一個用以解決工作匹配與排程問題之快速螞蟻系統優化法

江傳文
國立高雄第一科技大學電腦與通訊工程系
ccw@ccms.nkfust.edu.tw

摘要

本文中，我們將以快速螞蟻系統 (Fast Ant System, FANT) **的設計方法為基礎，提出一個名為** FANT-TMS (Fast Ant System for Task Matching and Scheduling) **且應用在工作匹配與排程問題求解之演算方法。此一方法旨在有效地決定各個工作在執行時期的順序，並將工作分派給叢集運算環境中合適的處理單元** (processing element, PE) **執行。** FANT-TMS **採用一種間接解的表示方法用以降低演算方法在執行時期的複雜度。相較於以基因演算法為基礎的工作匹配與排程技術，實驗結果顯示，本文所提出之** FANT-TMS **演算法確實優於傳統的方法。此外，為了強化演算法的執行效能，我們亦提出一種嶄新的區域搜尋方法。實驗結果亦呈現出此一設計方法的優異性。**

關鍵詞：個人電腦叢集、工作匹配與排程、螞蟻系統、區域搜尋

Abstract

In this paper, an efficient algorithm based on the Fast Ant System (FANT) is proposed. This algorithm, namely FANT-TMS, concentrates on properly sequencing the execution of the tasks and allocating the tasks to the processing elements of the cluster system. FANT-TMS is different from the previously proposed approaches in twofold. First, it employs an indirect representation scheme to represent a solution of the original problem. Second, it couples a local search procedure with a mechanism to improve the performance. FANT-TMS is evaluated through a comparison with the genetic algorithm (GA) based scheduling technique in terms of overall execution time of the parallel program. Simulation results show the efficiency and effectiveness of the proposed algorithm. With regarded to the performance of the proposed local search algorithm, these experimental results also demonstrate that significant improvement over existing methods (such as FASTEST and TASK) can be obtained.

Keywords: PC Clusters; Task Matching and Scheduling; Ant System; Local Search;

1. 緒論

隨著微處理器製造技術的快速發展以及網際網路的普及，個人電腦叢集 (PC Cluster) **在各個領域的應用已經引起廣泛的重視。個人電腦叢集主要是藉由高速網路裝置連結為數眾多的處理單元** (processing element, PE) **進而構成一個低成本且支援平行處理的高效能運算平台。由於在此一平台上所執行的平行應用程式係由許多的工作所組成，因此為了讓此一平台上的計算資源可以被充分利用，一個有效的工作匹配與排程演算方法無疑地成為高效能運算得以實現的重要技術。**

工作匹配與排程旨在探討如何在不違反工作執行先後順序限制條件的情況下，將組成平行應用程式的工作分派給適當的處理單元執行，進而使得該平行應用程式能在最短的時間之內完成。一般而言，一個工作匹配與排程演算方法必須處理以下的兩個議題：

- **決定平行應用程式中所有工作在執行時的優先順序，以及**
- **將所有工作依其執行優先順序分派給適當的處理單元執行。**

基於以上的問題特性，除了少數的特殊情況之外 [10]**，工作匹配與排程問題的一般型式已被證實為一** NP-**完備問題** [11]**。由於求解此一問題所需耗用之時間會隨著問題規模的增加而呈現指數方式的爆炸性成長，因此許多學者一直以來皆致力於發展各種近似解演算法** (approximation algorithm)**，試圖在可接受的合理時間內求得問題的近似解** [4-7]**。**

求解工作匹配與排程問題的近似解演算法可被概分以下兩種類型：以優先權為基礎的串列演算方法 (priority-based list scheduling) [4] **與泛用啟發式演算方法** (metaheuristics) [5-7]**。串列演算方法** (list scheduling) **在處理工作匹配與排程問題時首先會分析每一個工作的特性** (**例如：工作量**) **以及工作間彼此的關聯** (**例如：工作執行先後順序限制**)**，然後決定所有工作的優先權。之後，以特定的準則** (**例如：工作最早開始時間或最早完成時間**) **為依據，將適當的處理單元分派給具有最高優先權且尚未被處理的工作。各演算法依所使用的準則不同而有所不同。串列演算方法的主要特色在於具有執行速度的優勢，因此在處理小規模問題時往往可以在極短時間內找出問題的最佳解。然而，此一類型的演算方法同時也有為人所詬病的缺點。例如：吾人無法藉由增加程式執行時間以改進解的品**

質。此外，對於大規模的工作匹配與排程問題，串列演算方法亦無法保證其所求解的品質 [13]。有鑑於此，許多文獻便提出以泛用啟發式演算方法求解工作匹配與排程問題的構想。

傳統的泛用啟發式演算方法包括基因演算方法 (Genetic Algorithms, GAs) [3, 13]、模擬退火方法 (Simulated Annealing, SA) [1] 以及禁制搜尋方法 (Tabu Search, TS) [8]。在求解組合最佳化的研究領域中，泛用啟發式演算方法已經被廣泛應用且成功地獲致相當不錯的成效。例如：旅行推銷員問題 (Traveling Salesman Problem, TSP)、二次分派問題 (Quadratic Assignment Problem, QAP)、資源限制專案排程問題 (Resource-Constrained Project Scheduling Problem, RCPSP) 等。模擬退火方法、禁制搜尋方法以及基因演算方法皆可被視為是迭代式演算法 (iterative algorithms) 的一種。其基本運作方式係由一個或多個已知的問題初始解開始，然後反覆執行改進現行解的處理程序直到滿足終止條件為止。以上所提及之泛用啟發式演算方法應用在求解最佳化問題時大多以隨機方式產生初始解。此一方法之優點在於產生初始解的速度極快，但卻也往往導致問題求解的收斂時程過長。

為了改進傳統的泛用啟發式演算方法在設計上的缺陷，馬可・朵麗哥 (Marco Dorigo) 等學者於 90 年代初期提出著名的螞蟻系統 (Ant System, AS) 演算方法 [7]。螞蟻系統的設計構想主要源自於對自然界螞蟻覓食行為的觀察。在真實世界中，螞蟻利用一種名為「費洛蒙」 (pheromone) 的化學物質彼此溝通進而達成協同合作的目的，藉此找出巢穴與食物之間的最短路徑。螞蟻系統便是以人工螞蟻模擬此一行為模式並將之應用在旅行推銷員問題的求解。基本上，螞蟻系統在運作時的核心為狀態轉移規則與費洛蒙更新規則。狀態轉移規則主要是以費洛蒙機率模型為工具，用以引導人工螞蟻在決定其下一步的行進路線時有較大的機率選擇費洛蒙累積濃度較高的路線前進，如此反覆執行直到該人工螞蟻完成一條合法的路徑為止。換言之，狀態轉移規則係採用逐步建構的方式以產生一個完整的問題可行解。此係螞蟻系統與傳統泛用啟發式演算方法在設計上的最大不同之處。另一方面，就費洛蒙更新規則而言，當系統中所有的人工螞蟻都已經順利建構出問題解之後，便將所有人工螞蟻於本迭代中所走過的路徑累積適量的費洛蒙，以使得人工螞蟻在下一次執行狀態轉移規則之時有較高機率選擇較多螞蟻所行經過的路線。如此經過一定的迭代數之後，所有的人工螞蟻便會趨向於往費洛蒙累積濃度最高的路線前進，吾人最後便可以獲致理想的問題解。基於此一設計概念，近年來陸續有許多文獻提出新的設計方法，而這些方法也都被成功地應用在傳統組合最佳化問題的求解 [7]。鑑於螞蟻系統相關演算方法在求解眾多組合最佳化問題時具有相當卓越的成效，本文中，我們將提出一個可用以求解工作匹配與排程問題的螞蟻系統。

在嘗試以螞蟻系統演算方法求解工作匹配與排程問題的過程中，吾人面臨一些必須要克服的問題。例如：系統中人工螞蟻數量的設定。毫無疑問地，人工螞蟻的數量會直接影響演算方法的執行效能。若人工螞蟻的數量太多，將可能產生過多的重複運算且導致陷入區域最佳解的困境；反之，若人工螞蟻的數量太少且又缺乏相關的配套措施，則將無法有效地進行解空間的搜尋。為了解決此一問題，我們將以快速螞蟻系統 (Fast Ant System, FANT) 的演算方法為基礎 [2]，設計出一個名為 FANT-TMS (Fast Ant System for Task Matching and Scheduling) 的演算法。快速螞蟻系統係以單一人工螞蟻進行解空間的搜尋，搭配適當的費洛蒙重置機制，此一系統可以有效地運作並加快求解問題的速度。此外，我們也將採用一種間接解的表示方法用以降低 FANT-TMS 演算方法在執行時的複雜度。為了強化 FANT-TMS 演算方法的執行效能，我們同時也設計了一個適用於求解工作匹配與排程問題的區域搜尋方法。

本文在後續的內容中，首先會在第二節針對所探討的工作匹配與排程問題加以描述且定義。其次，我們將在第三節詳細說明本文中所提出方法的設計細節。接著我們會在第四節呈現評估演算法效能的實驗結果。最後，我們將第五節為本文內容進行結論並說明後續相關的研究工作。

2. 問題塑模

為了能有效描述平行程式的特性，吾人一般會利用一個有向非循環圖 (directed acyclic graph，DAG) $G = \{V, E\}$ 來表示一個欲進行排程的平行程式。針對任一特定的 DAG，節點係用來表示平行程式中的工作 (task)。DAG中的有向邊則係用來表示工作與工作之間的執行先後順序關係 (precedence relation)。此一模型中各維度的定義與相關特性說明如下：

- $V = \{t_1, t_2, ..., t_{|T|}\}$ 為平行程式中所有工作的集合，符號 $|T|$ 則表示為此集合中所有工作的個數。每一個工作都有相對應的工作量 (work load)，對工作 t_i 而言，我們以 $WL(t_i)$ 表示該工作之工作量。
- $E = \{e_{ij}\}$ 為 DAG 中所有有向邊的集合，e_{ij} 表示工作 t_i 與工作 t_j 之間具有執行先後順序關係之限制。換言之，若吾人欲開始執行工作 t_j，則必須先將工作 t_i 完成。我們以 $DATA(t_i, t_j)$ 表示工作 t_i 在執行完之後必須傳送給工作 t_j 的資料量。此外，在此限制條件下，我們稱 t_i 為 t_j 的立即前置工作，而 t_j 則被稱為是 t_i 的立即後繼工作。當一個工作沒有任何的立即前置工作，或者是所有的立即前置工作都已經完成排程與匹配，則稱該工作為準備工作 (ready task)。對於所有屬於工作 t_j 的立即前置工作所成的集合，我們以 $PRED(t_j)$ 表示；而 $SUCC(t_i)$ 則用以表

示工作 t_i 的所有立即後繼工作所成的集合。若有一工作 t_i，其 $SUCC(t_i)$ 的內容為空集合，我們稱之為起始工作。若有一工作 t_j，其 $PRED(t_j)$ 的內容為空集合，我們稱之為結束工作。在不失一般性的情況下，我們假設每一個 DAG 中都各只有一個起始工作與結束工作，並將之分別表示為 t_{entry} 與 t_{exit}。顯而易見地，t_{exit} 的完成時間即代表整個平行程式的完成時間。

圖 1 是一個有向非循環圖範例。此一 DAG 中共包含了 t_1、t_2、...、t_{10} 等 10 個工作節點。在工作節點 t_i 的圖形中，圓形中上半部份表示工作的名稱 t_i，而下半部份則表示工作 t_i 的工作量 $WL(t_i)$。由此 DAG 中我們可得知：工作節點 t_1 為起始工作，而工作節點 t_{10} 為結束工作；對工作節點 t_3 而言，其存在一個立即前置工作 t_1 以及兩個立即後繼工作 t_7 和 t_8；工作 t_1 與 t_7 之間所傳輸的資料量 $DATA(t_1, t_7)$ 為 20。

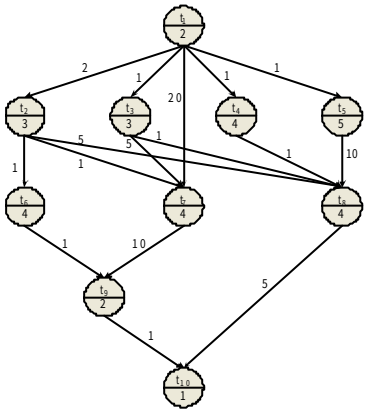


圖 1 一個有向非循環圖範例

另一方面，本文中所探討之叢集系統係由眾多處理單元所構成的集合，我們以符號 $P = \{p_1, p_2, ..., p_{|P|}\}$ 描述之。其中，$|P|$ 表示系統中處理單元的個數。平行程式中的所有工作便是由這些處理單元所負責執行。一旦某一個工作開始由某一處理單元所執行，其執行過程將不能被中斷直到該工作完成為止。除此之外，每一個處理單元不能在同一時間內同時執行多個工作，而每一個工作僅能被任意的一個處理單元所執行。

此一系統可被視為是一個異質性計算環境。換言之，系統中所有處理單元的計算能力都不盡相同。為了描述一個處理單元的計算能力，我們以符號 CP(pu) 表示處理單元 pu 在一個單位時間內所能處理的工作量。因此，將工作 ti 分派給處理單元 pu 執行所必須付出的計算成本 COMP(ti, pu) (亦即，完成 ti 所需耗用的 CPU 時間) 可被定義為工作 ti 的工作量除以處理單元 pu 之計算能力所得的結果。亦即

$$COMP(t_i, p_u) = \frac{WL(t_i)}{CP(p_u)}$$

此外，我們同時假設各個處理單元彼此之間係藉由高速網路元件以完全連接的網路拓樸相互連結。我們以 $BW(p_u, p_v)$ 表示處理單元 p_u 與 p_v 之間的網路頻寬。本文中，我們以 $PE(t_i)$ 表示負責執行工作 t_i 的處理單元，因此 t_i 在執行完成之後傳送資料給 t_j 所耗用的時間便等於 t_i 與 t_j 之間的資料傳輸量除以處理單元 $PE(t_i)$ 與 $PE(t_j)$ 之間的頻寬所獲得的結果，我們以通訊成本 $COMM(t_i, t_j, PE(t_i), PE(t_j))$ 表示此一耗用時間。必須要說明的是，若 $PE(t_i)$ 與 $PE(t_j)$ 所表示的是相同的處理單元，則因處理單元內部資料傳輸耗時極小，故此通訊成本可忽略不計。於是，$COMM(t_i, t_j, PE(t_i), PE(t_j))$ 的數學表示式可被描述如下：

$$COMM(t_i, t_j, PE(t_i), PE(t_j)) = \begin{cases} \dfrac{DATA(t_i, t_j)}{BW(p_u, p_v)}, & if \ \ p_u \neq p_v \\ 0, & otherwise \end{cases}$$

基於以上之定義，工作 t_j 在某一特定的排程結果 r 的完成時間 $FINISH_r(t_j)$ 之數學定義式可計算而得如下：

$$FINISH_r(t_j) = \max_{\forall t_i \in PRED(t_j)} [FINISH_r(t_i) + COMM(t_i, t_j, PE(t_i), PE(t_j))] + COMP(t_j, PE(t_j))$$

吾人並可進一步得出求解工作匹配與排程問題的目標函式：

$$\max_{\forall r} F(r) = FINISH_r(t_{exit})$$

本文之目的便在於根據此一目標函式設計出適宜的演算法則，進而找出合乎要求的工作排程結果，使得平行應用程式能在最短的時間之內完成。

3. 設計方法

3.1 狀態轉移規則設計

FANT-TMS 演算方法在初始化時會先將 DAG 檔案的資訊讀入，設定演算法所需要的參數。在螞蟻的數量部份，此一演算方法與傳統螞蟻系統演算法不同之處在於採用單一螞蟻的架構。其目的是為了節省每一迭代所要花費的計算時間，以及加快其收斂的速度。在費洛蒙結構的設計方面，我們會建構出一個 $|T| \times |T|$ 大小的費洛蒙表。在表中的每一列代表的是工作排程的順序，而每一行則是表示工作的編號。換句話說，人工螞蟻所行進路徑的結果係表示工作執行的順序。除此之外，我們也會建構出一個大小為 L 的禁制串列，用來存放前 L 個迭代的最佳解，其目的是為了當演算法陷入區

域最佳的困境時能有跳脫的機會。

在演算法一開始，會先以機率選擇算式決定工作執行的先後順序。各個工作會依據其費洛蒙值與依據全域最佳解所計算而得的 *B-Level* 值來決定其被選取到的機率，進而建構出一組符合拓樸排序的工作順序。最後，當所有工作的優先權順序皆已決定之後，便以工作最早完成時間為準則，依序將工作分派給最適合的處理單元執行，藉此建構出一組完整的問題解。本方法中的機率選擇算主要是計算工作 t_i 的執行優先權為第 s 順位的機率 $PROB(s, t_i)$，其被定義為

$$PROB(s,t_j)=\begin{cases}\dfrac{[t(s,j)]^a\times[\dfrac{BL(t_j)}{BL(t_{entry})}]^b}{\sum_{t_k\in Ready}[t(s,t_k)]^a\times[\dfrac{BL(t_k)}{BL(t_{entry})}]^b}, & if\ t_j\ is\ a\ ready\ task;\\ 0, & otherwise;\end{cases}$$

其中，$BL(t_j)$ 為工作 t_j 之 *B-Level* 值；a = 1、b = log(迭代數)。我們對於 b 值的設計採用動態設計，主要目的是希望能見度對工作優先權的影響會隨著迭代數的增加而越來越小。

3.2 區域搜尋方法設計

當以狀態轉移規則建構出一組完整的解之後，演算法便會針對此一現行解進行區域搜尋。在我們所提出的區域搜尋方法中，包含了以下四個步驟：

步驟 1 根據現行解之工作執行順序逐一檢查每一個工作置換所有處理單元後所得的結果。若存在一置換可使得目前的解有所改善，則將該工作予以重新分派。重複執行此一步驟直到檢查過程中都沒有置換發生的情況為止。

步驟 2 根據現行解之工作執行順序逐一檢查每一個工作向前調整其執行順序後所得的結果。若存在一新的執行順序可使得目前的解有所改善，則調整該工作的執行順序。重複執行此一步驟直到檢查過程中都沒有發生調整執行順序的情況為止。

步驟 3 找出目前排程結果中具有最高通訊成本的兩個工作 t_i 與 t_j。建立一個工作群組 G，並將 t_i 與 t_j 加入此一群組中。由 t_i 的所有立即前置工作中找出一個與 t_i 之間具有最大通訊成本且該通訊成本大於等於 $COMM(t_i, t_j, PE(t_i), PE(t_j))$ 之值的工作 t_{hi}。若工作 t_{hi} 確實存在，則將 t_{hi} 加入群組 G 中。以同樣方式為工作 t_j 找出一個立即前置工作 t_{hj}，並將 t_{hj} 加入 G 中。由 t_i 的所有立即後置工作中找出一個與 t_i 之間具有最大通訊成本且該通訊成本大於等於 $COMM(t_i, t_j, PE(t_i), PE(t_j))$ 之值的工作 t_{ki}。若工作 t_{ki} 確實存在，則將 t_{ki} 加入群組 G 中。以同樣方式為工作 t_j 找出一個立即後置工作 t_{kj}，並將 t_{kj} 加入 G 中。將群組 G 中的所有工作分派給相同的處理單元執行。重複執行此一步驟直到現行解無法再被改善為止。

步驟 4 找出目前處理最多關鍵路徑工作的處理單元。檢查此處理單元上所有工作與其它處理單元上的所有工作置換後的結果。若存在一置換可使得目前的解有所改善，則執行此一置換。重複執行此一步驟直到現行解無法再被改善為止。

當以上所提及的四個步驟均無法對現行解有所改善時，便停止區域搜尋程序的執行。在此一方法中，步驟一的目的是要將各個工作分派給最佳的處理單元執行；而步驟二則是要找出各個工作的最適執行順序。至於步驟三則是為了讓現行解中具有最高通訊成本的兩個工作能夠被重新分派。最後，步驟四則是為了避免關鍵路徑上的工作被分派給處理能力較差的處理單元，致使前三個步驟都無法讓排程結果再有改善，使得解的改善空間有限。

3.3 費洛蒙濃度更新策略設計

在費洛蒙更新策略的部份，FANT-TMS 採用以下三種費洛蒙更新策略：費洛蒙重置、全域最佳費洛蒙更新與串列費洛蒙更新。費洛蒙重置的時機是每當全域最佳解有所改善時便將費洛蒙表的內容初始化。此一做法是為了避免演算法在執行許多迭代之後，費洛蒙表中有些較差路徑上的值會過高，進而影響螞蟻在選擇路徑時無法選擇到較佳的路徑。至於全域最佳費洛蒙更新與串列費洛蒙更新的執行時機則是以機率判斷式 $((SL_{現行解} - AVG) / (SL_{現行解} - SL_{gbest})) > Random[0, 1]$ 決定。在此一判斷式中，分子的部份是以現行解之目標函數值減去禁制串列中所有解的目標函數值的平均，而分母的部份則是以現行解之目標函數值減去全域最佳解的目標函數值。在我們所提出的設計方法中，若此一判斷式成立，則表示要執行串列費洛蒙更新。這是因為當演算法如果陷入區域最佳的困境時，此區域最佳解與串列中的解的內容會有所差異，因此在更新費洛蒙濃度之後，可增加其它不同路徑上的費洛蒙量，便可使其有較大的機會跳脫區域最佳的困境。串列費洛蒙更新規則定義如下：

$$t(s,t_j)=(1-r)\times t(s,t_j)+Dt(s,t_j)$$

上式中，$t(s, t_j)$ 表示工作 t_j 的執行優先權為第 s 順位的費洛蒙值；而 $\Delta t(s, t_j)$ 則是定義如下：

$$Dt(s,t_j)=\sum_{k=1}^{L}Dt_k(s,t_j)$$

其中，L 為串列的大小；$\Delta t_k(s, t_j)$ 則定義如下：

$$Dt_k(s,t_j)=\begin{cases}\dfrac{SL(t_{entry})}{SL_k}, & if\ (s,t_j)\ is\ used\ by\ ant\ k\\ 0, & otherwise\end{cases}$$

在此，$SL(t_{entry})$ 為不考慮通訊成本的關鍵路徑長度；而 SL_k 則是第 k 隻螞蟻求解所得的排程長度。

另一方面，若判斷式不成立，則表示要執行全域最佳費洛蒙更新。其目的是當演算法在剛開始執行時，搜尋求解過程尚未收斂，以全域最佳費洛蒙更新來更新費洛蒙，可以有效加速求解過程收斂的速度。全域最佳費洛蒙更新規則定義如下：

$$t(s,t_j)=(1-r)\times t(s,t_j)+Dt_{gb}(s,t_j)$$

上式中，$\Delta t_{gb}(s, t_j)$ 的定義如下：

$$Dt_{gb}(d,t_j)=\begin{cases}\dfrac{SL(t_{entry})}{SL_{gbest}}, & if\ (d,t_j)\in gbest\ solution\\ 0, & otherwise\end{cases}$$

4. 效能評估

我們以 DAG 產生器為產生測試案例的工具，我們依據以下的相關參數值產生 1080 種不同的參數組合：

- 工作個數($|T|$)：20、40、60、80、100、120、140、160、180、200。
- 處理單元個數($|P|$)：2、4、8。
- 平均通訊成本與計算成本比值(CCR)：0.1、1、10。
- 工作節點的平均分支度(Degavg)：$\frac{\sqrt{|T|}}{2}$、$\sqrt{|T|}$、$2\sqrt{|T|}$。
- 平均分支度之標準差(Deg_{sd})：$0.2\times Deg_{avg}$。
- 處理單元之平均執行能力($Power_{avg}$)：1。
- 處理單元執行能力之標準差($Power_{sd}$)：0、0.4。
- 各個處理單元之間的平均傳輸速率($CommRate_{avg}$)：1。
- 處理單元傳輸速率之標準差($CommRate_{sd}$)：0、0.4。

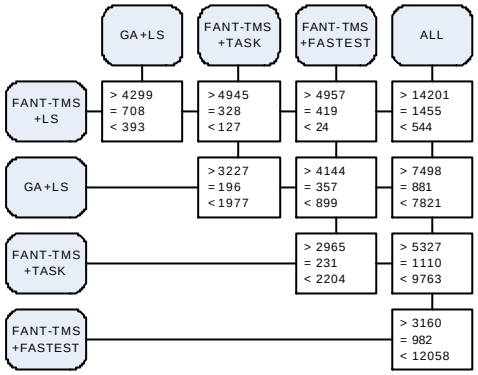
針對以上所提及的每一種參數組合，我們以隨機的方式產生 5 個測試的 DAG 檔案。因此一共有 5400 個不同的 DAG 檔案可做為我們在進行演算法效能評估時的測試案例。我們以執行時間上限為演算法執行之終止條件。其目的是要比較各種演算法在相同時間內對於處理各種 DAG 案例的效能。對每一個測試案例而言，其工作個數為 $|T|$，處理單元個數為 $|P|$，則我們可得知其解空間大小為 $|P|^{|T|}$。我們以 $\sqrt{|T|\times\ln|P|}$ 之值作為每個演算法執行該測試案例的單位時間。我們取演算法執行到單倍、雙倍、四倍以及八倍單位執行時間為演算法執行的停止條件。此可以比較出各種演算法在固定的時間內對於不同類型測試案例的優劣，同時也可以進一步比較出執行時間長短對於演算法的影響程度。

在後續的實驗結果中，我們以 "GA" 表示傳統的基因演算法。至於在區域搜尋演算法部份，則是以 "FAST"、"TASK" 以及 "LS" 分別表示 FASTEST [12]、TASK [9] 以及我們所提出的區域搜尋方法。我們將所有測試案例依據工作個數，處理單元個數來分類，並以 180 個不同的測試案例組成一個實驗群組。針對每一個實驗群組，以不同的 CCR 值再細分成三組不同的子群組。我們以異質性正規排程長度 (Heterogeneous normalized schedule length, HNSL) 為效能評估的工具。在此，HNSL 被定義為演算法所求得之排程長度與排程長度下限之比值；而排程長度下限則是指在 DAG 中選取一條包含起始工作與結束工作且工作量總和最大之路徑，然後將此路徑上所有工作的工作量加總後除以處理單元中最大工作能力所得到的值。

由實驗結果我們可以發現，在處理單元數量越多以及 CCR 越大的情況下，增加演算法的執行時間對於演算法的效能有較大的提升。這是因為當處理單元數越多，解空間就越大；CCR 越大，則解優劣的差異也會越大。因此，演算法就必須要花費較多時間來搜尋更好的解。相較於傳統的基因演算法，FANT-TMS 只有在較短執行時間、CCR=0.1 且 DAG 大小為 180 與 200 的情況下略遜於 GA+LS。這是因為 FANT-TMS 必須花費較多的時間決定所有工作的執行順序。儘管如此，二者之差距也僅有 0.05 而已。然而，在其它的情況下，FANT-TMS 都是明顯地優於基因演算法。此外，有關區域搜尋方法在效能上的比較，我們所提出的方法則是在所有的情況下都明顯優於其它方法。至於，在執行時間方面，我們所設計的演算法在執行時間增加越多，所改進的幅度也較其它演算法大。由此可以看出我們所設計的演算法有跳脫區域最佳解的能力，讓演算法不會因為陷入區域最佳解無法跳脫，其他演算法因為陷入區域最佳解無法跳脫，就算執行的時間加長，其改進的幅度仍然有限。

除此之外，我們也將所有測試案例在執行 8 倍時間所得之結果加以統計，進行不同演算法之間的全面性比較。我們在表 1 中標示出比較之後的結果。矩形中所表示的數據為矩形上方與矩形左方演算法的比較結果。" > "、" = "和" < " 分別表示左方演算法優於、等於、劣於上方演算法的排程案例個數。我們以 FANT-TMS+LS 對應到 ALL 為例，表示 FANT-TMS+LS 優於其它三種演算法一共 14201 次，相等的有 1455 次，較差的只有 544 次。整體看來，我們的演算法優於及等於 GA+LS、FANT-TMS+TASK 與 FANT-TMS+FAST 的比率分別為 92.72%、97.64% 與 99.55%。

表 1 排程結果優劣之全面比較



5. 結論

本文中，基於快速螞蟻系統的設計概念，我們提出了一個用以解決工作匹配與排程問題的快速螞蟻系統。此一演算法名為 FANT-TMS，其與傳統的螞蟻系統相關演算方法有以下的不同之處：

- 單一螞蟻求解：我們之所以採用單一螞蟻求解主要是有利於平行化的實作，而且可以讓求解的速度較快速較有效率。
- 兩階段求解： FANT-TMS 係以快速螞蟻系統決定工作執行的先後順序，然後再以 list scheduling 演算法的方式，分派工作給適當的處理單元執行。此一設計主要目的是要讓演算法的計算時間減少，讓演算法能在較短的時間內搜尋更多的解空間。
- 動態能見度設計：FANT-TMS 的能見度會隨著全域最佳解的改變而有所不同。此一設計方法可以讓演算法在求解剛開始的過程中有較佳的目標可以參考，而不會漫無目的的搜尋。
- 費洛蒙更新設計：FANT-TMS 使用機率判斷式來決定所要執行的費洛蒙更新規則，此使得吾人可以根據目前所求得的解選擇出最佳的費洛蒙方式更新。

為了有效獲得高品質的解，我們在本文中亦提出一個區域搜尋方法。此一方法與傳統區域搜尋技術最大的不同在於允許一次置換數個工作的處理單元。實驗結果顯示，我們所提出的 FANT-TMS 方法在單一機器執行上都較其它演算法要好。除此之外，我們也將 FANT-TMS 演算法以平行化的方式實作。在實驗的過程中我們發現以較多機器以及每部機器間具有資訊交換機制的結果會較好。我們希望將來能再加強此一平行化的架構，讓 FANT-TMS 演算法能在使用最少的機器數的情況下獲致最佳的解。

參考文獻

[1] E. Aarts and J. Korst, Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing, John Wiley & Sons, 1989.

[2] E. D. Taillard, "FANT: Fast Ant System," Technical report IDSIA-46-98, IDSIA, Lugano, Switzerland, 1998.

[3] E. S. H. Hou, N. Ansari, and H. Ren, "A Genetic Algorithm for Multiprocessor Scheduling," IEEE Trans. Parallel and Distributed Systems, Vol. 5, No. 2, pp. 113-120, February 1994.

[4] H. EI-Rewini, H. Ali, and T. Lewis, "Task Scheduling in Multiprocessing Systems," Computer, Vol. 28, No. 12, pp. 27-37, December 1995.

[5] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach," J. Parallel and Distributed Computing, Vol. 47, No. 1, pp.8-22, November 1997.

[6] M. Dorigo, and L. M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," IEEE Trans. Evolutionary Computation, Vol. 1, No. 1, pp. 53-66, April 1997.

[7] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents," IEEE Trans. System, Man and Cybernetics-Part B: Vol. 26, No. 1, February 1996.

[8] M. Lin, L. Karlsson, and L. T. Yang, "Heuristic Techniques: Scheduling Partially Ordered Tasks in a Multi-processor Environment with Tabu and Genetic Algorithm," Proc. of the 7th International Conference on Parallel and Distributed Systems, pp.515-523, July 2000.

[9] M. Y. Wu, W. Shu, and J. Gu, "Efficient Local Search for DAG Scheduling," IEEE Trans. Parallel and Distributed System, Vol. 12, No 6, pp. 617-627, June 2001.

[10] R. Sethi, "Scheduling Graphs on Two Processors," SIAM J. Computing, Vol. 5, No. 1, pp. 73-82, March 1976.

[11] W. H. Kohler and K. Steiglitz, "Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems," J. ACM, Vol. 21, No. 1, pp 140-156, January 1974.

[12] Y. K. Kwok and I. Ahmad, "FASTEST: A Practical Low-Complexity Algorithm for Compile-Time Assignment of Parallel Programs to Multiprocessors," IEEE Trans. Parallel and Distributed System, Vol. 10, No 2, pp. 147-159, February 1999.

[13] Y. K. Kwok and I. Ahmad, "Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using a Parallel Genetic Algorithm," J. Parallel and Distributed Computing, Vol. 47, No. 1, pp.58-77, November 1997.