國立交通大學

資訊科學系

碩士論文

進階區域式對角型 Tuple Space 搜尋之快速封包分類演算法

Fast Packet Classification Using Advanced Regional Diagonal Tuple Space Search

研 究 生：高鳴遠

指導教授：陳 健 教授

中 華 民 國 九 十 四 年 十二 月

進接區域式對角型 Tuple Space 搜尋之快速封包分類演算法

Fast Packet Classification Using Advanced Regional Diagonal Tuple Space Search

研 究 生：高鳴遠　　Student：Ming-Chao Huang

指導教授：陳　健　　Advisor：Chien Chen

國 立 交 通 大 學

資 訊 科 學 系

碩 士 論 文

A Thesis
Submitted to Department of Computer and Information Science
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
In

Computer and Information Science

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年十二月

國立交通大學
研究所碩士班

論文口試委員會審定書

本校 資訊科學與工程 研究所 高鳴遠 君

所提論文: 進階區域式對角型 Tuple Space 搜尋之快速封包分類演算法

Fast Packet Classification Using Advanced Regional Diagonal Tuple Space Search

合於碩士資格水準、業經本委員會評審認可。

口試委員：

指導教授：

所長：教授兼資訊科學與工程研究所所長 曾文貴

系主任：

中華民國九十四年十二月二十七日

Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.

As members of the Final Examination Committee, we certify that we have read the thesis prepared by Min-Yung Kao

entitled Fast Packet Classification Using Advanced Regional Diagonal Tuple Space Search

and recommend that it be accepted as fulfilling the thesis requirement for the Degree of Master of Science.

Thesis Advisor:

Chairman:

Date: 2005/12/27

進階區域式對角型Tuple Space 搜尋之 快速封包分類演算法

研究生：高鳴遠　　　　　　　　　　　　　　　　指導教授：陳 健　教授

國立交通大學電機資訊學院　資訊科學所碩士班

摘要

為了提供安全防護，虛擬私有網路，品質保證等等網際網路的服務。網際網路路由器需要將收到的封包進行快速的分類。封包分類是利用在封包標頭所包含的資訊與路由器中事先定義的規則表進行比對，一般而言，多重欄位的封包分類是一個相當困難的問題，已經有許多不同的演算法被提出來解決這個問題。在這篇論文中，我們提出一個稱為Addvanced Regional Diagonal Tuple Space Search的封包分類演算法，採用Diagonal Tuple Space Search之觀念，並利用分區搜尋，鏈結搜尋，單向搜尋等三種搜尋加以改良演算法：分區搜尋將整個Tuple space切割成許多區塊；鏈結搜尋使得搜尋進展能跳越至指定區域；單向搜尋則能將Diagonal seacr的至多三次二元搜尋減少為二次二元搜尋。透過實驗的觀察，可以將搜尋次數提升為只需 2logw，w為維度之長度；同時對儲存空間的複雜度由$O(n2^{w}\log w)$改變為$O(n\log w + n^{2}w)$，n表示規則之總數。在封包分類的效能上。因為Addvanced Regional Diagonal Tuple Space Search只需要比Diagonal Tuple Space Search更少的記憶體存取時間，而在封包分類的問題中，記憶體的存取往往決定了整體的查詢時間，所以，即使Addvanced Regional Diagonal Tuple Space Search需要另外對區塊作搜尋的步驟，仍會有比Diagonal Tuple Space Search更快的分類速度。

Fast Packet Classification Using Advanced Regional Diagonal Tuple Space Search

Student: Ming-Yung Kao　　　　　　Advisor: Prof. Chia-Hoang Lee

Department of Computer and Information Science
National Chiao Tung University

英文摘要

Abstract

In order to support Internet security, virtual private networks, QoS and etc., Internet routers need to classify incoming packets quickly into flows. Packet classification uses information contained in the packet header to look up the predefined rule table in the routers. In general, packet classification on multiple fields is a difficult problem. A variety of algorithms had been proposed. This thesis presents a novel packet classification algorithm, called advanced regional diagonal tuple space search algorithm using the concept of diagonal tuple space search algorithmand three search types: block seach, link-list search and single-direction search. Block search can divide all tuple space into many small blocks; link-list search can jump to assigned blocks to start process; single-direction search oly need two binary search. Our experiment results show that the advanced regional diagonal tuple space search algorithm only need 2 log w, where w denotes the length of dimensions, and the storage complexity from

$O(n2^w \log w)$ of the diagonal tuple space search algorithm to $O(n\log w + n^2 w)$, where n represents the number of rules. On the classification performance, the advanced regional diagonal tuple space search algorithm requires much less memory access time than diagonal tuple space search algorithm. Since memory access dominates the lookup time. Even though extra processing time for searching areas is required for the advanced regional diagonal tuple space search algorithm, the advanced regional diagonal tuple space search algorithm still outperforms diagonal tuple space algorithm on the classification speed.

誌謝

本論文得以順利完成，首先要感謝指導教授陳健老師與李嘉晃老師，其嚴謹的研究態度及細心的指導，使我在這些日子裡受益良多。其次要感謝口試委員簡榮宏教授、陳志成教授的指正與建議，使得本論文能更加完善。

此外，感謝陳健教授的Wireless/Wireline Convergence Networks實驗室的游宸同學，以及我研究室的明超、駿豪、沛言、佑銘、建良、國懿、信宏、志鵬、開國、仁信等同學與學弟們對我的關心與幫助，豐富了我的實驗室生活。

最後感謝我的家人，因為他們的支持是我努力的最大動力。謹以此篇論文給我的摯愛們。

高鳴遠 謹誌於交通大學

民國九十四年十二月

目錄

中文摘要...i
英文摘要...ii
誌謝...iv
目錄...v
表目錄...vii
圖目錄...viii
第一章　緒論...1
1.1 背景...1
1.2 Performance Metrics for Packet Classification Algorithm.....4
1.3貢獻...4
1.4論文架構...5
第二章　與封包分類有關之演算法.................................6
2.1前言...6
2.2演算法之簡介...6
2.2.1 Linear search..6
2.2.2 Hierarchical tries...7
2.2.3 Set-pruning tries..7
2.2.4 Grid of tries..7
2.2.5 Cross-producting...8
2.2.6 Are-base quadtree..8
2.2.7 Hierarchical intelligent cuttings..........................9
2.2.8 Hyper-Cuts...9
2.2.9 Recursive flow classification..............................9
2.2.10 Ternary content addressable memory........................10
2.2.11 Bit-map intersection......................................10
2.2.12 Aggregated Bit Vector.....................................11
2.3演算法之比較...11
第三章　相關研究...13
3.1 Tuple Space..13
3.1.1 Tuple Space之定義...13
3.1.2 Tuple Relationship..15
3.1.3 Marker 與 Precomputation..................................16
3.2 Rectangle Search...18
3.3 Diagonal Tuple Space Search................................20
第四章　Advanced Regional Diagonal Tuple Space Search..........24
4.1動機...24

4.2資料結構的描述 26
4.3演算法的描述 27
4.3.1 分區... 27
4.3.2 Link-list與marker生成.................................. 30
4.4衝突(Conflict)的處理 41
4.4.1單方向衝突的處理.. 41
4.4.2雙方向衝突的處理.. 43
4.5流程的說明 49
4.5.1 Marker生成的流程....................................... 49
4.5.2搜尋的流程.. 50
4.6演算法的証明 52
4.7搜尋範例 54
第五章 模擬與分析.. 56
5.1 Advanced regional diagonal tuple space search之分析環境 56
5.1.1分區之測試.. 56
5.1.2 Link-list之測試.. 59
5.1.3單方向之測試.. 60
5.2 總合模擬.. 62
5.2.1總合模擬之環境.. 62
5.2.2模擬結果.. 63
5.3複雜度的計算 65
第六章 結論與未來工作.. 67
參考資料... 68

表目錄

表1-1：封包分類之規則表範例.. 3
表2-1：封包分類演算法之分類。....................................... 6
表2-2：封包分類演算法之複雜度比較。................................. 12
表3-1：規則對應至Tuple之範例。...................................... 14
表3-2：表 3-1 之Tuple space。 14
表5-1：當有 0%的規則位於Wildcard區域時，各個演算法分別生成的marker總數量以及平均搜尋結果。....................................... 63
表5-2：當有 20%的規則位於Wildcard區域時，各個演算法分別生成的marker總數量以及平均搜尋結果。....................................... 63
表5-3：當有 50%的規則位於Wildcard區域時，各個演算法分別生成的marker總數量以及平均搜尋結果。....................................... 63
表5-4：Rectangle search，Diagonal tuple space search及Advanced regional diagonal tuple space search之複雜度比較。..................... 66

圖目錄

圖1-1：封包分類處理之流程圖.. 2
圖3-1：Tuple space可視為一二維空間。............................. 15
圖3-2：Tuple(x，y)與其他tuple之關係之二維空間表示圖。............... 16
圖3-3：Precomputation所包含之範圍。.............................. 18
圖3-4：Marker的生成。.. 18
圖3-5：Rectangle search之範例。.................................. 19
圖3-6：Rectangle search之最糟情形。.............................. 20
圖3-7：Diagonal tuple space search之marker生成方向。............. 21
圖3-8：Diagonal tuple search之範例。............................. 22
圖4-1：真實網路環境的規則配置情形。.............................. 24
圖4-2：Advanced Diagonal Tuple Space的區域配置圖示。.............. 27
圖4-3：區域內的marker配置圖示。.................................. 28
圖4-4：Linklist區間配置方式與header tuple之設置。................. 30
圖4-5.a：簡單的非對角線tuple之marker生成流程範例，加入R1。.......... 32
圖4-5.b：簡單的非對角線tuple之marker生成流程範例，加入R2。.......... 33
圖4-5.c：簡單的非對角線tuple之marker生成流程範例，加入R3。.......... 33
圖4-6：每一列(行)可與Diagonal tuple組成Link-list樹狀圖。............ 34
圖4-7：圖 4-6 之搜尋路徑圖。..................................... 35
圖4-8：二元搜尋較線性搜尋佳之例子。.............................. 36
圖4-9：線性搜尋較二元搜尋佳之例子。.............................. 36
圖4-10：動態決定區域搜尋演算法之流程圖。.......................... 36
圖4-11：最後對二元搜尋範圍內鏈結之規則或marker之處理。............. 38
圖4-12：封包單方向搜尋之流程圖。.................................. 40
圖4-13：產生衝突的範例圖示。...................................... 41
圖4-14：解決衝突之範例。.. 43
圖4-15：解決雙方向衝突之範例。.................................... 45
圖4-16.a：最佳配對規則可能分布範圍(a)。............................ 46
圖4-16.b：最佳配對規則可能分布範圍(b)。............................ 46
圖4-16.c：最佳配對規則可能分布範圍(c)。............................ 47
圖4-17：雙方向衝突的例外情形。.................................... 49
圖4-18：最深tuple(d，d)之最佳符合規則範圍圖。...................... 52
圖4-19：LongTuple與IncomparableTuple之範圍圖。..................... 53
圖4-20：Tuple space之範例圖。..................................... 54
圖5-1：演算法之差異性：採用分區之影響。............................ 57
圖5-2：規則總數為 1000 時，tuple(0，0)至tuple(15，15)中各個diagonal tuple 中的marker總數之折線圖。.................................... 57

圖5-3：規則總數為 5000 時，tuple(0，0)至tuple(15，15)中各個diagonal tuple 中的marker總數之折線圖。.................................... 58
圖5-4：規則總數為 10000 時，tuple(0，0)至tuple(15，15)中各個diagonal tuple 中的marker總數之折線圖。.................................... 58
圖5-5：演算法之差異性：採用link-list機制之影響。.................... 59
圖5-6：在不同的規則總數以及conflicting狀態下對tuple搜尋之平均結果折線圖。... 60
圖5-7：演算法之差異性：採用單方向機制之影響。...................... 60
圖5-8：在不同的規則總數以及conflicting狀態下對tuple搜尋之平均結果折線圖。... 61
圖5-9：在不同的規則總數以及conflicting狀態下的marker總數量之折線圖。 61

第一章 緒論

1.1 背景

隨著網際網路迅速的發展，使用網路的人數逐年增加，同時對於網路頻寬與品質的需求亦逐漸提高，進而邁向寬頻普及化的時代。人們逐漸能夠藉此從事各種通訊行為及互動，諸如網路安全性，防火牆(firewall)，Virtual private network(VPN)以及 Quality of service(QoS)等等的網路應用服務技術也隨著網際網路的重要性的提升而逐漸發展與成熟。為了達到這樣的應用，因此我們需要封包分類(Packet classification)技術的支援。封包分類技術是建立在路由器或網路閘道器上，其功能是將封包分類到不同的資料串流(Flows)，使得相對應資料串流的動作能夠快速地被執行，能有更好的封包處理效能。

網際網路的路由器(router)可以利用所收到的封包的表頭(header)作為資訊，並檢索預先定義的規則表(Predefined rule table)中所定義的規則，將封包分類至適當的類別中，規則表負責記載並管理著每一條規則以供檢索，範圍從數十條甚至是數千條規則不等，每一條規則的制定則是依據封包的表頭的欄位內容來設限，其中包含著網路來源端位址(Network source address)，網路目的端位址(Network destination address)，來源端接口(Source port)，目的端接口(Destination port)，協定型式(Protocol)等等可用資訊，而規則的定義可能是來源端位址或目的端位址的前幾位元字首(稱為 prefix)，可能是來源端接口或目的端接口中一段指定的範圍(稱為 range)，或是協定型式中用來表示某些協定的數字(如 TCP，UDP 或 ICMP)。

圖 1-1 為封包處理的流程 [1]，當接收到一個封包時，首先會截取出封包的表頭資訊並比對規則表中各個規則中的每一個欄位是否相符合，當一個規則能夠符合所有的欄位條件時，便將該規則視為可配對的規則(matching rule)，路由器會將所擷取到的封包表頭資訊與規則表中的規則進行比對分類動作，並找出數個可

符合的規則，從中挑選出擁有最高優先權的可配對規則，即為最佳符合規則(Best-matching rule)，也是分類該封包最佳的類別，最後路由器會根據最佳符合規則的內容而對該封包採取相對應的動作(接受或拒絕)。一般而言，處理封包分類的應用環境可以分成靜態與動態二種：在靜態的環境下較不需要去處理規則的插入及刪除等狀態，或是規則表的變動較不頻繁，相對的在動態環境中規則表可能有較為頻繁的更新動作，因此一個好的封包分類演算法必須能夠適應到各種的環境。

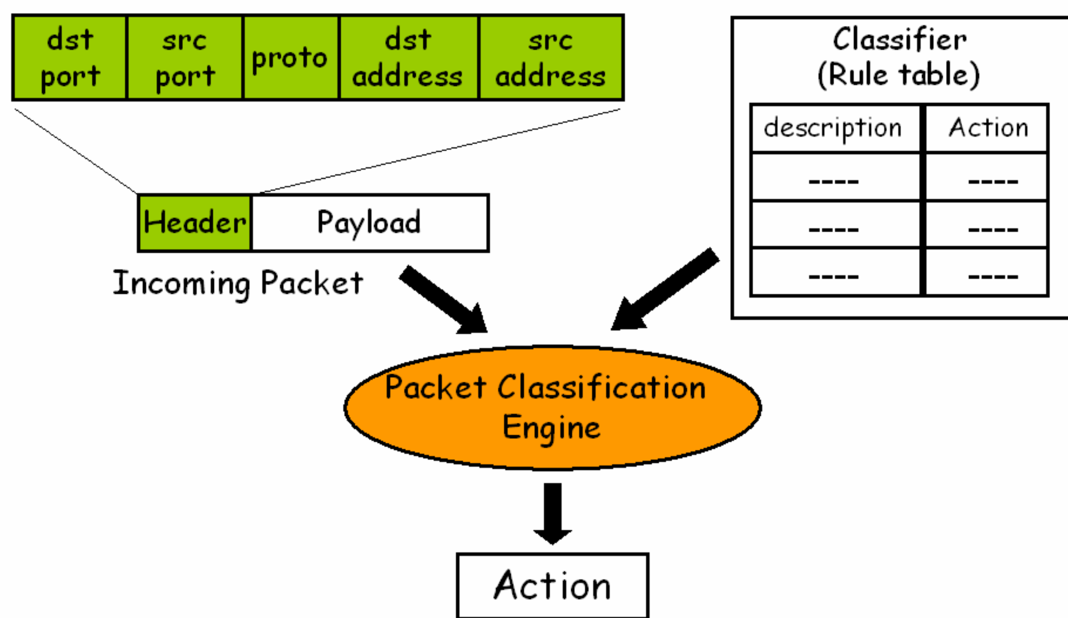


圖 1-1：封包分類處理之流程圖。

表 1-1 為規則表的範例 [2]。我們假設來源端位置與目的端位址皆為三位元字首長，位址欄裡頭的一個星星符號”*”視為一個遮罩位元，當一規則某一欄位內容為 11*時，則封包表頭資訊中相對應該欄位內容為 110 或 111 皆是為羽根規則相符合。當規則中某一欄位全是用星星符號表示時，則將規規則視為泛用規則(Wildcard rule)，亦即不論封包表頭資訊中該欄位內容為何，都能通過泛用規則在該欄位的檢查；而在來源端接口或是目的端接口等欄位中則是用一段數字範圍或是萬用字元來表示，最後的協定欄位則是指出明確的協定型式，如 TCP，UDP 或 ICMP，甚至是萬用字元等，規則都有指定各自對應的行為，分為接受與拒絕二種，

當一封包能通過規則表中的某一規則中的來源端位址，目的端位址，來源端接口，目的端接口，通訊協定等五個欄位的檢查比對時，便會採取該規則所對應的動作對封包做處理，每個規則都有各自的優先權設定，當封包與多各規則相符合時，會選取最高優先權的規則的動作為優先。在表 1-1 中，優先權高低的設定以編號 1 的規則為最高優先權，編號 5 的規則為最低優先權。假設一封包 P 表頭資訊中的來源端位址為 110，目的端位址為 011，來源端接口為 4，目的端接口為 6，通訊協定為 TCP，根據表 1-1 我們可以得知封包 P 能夠與規則 1 與規則 4 相符合，然而規則 1 有著較高的優先權，所以規則 1 便是封包 P 的最佳符合規則，接著根據規則 1 所對應的動作.封包 P 便被”拒絕接受”。

Rule	Source address	Destination address	Source port	Destination Port	Protocol type	Action
1	1**	010	2-4	6-9	TCP	Deny
2	101	***	1-7	4-6	UDP	Pass
3	00*	10*	*	*	ICMP	Deny
4	11*	01*	4-8	*	TCP	Pass
5	***	***	*	10-15	*	Deny

表 1-1：封包分類之規則表範例。

我們可以定義出 d-維封包處理問題：假設一規則表有一集合的規則 R={R1，R2，…，Rn}表現在 d 維空間上，每一個規則都是由 d 個欄位組成，Ri={$F_{1,i}$，$F_{2,i}$，…，$F_{d,i}$}，其中 $F_{j,i}$ 即為規則 i 的第 j 個欄位中的內容，同時每一個規則也需要一個定義優先權的數值，當收到一封包 P(p1，p2，…，pd)，若表頭資訊的所有欄位內容 p1，p2…，pd 皆能符合於某一規則 Ri 的所有欄位內容 $F_{1,i}$，$F_{2,i}$，…，$F_{d,i}$ 的範圍之內，便能將封包 P 視為能與規則 Ri 配對。如果封包 P 能與多個規則相符合，則路由器便會選擇擁有最高優先權的規則為所求，並對封包 P 採取相對應的動作。

1.2 Performance Metrics for Packet Classification Algorithm

而一個好的封包處理方法必須要滿足下列五點特性：

- 搜尋速度(Search speed)：封包分類的目標即為能夠快速的分類，達到線路的速度。目前網路環境能夠以極高速的方式運作，如一些路由器及侵入阻止裝置甚至能夠達到每秒傳輸約 125000000 個封包(一個 IP 封包至少約是 40bytes)，因此封包分類裝置的處理效率也必須儘可能地達到此一速度。
- 儲存空間(Storage space)：用來儲存規則表的記憶體需求量必須要盡可能的小以減少消耗，越小的記憶體需求便能允許演算法能夠更快速的被執行，但同時也會要求更高超的記憶體使用技術以達成更好的效能，如 on-chip SRAM 提供高速記憶體存取能力，但儲存空間卻相對的減少，
- 更新效率(Update)：每當規則表有所改變時，資料結構就必須要被更新，而更新的頻率取決於不同的應用環境，如 QoS 需要動態地驗證每個類別，因此規則需要快速的更新，相反的如防火牆因為其中定義的規則較少被更動因而可以接受緩慢的更新速度，一般而言一個封包分類演算法必須能夠高速地處理更新動作以應付各種處理環境
- 可規劃性(Scalability)：不同的網路應用服務技術會強調封包表頭資訊中的不同欄位，因此一個好的封包分類演算法必須能夠對多個欄位資訊加以規劃處理，以利於目前不同的網路環境甚至是未來的網路發展所使用，避免過時淘汰。
- 彈性(Flexibility)：規則的規格應該要能夠廣泛且充分地陳述出封包表頭資訊中的不同欄位。為了能夠適用於真實的網路環境，一個好的封包分類演算法必須盡可能的接受各種欄位的規格，如精確的數值，字首與遮罩位元，一段的範圍，以及泛用字元

大部分的封包分類演算都致力於提升搜尋速度的效率，但卻犧牲了儲存空間及更新時間，然而為了提供一更具備更新能力的演算法，則所需考慮的層面便不能僅僅只是侷限在搜尋效益的提升，也須放眼於更新速度的表現以及儲存空間的消耗等方面。

1.3 貢獻

在這篇論文中，我們將提出一個能適用於靜態，動態等多種環境之封包分類

演算法，此方法不僅能滿足於 Section 1.2 的五點需求，並能夠在搜尋的表現中比起其他演算法較佳的成果，同時也能減少空間的浪費。更甚者我們的方法採用動態規劃的方式，更能避免一些極端的分類問題。

1.4 論文架構

本篇論文全文共分為六章，除了本章為簡介外，第二章針對眾多的封包分類演算法做個簡短的介紹，第三章介紹雜湊表的使用以及 Tuple space 的概念及相關的演算法，第四章對 Advanced Regional Diagonal Tuple Space Search 做詳細的描述，第五章為實驗實做及結果分析，比較各演算法的優劣，第六章為結論並說明本研究接下來的一些可行方向。

第二章 與封包分類有關之演算法

2.1 前言

在最近幾年，眾多的學者紛紛提出許多的演算法來解決多維度的封包分類問題，使得處理的層次能勝於二維空間，我們可以藉由一些簡單的特徵來對這些演算法做區分，歸類成如表 2-1 中的四種形式 [3]，接著，我們會簡單的描述並比較其中一些較具代表性的演算法的優劣及特徵。

Category	Algorithms
Data structure-based	Linear Search，Hierarchical tries，Set-pruning tries，Grid of tries，Cross-producting
Geometric based	Are-base quadtree
Heuristic	Hierarchical intelligent cuttings，Hyper-Cuts，Recursive Flow Classification，Tuple space search Diagonal tuple space search
Hardware-based	Ternary Content Addressable Memory，Bit-map intersection，Aggregated Bit Vector

表 2-1：封包分類演算法之範疇。

2.2 演算法之簡介

2.2.1 Linear Search

Linear search 是眾多演算法之中最為簡單的演算法，它擁有最簡單的資料結構，使得更新最為快速容易，只需從規則索引表中依序比對每一條規則即可，此演算法能夠有效地使用記憶體，但卻有可能因為數量龐大的規則而增加了搜尋的時間與次數。

2.2.2 Hierarchical tries

Hierarchical tries (又稱 Multilevel trie 或 backtracking trie)主要是藉由遞迴規則的維度領域建構而成的二元樹，首先選定所指定的第一個的維度領域的 bit(0，1)建立第一層的二元樹，接著再由第一層二元樹中每個葉部節點建立起第二層的二元樹，並以此方式依序處理剩餘的維度空間，因此 Hierarchical tries 中的每一條由根部節點至最底層的葉部節點間的路徑都可以視為能與規則相符合的搜尋路徑，當我們能夠依據一封包的表頭資訊從根部節點搜尋至尾端葉部節點時，我們便可以判斷跟路徑的規則能夠與封包相符合，而不管規則是否符合，都有必要繼續搜尋其他的路徑以找出更佳的規則，因此便有回朔的必要，造成搜尋次數過多的問題。

2.2.3 Set-pruning tries

Set-pruning tries [4]與 Hierarchical tries 近似，但能夠藉由將同一規則配置至不同分支以避免 Hierarchical tries 的遞迴與回碩的問題來減少搜尋時間，如規則 R(011，110)對 0*->110 的分支多配置一條新的支線，但也由於規則的重複配置，使得記憶體空間的需求將隨著規則表規模的增加而變得更為龐大。

2.2.4 Grid of tries

Grid of triesy [5]在 Data structure-based 類別的方法之中對搜尋時間及儲存空間的處理是最為有效率的演算法，藉由配置單一規則至 Hierarchical tries 的單一節點的方式來減少對儲存空間的需求，以及利用 Set-pruning tries 的預先計算和轉換指標來達到與 Set-pruning tries 般少量的搜尋次數，當封包分類的問題能夠壓縮至二維空間來思量時，Grid of tries 會是最好的方法之一，然而 Grid of tries 卻無法

把演算法理念擴展至超過二維的多維空間上。

2.2.5 Cross-producting

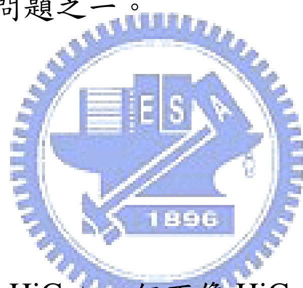
Cross-producting [2]將整個維度以多維空間的方式表示出來，每個規則都擁有屬於自己的位置及範圍大小，而 Cross-producting 再將整個多維空間做若干的切割使得切割之後的每一個區塊僅能存在一個最佳的規則，接著建立一索引表紀錄每一個區塊空間的組合位置以及在該區塊內的最佳符合規則，當搜尋進行時只需分別找出在每一個維度空間切割之後的位置區塊再進行組合以便查詢索引表找出屬於該封包的最佳符合規則，然而當維度增加或規則的數目增加時，可能會造成整個維度空間的切割變得更加複雜、細微，間接造成索引表變的龐大化，以致於需要更多的記憶體空間去儲存索引表。

2.2.6 Are-base quadtree

Are-base quadtree(AQT) [6]也是將整個維度空間以多維空間的方式表示出來，與 Cross-producting 不同的是 Are-base quadtree 將整個維度空間以遞迴的方式不斷的切割成編號為 00，01，10 及 11 的等份的四個子區塊，若以節點表示，第一層節點變為整各維度空間，第二層節點為第一層節點之子樹，以第一層分割來說，各自表示著(0*，0*)，(0*，1*)，(1*，0*)及(1*，1*)等字首組合，因此理論上來說每各歸則都能被分配至四各子區塊之一，即儲存於四個子節點中，但若存在能夠覆蓋該區塊某一維度的規則時，如(001，*)，則將此規則配置於原本區塊而非子區塊中。當收到封包進行搜尋，只需對四元樹進行檢索即可，並對所在節點之規則進行比對，當搜尋至葉部節點時便能找出最佳符合規則並回傳。

2.2.7 Hierarchical intelligent cuttings

Hierarchical intelligent cuttings(HiCuts) [7]藉由規則表的資料結構所衍生的啟發法(heuristics)以對每一個維度空間做切割並建立起一決策樹(decision tree)，決策樹的資料結構則是以規則表為基礎預先處理而成的，在建立決策樹的過程中，每一步僅對單一的維度空間做切割，最後，每個葉部節點都表示著一小群的資料集合，在這裡則是一小群性質類似的規則集合，每當接收到封包時，便會不段地追蹤整個決策樹結構直到找到最理想的葉部節點，再針對該葉部節點所對應的規則及合作線性搜尋，以便找出能與該封包符合的最佳符合規則，這一演算法強調規則表中的每個規則各有個自的特性，也會有相近似的特性，並以此為基礎發展而成，然而如何在龐大的規則表中建立起最理想的決策樹以使得搜尋能更加有效率仍是需要再深入研究的問題之一。



2.2.8 Hyper-Cuts

Hyper-Cuts [8]類似 HiCuts，但不像 HiCuts 在建立決策樹的過程中每一次只對單一維度切割，Hyper-Cuts 會在每一次遞迴的建樹過程中對多個維度空間進行更深的切割動作，因此決策樹中的每一個節點都可以表示成跨越了多維空間的多維體，而非 HiCuts 般僅只是代表著某一平面帶，因此 Hyper-Cuts 對檢索效能有著優越的表現，也能在許多的狀況下對儲存空間做更有效率的運用，但不幸地，在泛用規則的處理上卻無法達成理想的結果。

2.2.9 Recursive Flow Classification

Recursive Flow Classification(RFC) [9]也是最早期的啟發式分類法之一，首先先設置 T 個識別用位元，其中 T=log N (N 是規則的總數)，接著將 S 個封包表頭

資訊位元映射至識別位元，並且 S 必須遠大於 T，而映射的過程可能是以遞迴的方式分成數個階段，每個階段都不斷地消減位元的數量直到 T 個位元為止，而映射的方式可以採用 Cross-producting 逐漸分群，因此 RFC 演算法能夠對封包做極高速的分類，然而卻有著即為可觀的記憶體使用量，也無法有效率的對更做處理。

2.2.10 Ternary Content Addressable Memory

Ternary Content Addressable Memory(TCAM) [10]的每個單位組織有三種值:0，1，X，而 X 如同遮罩位元般可以接受 0 或 1 的值，所以 TCAM 可以支援較精確的數值或字首配對，每一段範圍也能轉換成字首或數值，如範圍大小為 1023 可以表示成六種字首表示法:000001*，00001**，0001***，001****，01*****，1******，因此 TCAM 在接收到封包之後能夠同步的比較每一條規則，此一演算法適用於較小型的規則表，然而對於龐大的規則表，TCAM 就需要大量的空間，造成極大的效能被消耗掉。

1896

2.2.11 Bit-map intersection

Bit-map intersection [11]亦是如同 Cross-producting 般地使用維度空間分割的概念來處理封包分類的問題，然而不同的是它將分割成數個子問題再組合出結果，首先，針對每一層維度空間，以幾何空間的型式針對每一個規則在該層維度空間所映射的位置做切割，因此若有 N 條規則，每一層維度最多將會產生 2N+1 個區間，而規則與區間的關係就僅只有覆蓋與未覆蓋二種，接著再針對每一塊區間，給予相對應且大小為 N 位元的位元向量，其中每一個位元 i 表示相對應於規則表中的第 i 條規則，當某一區間的某一位元 j 設為 1 即表示規則 j 所涵蓋的範圍能夠包含覆蓋該區間，反之當位元 j 設為 0 即表示規則 j 並無覆蓋該區間，當接收到一封包時，便針對每一維度的區間做檢查，找出該封包座落於何區間，並取

得相對應的位元向量表，因此便能得知在不同維度下有哪些規則能與風包廂配合，最後針對由每一層維度取得的位元向量做聯集的運算，並找出擁有最高優先權的規則，便為該封包的最佳符合規則，然而也如同 Cross-producting 般當規則表極為龐大時，會造成整個維度空間與幾何空間複雜化，需要配置更多位元的位元向量給更多的區間，然而沒有規則覆蓋的區間也會大量增加，便會形成即為可觀的記憶體空間的浪費，這也是眾多研究的議題之一。

2.2.12 Aggregated Bit Vector

Aggregated Bit Vector(ABV) [12]是 bit －map intersection 演算法的改良，主要著重於規則表中能與封包相符合的頻率較為稀少的規則，以及位元向量中散佈較為稀疏的位元集合，並採用了位元向量的聚集以及規則的再排列兩種技術來達成，聚集乃是藉由建立一小型的位元向量(稱為 ABV)來紀錄部分位元向量的資訊以減少記憶體的存取，然而卻也會產生不良的影響，當對 ABV 做聯集取得最理想結果時，有可能會發生規則配對錯誤的情形，造成更多的記憶體存取次數，而規則的在排列便能減輕這樣的錯誤情形，雖然 ABV 在記憶體存取次數上勝於 bit－map intersection，但它卻也無法改良 bit －map intersection 的問題，甚至增加了些許的不良情況。

2.3 演算法之比較

表 2-2 列出了上述所提的各個演算法在最糟情況下的搜尋時間複雜度，更新時間複雜度以及儲存空間複雜度 [13]，其中 N 為規則的個數，d 為規則所包含的維度數目，W 是每一維度的位元長度，T 維區間搜尋之時間。然而有些更新時間複雜度未獲取，便以 N/A 表示：

Algorithm	Search time complexity	Update time complexity	Storage complexity
Linear Search	N	log N	N
Hierarchical tries	W^d	d^2W	N^dW
Set-pruning tries	dW	N^d	N^ddW
Grid of tries	2W	NW	NW
Cross-producting	dW	NW	N^d
Area-based QuadTree	αW	$\alpha\sqrt{N}$	N
Hierarchical Cuttings	D	N/A	N^d
Recursive Flow Classification	D	N/A	N^d
Bit-map intersection	d(T+N/W)	N/A	dN^2

表 2-2：封包分類演算法之複雜度比較。

第三章 相關研究

在這一章節中，我們將介紹 Tuple space 之定義以及作為我們所提出演算法之基礎的二個使用 Tuple space 為概念的搜尋演算法。

3.1 Tuple Space

3.1.1 Tuple Space 之定義

每個規則都有定義各自的 source specified bits，destination specified bits 以及所涵蓋的範圍，而 source specified bit 個數及 destination specified bit 個數卻只分布於 0~32 等 33 種長度，藉由 source specified bit 個數及 destination specified bit 個數的配對可以產生共 33X33 種組合。這使得對規則的搜尋可以先透過簡單的分群而減少所需搜尋規則之數量。因為在同一群組中的規則都擁有相同的 specified bit 個數，我們便可以利用 specified bit 個數之值作為 hash key 值來檢索所欲搜尋的 specified bit 是否存在於規則資料庫中，因此透過 hash table 的建立以及 hash key 的檢索，使得群組間的搜尋僅需 O(w)，w 為 specified bit 之長度。

Tuple space search [14]最早是由 Srinivansan et al.所提出的，利用 Prefix specification 來處理多維度封包分類的問題。由於作為 hash key 值的是規則的 source specified bit 個數及 destination specified bit 個數，因此整個 Hash table 可以視為二維空間，如表 3-1 的例子便是依據 rule 中每一層維度的 specified bit 的個數對應至相對應的 tuple 欄位，R2 的 specified bite 個數分別為 1 與 2，便對應至 tuple(1，2)這一欄位中；R3 的 specified bite 個數分別為 1 與 1，便對應至 tuple(1，1)這一欄位中。表 3-2 則是整個 tuple space 的資訊，表示出全部的 tuple 以及每個

tuple 欄位所包含的規則的編號，接著當封包需要進行規則的比對時便是使用對雜湊表進行索引的方式來找出特定的 tuple 並從中得知所配置的每一個規則，繼而進行規則的比對。

由於 tuple space 使用了雜湊表索引的概念，因此封包分類的問題便由對每一個規則作比對的問題壓縮成對每一個 tuple 做比對的問題，再比對其中的每一個規則，相較於其他使用規則來做分類的演算法，採用 tuple space 的演算法更能擁有較快的平均搜尋時間以及較佳的更新時間。現今引用 tuple space 概念的演算法，多是採用二維空間作為處理的途徑(圖 3-1)，亦即使用(Source specified bits，Destination specified bits)當作 Hash key 來檢索每一個 tuple，然而 tuple 與 tuple 間除了二個欄位的 specified bit 個數之外並無任何直接及間接的關聯性可供快速的檢索，因此如何正確、有效的檢索出理想的 tuple 以找出最恰當的規則，便是眾多使用 tuple space 的演算法，以及本篇論文的議題所在。

Rule	Specification	Tuple
R1	(00*，00*)	(2，2)
R2	(0**，01*)	(1，2)
R3	(1**，0**)	(1，1)
R4	(00*，0**)	(2，1)
R5	(0**，1**)	(1，1)
R6	(***，1**)	(0，1)

表 3-1：規則對應至 Tuple 之範例。

Tuple	Hash Table Entries
(0，1)	{ R6 }
(1，1)	{ R3， R5}
(1，2)	{ R2 }
(2，1)	{ R4 }
(2，2)	{ R1 }

表 3-2：表 3-1 所建立之 Tuple space。

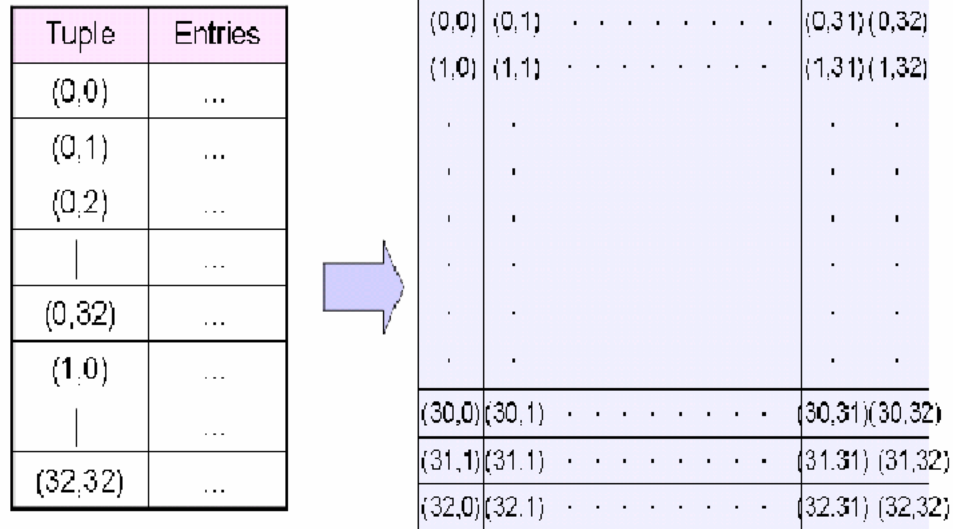


圖 3-1：Tuple space 可視為一二維空間。

3.1.2 Tuple Relationship

任一 tuple 與 tuple 之間存在著三者關係：ShorterTuple，LongerTuple，以及 IncompatibleTuple，假設存在二個 tuple： Ta 及 Tb，則其關係的定義如下所敘：

- 若∀i，1≤i≤d，Ta[i]≤Tb[i]，而且至少存在一個 i 使得 Ta[i]≠Tb[i]，則定義 Ta 屬於 Tb 的 ShorterTuple。
- 若∀i，1≤i≤d，Ta[i]≥Tb[i]，而且至少存在一個 i 使得 Ta[i]≠Tb[i]，則定義 Ta 屬於 Tb 的 LongerTuple。
- 若 Ta 既非 Tb 的 ShorterTuple，也不是 Tb 的 LongerTuple，則定義 Ta 屬於 Tb 的 IncompatibleTuple。

由上述的定義，我們可以知道 tuple(1，2)是 tuple(3，4)的 ShorterTuple，因 1≤3， 2≤4；tuple(5，6)是 tuple(3，4)的 LongerTuple， 因 5≥3， 6≥4； tuple(2，5)是 tuple(3，4)的 IncompatibleTuple，因 2≤3，5≥4。圖 3-2 即為二維空間中 tuple(x，y)的範圍分布。

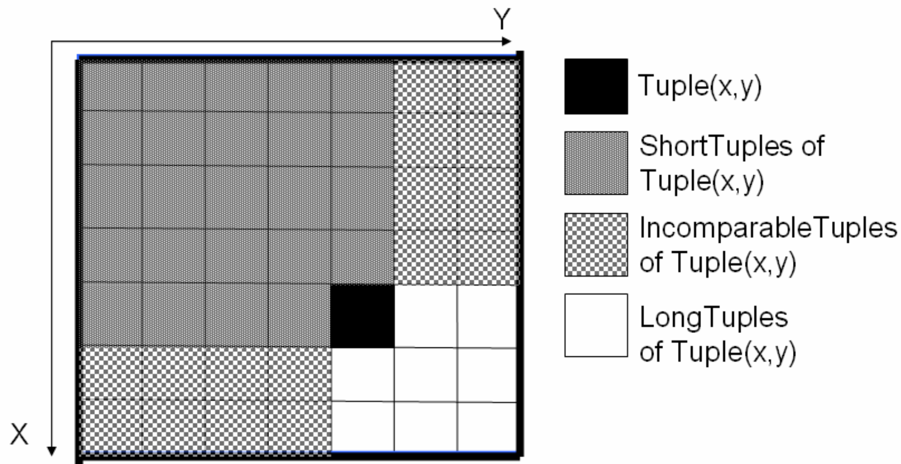


圖 3-2：tuple(x，y)與其他 tuple 之關係之二維空間表示圖。

當一個封包能擁有二個以上的規則滿足封包的配對，則根據每一個規則所配對的 tuple 之間的 ShorterTuple，LongerTuple，以及 IncompatibleTuple 三種關係，可以發展出規則與規則之間的三種關係： specific， general 以及 conflicting，假設當一封包 P 能滿足二個規則：Ra 及 Rb 分別對應 Ta 及 Tb，則 Ra 與 Rb 間的關係定義如下所敘：

- 若 Ta 是 Tb 的 ShorterTuple，則我們稱 Ra 較 Rb 更為 general，亦即當 Rb 能滿足封包的配對時，Ra 亦能滿足封包的配對，反之當 Rb 無法滿足時，Ra 卻有可能滿足配對。
- 若 Ta 是 Tb 的 longerTuple，則我們稱 Ra 較 Rb 更為 specific，亦即當 Rb 能滿足封包的配對時，Ra 未必能滿足封包的配對，反之當 Rb 無法滿足時，Ra 必定也無法滿足配對。
- 若 Ta 與 Tb 互為 IncompatibleTuple，則我們稱 Ra 與 Rb 互為 conflicting，當 Rb 能滿足封包的配對時，Ra 未必能滿足封包的配對，反之當 Rb 無法滿足時，Ra 卻有滿足配對的可能性。

3.1.3 Marker 與 Precomputation

Srinivansan et al.提出了 Rectangle search 演算法並提出了 precomputation 以及 marker 的概念來解決 tuple space 中 tuple 與 tuple 間毫無關聯的問題 [14]：

Marker 主要目的在於能為某些 tuple 的集合建立起關聯性，一般來說規則能夠在它的 ShortTuple 生成自己的 marker，而 marker 除了 specified bit 需與所對應

的 tuple 相同之外，其他資訊都與原來的規則相同，而隨著演算法的不同，marker 的生成也有著不同的方式。在 Rectangle search 中，對每一個規則而言，都必須不斷向左設置一個自己的 marker，如圖 3-4 所示，每一個規則都會向左邊的 tuple 生成一個 marker，R1 在(3，1)的 marker 便為(111*，1*)，在(3，0)的 marker 便為(111*，*)，R2 在(2，0)的 marker 便為(11*，*)，因此我們可以將每個規則的 marker 視為較該規則更為 generic，當封包對某個 marker 比對失敗時，同時也意味著生成該 marker 的規則亦無法比對成功，因此可以延伸出當對某個 tuple 搜尋失敗，該 tuple 的右邊區域皆不可能有會符合的規則存在，因不可能發生不滿足 marker 卻滿足規則的情形發生，另外，根據每一個演算法的需要，每一個規則並不限定只向左邊的 tuple 生成 marker，而是會有向上方，向左上方等不同方向的可能產生。

Precomputation 主要的目的在於能為每一個規則找出包含它自身以及該規則所對應 tuple 的所有 ShorterTuple 中擁有最高優先權(最佳選擇)的規則，如圖 3-3 所示，灰色區域便是 R1 的 precomputation 範圍(自己所對應的 tuple 以及 ShorterTuple)，在這範圍之中搜尋出與 R1 相同或是比 R1 更為 general 的規則，結果找出 R1 與 R2 這二個規則，若 R1 的優先權高於 R2，則我們便紀錄 R1 為 R1 這個規則所能對應到的最佳選擇，反之則紀錄 R2 為 R1 這個規則所能對應到的最佳選擇。

由於封包的搜尋動作會受到 marker 所引導，然而當封包能與 marker 配對卻不一定表示能與規則配對，因此當發生此一情形時便會造成整各搜尋演算法因為受到錯誤的規則所引導，導致無法找出最理想的規則，所以此一動作的優點便是當我們能找到一 marker 與封包相配對時，我們能自然的得到包含該 tuple 以內的所有 ShorterTuple(如圖 3-3 所示為該 tuple 的左上方塊)的最佳規則，無須再去顧慮發生上述錯誤情況時的處理。

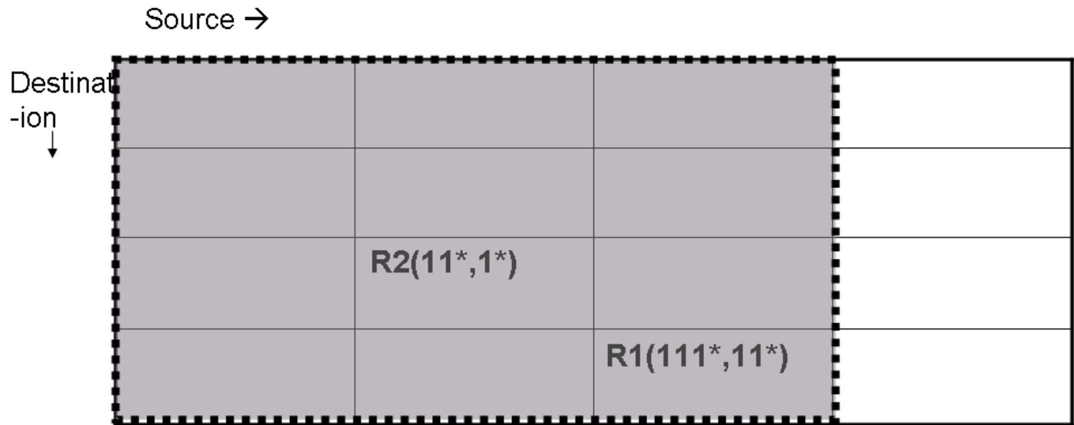


圖 3-3：Precomputation 所包含之範圍。

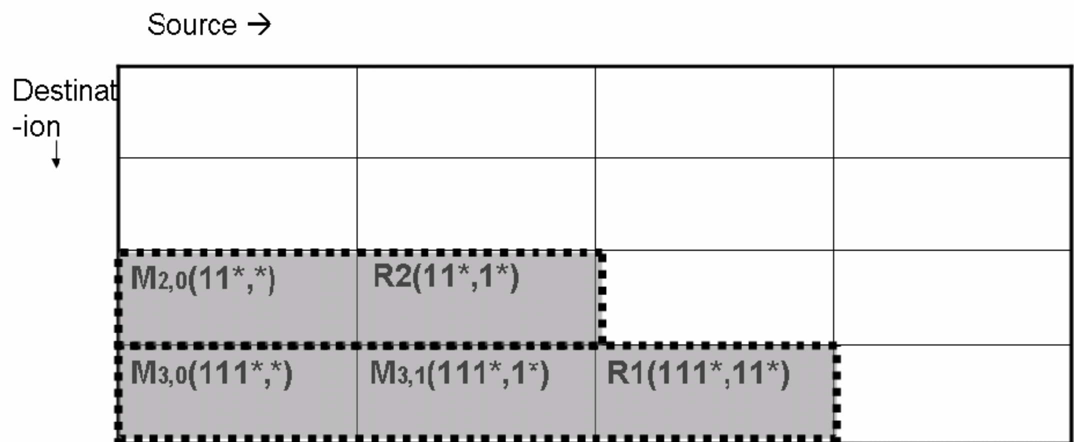


圖 3-4：Marker 的生成。

3.2 Rectangle Search

利用 precomputation 以及 marker 的觀念，便發展出 rectangle search 演算法 [15]：當被給予一封包 P，便從整個 tuple space 的最左下方((32，0)的位置)開始進行探測，若能找到至少一個規則或是 marker 能與 P 相配對，則表示該 tuple 的所有 LongerTuple 有可能存在更為 specific 的規則，因此繼續向右一行的 tuple 進行搜尋，否之無法找到任一個規則或是 marker 與 P 相配對，表示不存在更 specific 的規則可與之配對，此時向上一列的 tuple 進行搜尋。

如圖 3-5 所示，R1 至 R5 生成各自的 marker，當根據封包 P 的表頭資訊內

容(0000，0000)開始進行 rectangle search 時，首先先由最左下角的 tuple(4，0)開始，因能夠找到 marker 與 P 符合，所以向右一列的 tuple(4，1)繼續進行搜尋，並紀錄由 marker 得知的此時最佳符合規則為 R6，同樣的在 tuple(4，1)能找到 R2 配對，所以繼續向右一列的 tuple(4，2)進行搜尋，而由於 R2 的優先等級較 R6 高，因此符合封包 P 的最佳符合便由 R2 取代，而在 tuple(4，2)因為無法找到能夠與 P 配對的規則或 marker，所以往上一行的 tuple(3.2)繼續進行搜尋，如同圖 3-5 箭頭的搜尋過程，最後會結束於 tuple(0，3)，封包 P 所能配對到的最佳規則為 R1。

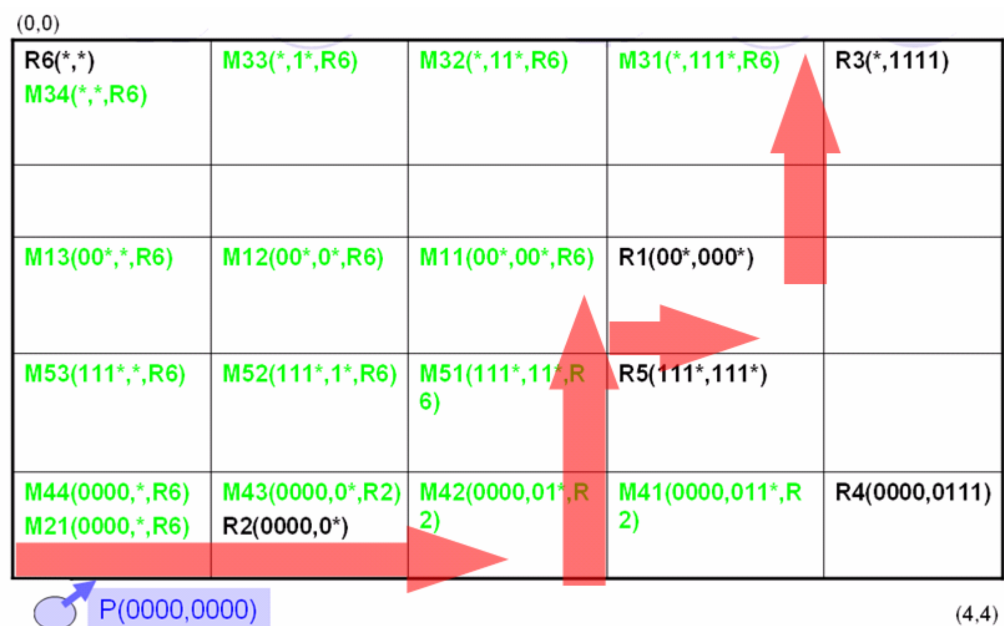


圖 3-5：Rectangle search 之範例。

然而我們也可以由圖 3-6 看出 rectangle search 的問題，當一封包在 tuple space 之中對 tuple 的比對路徑如圖所示時，此演算法需要對(2w-1)個 tuple 做檢測，亦及對雜湊表做了(2w-1)次的存取，即使在最佳狀況(對每個 tuple 都能找到配對的規則或 marker 的情況或對每個 tuple 都無法找到可配對規則或 marker 的情況)下也需要高達 w 次的存取，同時每一個規則都會像左生成 marker，這結果造成 marker 的數量遠大於規則的數量的幾十倍以上，假設 tuple space 有 n 各規則，則最糟情形為每個規則都分別向左生成(w-1)個 marker，共須 n+nx(w-1)=nw 的儲存空間需求。因此 rectangle search 在搜尋時間 O(w)及記憶體空間的表現 O(nw)上都不甚想

理。

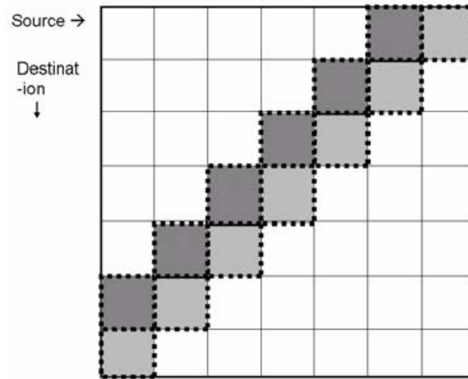


圖 3-6：Rectangle search 之最糟情形。

3.3 Diagonal Tuple Space Search

為了解決 Rectangle search 對 tuple 搜尋次數過多的問題，Mikko Alutoin 和 Pertti Raatikainen 共同提出了 Diagonal tuple space 的概念 [16]，在這演算法中，marker 的生成方向如圖 3-7 所示，對任一個 tuple(i，j)， 0≤i，j≤32，可以有以下的三種生成 marker 的情形：

- 當 i<j，則 tuple(i，j)的每一個規則都要向上增加自己的 marker 直至 tuple(j，j)為止。
- 當 j<i，則 tuple(i，j)的每一個規則都要向左增加自己的 marker 直至 tuple(i，i)為止。
- 當 i=j，則 tuple(i，j)的每一個規則都要向左上方增加自己的 marker 直至 tuple(0，0)為止。

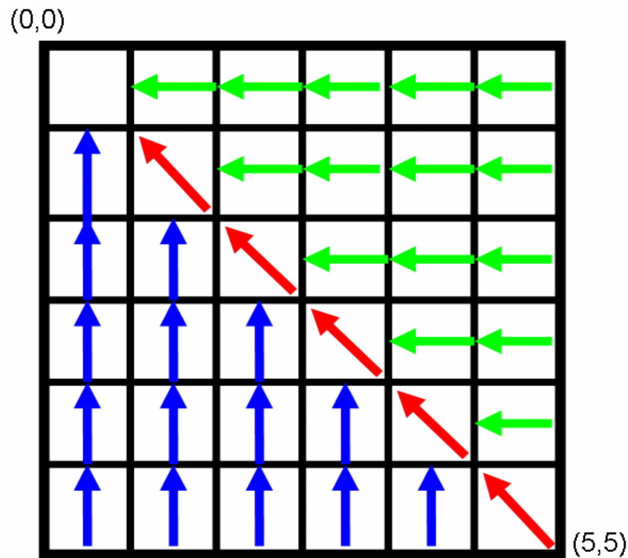


圖 3-7：Diagonal tuple space search 之 marker 生成方向。

因此我們能夠觀察到對每一列或是每一行而言，他們之間的關聯性只有靠位於對角線的 tuple 所生成的 marker 來建立，而在 diagonal tuple search 中使用了”映射”作為搜尋方式，當找到位於某對角線上的 tuple(d，d)時，需要分別對 tuple[d，y≥d]以及 tuple[d，x≥d]再多做二次的二元搜尋，而決定 tuple(d，d)的位置又再多需一次 binary search，因此整個 diagonal tuple search 更需三次的 binary search，使得在搜尋上的時間複雜度僅須 O(log w)，然而”映射”會對一些衝突的規則或是 IncompatibleTuple 造成誤判的錯誤，因此為了使搜尋能夠更為完整與正確，在 precomputation 需具備增加 mirrir rule 的動作，當規則 R 在其 IncompatibleTuple 的對角線 tuple 找到一 marker M，使得規則 R 會與 marker M 產生衝突的問題，則必須在規則 R 與 marker M 的連結位置產生一 mirror rule，若規則 R(111***，11111*)與 marker M(1111**，1111**)，則產生的 mirror rule 為 (1111**，11111*)。

在搜尋上需要三次二元搜尋，首先根據被給予的封包 P 針對 tuple space 對角線上的 tuple 做第一次二元搜尋以找出 tuple(d，d)，以能夠找到最大的 d 值為優先，若是 tuple(d，d)上的規則與 P 相符合，可以得之封包 P 所能找到的最佳符合為該規則，回傳該規則而後結束搜尋；若是 tuple(d，d)上的 marker 與 P 相符合時，則

分別對 tuple[d，y≥d]以及 tuple[d，x≥d]坐二元搜尋，同樣的以找到最大值為優先，而後再選擇出擁有最大優先權的規則或是 marker 中的最佳符合規則，即可完成封包 P 的搜尋動作，如圖 3-8 所示，R1 向左生成 marker 直至 tuple(1，1)為止，R2 向上生成 marker 直至 tuple(1，1)為止，R3 向左生成 marker 直至 tuple(0，0)為止，R4 及 R5 向左上生成 marker 直至 tuple(0，0)為止，當給予一封包 P 之後，對對角線的 tuple 做第一次的二元搜尋，最初是 tuple(2，2)檢測失敗後再對 tuple(1，1)進行搜尋，爾後發現有 marker 與 P 相符合之後，接著針對 tuple(1，1)右邊以及下面的 tuple 各自進行二元搜尋，分別找到 R1 及 R2 兩個可以符合的規則，最後依據優先權的比較，可以得知封包 P 能夠分類至 R1。

(0,0) P(0000,0000)

R6(*,*) M14(*,*,R6) M24(*,*,R6) M32(*,*,R6) M44(*,*,R6) M53(*,*,R6)	M31(*,0*,R6)	R3(*,00*)		
	M13(0*,0*,R6) M23(0*,0*,R6) M43(0*,0*,R6) M52(1*,1*,R6)	M12(0*,00*,R3)	M11(0*,000*,R3)	R1(0*,0000)
	M22(00*,0*,R6)	M42(00*,01*,R6) M51(11*,11*,R6)		
	M21(000*,0*,R6)		R5(111*,111*) M41(000*,011*,R6)	
	R2(0000,0*)			R4(0000,0111)

(4,4)

圖 3-8：Diagonal tuple search 之範例。

由於至多使用了三次的二元搜尋來完成整各封包分類，因此在搜尋時間上能壓縮至 $O(\log w)$，若有n各規則存在於tuple space，當一對角線tuple(x，x)以及IncomparableTuple的某一規則產生衝突時，可能會生成 2^{d-x} 個marker以解決衝突問題，因此最糟情形需要 $O(n2^{w}w)$ 的儲存空間需求，然而在此情況下產生衝突的列或行因為會有marker重複的情形，因此至多只需生成 $\log w$ 個marker即可，使得

diagonal tuple search需要$O(n2^{w} \log w)$ 的儲存空間，比起rectangle search更為不理想。

第四章 Advanced Regional Diagonal Tuple Space Search

4.1 動機

一般來說，真實網路環境下的規則在 tuple space 的分布情形大致上如圖 4-1 所示，整個 tuple space 以二維空間所表示，顏色越深之區域表示所對應的規則數量越多，反之顏色越淺或接近白色表示該格所對應的規則稀少或無。其中以 tuple[16~32，16~32]這一區間分布的最為密集，亦或是在 wildcard 區域([0，0~32]或[0~32，0])有較為顯注的密度集中於此，考慮這樣的情況下，使用 Diagonal tuple space search 便有明顯的空間的浪費以及搜尋次數的增加。

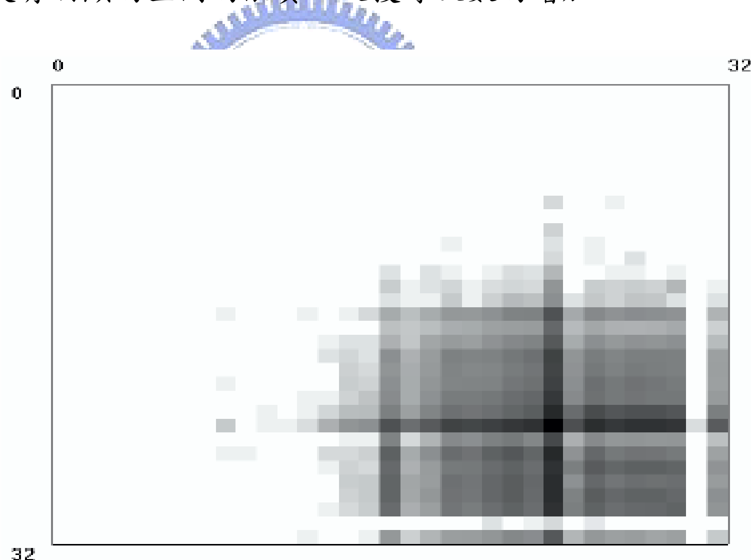


圖 4-1：真實網路環境的規則配置情形 [17]。

- 空間的浪費：

(1)、在 Diagonal tuple space 定義中，每一條規則皆會朝著 tuple(0， 0)的方向增加 marker，可能造成 tuple(0，0)~tuple(15，15)這些對角線位置的 tuple 由於這些規則的存在而生成大量的 marker，但這些對角線 tuple 的右邊及下面卻可能沒有或僅只有極為稀少量的規則被配置，因此對這些區域的 tuple 來說，它們可能只需要配置及少量的空間比例來儲存這些來自右方及下方的規則所生成的

marker，卻需要極大量的空間比例來儲存來自於右下方的規則所生成的 marker。

(2)、同時對這些極少量的規則來說，它們仍需要朝向對角線位置生成 marker，如存在於 tuple(32，5)的規則得對 tuple(5，5)至 tuple(31，5)等 27 個 tuple 分別生成該規則的 marker，在空間的使用上是極為可觀的。

- 存取次數的增加：

由於空間的浪費的第二點所述，若當第一次二元搜尋所能找到最深的 tuple 是介於 tuple(0，0)~tuple(15，15)等規則分佈較為稀少之區域時，卻有可能因為該 tuple 的右邊或下方僅存在一條規則而多需要二次的二元搜尋，逐次的藉由 marker 的指引才能找到該規則所對應的 tuple，但所搜尋的區域可能並不存在其他的規則或 marker，過多的 marker 的指引反而造成不必要的搜尋。如上例中欲搜尋 tuple(32，5)的規則，我們必須由 tuple(5，5)出發，分別對 tuple(i，5)，5≦i≦32 以及 tuple(5，j)，5≦j≦32 做二元搜尋，其中必須搜尋(5，18)、(5，25)、(5，29)、(5，31)、(5，32)以及(18，5)、(11，5)、(7，5)、(5，5)，但可能這些 tuple 只存在該規則所對應的 marker 甚至無任何規則與 marker 存在。

因此為了解決 Diagonal tuple space search 不符合真實網路環境的問題，我們便提出了幾點想法：

- 若對應於 tuple[16~32，16~32]的規則僅需生成 marker 至一定範圍的 tuple，而無須朝著 tuple(0，0)的方向逐漸生成，則 tuple(0，0)~tuple(15，15)等左上段對角線 tuple 的 marker 必能減少許多，同時也能使得搜尋所需的存取次數下降。
- Diagonal tuple space search 需要至少三次的二元搜尋是因為對應於對角線 tuple 上的 marker 無法告訴我們來自於哪個方向，因此第二次以及第三次的二元搜尋得分別往右以及往下做搜尋以進行確認。若我們能建立起方向性，便能依據方向性朝著正確的方向進行一次的二元搜尋即可。
- 若對某些 marker 建立起類似指標之性質，則透過這些 marker 的指引便能直接對生成該 marker 的規則所對應的 tuple 做搜尋，如此在規則散佈密度稀散的區域，便無須作多次累贅的搜尋。

為了達成以上三點的想法，我們嘗試改變規則及 marker 的資料結構，

並進而提出了以下的方法來加以改良，使得 marker 的生成數量得以減少，並在搜尋時增添一變數輔助，已達成第一次二元搜尋之後僅需往單一方向做深入搜尋的想法，必藉此減少存取的次數。

在 Section 4.2 中，我們將介紹規則與 marker 的資料結構；在 Section 4.3 中，我們將加以描述整個 tuple space 的 marker 生成流程；在 Section 4.4 中針對一些渴能發生的問題做進一步處理；在 section 4.5 中，我們將描述整個初始化動作之流程以及搜尋流程，並以簡單的例子做說明； Section 4.6，我們將證明整個搜尋演算法的可行性；而最後的 Section 4.7 中將以簡單的例子說明整個搜尋流程。

4.2 資料結構的描述

我們對規則及 marker 額外增設了二個欄位，用來記錄該 marker 是由哪一個 tuple 生成而來，我們稱之為來源資訊(Source information)，如 R(11111，1111*)在 tuple(2，3)生成了 marker(11***，111**)，則該 marker 的來源資訊便填入(5，4)。由於在 marker 生成階段時可能因為與某些規則相重複而不被生成，便改為對與該 marker 重複的規則填入來源資訊，另外由於 Precomputation 在我們的演算法中仍是有必要的，因此一個 marker 的資料結構便為：

```
Marker(Source specified bits，Destination specified bits，
       Best matching rule，Source information)
```

而一個規則的資料結構便為：

```
Rule(Source specified bits，Destination specified bits，Source information)
```

增設此來源資訊欄位的目的有二：

- 為二元搜尋增加方向性，並限定二元搜尋範圍。對位於 tuple(5，5)的 marker 來說，若來源資訊為(5，10)，我們便可得知該 marker 來自於右方的 tuple 的某一規則所生成的。一般來說位於對角線的規則或 marker 的來源資訊可分為三種資料型態：

NULL，來源資訊填入(Null，Null)。

單方向，當該 marker 僅是來自於右方或下方其中之一時，此時紀錄來源的 tuple 位址，來源資訊填入(Source specified bits length，Destination specified bits length)。

雙方向，當該 marker 同時為來自兩方的規則所生成，如 R1(11***，1111*)與 R2(1111*，11***)在 tuple(2，2)所生的 marker 皆是(11***，11***)，此時記錄 marker 所在的 tuple 與一空值，即(2，Null)。

- 具備指標之功能。由於來源資訊記錄著生成該 marker 的規則所對應的 tuple 位址，因此我們可以透過來源資訊的紀錄直接對相對應的 tuple 做搜尋即可。

4.3 演算法的描述

4.3.1 分區

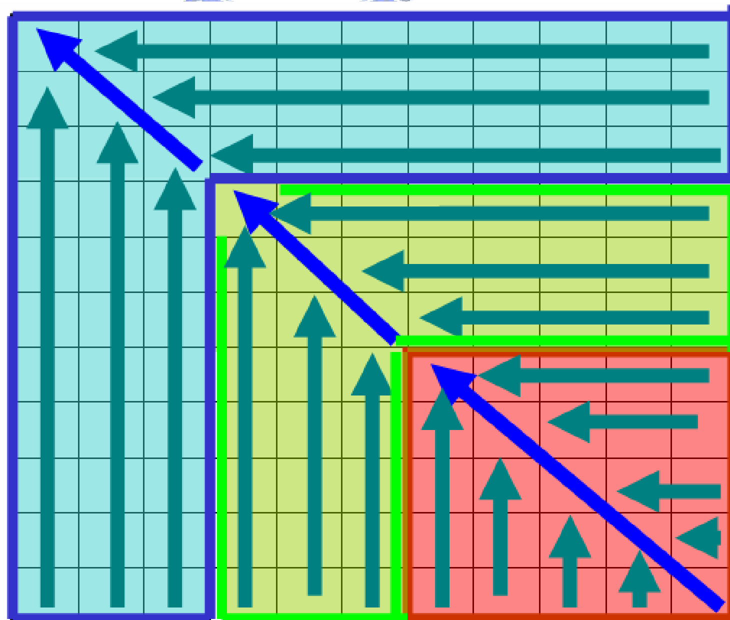


圖 4-2：Advanced Diagonal Tuple Space 的區域配置圖示。

如 Section 4.1 小節所描述，一般的規則在 tuple space 的分布情形大致可分為

三大類： tuple[16~32，16~32]等較密集區域，wildcard 區域(tuple[0，0~32]或 tuple[0~32，0])以及其他較為疏散區域，而為了避免位於對角線上的 tuple 對又下方的規則生成過多的 marker，因此我們可以將整個 tuple space 分為三各區域，如圖 4-2 所示，區域與區域之間的間隔可以為 0~7，8~15 及 16~32 等三大塊，而在每各區域內的所有的規則都能再對該區域進行搜尋時有機會被檢索，而區域以 0~7，8~15，16~32 之分區法正如同二元搜尋之順序般，當逐漸由最下層區域往上層區域進行搜尋時，正如同 Diagonal search 之第一次二元搜尋失敗時地向左上深入搜尋。

每一層區域的對角線 tuple 必須能夠配置該區域內規則的 marker，使能夠藉由該 marker 搜尋至該規則，因此每一個規則都必須在相對應的對角線位置之 tuple 留下 marker，如圖 4-3 所示，對任一個 tuple(i，j)， up_bound≤i，j≤down_bound，up_bound 為該區域的上界， down_bound 為該區塊的下界，在這三個區域內的規則可以有以下的三種生成 marker 的情形：

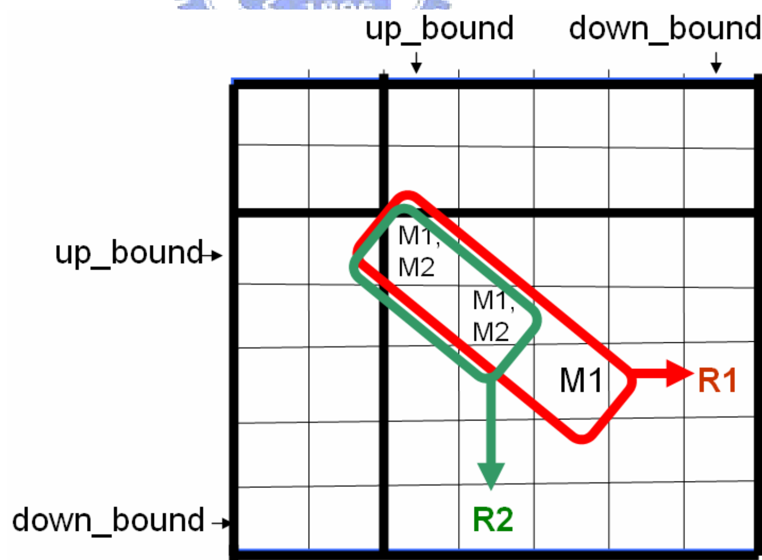


圖 4-3：區域內的 marker 配置圖示。

Case 1：如圖 4-3 的規則 R2 所示，當 j<i，則 tuple(i，j)的每一個規則都要向上在 tuple(j，j)增加自己的 marker，接著再向左上方增加自己的 marker 直至 tuple(up_bound，up_bound)為止。而後，我們會對位於 tuple(j，j)所生成的 marker

增設來源資訊已建立起方向性，若該 marker 因與其他的規則或 marker 重複而不被生成，則對被重複的規則或 marker 填入來源資訊。

Case 2：如圖 4-3 的規則 R1 所示，當 i<j，則 tuple(i，j)的每一個規則都要向左於 tuple(i，i)增加自己的 marker，接著再向左上方增加自己的 marker 直至 tuple(up_bound，up_bound)為止。而後，我們會對位於 tuple(j，j)所生成的 marker 增設來源資訊已建立起方向性。若該 marker 因與其他的規則或 marker 重複而不被生成，則對被重複的規則或 marker 填入來源資訊。

Case 3：當 i=j，則 tuple(i，j)的每一個規則都要向左上方增加自己的 marker 直至 tuple(up_bound，up_bound)。

藉由如此的 marker 生成方式使得每一個對角線上的 tuple 都有來自其右方以及下方的規則所生成的 marker，每一個區域內的規則亦不會在其他的區域生成任何的 marker，因此區域與區域之間可以說是互相獨立的，當最右下方的區域在最左上角的 tuple 生成大量的 marker 時，並不會影響到另外二個區域內的 marker 生成，這二個 Diagonal tuple space 也因此不再有來自右下方區域的規則所生成的 marker 的空間負擔。

另一方面，由於每一個規則最終都會生成 marker 於 tuple(up_bound，up_bound)，亦即該規則所屬區域的最左上角的 tuple，藉由透過每一個區域的最左上方的 tuple(tuple(up_bound， up_bound))的檢查，若檢查失敗則表示該區域並無任何可能會與封包相配合的規則存在，換至下一各區域做相同的檢查；當檢查成功，我們便能得知該區域可能存在著能與封包相配合的規則。因此對每一塊正方形區域搜尋的順序為最右下方區域開始，當最右下方區域的最左上方 tuple 無任何規則或 marker 能與封包配對時，便跳往上一層的區域的最左上角 tuple 做相同的檢察，直至 tuple(0，0)所屬的最上一層區域為止；若區域的最左上方 tuple 存在著能與封包配對的規則或 marker 時，便能對該區域進行內部的搜尋，若搜尋

1896

至最上層，由於 tuple(0，0)的規則必定能與封包符合，可直接對內部進行第一次二元搜尋無須再檢察 tuple(0，0)是否符合。

第一次的二元搜尋範圍便為區域內的對角線，tuple(up_bound，up_bound)至 tuple(down_bound， down_bound)，當找到最深的 tuple 時，又可分為二種情形，若與封包相符合的規則或是 marker 所填入的來源資訊為單一方向之型式時，我們便可以透過來源資訊得知可能在哪各方向(右方，下方)存在著某一條規則能與封包相配合。

4.3.2 Link-list 與 marker 生成

對其他對角線 tuple 之外的規則生成 marker 的方式，我們亦可以對對角線向右延伸的每一列以及向下延伸的每一行做更進一步的分區，如圖 4-4 所示，我們將

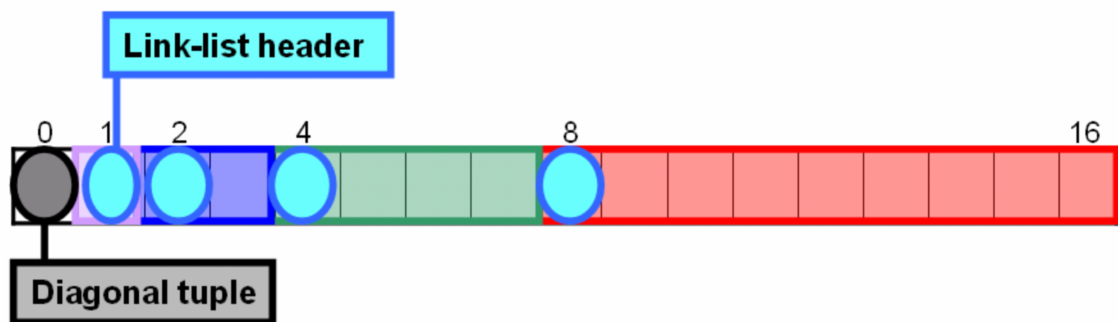


圖 4-4：Linklist 區間配置方式與 header tuple 之設置。

一列(或一行)分成若干小區塊，每一個小區塊的起始位置亦可以為二元搜尋失敗時所需經過的 tuple，在圖 4-5 中便為編號 8、4、2、1 為各個區塊之起始位置，這些 tuple 可以視為一個 link-list 之起頭，接著便可以由下列步驟來說明位於非對角線 tuple 上的每各規則如何生成 marker：

Step 1：若規則R對應於tuple(x，y)中，若x≠y，則規則R對應於非對角線之

tuple。接著根據該規則R所對應的tuple位置，找出相對應於該列或該行的link-list header tuple位置，若y>x時，則所對應的link-list header位置為tuple(x，x + $2^{|\log 2(y-x)|}$)；若x>y時，所對應的區域邊界位置為tuple(y + $2^{|\log 2(x-y)|}$，y)。

Step 2：規則 R 在生成 marker 於所對應的 link-list header tuple 中，並填入來源資訊為(x，y)。若該 marker 因與其他規則或 marker 發生重複而取消時，我們又能夠分為三種情形：

- Case 1：若該規則或 marker 尚未填入來源資訊，則填入(x，y)。
- Case 2：若該規則或 marker 已填入來源資訊(x`，y`)，則比較(x`，y`)與(x，y)，當 x`=x 且 y`<y，或是 x`<x 且 y`=y 時，我們可以得知規則 R 所對應的 tuple 位置(x，y)與 link-list header tuple 間的距離較(x`，y`)與 link-list header tuple 間的距離遠，我們無須更動該來源資訊，接著規則 R 便在 tuple(x`，y`)位置生成 marker，marker 的來源訊息便為規則 R 的(x，y)，若又存在重複的規則或 marker 使得規則 R 的 marker 不被生成，則採用相同的步驟遞回進行，比較該規則或 marker 的來源資訊與規則 R 的位置，並做相同的處理。
- Case 3：若該規則或 marker 已填入來源資訊(x`，y`)，則比較(x`，y`)與(x，y)，當 x`=x 且 y`>y，或是 x`>x 且 y`=y 時，我們可以得知規則 R 所對應的 tuple 位置(x，y)與 link-list header tuple 間的距離較(x`，y`)與 link-list header tuple 間的距離近，則 link-list header tuple 之 marker 或規則的來源資訊便取代為(x，y)，而原來位於 tuple(x`，y`)的規則 R`便生成 marker 至 tuple(x，y)位置上，來源資訊填入(x`，y`)，同樣的，若存在重複的規則或 marker 存在使得規則 R`的 marker 不被生成，則採用相同的步驟遞回進行，比較該規則或 marker 的來源資訊與規則 R`的位置，並做相同的處理。

Step 3：當所有非對角線上之 tuple 的規則皆生成 marker 於相對應的 link-list header tuple 之後，我們便可以開始對對角線 tuple 上的每一個規則或是 marker 填入相對應的來源資訊。對每一個對角線上的規則或 marker，都須向右以及向下檢查每一個 link-list header tuple，若該 marker 或規則是由右方或下方的的某一規則所生成而來，因該規則亦會對相對應的 link-list header tuple 生成 marker，因此原則上這些對角線的規則或 marker 至少能找到一個 link-list header tuple 的規則或 marker 使得對角線之規則或 marker 能包含住 link-list header tuple 的規則或

marker。若能找到能包含的 link-list header tuple 之規則或 marker 時，則此一對角線之規則或 marker 的來源資訊則填入該 link-list tuple header 之位置。若存在二各以上不同位置的 link-list header tuple 滿足此一情形時，澤我們需判斷這些 link-list header tuple 是否都在同一端(都在右端或都在下方)：若是，選擇距離對角線位置最遠之 link-list header tuple 為來源資訊位置；若是不同端時，此時便會產生雙方向衝突，我們暫時將來源資訊填為(d，-)，d 為對角線之位置。

圖 4-5 為非對角線 tuple 之規則生成 marker 的簡單範例，其中(2，3)與(2，5)皆為 link-list header tuple，圖 4-5.a 加入 R1，因已在所對應的 link-list header tuple，故無須生成 marker，圖 4-5.b 加入 R2，在對應的 link-list header tuple(2，5)內生成 marker 並填入來源資訊為(2,7)，圖 4-5.c 加入 R3，由於 R3 所對應的 link-list header tuple(2，5)已存在 marker 且該 marker 的來源資訊(2，7) 較 R3 所對應的 tuple(2，6)距離對角線更遠，因此將該 marker 的來源資訊更新為(2，6)，並找出 tuple(2，7)的規則或 marker 再生成 marker 於 tuple(2，6)上，填入來源資訊(2，7)。

R4(*, *,)							
	M(1*,1*, R4)						
		M(11*,11*, R4,(2,5))			R1(11*, 11111*)		
			Header tuple		Header tuple		

圖 4-5.a：簡單的非對角線 tuple 之 marker 生成流程範例，加入 R1。

R4(*, *,)							
	M(1*,1*, R4)						
		M(11*,11*, R4,(2,5))			R1(11*,11111*) M(11*,10101*, R4,(2,7))		R2(11*, 1010101)

Header tuple

圖 4-5.b：簡單的非對角線 tuple 之 marker 生成流程範例，加入 R2。

R4(*, *,)							
	M(1*,1*, R4)						
		M(11*,11*, R4,(2,3))			R1(11*,11111*) M(11*,10101*, R4,(2,6))	R3(11*,101011*) M(11*,101010*,R4, (2,6))	R2(11*, 1010101)

Header tuple

圖 4-5.c：簡單的非對角線 tuple 之 marker 生成流程範例，加入 R3。

透過 Step 2 的處理之後，我們可以知道每一個位於 link-list header tuple 之規則或 marker 都是一個 link-list 起點，當該 tuple 存在越多的規則或 marker 即表示存在越多的 link-list，我們可以以一樹狀圖來表示之：

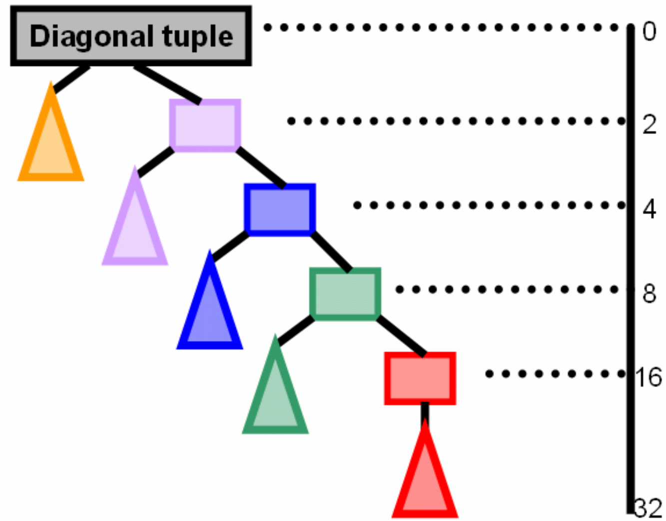


圖 4-6：每一列(行)可與 Diagonal tuple 組成 Link-list 樹狀圖。

圖 4-6 中假設以(0，0)為對角線 tuple，每個矩形為 link-list header tuple，三角型為該小區域內所生成之 link-list 之集合，越下方之三角型，所對應的區域長度越長，因此有可能會形成越長串的 link-list。

當封包在第一次的二元搜尋中找到最深的對角線 tuple(d，d)時，透過與封包相符合的規則或 marker 所填入的來源資訊(x，y)來判斷搜尋的範圍，當來源資訊為雙方向時留至下一小節再加以討論，當來源資訊為單方向時，因記載著可能存在著的小區域中最遠之一的來源位置，而較該小區域更遠的的其他區域因並無任何的規則或 marker 能與封包相符合，便無須考慮那些區域內的規則是否與封包相符合，所以接下來的搜尋範圍便跳至 tuple(x，y)，以該 tuple 為起點首先先判對是否該區域可能存在能夠與封包相符合之規則，若對 tuple(x，y)搜尋成功時，即可對所在的區域進行進一步的搜尋；反之若 tuple(x，y)搜尋失敗，我們必須跳至前一個 link-list header tuple 進行相同的判斷，因此當不斷的搜尋失敗時，會有與二元搜尋相同的表現效率。圖 4-7 即為圖 4-6 之搜尋方式，根據對角線上的來源資訊跳至最遠的 link-list header tuple 進行檢查，當檢查成功則指使向下對子樹進行細部搜尋；若檢查失敗則跳往上層再進行相同之檢查。

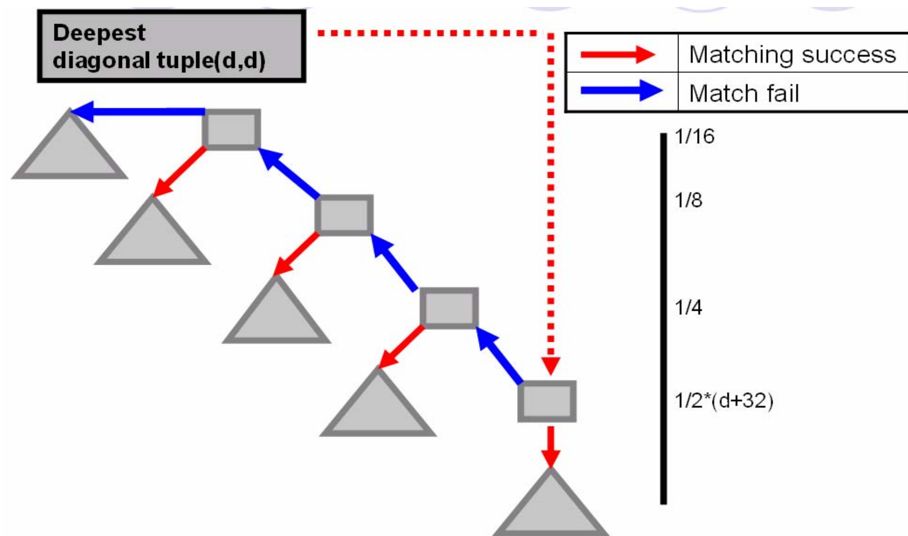


圖 4-7：圖 4-6 之搜尋路徑圖。

接著我們將介紹對小區域內部如何進行細部的搜尋動作。透過上述步驟的鏈結型式，我們能夠採取簡單的線性搜尋法來進行搜尋，當對某個 tuple 搜尋失敗便可結束整個搜尋流程。然而這樣的配置方式卻有可能導致如圖 4-8 的問題，採用上述所提的線性搜尋法時，若一封包 P(11111111111，1111111111)搜尋到 tuple(2，4)比對到規則 R1 時，根據其來源資訊接下來搜尋 tuple(2，5)比對到規則 R2，再根據 R2 的來源資訊比對 tuple(2，6)的 R3，以此類推再比較 R4 與 R5，共搜尋了五個 tuple；但若採用二元搜尋的方式時，則僅只是從 R3 到 R4 到 R5，只需三次的搜尋。因此當該列(該行)的規則散佈稀少時，使用線性搜尋的方式能取得較佳的效果；若散佈密集時，使用二元搜尋能夠獲得較線性搜尋更理想的效果。然而使用二元搜尋的方式時，則有必要將 tuple(2，4)之後的每個 tuple 內的規則向前生成 marker 直至 tuple(2，4)為止，而若該區域之規則分布稀疏時，如圖 4-9 所示般情形，使用二元搜尋之效率就不如線性搜尋般理想了。在圖 4-8 中採用二元搜尋時，由於整個搜尋範圍為(2，4)至(2，9)，因此我們必須找到標示著來源資訊為(2，4)的 marker 或規則，將來源標示更新為(L_range，R_range)，L_range 為二元搜尋的左邊(或上面)邊界，R_range 為二元搜尋的右邊(或下面)邊界，在圖 4-8 中即為(4，9)，當找到的 marker 來源資訊為此一格式時，便為二元搜尋模式，即可在此一範圍內展開二元搜尋。那麼如何選擇該使用線性搜尋或是二元搜尋便也影響了

搜尋的效率。

(2,4)	(2,5)	(2,6)	(2,7)	(2,8)	(2,9)
R1(11*,1111*,(2,5))	**R2**(11*,11111*,(2,6))	**R3**(11*,111111*,(2,8))		**R4**(11*,11111111*,(2,9)) **R5**(11*,11111100*,(-,-))	**R6**(11*,111111111*,(-,-))

圖 4-8：二元搜尋較線性搜尋佳之例子。

(2,4)	(2,5)	(2,6)	(2,7)	(2,8)	(2,9)
R1(11*,1111*,(2,9))					R2(11*,111111111*,(-,-))

圖 4-9：線性搜尋較二元搜尋佳之例子。

在這問題中，我們採用透過遞迴的方式來判斷單一行(列)需採用的搜尋演算法 [18]。由於線性搜尋不適用於規則散佈密集的情形，二元搜尋不適用於規則散佈稀疏的情形，因此我們嘗試合併線性搜尋與二元搜尋演算法，利用遞迴的方式動態地產生演算法，我們可以根據規則分配狀況來即時決定搜尋演算法，亦可以在加入或刪除規則之後重新調整搜尋演算法。

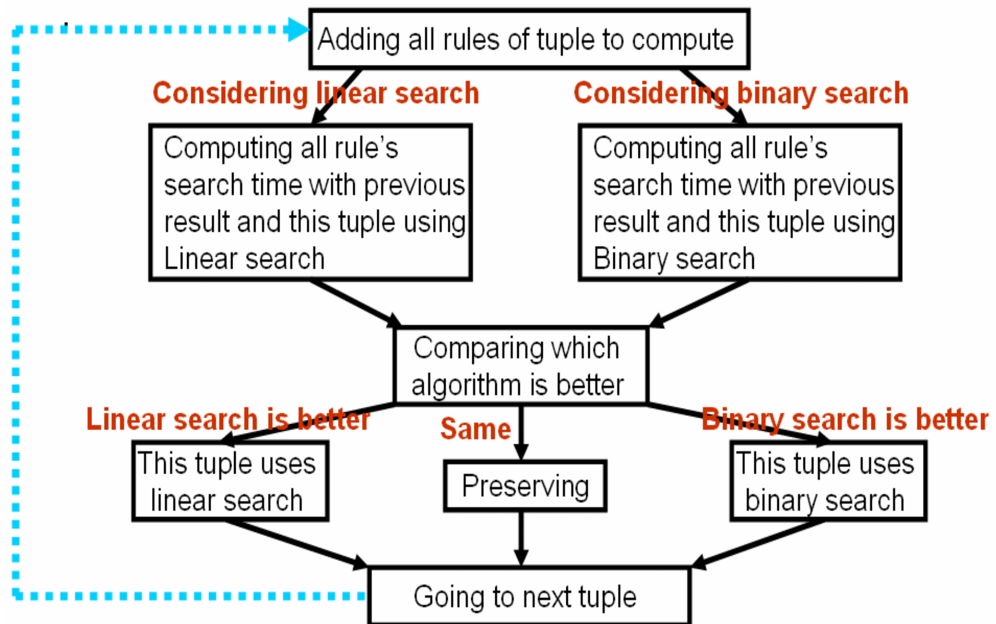


圖 4-10：動態決定區域搜尋演算法之流程圖。

圖 4-10 為整個演算法建立流程圖。當我們找到一位於 link-list header tuple 之規則或 marker 所建立的鏈結時，我們便從該鏈結所指向的第一個 tuple 開始進行處理(非 link-list header tuple)，首先依照 tuple 的先後順序，依序將最前方的 tuple 的所有規則加入，並計算每個規則的搜尋總次數。當加入一新的規則時，必須計算新規則採用線性搜尋或採用二元搜尋，何者有較好的效果，則選擇較佳效果之演算法；若次數相同時，則保留選取方式，並繼續加入下一個規則重複地計算，若結束規則的增加時次數仍然相同，則選取線性搜尋方式，因線性搜尋所需生成的 marker 個數少於二元搜尋。當整個列皆計算完畢之後，再與完全採用二元搜尋的總次數做比較並選取較佳的演算法。若選取的是線性加二元搜尋混用演算法，則必須將來源資訊為採用二元搜尋演算法的 tuple 更改為上述的格式。以下為圖 4-8 的演算法生成方式：

Step 1：加入 R1，使用線性搜尋共需 1 次

，使用二元搜尋共需 1 次

=>保留

Step 2：加入 R2，使用線性搜尋共需 1+2=3 次

，使用二元搜尋共需 1+2=3 次

=>保留

Step 3：加入 R3，使用線性搜尋共需 1+2+3=6 次

，使用二元搜尋共需 2+2+2=6 次

=>保留

Step 4：加入 R4 與 R5，使用線性搜尋共需 1+2+3+4+4=14 次

，使用二元搜尋共需 3+2+2+3+3=13 次

=>(2，4)至(2，8)間選取二元搜尋

Step 5：加入 R6，使用線性搜尋共需 3+2+2+3+3+4=17 次

，使用二元搜尋共需 3+2+3+3+3+3=17 次

=>(2，4)至(2，9)間選取二元搜尋

而最後我們必須判斷完整只對 tuple(2，4)至 tuple(2，9)間使用二元搜尋之效果是否會較動態演算法之效果佳，若是則由完整的二元搜尋所取代，因動態生成演算法能適應最佳狀況但並非有最平均之表現。在圖 4-8 中與 Step 5 之後的結果相同，故採用動態演算法對該區間做搜尋。當最終決定完演算法之後，我們必須將二元搜尋範圍內非尾端的 tuple 所對應鏈結的規則或 marker 之來源資訊給取消以避免誤導其他鏈結的搜尋動作，同時也向前生成 marker 直至二元搜尋之前端為止，如圖 4-11 所示。

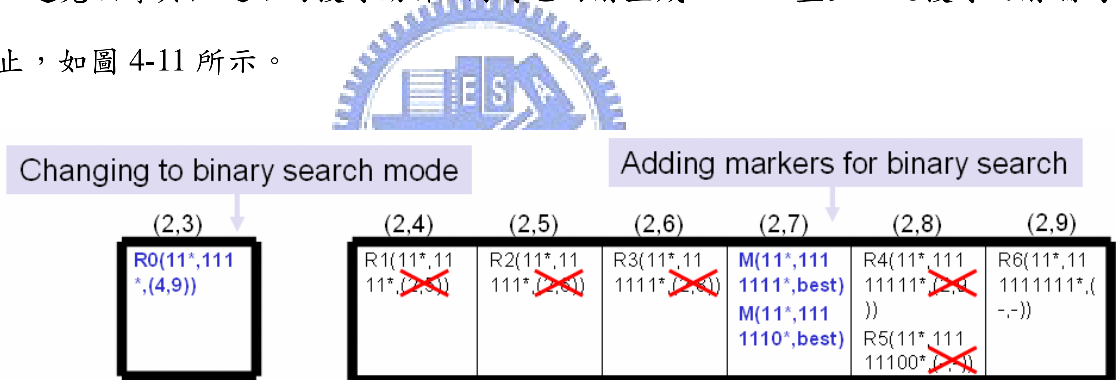


圖 4-11：最後對二元搜尋範圍內鏈結之規則或 marker 之處理。

因此我們可以對非對角線 tuple 的規則或 marker 做檢查，當判斷 link-list headertuple 內的規則 R 或 marker M 填入來源資訊時，可以知道存在著一條以來源資訊的位置為起點的 link list，便可以對該 link list 內所有比 R 或 M 更 specific 的所有規則做動態的處理，因此對每一行或每一列來說，有可能存在著二者以上的不同搜尋演算法。

圖 4-12 為至此整個單方向搜尋的流程圖，首先我們根據每個區域之最左上方 tuple 的檢查來判斷是否可能存在能符合的規則，若檢查失敗則往上一層區域做相同的檢查，當檢查成功或檢查至最上層便可直接進行二元搜尋，其範圍為該區

域之上界至下界以找出最深的 tuple，若找到能符合的規則或 marker 存在來源資訊時，若是單方向，便可直接跳至該來源資訊所指之 link-list header tuple 做檢查，若檢查失敗則往上一層的 link-list header tuple 做檢查，若檢查成功則判斷符合的規則或 marker 是否有來源資訊，若有且為二元搜尋模式時，則根據來源資訊所標示的上限與下限間的範圍做搜尋，若為線性搜尋型式時則直接前往該來源資訊所紀錄之 tuple 做檢查即可，若搜尋失敗或符合的規則或 marker 不再有來源資訊時便可結束搜尋;若符合的規則或 marker 有標記來源資訊時則再次判斷是屬於二元搜尋或是線性搜尋，一直遞迴。

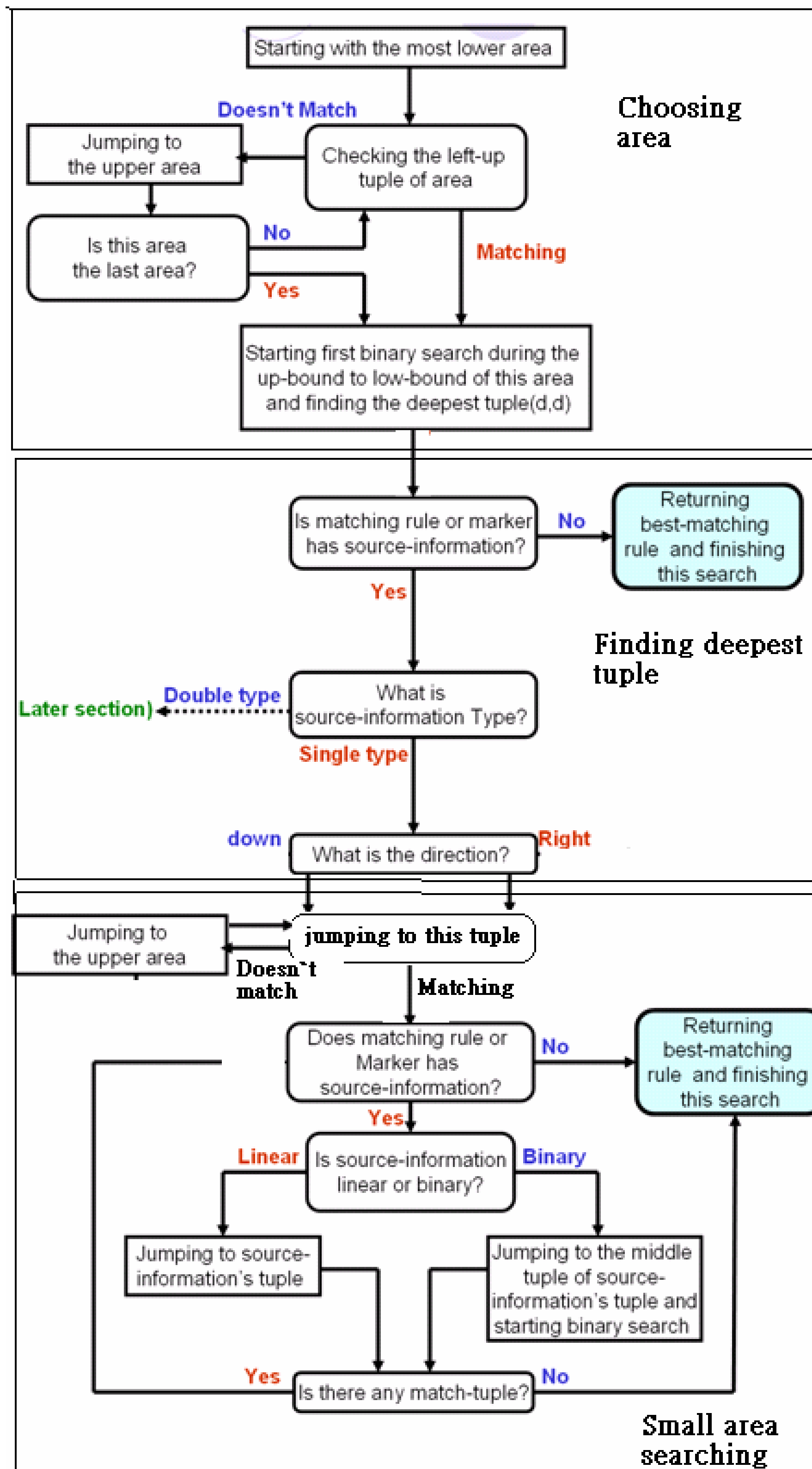


圖 4-12：封包單方向搜尋之流程圖。

4.4 衝突(Conflict)的處理

在這一演算法中，我們將衝突的情況分為二種：單方向的衝突，以及雙方向的衝突：

4.4.1 單方向衝突的處理

當收到一封包對對角線間的 tuple 進行第一次二元搜尋時，若規則 R 向左在對角線 tuple(k，k)生成 marker M，規則 R`向左在對角線 tuple(k`，k`)生成 marker M`，k`>k，marker M`的範圍能夠被 marker M 的範圍所覆蓋住，如此第一次二元搜尋極有可能受到 marker M`的引導而搜尋至 tuple(k`，k`)而非 tuple(k，k)的位置，儘管對封包來說規則 R 可能會是它的最佳符合規則，如圖 4-13 所示，此為單方向衝突問題，當收到封包 P 進行二元搜尋，因 R2 生成 marker(11*，11*)導致封包 P 搜尋至該 tuple，然而進一步進行深入二元搜尋時卻會於 tuple(2，3)造成搜尋失敗而無法理想地搜尋至 R1。

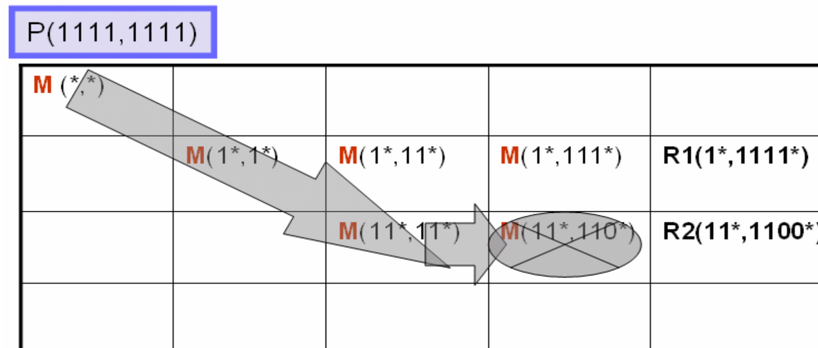


圖 4-13：產生衝突的範例圖示。

為了解決這類的問題發生，如同 Diagonal tuple space search 般，每個規則必須對存在於 IncompatibleTuples 的所有對角線 tuple 做檢查，若這些 tuple 的規則或是 marker 可能與規則 R 產生衝突的情形，便需增加一條 Mirror rule [16]。假設規則 R 位於 tuple(x，y)，若 y>x 時，便須對 tuple(x+1，x+1)至 tuple(y-1，y-1)間的每一個對角線 tuple 做檢查，若存在某一 tuple(k，k)的 marker 或規則使得該 marker

或規則的前 x 個 source specified bits 與規則 R 的 source specified bits 相同，規則 R 的前 k 個 destination specified bits 與該 marker 或規則的 destination specified bits 相同，便有可能產生單方向衝突問題；若 y<x 時，便須對 tuple(y+1，y+1)至 tuple(x-1，x-1)間的每一個對角線 tuple 做檢查，若存在某一 tuple(k，k)的 marker 或規則使得該 marker 或規則的前 k 個 source specified bits 與規則 R 的 source specified bits 相同，規則 R 的前 y 個 destination specified bits 與該 marker 或規則的 destination specified bits 相同，便有可能產生單方向衝突問題。

為了避免這種問題造成搜尋失誤，必須在規則 R 與 tuple(k，k)的聯集位置生成 Mirror rule，亦即聯集規則。當 y>x，聯集規則位於 tuple(k，y)；當 y<x，聯集規則位於 tuple(x，k) 。根據聯集規則的位置而生成 marker 的方式則與一般的規則無異，但聯集規則仍須與 marker 有著相同資料結構，即必須紀錄該位置的最佳符合規則。

因此當收到封包進行第一次二元搜尋時，若規則 R 向左在對角線 tuple(k，k)生成 marker M，規則 R`向左在對角線 tuple(k`，k`)生成 marker M`，k`>k，以及 marker M`的範圍能夠被 marker M 的範圍所覆蓋住，若封包能夠符合規則 R，當第一次二元搜尋至 tuple(k`，k`)位置時，由於 k`>k，封包必能滿足規則 R 的 source specified bits，又由於聯集規則的 destination specified bits 與規則 R 相同，因此搜尋必定能到達規則 R 與 marker M`的聯集位置甚至之後，如此便可解決單方向衝突問題。如圖 4-14 所示，增加了 R3 這個聯集規則以及相對應的 marker 之後，封包 P 便能搜尋至 R3 的位置並得知 R3 能與封包 P 相配合，再透過 R3 的最佳符合規則便能獲得 R1。

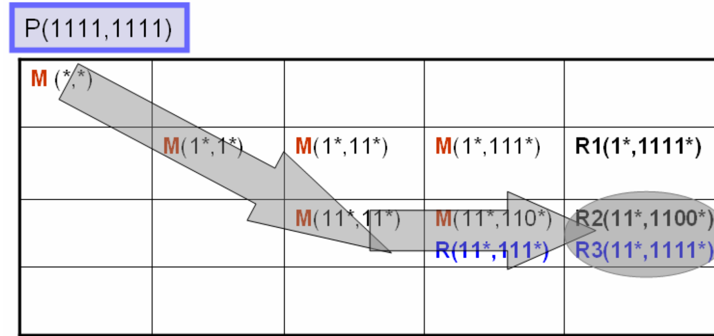


圖 4-14：解決衝突後之範例。

4.4.2 雙方向衝突的處理

當對對角線間的 tuple 進行第一次二元搜尋時，若所找到能與封包相配合的規則或是 marker 的來源資訊類別為雙方向時，即表示下方及右方皆有可能存在可能與封包相配合的規則，此時演算法會因不知該對哪一方向做進一步的二元搜尋而產生問題，此便為雙方向的衝突問題。

當在建立 marker 階段時，若一規則 R 位於 tuple(x，y)生成 marker 於 tuple(x，x)時，若已存在相同的 marker 且來源資訊與規則 R 為不同方向，此時便將該 marker 的來源資訊設為雙方向，此時可分為以下的步驟：

Step 1：檢查另一方向的所有 tuple，若存在規則 R`使得 R`在 tuple(x,x)所生成的 marker 與規則 R 相同。

Step 2：取得規則 R 與規則 R`的連集規則，若 R 為(00*****，0000000)，R`為(00000**，00*****)，則他們之間的聯集規則為(00000**，0000000)。

Step 3：再從聯集規則生成相對於對角線 tuple 的 marker，如(00000**，0000000)會在 tuple(5，5)生成 marker(00000**，00000**)，此時該 marker 的來源資訊便為(2，count)，即(x，count)。

Step 4：再由 tuple(x，x) (即(2，2)) 沿著對角線向右下方生成 marker 直至該

對角線 tuple 的 marker，同樣的，這些 marker 的來源資訊亦填入(x，count)，count 為上一層 marker 生成於該 tuple 之第幾各位置，如 R 與 R`在 tuple(3，3)生成 marker 於第七各位置時，則 tuple(4，4)的 marker 之來源資訊便記錄為(2，count=7)。若存在其他的規則或 marker 使得 marker 的生成因為重複而被取消，檢查這些 marker 的來源位置，若這些 marker 的來源資訊位置(d，count)較(x，count)為深，即 d>x，則不予變更；若(x，count)較(d，count)為深，即 x>d，則這些 marker 的來源位置由(x，count)來取代(d，count)。

對 Step 4 來說，當向左上方生成 marker 直至 tuple(x，x)，若存在某重複的規則或 marker，它們的來源資訊只會有 NULL 及雙方向兩種，不存在單方向的來源資訊。若存在某一 marker 的來源資訊為單方向且會造成 Step 4 的 marker 生成重複，如 marker(0000***,0000***)且來源資訊為(4,7)，而在 Section 4.4.1 小節所描述的對單方向衝突的處理中，必定能使規則 R(00*****，0000000)與規則 R`(00000**，00*****)對 marker(0000***,0000***)產生衝突問題，因此分別在 tuple(4，7)產生 Mirror rule(0000***，0000000)及 tuple(5，4)產升 Mirror rule(00000**，0000***)使得 marker(0000***，0000***)的來源資訊變更為雙方向。

由於 Step 3 所生成的 marker 的(source specified bits，destination specified bits)是分別取自於規則 R 的 destination specified bits 與規則 R`的 source specified bits，因此在 Step 4 中向左上方生成的 marker 亦是取自於規則 R 的 destination specified bits 與規則 R`的 source specified bits，因此如圖 4-15 所示，當第一次二元搜尋能找到最深的 tuple 為(k，k)的某一 marker 且 k>x 時，我們可以得知以下二項資訊：

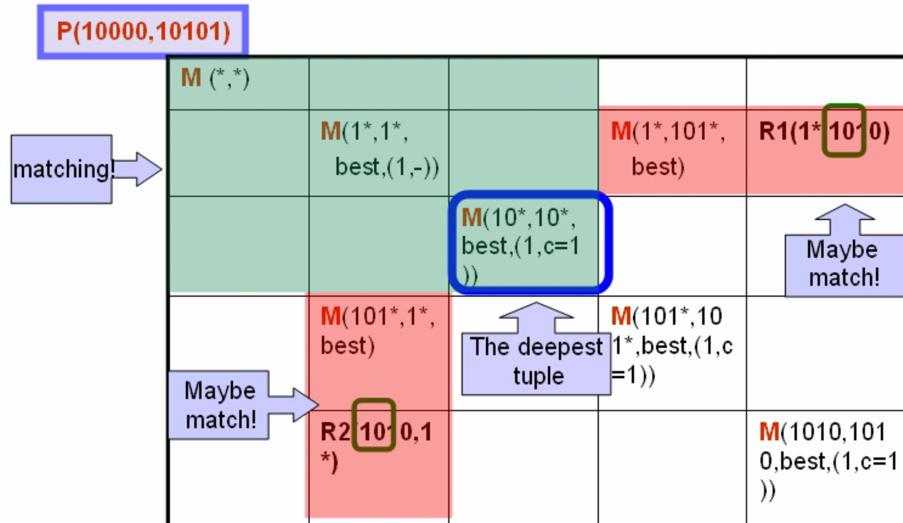


圖 4-15：解決雙方向衝突之範例。

- 由於 tuple(x，x)在 tuple(k，k)的 ShortTuples 範圍內，當封包能滿足 tuple(k，k)的 marker 時，必能滿足規則 R 與規則 R`在 tuple(x，x)所生成的 marker，因此我們可以得知封包必能滿足規則 R 的 source specified bits，以及規則 R`的 destination specified bits。
- 當封包滿足 tuple(k，k)的 marker，我們能解釋為該封包至少能滿足規則 R 的前 k 個 destination specified bits，以及規則 R`的前 k 個 source specified bits，如圖七中，最深 tuple 的 marker 為(11*，11*)，我們便能得知該封包至少滿足 R1 的前二個 destination specified bits 以及 R2 的前二個 source specified bits。

然而由於 tuple(k，k)為第一次二元搜尋所能找到最深 tuple，因此可能產生以下三種情形之一：

Case 1：封包只能滿足規則 R 的前 k`個 destination specified bits，以及規則 R`的前 k 個 source specified bits，k`>k。可以得知封包由 tuple(x，x)開始，向右搜尋能到 tuple(x，k`)的位置，向下搜尋至多只能到 tuple(k，x)的位置，而由於 tuple(k，x)在 tuple(k，k)的 ShortTuples 的範圍內，因此只需比較在 tuple(k，k)的 marker 及在 tuple(x，k`)的 marker 所記錄的最佳符合規則資訊何者優先權為大即可，如圖 4-16.a 中，tuple(k，k)的 marker 能找出整個淺灰色區域的最佳符合規則，而深灰色區域有一部分未被淺灰色區域所包含住，其中可能存在著滿足封包的最佳符合規則，因此也必須對該區域做搜尋。

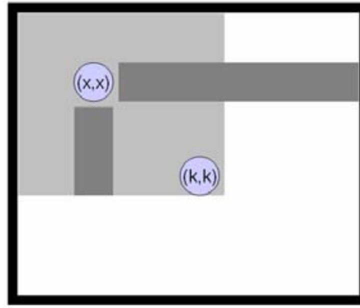


圖 4-16.a：最佳配對規則可能分布範圍(a)。

Case 2：封包只能滿足規則 R 的前 k 個 destination specified bits，以及規則 R`的前 k`個 source specified bits，k`>k。可以得知封包由 tuple(x，x)開始，向右搜尋至多只能到 tuple(x，k)的位置，向下搜尋能到 tuple(k`，x)的位置，而由於 tuple(x，k)在 tuple(k，k)的 ShortTuples 的範圍內，因此只需比較在 tuple(k，k)的 marker 及在 tuple(k`，x)的 marker 所記錄的最佳符合規則資訊何者優先權為大即可，如圖 4-16.b。

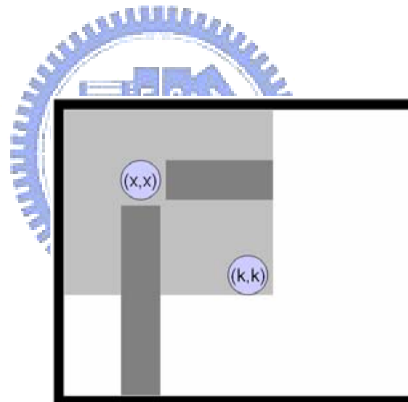


圖 4-16.b：最佳配對規則可能分布範圍(b)。

Case 3：封包只能滿足規則 R 的前 k 個 destination specified bits，以及規則 R`的前 k 個 source specified bits。可以得知封包由 tuple(x，x)開始，向右搜尋至多能到 tuple(x，k)的位置，向下搜尋至多能到 tuple(k，x)的位置，而由於 tuple(x，k)及 tuple(k，x)在 tuple(k，k)的 ShortTuples 的範圍內，因此 tuple(k，k)的 marker 所記錄的最佳符合規則資訊便為最佳規則，如圖 4-16.c，由於整個深灰色區域接被淺灰色區域所包含住，因此淺灰色區域的最佳符合規則即為所能找到的最佳規則。

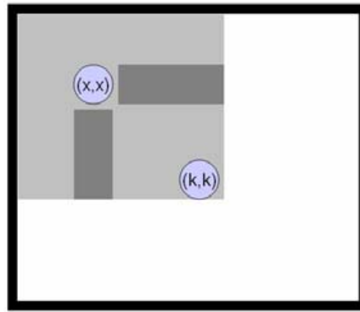


圖 4-16.c：最佳配對規則可能分布範圍(c)。

二元搜尋的特色為：當搜尋成功時，更進一步的深入進行二元搜尋；當搜尋失敗，則退一步的深入進行二元搜尋。因此當收到封包進行第一次二元搜尋時，我們需要一變數 Dim 來記錄比對的規則或 marker 的 source specified bits 或 destination specified bits 能否與封包相符合，當封包檢察能與規則或 marker 的 source specified bits 及 destination specified bits 符合時，則最深的 tuple 便由該規則或 marker 所在的 tuple 所取代，並向前進行更細的二元搜尋；反之都不符合或僅符合 source specified bits 或 destination specified bits 其中之一時，則視為比對失敗向後進行細部的二元搜尋。當進行第一次二元搜尋找到能與封包符合的 tuple 時，我們會將該 tuple 暫時紀錄為最深的 tuple，並開始在之後的搜尋中進行 Dim 的紀錄：若對某個來源資訊為(x，count)的規則或 marker 搜尋失敗且能夠符合 source specified bits 或 destination specified bits 其中之一時，而 count 之值也與最深 tuple 之規則或 marker 相同時，便用 Dim 紀錄之，因此 Dim 的狀態有三種：滿足 source、滿足 destination 以及都不滿足(NULL)。另外當搜尋進行至 tuple(1，1)時也須進行 Dim 的紀錄，因為 tuple(0，0)必定能與封包符合，然而欲搜尋至 tuple(0，0)之前得不斷的搜尋失敗，因此 tuple(1，1)的搜尋便得附加 Dim 的紀錄動作。

透過 Dim 的狀態，我們可以得知該封包的搜尋是屬於 Case 1~3 的何者。當 Dim=source 時，我們可以知道封包的 source 欄位能符合規則 R`至少大於 k 個 source specified bits，因此屬於 Case 2 的情形，接著根據 tuple(k，k)的 marker 來源資訊，得知該 marker 來源為(x，x)，藉由圖 4-16.b 所示，直接由 tuple(k，x)開始進行搜尋而非由 tuple(x，x)開始；當 Dim=destination 時，我們可以知道封包的

destination 欄位能符合規則 R 至少大於 k 個 destination specified bits，因此屬於 Case 1 的情形，接著根據 tuple(k，k)的 marker 來源資訊，得知該 marker 來源為(x，x)，藉由圖 4-16.a 所示，直接由 tuple(x，k)開始進行搜尋而非由 tuple(x，x)開始；當 Dim=NULL，我們可以得知封包的每個欄位至多只能滿足規則 R 的 k 個 destination specified bits 以及規則 R`的 k 個 source specified bits，因此封包至遠只能滿足 tuple(k，k)的規則或 marker，該規則或 marker 所紀錄的最佳規則便為滿足該封包的最佳符合規則。

然而這樣的處理方式會存在著一個例外情形，當第一次二元搜尋時直到最後一次的搜尋才找到最深的 tuple 時，此時無法紀錄 Dim 之值，因第一次二元搜尋已經結束。如圖 4-17 的例子中，R1 與 R2 會造成雙方向衝突，因此分別對 tuple(2，2)、tuple(3，3)及 tuple(4，4)生成來源資訊為(1，count)的 marker。當收到一封包 P(10000，10111)先對最下層正方形區域的最左上方 tuple(tuple(3，3))進行比對，比對失敗後往上一層的正方形區域進行第一次二元搜尋，此時將會紀錄 tuple(2，2)為最深的 tuple，但我們無法開始對 Dim 進行紀錄，因為整個二元搜尋業已結束。因此當發生此一情形時，我們必須再對 tuple(d+1，d+1)多進行一次搜尋以進行 Dim 的紀錄，d 為最深 tuple 之位置，在圖 4-16 中就得再對 tuple(3，3)進行一次搜尋，此時便能紀錄 Dim 為 Destination，再藉由 tuple(2，2)的 marker 填入的來源資訊知道是由 tuple(1，1)產生雙方向衝突問題，因此我們便可直接對 tuple(1，1)右方進行橫向的細部搜尋。

M (*,*)	P(10000,10111),Dim=Null			
	M(1*,1*,(1,-))	M(1*,10*)	M(1*,101*)	R1(1*,1010)
	M(10*,1*)	P(10000,10111),Dim=Null M(10*,10*,(1,c=1))		
	M(101*,1*)		M(101*,101*,(1,c=1))	
	R2(1010,1*)			M(1010,1010,(1,c=1))

圖 4-17：雙方向衝突的例外情形。

4.5 流程的說明

4.5.1 Marker 生成的流程

由於單方向的衝突情形必須生成聯集規則，對我們的演算法來說聯集規則與一般的規則無異，所以在處理正方形區域外之演算法選擇與 marker 生成之前，我們必須先將聯集規則的資訊儲存於 hash table 的欄位中。我們將整個生成流程以下列的步驟來表示：

Step 1：將規則根據 source specified bit 之個數及 destination specified bits 之個數儲存於相對應的 hash table 之欄位中。

Step 2：生成 marker：先對每各規則所對應的對角線 tuple 生成 marker，並往左上方亦生成 marker 直至區域之上界為止。此時尚不需對規則所對應之對角線之 marker 填入來源資訊。

Step 3：對每個規則以及對角線上的規則及 marker 檢查是否可能發生單方向衝突，若是，則生成一個聯集規則，並依據其位置生成 marker。

Step 4：對配置於非對角線 tuple 之規則生成 marker 至相對應的 link-list header tuple，並填入來源資訊，如果已被填入，判斷何者距離對角線位置較近，由較近者的位置取代來源資訊，而較遠者生成 marker 至較近者位置。

Step 5：對每個對角線 tuple 上的規則或 marker 做檢查已填入來源資訊，判斷是否為單方向，若是則找出最遠的 link-list header tuple 為其來源資訊。

Step 6：利用遞迴方式動態決定每個小區域之演算法

Step 7：若對角線 tuple 的規則或 marker 之來源資訊為雙方向時，則分別向右及向下尋找出相對應的兩個規則(包含單方向衝突所產生的聯集規則)，並對此二規則的聯集位置所對應的對角線位置生成 marker，並向左上方生成 marker 直至造成雙方向衝突的 tuple 為止，並更改來源資訊。

4.5.2 搜尋的流程

在這一小節中，我們將把整個搜尋演算法分成下列的幾個步驟：

Step 1：當收到一封包時，針對最右下方正方形區域的最左上方 tuple 做檢查比對：若檢查失敗時，則跳至上一層區域的最左上方 tuple 進行檢查，以此遞迴；檢查成功時，前往 Step 2。

Step 2：進行區域內對角線 tuple 的二元搜尋，並進行變數 Dim 的紀錄，找到最深的 tuple(d，d)，並記錄此時的最佳符合規則 R_D，若所找到的規則或 marker 其來源資訊屬於單方向時，前往 Step 3.1，並記錄此時的最佳符合規則；若所找到的規則或 marker 其來源資訊屬於雙方向時，前往 Step 3.2，並記錄此時的最佳符合規則；若找到的規則未填入任何來源資訊時，則直接回傳該規則。

Step 3.1：根據位於 tuple(d，d)且能與封包相符合的規則或 marker 所填入的來源資訊(x，y)，前往 tuple(x，y)檢查是否符合，若是則進行 tuple(x，y)所對應小

區域間的搜尋，並將最終找到的規則或 marker 所記錄的最佳符合規則與目前所記錄的最佳符合規則做比較選出最高優先權者，若最終找到的規則或 marker 無任何來源資訊，則回傳此最佳符合規則；若最終找到的規則或 marker 填入來源資訊，則前往 Step 4。

Step 3.2：若找到最深 tuple(d，d)時 Dim 仍為 Null，再檢察一次 tuple(d+1，d+1)以記錄 Dim。根據位於 tuple(d，d)且能與封包相符合的規則或 marker 所填入的來源資訊(k，count)，以及 Dim 的紀錄來做判斷：若 Dim 紀錄為 Destination 時，則由 tuple(k，(32-k)/2)進行搜尋以判定該小區域是否可能存在符合的規則；若 Dim 紀錄為 Source 時，則由 tuple((32-k)/2，k) 進行搜尋以判定該小區域是否可能存在符合的規則。並將最終找到的規則或 marker 所記錄的最佳符合規則與目前所記錄的最佳符合規則做比較選出最高優先權者，若最終找到的規則或 marker 無任何來源資訊，則回傳此最佳符合規則；若最終找到的規則或 marker 填入來源資訊，則前往 Step 4。

Stpe 4：根據來源資訊型別做判斷，若該位置為 tuple(m，n)：

- Case 1：若為二元搜尋型別(l，r)時，則根據 m，n 來做二元搜尋的範圍：若 n>m，則由 tuple(m，l)至 tuple(m，r)間進行二元搜尋；若 n<m，則由 tuple(l，m)至 tuple(r，m)間進行二元搜尋，最後將最終找到的規則或 marker 所記錄的最佳符合規則與目前所記錄的最佳符合規則做比較選出最高優先權者，若無填入來源資訊時則回傳之，即可結束整個搜尋流程，否則重複進行 Step 4 的判斷。
- Case 2：若為線性搜尋型別(m，next)或(next，n)時，則跳往該來源資訊所指引位置做檢查，將找到的規則或 marker 所記錄的最佳符合規則與目前所記錄的最佳符合規則做比較選出最高優先權者，再根據所記錄的來源資訊重複進行 Step 4 的判斷，直至整個 tuple 搜尋失敗或是所搜尋到的規則或 marker 未填入來源資訊為止，最後將最佳符合規則回傳之，即可結束整個搜尋流程。

4.6 演算法的証明

(1)、當第一次搜尋所能找到的最深 tuple(d，d)，則封包的最佳符合規則可能存在的範圍為 Tuple(d，d)∪Tuple(d，d)之 ShortTuples∪T(d+1，d+1)之 IncompatibleTuples。

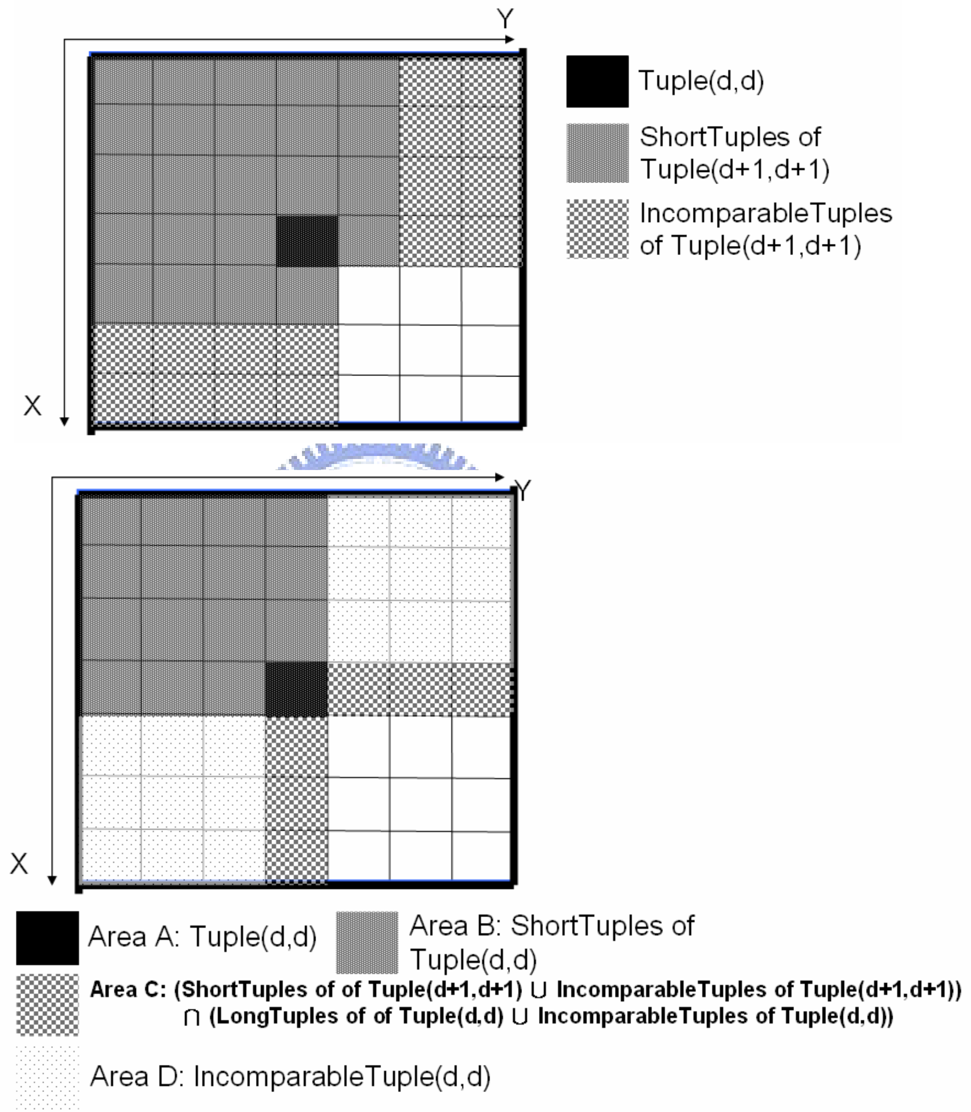


圖 4-18：最深 tuple(d，d)之最佳符合規則範圍圖。

圖 4-18 為最深 tuple(d，d)時的最佳符合規則，上下二圖為不同的表示方法。在圖 4-18 之下圖中，若最佳符合規則位於 Area A 時，當搜尋至 tuple(d，d)時即可取得；若最佳符合規則位於 Area B 時，能藉由 precomputation 過程中求得；若最佳符合規則位於 Area C 時，由於 Area C 的規則會生成 marker 至 Area A，所以

可以藉由第二次之後的二元搜尋來搜尋出最佳符合規則；若最佳符合規則位於 Area D 時，此時該區的規則將會對 tuple(d，d)的規則或 marker 產生單方向衝突，因此會在 Area C 區產生聯集規則，藉由對 Area C 的搜尋亦能找出存在於 Area D 的最佳符合規則；若最佳符合規則位於其他區域時，則會對 Area A 產生 marker，因此第一次的二元搜尋將會因此 marker 的引導而使最深 tuple 往前推進至 tuple(d`，d`)，d`>d，與最深 tuple 為(d，d)的前提產生矛盾，故此假設不成立。

(2)、若 tuple(x，y)存在一可與封包配對的規則，而 tuple(x，y)的所有 LongTuple 不存在可與封包配對的規則，則 tuple(x，y)的 IncomparableTuple 亦不存在可供配對的規則。

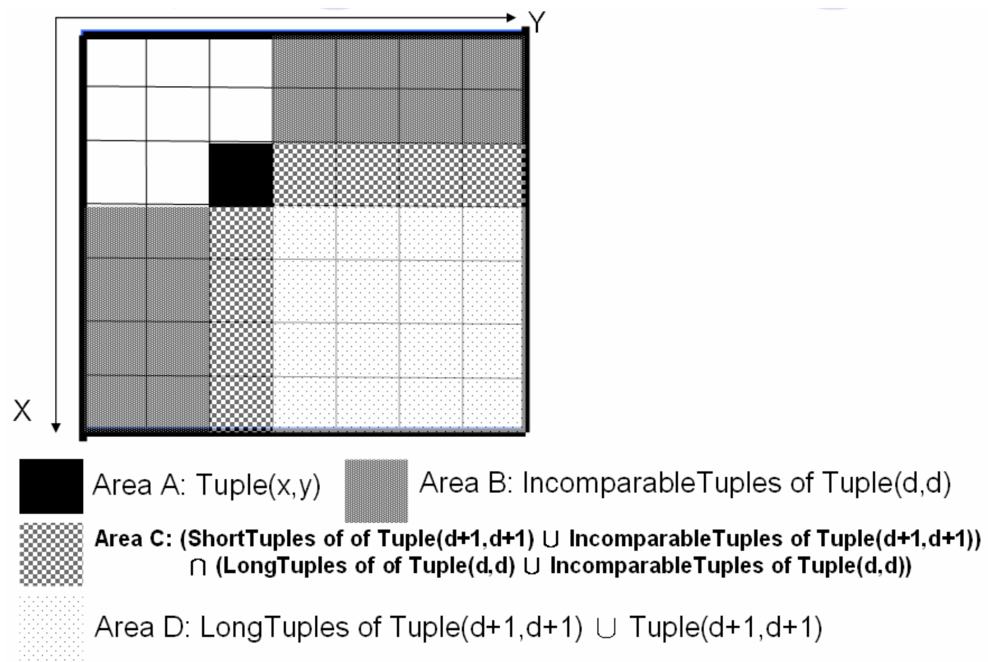


圖 4-19：LongTuple 與 IncomparableTuple 之範圍圖。

Tuple(x，y)的 LongTuple 為圖 4-19 中 Area C 與 Area D 的交集，若 tuple(x，y)的 IncomparableTuple 存在可供配對的規則時，此時將會對 Tuple(x，y)的規則或 marker 產生單方向的衝突問題，因此在 Area C 中產生聯集規則，由於聯集規則可視為規則之一，以及 Area C 位於 Tuple(x，y)的 LongTuple 範圍內，因此將使得 Tuple(x，y)的 LongTuple 存在可供配對的規則，與 tuple(x，y)的所有 LongTuple 不存在可與封包配對的規則的大前提產生矛盾，故此假設不成立。

此二驗證不僅能適用於 diagonal Tuple Space search 演算法也能適用於我們所提出的 Advanced Regional Diagonal Tuple Space Search 演算法。當開始進行搜尋時，正方形區域間的跳躍搜尋以及第一次的二元搜尋以決定最深的對角線 tuple，透過定理一的驗證能夠決定出最佳符合規則的分布範圍；當方向為雙方向時，透過 Section 4.4.2 的驗證，我們亦能證明所選出的方向的正確性；當開始進行第二次二元搜尋之後的搜尋動作所找到的距離對角線最遠的 tuple，便能透過定理二的驗證得知該 tuple 中能與封包配對的規則或是 marker 所記錄的最佳規則即是該方向搜尋中所能找到的最佳規則。

4.7 搜尋範例

如圖 4-20 所示為簡單的規則與 marker 的配置圖，我們將整個 tuple space 分

R10(*, *)					M(*,11111*,R10,(0,6))	R3(1*,111111*)		
	M(1*,1*,R10,(1,-)) M(0*,0*,R10)				M(*,11111*,R10,(1,7))	M(1*,111111*,R3,(1,7))	R2(1*,1111111*)	
		M(00*,00*,R10,(3,2)) M(11*,11*,R10,(2,c=1))				R7(11*,111111*,(2,7)) M(11*,1111000*,R10,(2,7))	R8(11*,1111000*,(2,8)) M(11*,1111111*,R2)	R9(11*,11110001*)
		R4(000*,00*)	M(111*,111*,R10,(2,c=2))					
				M(1111*,1111*,R10,(2,c=1))				
	M(11111*,1*,R10,(6,1))	M(11111*,11*,R10,(6,2))			M(00000*,00000*,R4) M(11111*,11111*,R10,(2,c=1))			
	R1(111111*,1*)	M(111111*,11*,R1)				M(000000*,000000*,R4,(6,-)) M(111111*,111111*,R7,(2,c=2))	R6(000000*,0000000*)	
						R5(0000000*,000000*)	M(000000*,000000*,R5,(6,c=1))	

圖 4-20：Tuple Space 之範例圖。

為二區塊，並以 tuple(0，0)及 tuple(5，5)為區塊上界，位於非對角線 tuple 之規則則生成 marker 直至該 tuple 所屬的 link-list header tuple，並考慮單向及雙向衝突問題，其中 R1~R10 為規則編號，R1 之優先權為最高，M 為 marker 之代稱。若收到封包 P1 之檔頭資訊為(11110000，11111111)時，由於 tuple(5，5)不存在可與 P1 配對之規則或 marker，因此往上一層區域做搜尋，由於為最後一層區域固直接開始進行二元搜尋，首先於 tuple(2，2)找到能配對的 marker，此時記錄最深 tuple 暫為(2，2)，最佳規則暫為 R10，並開始進行 Dim 的記錄，接著 tuple(3，3)及 tuple(4，4)皆存在規則或 marker 能與 P1 配對，此時的最深 tuple 為(4，4)，最佳規則暫為 R10，而 Dim 為 NULL，因第一次二元搜尋結束時最深 tuple 的 marker 所記錄的來源資訊為雙方向，而 Dim 依然為 NULL，因此我們必須再向下一層的 tuple 搜尋一次來記錄 Dim，即對 tuple(5，5)再搜尋一次，當我們找到第二個 marker M(11111*，11111*，R10，(2，c=1))時，根據所紀錄的 c=1 與 tuple(4，4)中能與 P1 相符合的 marker 在 tuple 中所儲存的次序是相同的，而此一 marker 之 destination specified bits 皆與封包 P1 相符合，我們便能紀錄 Dim 為 destination，因此我們便能從 tuple(2，2)為起點，朝著 destination 之方向進行 link-list 搜尋。由 tuple(2，6)開始，找到能符合的 marker，其來源資訊為 tuple(2，7)，因此下一走我們對 tuple(2，7)進行搜尋，在 tuple(2，7)中我們找到 marker(11*，1111111*，R2)能與封包相符合，我們便更改最佳配對規則為 R2，又因此 marker 無紀錄來源資訊，我們便無需繼續搜尋，因此整個搜尋流程即可結束，而最佳配對規則為 R2。

第五章　模擬與分析

5.1.1 Advanced regional diagonal tuple space search 之分析環境

我們所提出的 Advanced regional diagonal tuple space search 演算法之概念，可以分解為三部份：分區，link-list 處理與單方向等三部份的 marker 生成與搜尋，為了測試這三項概念有益於 marker 數量的減少以及搜尋次數的減少，我們分別以這三項概念為主題作測試，實驗組為 Diagonal search 加上分區，加上列與行的來源資訊與 link-list 處理，以及加上對雙方向衝突處理等三大項，對照組則為傳統的 Diagonal search，分別對 1000 條規則，5000 條規則以及 10000 條規則做測試，規則的生成方式分為 Prefix length 的生成以及 Conflicting 之模擬：

(1). Prefix length：40%之規則配置於[16≦x≦32，16≦y≦32]範圍內，x 與 y 分別為 source mask bit 個數及 destination mask bit 個數,亦即在 tuple space 中所對應之 tuple，其餘之 60%規則則成亂數分佈。

(2). Conflicting：分別將其中的 80%, 40%, 0%之規則設為會與其他規則產生 conflicting (single-conflicting)，其餘規則則呈亂數產生。Conflicting rule 之產生為隨機取一位於 tuple(x，y)規則 R，將規則 R 亦生成於 tuple(x`，y`)中為 R`，(x`≦x，y≦y`)或(x≦x`，y`≦y)，並分別隨機取規則 R`的前 x``各 source mask bit 及 y``各 destination mask bit，其值不與變更，x``<x`，y``<y`，而[x``~x`]與[y``~y`]間之 bit 必須隨機修改以產生 conflicting 狀態。

5.1.1 分區之測試

模擬環境：同為 diagonal tuple space 中，圖 5-1 之右圖乃以(0，0)~(8，8)，(9，

9)~(15，15)，(16，16)~(32，32)等三各獨立區塊之 diagonal tuple space，比較 tuple(0，0)，tuple(1，1)，…tuple(15，15)所包含之 marker 之差異：

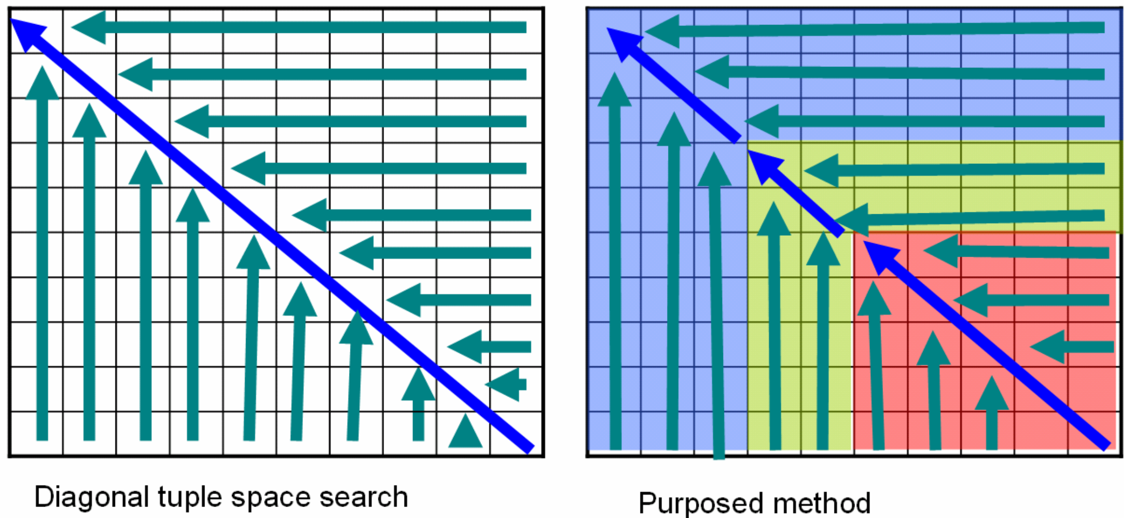


圖 5-1：演算法之差異性：採用分區之影響。

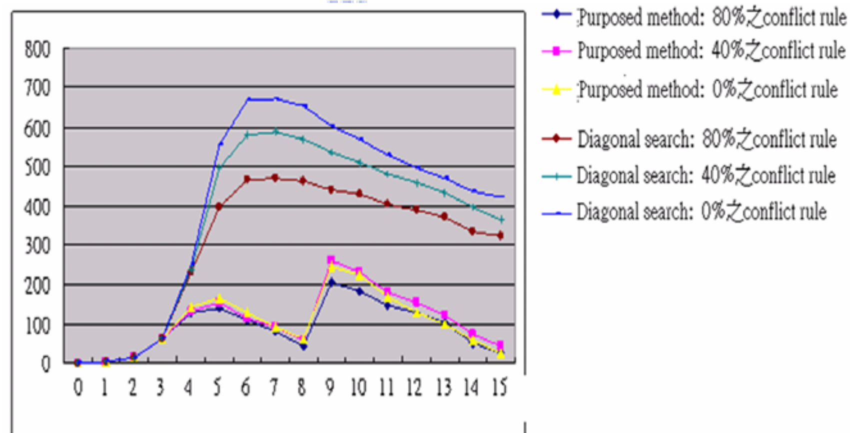


圖 5-2：規則總數為 1000 時，tuple(0，0)至 tuple(15，15)中各個 diagonal tuple 中的 marker 總數之折線圖。

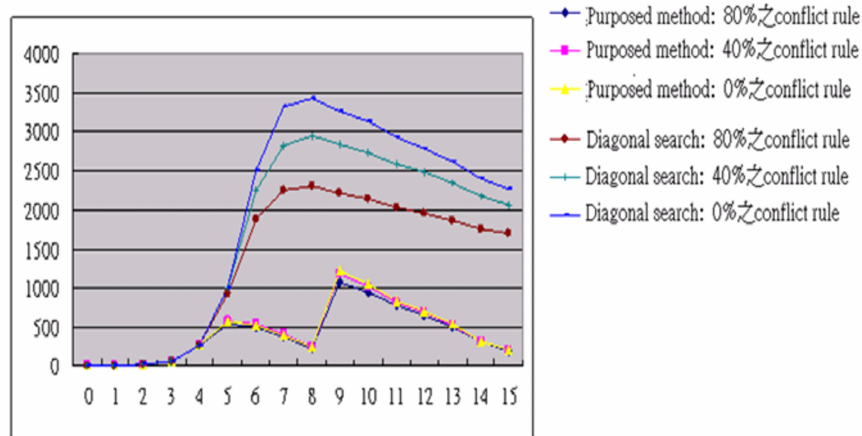


圖 5-3：規則總數為 5000 時，tuple(0，0)至 tuple(15，15)中各個 diagonal tuple 中的 marker 總數之折線圖。

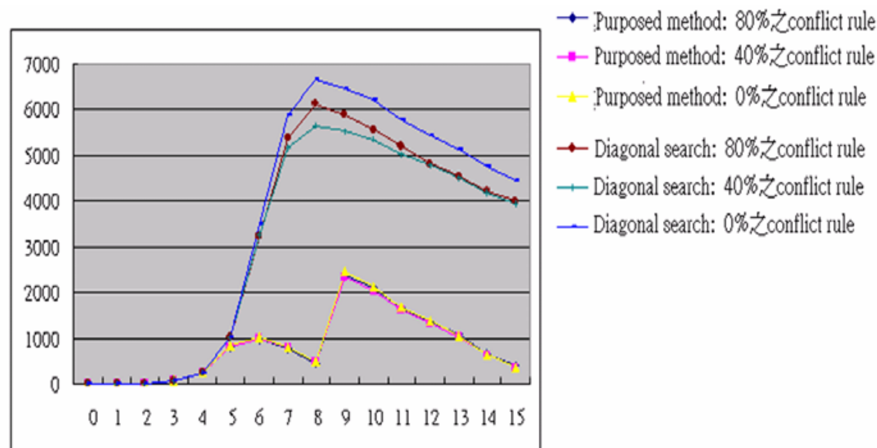


圖 5-4：規則總數為 10000 時，tuple(0，0)至 tuple(15，15)中各個 diagonal tuple 中的 marker 總數之折線圖。

圖 5-2 至圖 5-4 分別為規則總數為 1000，5000 與 10000 時，tuple(0，0)至 tuple(15，15)中各個 diagonal tuple 中的 marker 總數。Diagonal search 中，從 tuple(8，8)開始往回至 tuple(0，0)為止，marker 之總數量會呈現倍減的現象，因對這些 tuple 來說，marker 之總數量已呈現飽和狀態無法再增加，而從 tuple(15，15)往回至 tuple(8，8)為止，因 marker 重複率並不高，而每一層不僅接收來自右下規則的 marker，也接收來自右方與下方規則之 marker, 因此呈現遞增狀態。而加入分區機制的 diagonal search 則於由在 tuple(8，8)及 tuple(15，15)重新開始生成 marker，因此不接收來自右下方區域之規則的 marker，不會造成這些區域的 marker 數量過

高，而在 tuple(0，0)至 tuple(5，5)因已接近飽和，所以這區域之圖形與 Diagonal search 相似。

5.1.2 Link-list 之測試

模擬環境：同為 diagonal tuple space 中，圖 5-5 之右圖乃在對非對角線 tuple 之規則與 marker 建立 link-list，並對對角線 tuple 的規則與 marker 分別加入對右方以及對下方的來源資訊，使得當找到最深之 tuple 時能分別向右及向下進行跳躍式搜尋，分別比較平均的搜尋次數：

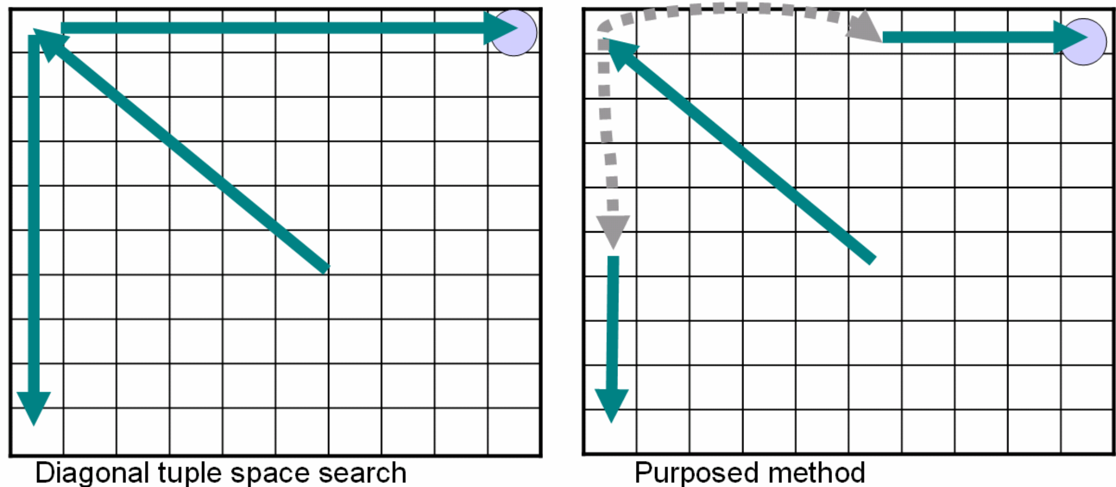


圖 5-5：演算法之差異性：採用 link-list 機制之影響。

由於對 diagonal tuple 之外的所有 tuple 採用 link list 建樹式搜尋，因此若對某一 header tuple 搜尋成功時，能夠有與二元搜尋相等或較佳的結果，而若對 header 搜尋失敗時，則搜尋次數會與二元搜尋相同。圖 5-6 中，當規則的總數提升時，即每各 tuple 配置到規則機率亦跟著提升，密度亦增大，因此動態設計演算法採用二元搜尋的機率也跟著提升。當規則總數為 1000 時採用線性搜尋之機會較高，因而能有較佳之結果，同時當規則總數為 5000 或 10000 時，則因 tuple 之規則密度增加因而易採用二元搜尋。當 conflicting 比例提昇，第一次二元搜尋較容易至較深的 tuple，而對這些區域來說因規則密度較高多是二元搜尋，因此搜尋結果容易

與 diagonal search 相似，但平均來說仍較佳。

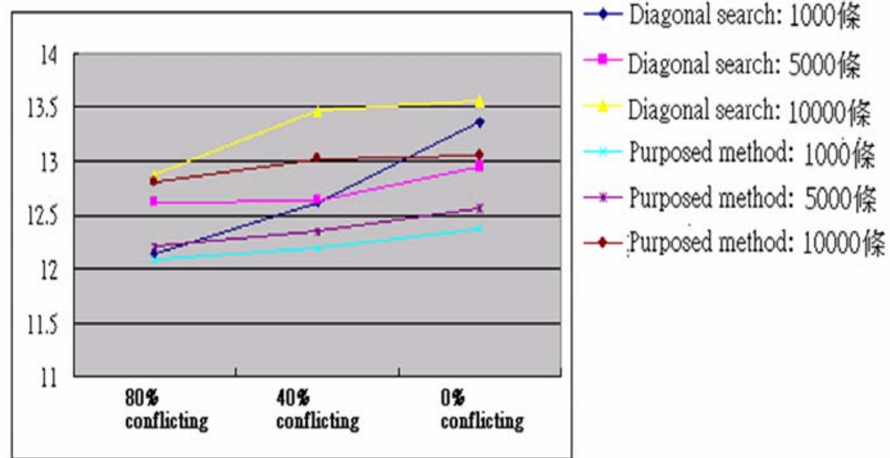


圖 5-6：在不同的規則總數以及 conflicting 狀態下對 tuple 搜尋之平均結果折線圖。

5.1.3 單方向之測試

模擬環境：圖 5-7 之右圖乃對 Diagonal tuple space 中的 diagonal tuple((0，0)~(32，32))的所有規則與 marker 增加 source information，使產生方向性，並且對可能產生 double-direction conflicting 的規則與 marker 增加 marker 於對角線上。最後針對搜尋次數，以及 marker 的總數量做比較：

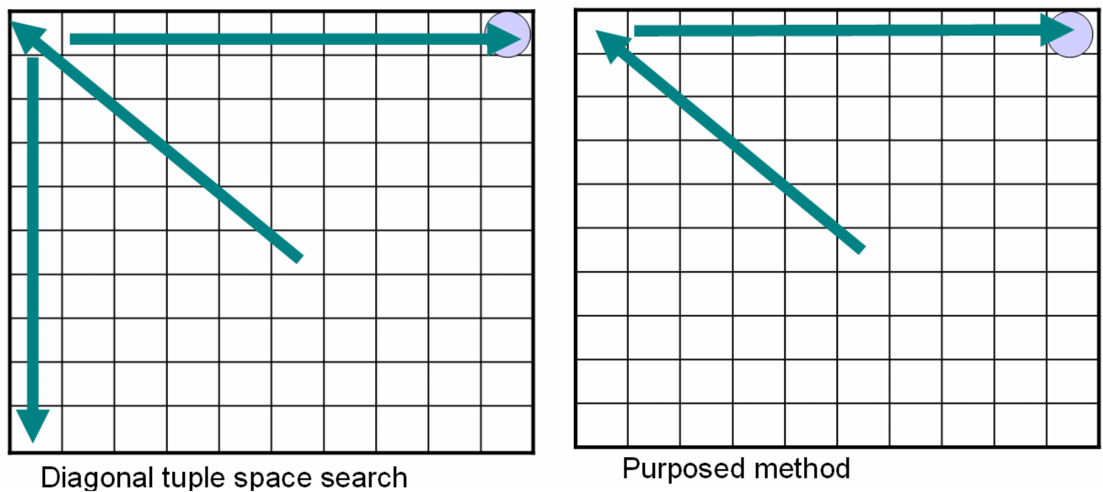


圖 5-7：演算法之差異性：採用單方向機制之影響。

圖 5-8 中，由於 conflicting 比例減少，因此 conflicting marker 數量亦隨之減

少，因此第一次二元搜尋所搜尋到的最深 diagonal tuple(d，d)會越接近(0，0)，對 Diagonal search 來說，二次及三次二元搜尋之範圍亦隨之增大，d~32，然而當 d<16 時，二次及三次二元搜尋之平均搜尋次數介於 4 至 5 之間，因此對整體搜尋影響並不很大。而所測試方法則由於儘需二次二元搜尋，因此整體表現而言較 Diagonal search 佳，由於第一次二元搜尋需六次的搜尋(最後一次儲存 Dim 之值)，因此整體來說搜尋次數較 Diagonal search 少約三次左右。另外也由於雙方向衝突之關係必須多增加 marker 至 diagonal Tuple 中，因此在圖 5-9 中的結果，這實驗裡所測試之方法會較 diagonal tuple space 多許多 marker

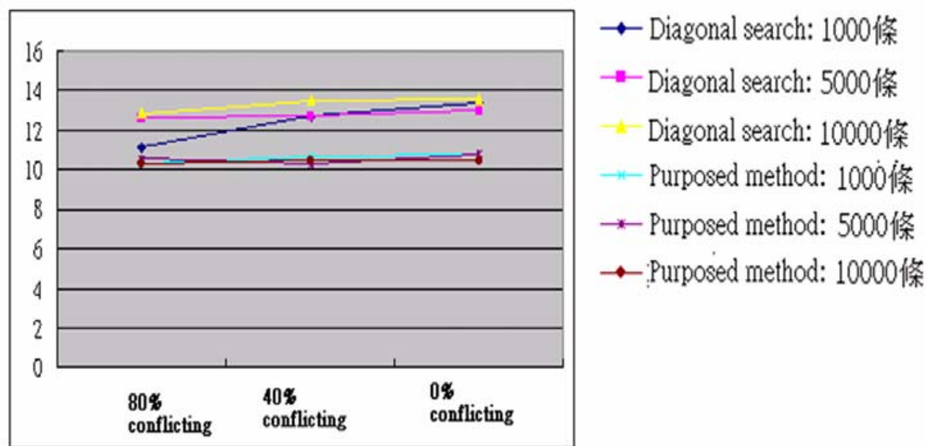


圖 5-8：在不同的規則總數以及 conflicting 狀態下對 tuple 搜尋之平均結果折線圖。

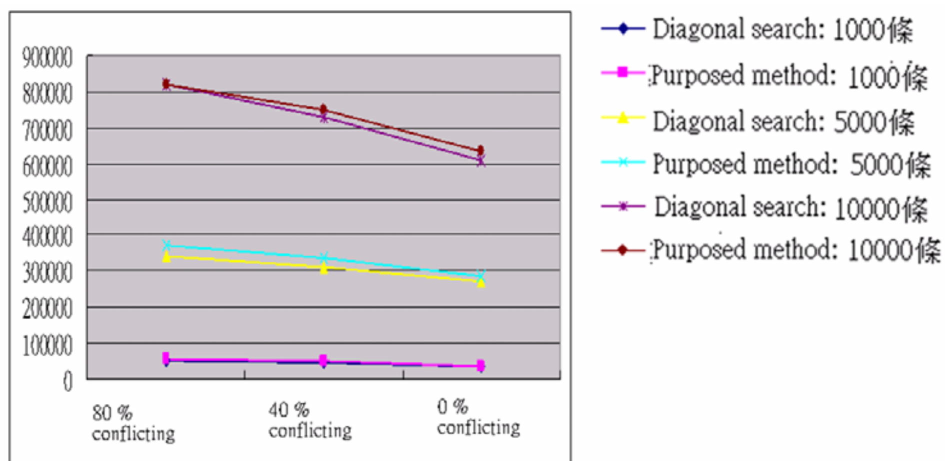


圖 5-9：在不同的規則總數以及 conflicting 狀態下的 marker 總數量之折線圖。

5.2 總合模擬

5.2.1 總合模擬之環境

如 2.2.1 小節所述，一般的真實網路環境下的規則分布都散佈於 tuple[16~32，16~32]以及 wildcard(tuple[0，0~32]及 tuple[0~32，0])之間，另一方面也必須考慮衝突對整個演算法的影響，傳統的 tuple space 搭配 Rectangle search 演算法中因無須考量 tuple 跟 IncompatibleTuple 間的影響，然而在 Diagonal tuple space search 以及我們所提出的 Advanced regional diagonal tuple space search 演算法卻會因為避免衝突的問題產生而須加進 Mirror rule 以及相對應的 marker，因此對此二個演算法來說，會產生衝突的 marker 越多，越會在空間以及時間造成不利的影響，而會產生衝突的 marker 的產生則是因為存在著會產生衝突的規則。因此為了比較 Rectangle search，Diagonal tuple space search 以及 Advanced regional diagonal tuple space search 這三個演算法在不同環境下的影響，我們選擇了以下的幾個因子來做控制變因並分析其結果：造成衝突規則的多寡以及位於 wildcard 區域的規則的比例，並以 1000 調規則來分析其結果。

5.2.2 模擬結果

Conflict rule number	Tuple Space	Diagonal Tuple Space	Proposed method
0%	17840 47.63	36912 13.435	9145 9.251
40%	17502 48.34	43534 12.667	9877 9.203
80%	16950 46.41	50595 11.182	10808 9.021

表 5-1：當有 0%的規則位於 Wildcard 區域時，各個演算法分別生成的 marker 總數量以及平均搜尋結果。

Conflict rule number	Tuple Space	Diagonal Tuple Space	Proposed method
0%	16429 52.20	53557 12.828	21231 9.820
40%	16355 49.60	69322 12.620	23327 9.716
80%	15941 45.33	76953 12.02	26540 9.542

表 5-2：當有 20%的規則位於 Wildcard 區域時，各個演算法分別生成的 marker 總數量以及平均搜尋結果。

Conflict rule number	Tuple Space	Diagonal Tuple Space	Proposed method
0%	14356 39.87	69842 13.022	40500 10.152
40%	14237 41.22	80572 12.855	39458 9.802
80%	13925 42.5	86352 12.618	37689 9.852

表 5-3：當有 50%的規則位於 Wildcard 區域時，各個演算法分別生成的 marker 總數量以及平均搜尋結果。

表 5-1，表 5-2 及表 5-3 分別為規則總數為 1000 且其中的 0%，20%，50%散

佈於 Wildcard 區域時的情況，另外又有固定的 40%的規則散佈於 tuple[16~32，16~32]之間，而 Conflict rule 的生成方式為由剩餘的規則之中隨機取一個，並隨機修改 Source prefix length，Destination prefix length，Source specified bits 的隨機長度的末幾 bit 的內容，Destination specified bits 的隨機長度的末幾 bit 的內容以及規則的優先權，以製造出容易產生衝突的環境，而測量方式又分為規則總數的 0%，40%以及 80%與剩餘的 100%，60%以及 20%的規則產生衝突。而透過實驗結果可以發現當衝突的規則數量增加時，由於 Rectangle search 並無必要針對衝突作額外的處理，因此當衝突規則增加時，對 marker 的生成並無顯著的負面影響，甚至能夠因為衝突規則與規則之間生成的 marker 有極大的重複性而不被生成，當規則越集中於 Wildcard 時能夠取消更多的 marker；相反地，對 Diagonal tuple space search 來說，儘管也能因為規則集中而將重複的 marker 消除，但因為針對衝突必須增設 mirror rule 以及相對應的 marker，反而使得 marker 的生成量大為增加，如在 tuple(0，32)的規則與 tuple(31，31)的某一 marker 產生衝突，亦等於說對 tuple(1，1)至 tuple(30，30)都存在著衝突的影響，因此所增加 marker 數量極為可觀；而我們所提出的方法雖然亦存在著相同的問題，但透過跳躍式的生成 marker 以及各區域間的 marker 不互相流通的方式，使得 marker 大量生成的問題能夠稍微減緩些，而透過跳躍生成的方式也使得在搜尋階段能夠避開一些無用的 tuple 進行搜尋，然而當規則衝突情形極為嚴重時，marker 所佔用的記憶體空間仍較 Rectangle search 嚴重，另外也由於雙方向衝突之問題亦會增加 marker，而當規則越分佈於 Wildcard 區域時越容易造成此一問題，因此因應這問題而生的 marker 便極為龐大。

而在平均每個封包對 tuple 的搜尋次數上，由於 rectangle search 至少必須搜尋一個維度長的 tuple 量，因此該演算法在搜尋的表現上是極為不理想的，然而衝突的規則增加對該演算法的影響卻不大，並且相較於 Diagonal tuple space search 以及我們所提出的演算法皆會隨著規則散佈於 Wildcard 區域的密集度增加而使搜尋效率降低的情形，Rectangle search 更能夠因為 Wildcard 區域的密集度增加而使

搜尋效率提升，其原因在於因為 Wildcard 的規則數量增多，而使得在 tuple space 的搜尋路徑幾乎能成為二直線而不會產生太多的稜角，然而整體搜尋狀況仍是相當不理想；而對 Diagonal tuple space search 來說，當一封包必須搜尋至 wildcard 區域時為該演算法之最糟狀態，因所需的三次二元搜尋皆是整個 Diagonal tuple space search 的最長路徑(16 長度,33 長度,33 長度),因此當規則越集中於 Wildcard 區域時，平均的搜尋次數亦會跟著提升，然而當單方向衝突之比例越上升時，月有助於整體搜尋效率提升，因透過 mirror rule 之引導能使封包再第一次二元搜尋不至於找到 tuple(0，0)；而在我們提出的演算法中，雖然在最糟情況下可能也需要二次的二元搜尋，然而第二次之搜尋除非在最糟情況下才會與二元搜尋之表現無異，而最糟情況在於對小區域內的規則之密度提昇。

5.3 複雜度的計算

在搜尋上，由於最初採用區域空間跳躍式搜尋，而區域空間之劃分因與二元搜尋失敗會經過的點相同，因此從區域跳躍至第一次二元搜尋需$\log w+1$，加一乃因為必須多對最深tuple之下依對角線tuple多判斷一次以記錄Dim之值，當開始跳躍至小區域進行link-list搜尋，最糟情形為不斷搜尋失敗，因為小區域建立方式也是採用以二元搜尋失敗時所經過的tuple為分隔，因此不斷失敗的情況下需搜尋$\log w$，因此整個搜尋需 $2\log w +1 = O(\log w)$。在儲存空間的需求上，由於大區域與小區域之劃分皆是採二元搜尋長度所建立，所以需 $2\log w$，若二二規則產生雙方向衝突，則最糟情形將生成$n^2w/4$，因此整個儲存空間需求為 $2n\log w+ n^2w/4$。

	Time complexity	Space complexity
Rectangle search	O(w)	O(nw)
Diagonal tuple space search	3log w =O(log w)	$O(n2^{w}\log w)$
Advanced regional diagonal tuple space search	2log w =O(log w)	$2n\log w + n^{2}w/4$ $=O(n\log w + n^{2}w)$

表 5-4：Rectangle search，Diagonal tuple space search 及 Advanced regional diagonal tuple space search 之複雜度比較。

第六章 結論與未來工作

在此研究中，我們將現實網路環境因素考量於Diagonal tuple space search演算法並加以改良該演算法對儲存空間的嚴重需求，加速封包的搜尋速度，透過一些想法的提出以及資料結構的改變，發展出我們所提的Advanced regional diagonal tuple space search演算法，並以分區搜尋來解決對角線tuple配置過多marker之問題；以鏈結搜尋加上動態決定演算法使得每一行(列)再度切割成許多小區段並根據演算法生成marker，使得規則密集之區段能以二元搜尋達到平均的效果，規則稀疏之區段能以線性搜尋以快速搜尋至合適的規則；最後透過對雙方項問題的處理，使得當找到最深的對角線tuple之後無需分別向右，向下進行二次確認，可減少一次二元搜尋。因此Advanced regional diagonal tuple space search能夠達到 $2\log w$的搜尋效果以及 $2n\log w+n^2w/4$ 的空間需求。

接著以模擬的網路環境來展現其成果，我們分別驗證出所提的三項改良皆有益於搜尋的進行。而從數據與結果中顯示 Advanced regional tuple space search 不僅將空間需求壓縮至一定程度，也藉由選擇較有利的局部演算法進行搜尋。然而跟其他舊有的演算法相比較之下，在某些狀況下仍須有著配置較大的空間來儲存 marker 的問題，以及雖然改善 Diagonal tuple space search 的搜尋效率但幅度並不大的情形：由於為了解決雙方向衝突必須增加額外的 marker，當規則分布於 Wildcard 區域之 tuple 數量越多時，所配置的 marker 也跟著增多，這也是值得未來繼續研究的方向。而 Advanced regional tuple space search 對 marker 生成方式不如 Rectangle search 及 Diagonal tuple space search 之單純，因此也造成更新亦變的複雜。由於模擬之網路環境預設為會對本論文造成最糟效果之模擬環境，未來也可透過實際環境之測試來評估表現效益，以滿足實際應用需求。

參考資料

[1] Chia-Ren Hsu, Chien Chen and Chun-Yuan Lin, "Fast Packet Classification Using Bit Compression", Department of Computer Information Science Natio n Chiao Tung University, Hsin Chu, Taiwan, 2004.

[2] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "Fas and Scalable L ayer Four Switching," Proceedings of ACM SIGCOMM, volume 28.

[3] P. Gupta and N. McKeown, "Algorithms for Packet Classification, "IEEE N etwork Special Issue, volume 15, number 2, pages 24-32, March/April 2001.

[4] P. Tsuchiya, "A Search Algorithm for Table Entries with Non-contiguous Wi ldcarding," unpublished report, Bellcore, 1992.

[5] V. Srinivasan, S. Suri, G. Varghese, and M. Waldvogel, "Fast and Scalable Layer four Switching," Proceedings of ACM Sigcomm, pages 203-14, Septe mber 1998.

[6] M. M. Buddhikot, S. Suri, and M. Waldvogel, "Space Decomposition Tech niques for Fast Layer-4 Switching," Proceedings of Protocols for High Speed Networks, August 1999.

[7] P. Gupta and N. McKeown, "Packet Classification using Hierarchical Intellig ent Cuttings," IEEE Micro, vol. 20, no. 1, pages 34-41, January/February 20 00.

[8] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification usin g multidimensional cutting," Proceedings of SIGCOMM, pages 213-224, Aug ust 2003.

[9] P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," Proc. Sigcomm, Computer Communication Review, pages 147-160, September 199 9.

[10] Kostas Pagiamtzis, "Kostas Pagiamtzis :: Introduction to Content-Addressabl e Memory (CAM)," http://www.eecg.toronto.edu/~pagiamt/cam/camintro.html, 2005.

[11] T.V. Lakshman and D. Stiliadis, "High-Speed Policy-based Packet Forwardi ng Using Efficient Multi-dimensional Range Matching," Proceedings of ACM Sigcomm, pages 191-202, September 1998.

[12] F. Baboescu, G. Varghese, “Scalable Packet Classification,” Proceedings of ACM Sigcomm, pages 199-210, August 2001.

[13] Ya-Wen Lin, “Adaptive Packet Classification Using Kd-Tree Based Tuple,” National Chiao Tung University, 2002.

[14] V. Srinivasan, S. Suri, and G. Varghese, “Packet Classification using Tuple Space Search,” Proceedings of ACM Sigcomm, pages 135-46, September 19 99.

[15] Priyank Warkhede, Subhash Suri and George Varghese, “Fast Packet Classif ication for Two-Dimensional Conflict-Free Filters,” Proceedings of 20th IEEE Infocom, vol. 3 (2001) 1434-1443.

[16] Mikko Alutoin and Pertti Raatikainen, “Diagonal Tuple Space Search in Tw o Dimensions”, Globecom,2005.

[17] Pi-Chung Wang, Chia-Tai Chan, Wei-Chun Tseng and Shuo-Cheng Hu, “Op timal Tuple Reduction for Fast Two-Dimension Packet Classification”, Interna tional Computer symposium (ICS’02), National Dong Hwa University, Taiwa n, December 2002.

[18] Pi-Chung Wang, Chia-Tai Chan, Wei-Chun Tseng and Shuo-Cheng Hu, “Op timal Tuple for Fast Two-Dimension Packet Classification,” Proceedings of I CS’03, B2-A-1~B2-A-7, Dec. 2002.