

以虛擬鍊結改善 RRT-based 漸進式街圖法的路徑品質

許書璋 李蔡彥

國立政治大學 資訊科學所

E-mail: {g9204, li}@cs.nccu.edu.tw

摘要

路徑計畫是機器人自動化中的重要課題，而 Rapidly-exploring Random Tree (RRT) 是近年來十分受歡迎的方法。Reconfigurable Random Forest (RRF) 是以 RRT 為基礎所設計的漸進式街圖運動計畫器。RRF 可以透過學習的方式，經由每次的查詢，逐漸擴展街圖，並且具有街圖管理的機制。但由於 RRF 街圖是由 RRTs 所組成，而 RRTs 在空間中的成長型態與分佈，可能造成路徑查詢時產生品質較差的路徑。在本篇論文中我們提出以虛擬鍊結的概念來改善此問題。當被計畫物體的起始與目標組態都連接上街圖後，虛擬鍊結可以將 RRT 的樹狀結構(Tree)轉換成圖形結構(Graph)，提供做最佳路徑搜尋的機會，而不只是沿著樹狀結構取得路徑。實驗證明以虛擬鍊結的方式可以提升 RRF 產生最短路徑的機率，改善 RRF 的路徑品質。

關鍵詞：運動計畫、路徑品質、RRT、隨機式街圖。

Abstract

Path planning is an important issue in robot automation. Rapidly-exploring Random Tree (RRT) is one of the popular methods for this problem in recent years. Reconfigurable Random Forest (RRF), which extends RRT, is an incremental roadmap motion planner that can learn incrementally on every planning query and manage the learned roadmap effectively. Due to the structure of the RRT roadmap, the path generated by RRF may not be the shortest one. In this paper, we propose to use the concept of "Virtual Link" to improve the situation. When the initial and goal configurations are connected to the roadmap, the virtual links are restored in the roadmap. We can then transform the forest structure into a graph, which can be used to search for a shortest path. Our experiments show that the virtual link mechanism can improve the quality of the generated paths.

Keywords: Motion Planning, Path Quality, RRT, PRM

1. 概論

運動計畫的技術起源於機器人學的領域，主要目的是替機器人或機器手臂，從我們指定的起始狀態到目標狀態，規劃出一條不會與環境中障礙物碰撞的路徑，使得機器人或機器手臂可以順利的移動。除了機器人自動化之外，近年來運動計畫所發展出來的技術也被廣泛的應用到其他領域上，如電腦動畫、電腦輔助設計、虛擬實境、3D 人機介面操作輔助[1][2][3]等領域上。

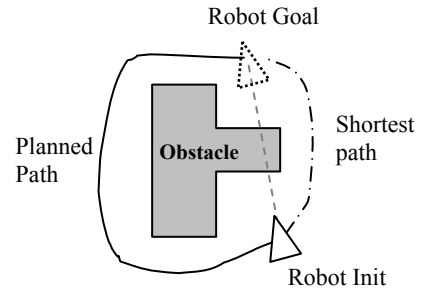


圖 1. 利用 RRF 搜尋路徑時所發生繞遠路的情況

在運動計畫的技術當中，我們稱要被規劃的物體為 robot，而 robot 所處的工作環境我們稱工作空間 (workspace)。在工作空間中，robot 的狀態是用組態 (configuration) 來描述，全部組態的集合稱 Configuration Space，記做 C 。在 C 之中，和障礙物 (obstacle) 沒有碰撞的子空間，稱為自由空間，記做 C_{free} 。舉例來說， X 是一個能夠在 2 維平面上移動的 robot，所以 X 的工作空間為一個 2 維空間。而 X 的組態，要看 X 在這個工作空間上的自由度而定。假設 X 是單一個剛體，可以移動、轉動，則描述 X 的組態形式為 $X=(x, y, \theta)$ ，我們說 X 組態空間的維度是 3 維。一個 robot 組態空間的維度是會隨著 robot 所在的工作空間及 robot 本身構造而改變。例如 X 的工作空間如果是 3 維空間， X 的組態空間則為 6 維 $X=(x, y, z, \alpha, \beta, \gamma)$ ，其中 α, β, γ 是對三個軸的旋轉角度。

隨機式街圖 (Probabilistic Roadmap Methods, PRM) 是近年來解決運動計畫問題十分受歡迎的方法之一。這個方法會預先在組態空間中，以隨機取樣的方式建構出近似自由空間的街圖連結，並將此街圖儲存起來；之後當需要查詢時，只要把起始組態和目標組態連上街圖，就可以迅速產生一條路徑。而 Reconfigurable Random Forest (RRF)[4] 是一種利用快速擴展隨機樹 (Rapidly-exploring Random Tree, RRT)[5] 這種資料結構，再搭配 RRT-Connect 演算法[6]，所改良出來的漸進式隨機街圖演算法。RRF 的本質上是由許多 RRTs 所構成的 forest。RRF 的特性在於它可以累積每一次的運動計畫查詢中所獲得之資訊，慢慢的擴大場景中街圖的規模，以便下次再做查詢時利用。此外，它還具備街圖管理的機制；當街圖成長太龐大時，可將街圖做適當的修剪，以保持查詢的效率。

除了使用在機器人的路徑計畫外，我們也成功地將 RRF 用在改善人機界面的應用上。而在這些應用上，我們發現利用 RRF 所搜尋到的路徑，有繞遠路的現象(圖

1)。這樣的現象如果是發生在機器人相關的應用上，會導致移動效率下降；若是發生在人機介面輔助的應用上，則是導致使用者感覺不自然。這個問題的來源，是由於 RRF 街圖本身是以一群 RRT 所組成，又因為我們在做路徑查詢時，會依靠到這些已經存在的 RRTs，因此我們找到的路徑，就受限於這些 RRTs 在空間中成長型態的限制。這也是 RRT-based 運動計畫器在所產生的路徑品質上的限制。

為了解決這個問題，我們提出了使用**虛擬鍊結 (Virtual Link)**的方式，來改善 RRF 所找出來路徑的品質，讓 RRF 盡可能找出最短的路徑。虛擬鍊結的概念是當 RRF 在成長時，我們讓 RRTs 保持 Tree 的資料結構，但是在搜尋路徑之前，我們把 RRTs 轉成 Graph 的型態，提供最佳路徑搜尋的機會。虛擬鍊結是隨著 RRT 本身的成長而逐漸加入的。在加入虛擬鍊結後，我們仍然可以保留住 RRF 漸進式成長的特性，又可以解決 RRF 可能產生較差的路徑的問題。

2. 相關研究

關於運動計畫的定義與方法的研究，在文獻[7]中有較為完整的記載。一般來說，三種常見的運動計畫解決方法分別為：cell decomposition、potential field 和 roadmap。

Cell decomposition 的方式，是先將自由空間劃分成小區域(cell)，接著在彼此相鄰的小區域找到通道並連結起來，組成類似 graph 的結構，然後在上面搜尋路徑[8]。

Potential field 的方式，則是在 robot 所處的工作空間建立人工位能場，當成路徑搜尋時的依據(heuristic)，利用 robot 有從由高位能往低位能移動的趨勢，來找出路徑[9]。利用 potential field 所設計的運動計畫器大多是具有完整性的(completeness)，而且在一些具有狹道(narrow passage)的環境中，potential field 比其他方法容易找到路徑。Potential field 的缺點是會有遇到局部最小值(local minimum)的問題以及在較高維度的工作空間或組態空間較不適用。

以 Roadmap 的方式來解決運動計畫的問題可以分為兩個階段(phase)：前處理階段(preprocessing phase)及查詢階段(query phase)。在前處理階段先建立自由空間中的街圖連結，以供查詢階段使用。在查詢階段時，如果能夠將起始組態和目標組態連接上現有的街圖，就可以得到一條路徑。街圖產生的方法有很多種，其中隨機式街圖(Probabilistic Roadmap Methods, PRM)[10]是在自由空間中，以隨機取樣的方式來建構街圖的一種產生方式，而 RRT[5][6]則為這種隨機式街圖產生的一種方式。一般而言，具高自由度的 robot，由於其搜尋空間為高維的組態空間，因此我們往往會採用隨機取樣的方式建立街圖。主要原因在於高維的環境中要建立一分完整的街圖，需要花費大量的時間，而隨機取樣是比較可行的方法。

3. 以 RRT 為基礎之漸進式街圖-RRF

這一節主要是概略說明 RRF 的建立與運作，以及

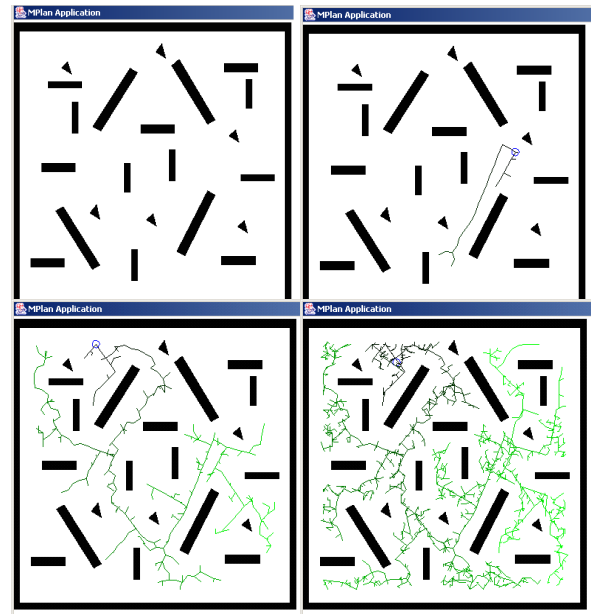


圖 2. RRF 街圖成長過程 (3D 自由空間投影到 2D)。左上、右上、左下、右下四個圖分別是：query 0 次、query 1 次、query 5 次及 query 50 次時的街圖分佈。其中顏色越接近黑色的表示該節點在 RRT 中的深度(depth)越淺，越接近綠色表示節點深度越深。

為什麼會發生類似繞路的問題，造成路徑品質下降。有關 RRF 的細節及完整演算法可以參閱[4]。

3.1 RRF 的建構

RRF 是利用 RRT 資料結構以及 RRT-Connected 演算法的觀念改良而來的漸進式街圖。基本上 RRF 仍是一群 RRTs 所構成的 forest，只是當中 RRTs 是漸進式加入的，並非一次就構成。每一次查詢的動作，RRF 會把起始組態 Init 和目標組態 Goal 分別當成 root 長出兩棵新的 RRT，這兩棵 RRTs 在擴張的過程中，都會嘗試著和 forest 中其他的 RRTs 合併，一直重複進行擴張與嘗試合併，直到包含 Init 和 Goal 的兩棵 RRTs 合併在一起，即可搜尋得到一條從 Init 到 Goal 的路徑。

3.2 RRF 的精簡

從圖 2 中可以看出 RRF 會隨著 query 次數的增加而慢慢成長。一般而言，街圖的節點數越多，表示我們對自由空間越能夠掌握，在查詢的時候，不僅增加 Init 和 Goal 連結上的機會，也大大的節省了探索的時間。但是當街圖複雜到一定程度時，也就是節點數成長到一個程度後，系統必須花費大量的空間來儲存整份街圖，而且在路徑查詢過程中，也因為要處理的節點數變多，反而會影響查詢的效率。所以必須要有精簡街圖的機制，來控制街圖的成長。RRF 提供了兩個精簡街圖的程序：垂直合併(V_MERGE)與水平合併(H_MERGE)。

當我們要修剪一棵 RRT 時，會分別就垂直與水平兩個方面來檢驗 RRT 中的節點是否太過密集，先是以 pre-order 方式 traverse 該顆 RRT 並進行垂直合併，接著

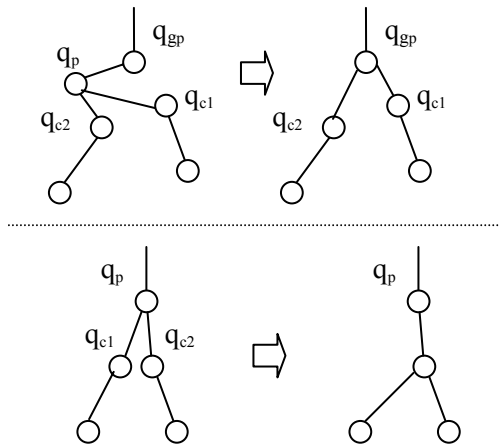


圖 3. 垂直合併(上圖)與水平合併(下圖)

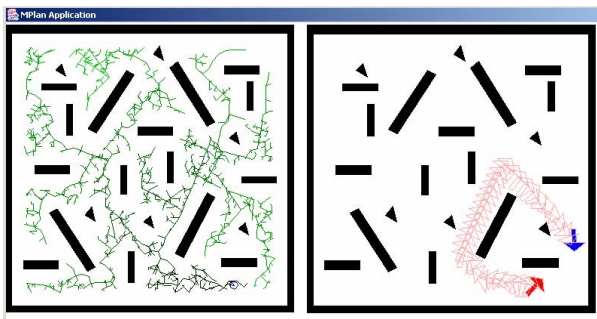


圖 4. 左邊為街圖投影, 右邊為路徑, 紅色箭頭為 Init, 藍色為 Goal。由於街圖中 RRT 生長情況, 使得繞遠路的情況發生。

再以 post-order 的方式 traverse 該類 RRT 並進行水平合併。我們以圖 3 上圖說明垂直合併原理。當我們檢驗某個非 root 或 leaf 的節點時(q_p), 如果發現其任一子節點(q_{ci})與 q_p 之父節點(q_{gp})的距離小於我們所設定的常數 MinVMergeD 時, 且 q_{ci} 與 q_{gp} 可以直接連上, 我們就把該子節點 q_{ci} 轉移到 q_{gp} 上。如果 q_p 所有的子節點全部轉移了, 那 q_p 就變成多餘的節點, 會被刪除掉。

水平合併原理類似(圖 3 下圖), 只是檢驗的是兄弟節點之間的距離, 如果 q_{c1} 與 q_{c2} 的距離小於我們設定的常數 MinHMergeD 時, 且 q_{c1} 全部的子節點和 q_{c2} 都可以直接連上, 那我們就把 q_{c1} 的子節點全部轉移到 q_{c2} 底下, 然後刪除 q_{c1} 。

過去的研究數據顯示經過這樣的精簡程序後, RRF 的街圖仍具有代表性[4]。

3.3 RRF 路徑的先天限制

雖然 RRT 有快速拓展的優點, 但是由於其樹狀結構及隨機生長的特性, 使得它在運動路徑規劃的應用上, 有一些先天上的限制。在 RRT-Connect 演算法中, 當 Init 與 Goal 連結成同一棵 RRT 之後, 所能取得的路徑與該類 RRT 生長的過程有很大的關係, 也和 merge 發生的節點所在區域有很大的關係。這使得我們只能夠保證能找出一條路徑, 但不一定是最短路徑。而 RRF

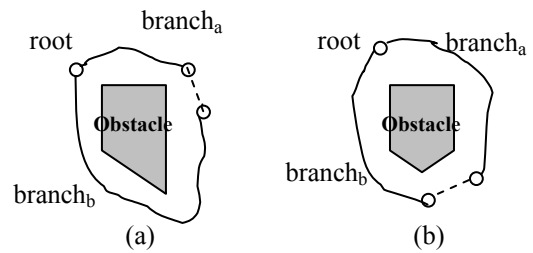


圖 5. 建立 Virtual Link 的地點 (a). 兩個長度不平衡的分支末端。(b). 兩個深度都達到一定程度的分支末端。

是由 RRT 組成的, 加上 query 的程序, 會使得 forest 中的 RRTs 有 merge 的動作, 因此 RRT 的形狀會一直改變。當我們把這種由 RRTs 所構成的街圖拿來重用時, 發現繞遠路的這種情況會變的比較頻繁。圖 4 便是一個繞遠路的例子。

4. 以虛擬鍊結改善路徑品質

在了解 RRF 運作的過程與分析了之前提過的繞遠路的問題之後, 我們提出使用虛擬鍊結(Virtual Link)這種機制來提升 RRF 在查詢時所產生路徑的品質。虛擬鍊結可以改善 RRT 的樹狀結構, 並且不會破壞原本 RRT 漸進式建立街圖的特性。我們目前是以路徑的長短當成品質的考量。路徑長度越短, 我們就認定其品質越好。

4.1 虛擬鍊結的性質

我們所使用的虛擬鍊結是一種 Tree 內部之間互相連結的 Link。它不像一般 Tree 中的 Link 是垂直連結 child 與 parent, 而是橫向的連結不同分支上的 node。因為有了這些連結, Tree 的結構, 就等於是被轉換成 Graph 的結構。虛擬鍊結的目的, 是為了放寬原本 RRT 中, Init 和 Goal 連起來後只能沿著 tree 生長的形狀 trace 出一條 path 的限制, 讓 Init 和 Goal 連上後, 可以在一個由 Tree 和虛擬鍊結組成的 Graph 上面做路徑搜尋的動作, 而不是單純的 traceback 的動作。

4.2 虛擬鍊結的產生

虛擬鍊結會隨著 RRT 在其擴張的過程中逐漸產生。每次 RRT Extend 出一個新的節點 (稱 q_{new}) 後, 我們取出 q_{new} 周圍 $10*10$ (workspace) 範圍內的所有節點, 當成可能與 q_{new} 互相透過虛擬鍊結連結的節點候選人集合。如果 q_{new} 和任一節點候選人之間滿足了下面兩個其中任一個關係, 我們就將這兩個節點之間放上一個虛擬鍊結:

1. q_{new} 與節點候選人的深度(depth)相差 K 以上。
2. q_{new} 與節點候選人來自不同的分支(branch), 而這兩個分支交會的節點和兩者的深度相差都大於 K 。(K 在目前的實做中為 15)

第一個條件的目的是為了讓 unbalance branch 之間建立連結, 第二個條件的目的是讓兩個來自不同方向 branch 有機會建立連結。圖 5 為兩個條件的示意圖。

這部分在實做上, 只需把原本 node 的資料結構加上一個額外的 List, 記錄該 node 與哪些 node 之間有虛

```

EXTEND*(T, Y_rand)
1. X_near ← Nearest_Neighbor(T, Y_rand);
2. X_new ← New_Configuration(X_near, Y_rand);
3. if (X_new is NOT Null)
4.   X_new.add_VirtualLink(
       X_new.search_VirtualLink());
5.   T.add_node(X_new);
6.   if (X_new = Y_rand)
7.     return Reached;
8.   else
9.     return Advanced;
10. return Trapped;
    
```

圖 6. 修改後的 EXTEND* 演算法

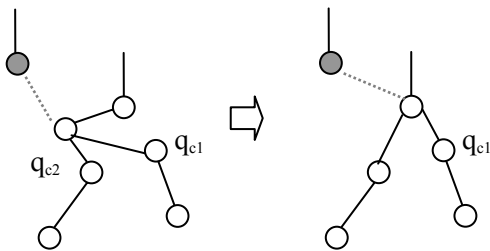


圖 7. 垂直合併時 Virtual Link 的轉移。q_p 全部的子節點都被垂直合併到 q_p 的父節點 q_{gp} 之後，q_p 的虛擬鍊結也要轉移到 q_{gp} 上。

擬鍊結的存在，但是對 RRT 在擴張的過程是沒有影響的。這種作法讓 RRT 仍可以漸進式的方式擴張。修改後之 RRT 擴張(Extend*)演算法如圖 6 所示。

4.3 虛擬鍊結的檢查

在虛擬鍊結的碰撞偵測部分，我們採用了 Lazy PRM 的精神[11]：在虛擬鍊結加入時，我們沒有立即檢查虛擬鍊結兩端點之間的內插組態是否會碰撞到環境中障礙物。相反的，而是等到 Init 與 Goal 連上之後，開始做 path search 的時候才去做碰撞偵測。這種 Lazy 的碰撞偵測檢查可以有效的提高 RRT 在建立街圖時的效率。

4.4 路徑搜尋

當 Init 與 Goal 連上之後，這兩顆 Tree 會被 Merge 成一棵 Tree。在進行搜尋之前，我們需要把 Tree 的結構變成 Graph 的結構。在原本的 Tree 結構上，我們將每個節點的 parent、children、以及虛擬鍊結所連接的節點，皆視為該節點的 adjacent node，然後做出一個 adjacency List。這個 adjacency List 就是我們 search 時需要的 Graph。然後我們用 Dijkstra's shortest path 演算法找出 Init 到 Goal 的最短路徑。當路徑找出來之後，需要對路徑做碰撞偵測的檢查。我們會檢查是否有那些段落是由虛擬鍊結構成的，因為只有這些部分是有可能會產生碰撞的，接著對這些部分做碰撞偵測的檢查。如果都沒有碰撞就傳回這條路徑，否則就把發生碰撞的虛擬鍊結刪除，重新搜尋一次。

```

RRF*_Planner(X_init, X_goal)
1. T_init.init(X_init); T_goal.init(X_goal);
2. Forest.add(T_init); Forest.add(T_goal);
3. Merge_RRTs(T_goal, T_goal.X_root);
4. if (Merge_RRTs(T_init, T_init.X_root))
5.   if (T_init.tree_id=T_goal.tree_id)
6.     L=buildAdjacencyList(T_init);
7.     return ShortestPathSearch(L, T_init, T_goal);
8. for k=1 to K do
9.   Yrand ← Random_Configuration();
10.  if (Extend(Tinit, Yrand) ≠ Trapped)
11.    if (Merge_RRTs(Tinit, Tinit.Xnew))
12.      if (Tinit.tree_id=Tgoal.tree_id)
13.        L=buildAdjacencyList(T_init);
14.        return ShortestPathSearch(L,
            T_init, T_goal);
15.  Swap(T_init, T_goal);
16. return Failure;
    
```

圖 8. 修改後的 RRF* Planner

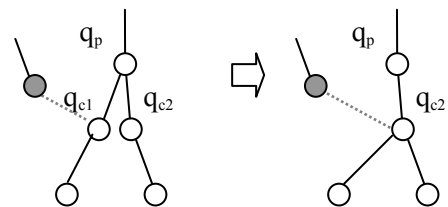


圖 9. 水平合併時 Virtual Link 的轉移。q_{c1} 被水平合併到 q_{c2} 之後，q_{c1} 的 Virtual Link 也需要轉移到 q_{c2} 上。

在這種機制下，我們可以確定傳回的路徑不會和障礙物發生碰撞，而且路徑的長度一定會小於或等於單純使用 RRF 所找到路徑的長度。根據這個方式所修改後的 RRF* 路徑搜尋演算法如圖 8 所示。

4.5 虛擬鍊結的精簡

虛擬鍊結是隨著 RRF 成長而加入的，當 RRF 成長至過於龐大的程度時，虛擬鍊結的數量也會隨之增多。基於這個原因，我們修改 RRF 原先的精簡街圖程序，並希望虛擬鍊結也能同時被夠被精簡。在精簡過程中，有節點必須被刪除的時候，也就是虛擬鍊結需要被調整修剪的時候。在垂直合併的精簡過程中(如圖 7)，假設 q_p 的全部子節點，q_{c1} 及 q_{c2} 都可以被轉移到 q_p 的 parent q_{gp} 上，這時候 q_p 就變成多餘的節點而被刪除掉。如果 q_p 上有虛擬鍊結，我們就將這些虛擬鍊結也轉移到 q_{gp} 上。而在水平合併精簡過程中(如圖 9)，假設 q_{c1} 為即將被 merge 的節點，它全部的 children 都會被轉移到 q_{c1} 的兄弟節點(sibling)q_{c2} 上面去。如果 q_{c1} 上有虛擬鍊結，我們也把這些虛擬鍊結轉移到 q_{c2} 的節點上；此轉移的目的即在保留虛擬鍊結。

在轉移的過程我們加入了一個限制，如果一個節點上已經有 M 個虛擬鍊結了(目前實做上 M 為 1)，但如果還有其他虛擬鍊結要轉移到該節點，就直接捨棄這些 Link 而不再轉移到該節點身上。我們觀察發現，一些關鍵的地方(如圖 5)，虛擬鍊結數量會特別多，但是這些

表 1. 虛擬鍊結花費時間與產生的數量。加入虛擬鍊結多花的時間為原本的 22.5%，[建立鍊結時間/(總時間-建立鍊結時間)]

計畫次數	總點數(個)	時間(ms)		虛擬鍊結數量(個)	
		總時間	建立鍊結	精簡前	精簡後
5	336	94	16	0	0
10	657	171	31	2	2
50	1505	500	109	39	18
100	1826	953	203	80	27
200	2387	1701	358	158	41
500	3575	4457	438	359	60

表 2. 路徑品質改善情況統計

隨機運動計畫測試範圍	路徑改善的次數	
	在第 N-1 次無精簡	在第 N-1 次進行精簡
第 51-70 次 (N=51)	10	5
第 181-200 次(N=181)	9	8

Link 大多數連結的是同一組 branch。所以在精簡的時候，我們假定會被轉移到某個點的虛擬鍊結應該都是連結到相同 branch 上，因此只需要保留下 M 個，就可以具有代表性，剩下的 Link 就可以捨棄了，達到精簡的目的。

5. 實驗結果

實驗所用的路徑計畫器是以[4]中的路徑計畫器為基礎加入第四小節所提的虛擬鍊結概念作修改。實驗所採用的工作空間為 2 維，robot 在工作空間有三個自由度(x, y, θ)，而組態空間的大小為 128*128*100。所採用的場景為圖 2 中的場景。實驗採用的電腦 CPU 為 AMD Athlon 1.2G，記憶體為 512MB。

首先我們針對加入虛擬鍊結後所需要負擔的運算時間做了一些評估。我們以隨機產生 robot 的 Init 與 Goal 的方式，分別連續做了不同次數的運動計畫的查詢，觀察加入虛擬鍊結前後所花費的時間。另外我們也觀察了虛擬鍊結在精簡前後的數量。

表 1 的總時間為隨機計畫 N 次中，在街圖建立過程所花的總時間，包含街圖中 RRT 的成長、合併、以及虛擬鍊結產生的時間。從數據我們可以估算出，新的 RRT 需要負擔比原本多約 22.5% 的時間來建立虛擬鍊結。而從精簡前後的虛擬鍊結數量比例，可以發現新的精簡演算法能有效的控制虛擬鍊結數量。

接下來我們針對路徑品質是否真的有改善的部分做了一些初步的實驗。我們去觀察隨機計畫 N 到 N+19 次之間，有哪幾次在最後搜尋到的路徑比沒有使用虛擬鍊結機制所產生的路徑來的好，路徑品質好壞評估是以路徑長短作衡量。圖 10 是一個所找到的路徑品質比原本好的例子。我們還比較了如果在第 N-1 次時，進行一次街圖的精簡(包含虛擬鍊結的精簡)，看看是否會產生

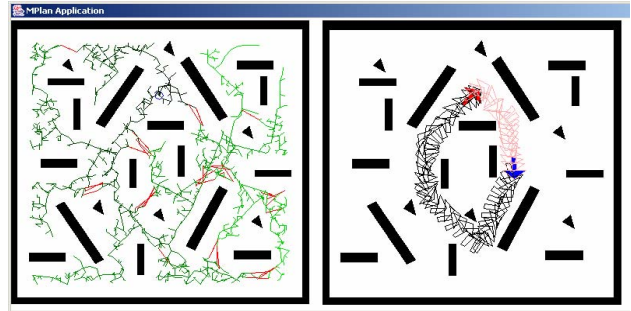


圖 10. 左邊紅色線段為 RRT 中的虛擬鍊結，右邊淡紅色路徑為改善後，黑色路徑為改善之前。

影響。N 我們分別取了 51 與 181 兩組下去做實驗。表 2 是統計的結果。

從表 2 中，可以估算出沒有進行精簡改善的比例為 47.5%，有進行精簡改善的比例為 32.5%。再從個案中去觀察，發現 51-70 這一組，有無精簡對品質改善比例的影響較大；不過 181-200 這組，影響則不大。進行精簡後改善的比例會降低，目前推測是因為精簡過程中，虛擬鍊結的一端會被連到其他節點身上，有可能造成虛擬鍊結長度過長，造成整段鍊結上和障礙物產生碰撞的機率提高，也降低了透過虛擬鍊結找到更好的路徑的機率。由實驗結果看來虛擬鍊結確實可以提高找到比原本更好的路徑的機率。

6. 結論

RRT-based 漸進式街圖的特性，在於它能保留查詢結果以供日後查詢時能重新利用，而且它提供了街圖管理的機制。不過其產生的路徑，會受限於街圖中 RRT 的樹狀結構以及隨機生長所形成的形狀，可能會造成路徑品質不佳的情形發生。我們以虛擬鍊結的概念來改善此現象。我們在 RRT 擴張的過程中，在一些分支末端嘗試加入虛擬鍊結，然後當 robot 的 Init 與 Goal 連上後，透過虛擬連結將 RRT 的樹狀結構轉換成 Graph，進行最短路徑的搜尋。整體來看，在建立街圖過程中，需要額外付出約 22.5% 的時間建立虛擬鍊結，而虛擬鍊結帶來的好處是讓我們有平均約有 47.5% 的機率可以搜尋出品質更佳的路徑。

7. 致謝

此研究在國科會 NSC 92-2213-E-004-001 計畫的支助下完成，特此致謝。

8. 參考文獻

- [1] Y. Koga, K. Kondo, J.J. Kuffner, and J.C. Latombe, "Planning motions with intentions," *SIGGRAPH '94*, pp.395-408, 1994.
- [2] H. Chang, and T.Y. Li, "Assembly Maintainability Study With Motion Planning," *IEEE International Conf. Robotics and Automation*, pp.1012-1019, 1995.
- [3] T.Y. Li, and H.K. Ting, "An Intelligent User Interface with Motion Planning for 3D Navigation," *IEEE Virtual Reality Conf.*, pp177-184, 2000
- [4] T.Y. Li and Y.C. Shie, "An Incremental Learning Ap-

- proach to Motion Planning with Roadmap Management," *IEEE International Conf. on Robotics and Automation*, 2002.
- [5] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," TR 98-11, Computer Science Dept., Iowa State University, 1998.
- [6] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," *In Proc. IEEE International Conf. on Robotics and Automation*, pp. 995-1001, 2000.
- [7] J. Latombe, *Robot Motion Planning*, Kluwer, Boston, MA, 1991.
- [8] R. Brooks and T. Lazano-Perez, "A Subdivision Algorithm in Configuration Space for Find-path with Rotation," *IEEE Transaction on System*, pp.799-806, 1983.
- [9] J. Barraquand, B. Langois, and J. C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Robotics and Automation*, 22(2):224-241, 1992.
- [10] L. Kavraki, P.Svestka, J. Latombe, and M. Overmars, "Probabilistic Roadmaps for Fast Path Planning in High-Dimensional Configuration Spaces," *IEEE Transaction on Robotics and Automation*, 12:566-580, 1996.
- [11] R. Bohlin and L.E. Kavraki, "Path planning using lazy PRM," *IEEE Int. Conf. on Robotics & Automation*, pp .521-528, 2000.