=3^e, d_1=3^{(e-1)},\ldots, d_e=1$。

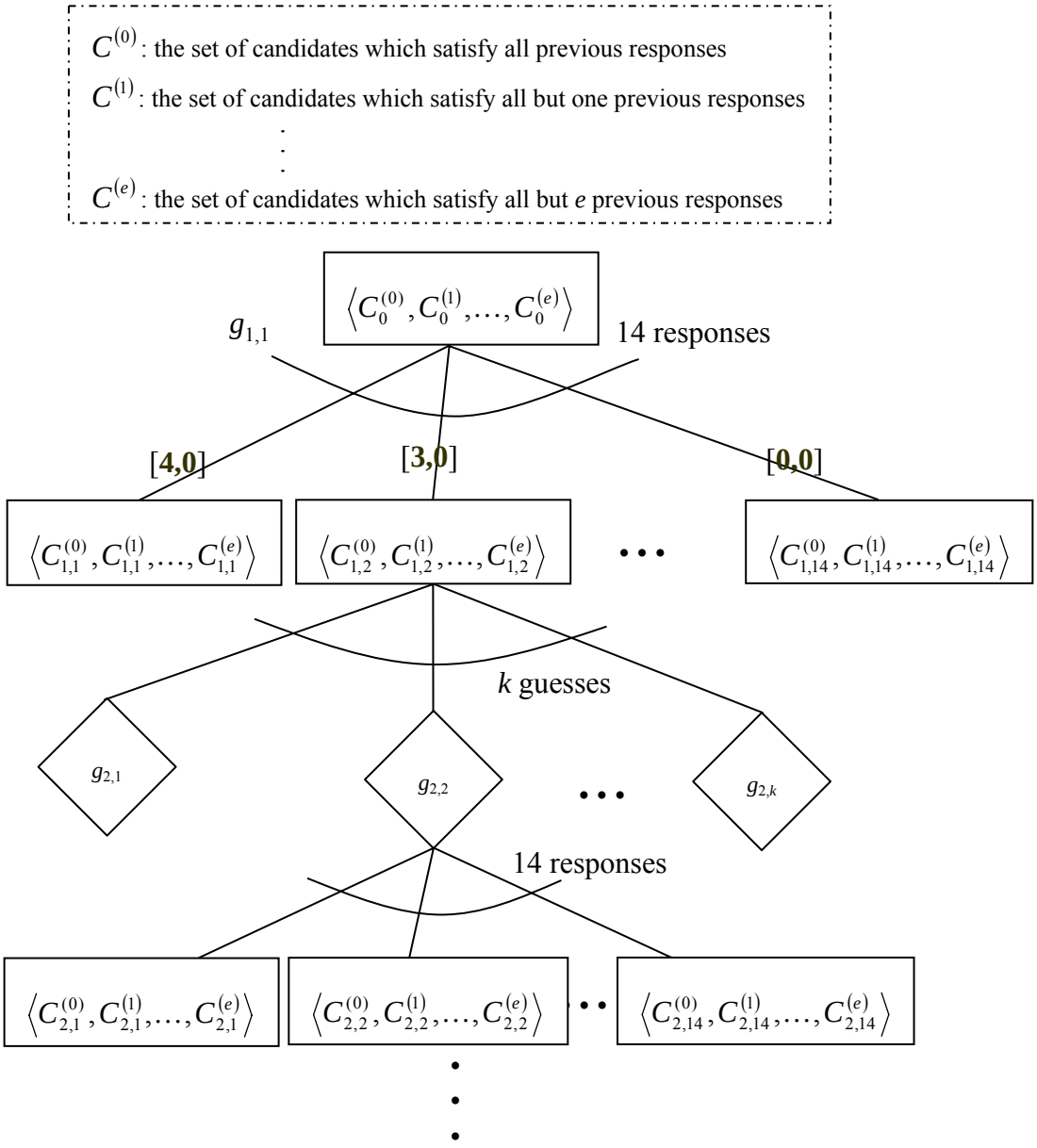

圖 13 :一般化的 KWB 演算法應用於容許 e 次錯誤回應之演繹競局問題

第二節 一般化的以鴿籠原理為基礎之快速式回溯驗證演算法

我們接著將以鴿籠原理為基礎的快速式回溯驗證演算法做推廣，稱為一般化的 PPBFB 演算法，並利用一般化的 PPBFB 演算法可求得容許 e 次錯誤回應的 Mastermind 和 AB game 之競局問題所需的最少猜測次數，亦即所謂的下限。

一般化的PPBFB演算法的原理相當類似在第二章與第三章的PPBFB演算

法，換言之，其亦以最差優先搜尋進行窮舉且在搜尋過程中利用延伸鴿籠原理預先估計下限而提早回溯，以節省窮舉搜尋的時間。圖 14 是一般化的PPBFB演算法之示意圖。其中 $g_{i,j}$ 為第i次猜測時的第j個可能密碼。$r_{i,\max}$代表在第i次猜測後，14 個類別中，會導致最多猜測次數的類別（亦即所謂的最差情況），M表示密碼破解者在當時所需考慮的可能密碼集合（|M|代表其集合元素個數），$l_{\max}$代表由該狀態到最終狀態的理論下限，h代表我們所要驗證的下限值。

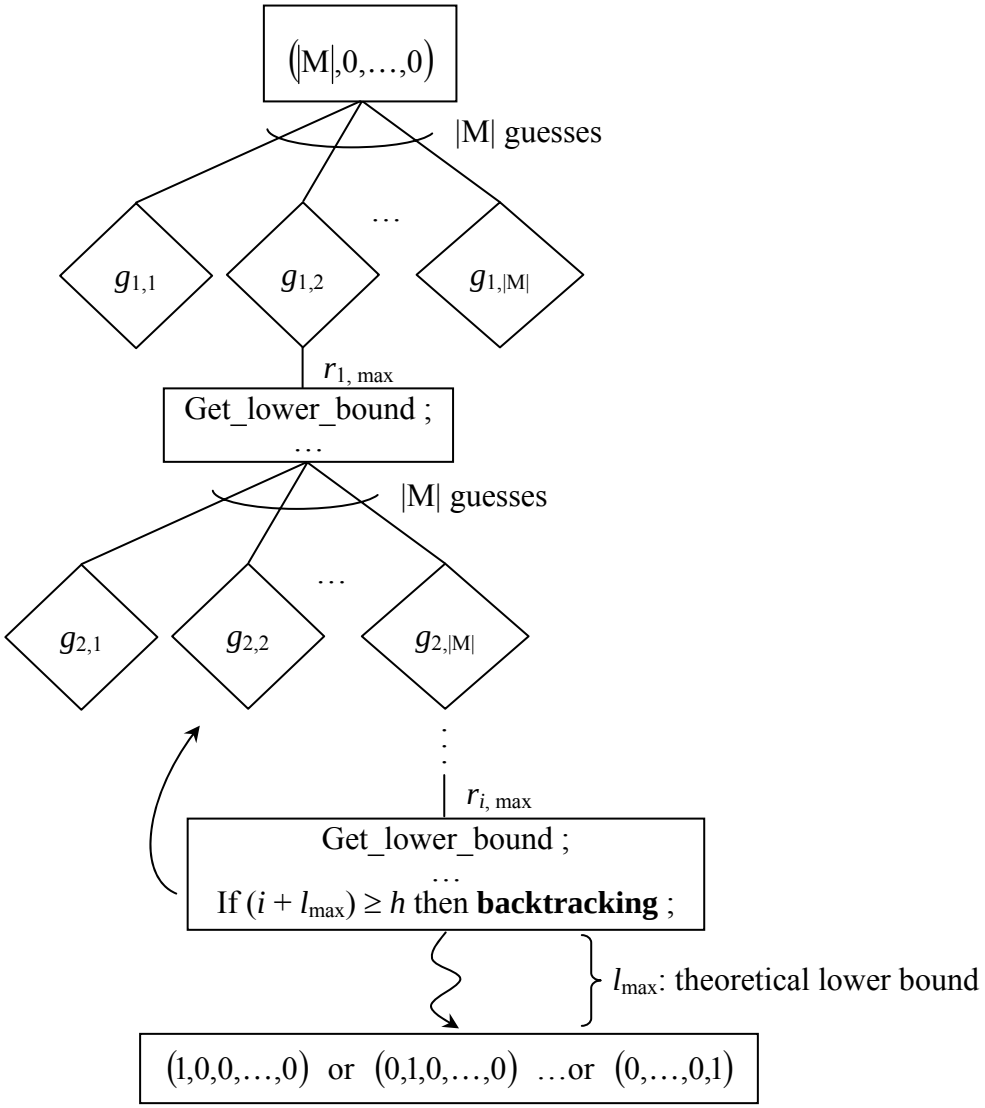

圖 14 :一般化的 PPBFB 演算法應用於容許 e 次錯誤回應之演繹競局問題

一般化的 PPBFB 演算法與 PPBFB 演算法的不同處在於在搜尋中的每個節點皆有 $e+1$ 個集合，對應到其狀態為 $e+1$ 個自然數。既然狀態變為 $e+1$ 個數，那 PPBFB 演算法中相關的演算法都需要跟著做修改。例如圖 5 的演算法求取容

量集合時，即必須求分別對應到$C^{(0)}, C^{(1)} \cup C^{(0)}, \ldots, C^{(e)} \cup \ldots \cup C^{(0)}$的 $e+1$ 個容量集合。對於圖 6 中的估計下限演算法也需要修改，因為每個狀態皆有 $e+1$ 數，因此必須使用所求得的 $e+1$ 個容量集合與延伸鴿籠原理並模仿圖 6 中的觀念去計算我們所需的下限。至於我們先前所提出的二種增加搜尋速度的作法，對於「容量更新」技巧一樣是在競局樹第二層以上的狀態才重新計算容量集合，而因為目前狀態有 $e+1$ 欄位，因此使用「預先處理」技巧時，必須窮舉 $e+1$ 個欄位所有的可能狀態。其他一般化的 PPBFB 演算法之核心觀念均與 PPBFB 演算法相同。

第三節 本章的結論

本章將 k 分支演算法及以鴿籠原理為基礎的快速式回溯驗證演算法推廣到至多容許 e 個錯誤回應之演繹競局問題上。在實務上我們針對 $e=2$ 將上述二個演算法實作成程式，亦即為容許二次錯誤回應的 Mastermind 與容許二次錯誤回應的 AB game 之競局問題，並使用一台 AMD Opteron 1.6Ghz 的個人電腦進行實驗。在合理的時間和空間因素的考量下，我們得到以下結果：在容許二次錯誤回

表 3 :容許二次錯誤回應的 Mastermind 與 AB game 之上、下限與其計算時間

競局問題	上限			下限	
	k 值	所需猜測次數 H	時間（分）	欲驗證猜測次數 h	時間（分）
Mastermind	1	13	96.85	7	391.87
	2	10	2859.67		
	3	10	19051.24		
AB game	1	16	3078.03	8	157.35
	2	15	> 100000.00		

應的 Mastermind 問題上，我們求得的上限與下限各為 10 與 7，而在容許二次錯誤回應的 AB game 問題上，所求得的上限與下限各為 15 與 8。請參考表 3。

由理論及實驗中，我們知道當 e 越大時，意謂著搜尋空間與時間成指數成長。以 Mastermind 為例，由表 1 與表 3，若我們欲求上限且希望得到較好的解（設 $k = 3$）時，當 $e = 1$ 時，程式執行時間為 65.11 分，而當 $e = 2$ 時，程式執行時間為 19051.24 分，因此我們估計當 $e \geq 3$ 且 $k = 3$ 時，程式的執行時間至少需要一年以上，這遠遠超過合理的執行時間；另外，對於下限的求得，我們發現對當 $e = 1$ 與 $e = 2$ 時，我們的驗證程式若限定在合理的時間內必須搜尋完成，則都只能驗證到 $h = 7$，若要驗證 $h = 8$ 則會因為要在競局樹多搜尋一層，所以搜尋時間至少為 $h = 7$ 時的 1296*14 倍以上，亦遠超過合理的搜尋時間，因此當 $e \geq 3$ 時，此驗證程式所能求得的下限必然無法大幅提昇。另外對於 AB game 而言，在 e 相同的情況下，其搜尋空間遠比 Mastermind 大得多，所以搜尋的時間自然會更久。因此在實務上，因為我們的方法對 $e \geq 3$ 的 Mastermind 和 AB game 效果有限，所以並沒有實作程式進行實驗。由於當 e 越大時，搜尋的難度會越大，因此探討容許多次錯誤回應的演繹競局問題時，如何進行更有效率的搜尋是一個重要的課題。