實驗設計結合適應性突變及多精英運算於實數編碼基因演算法之改良研究

湯秉宏
中山醫學大學應用資訊科學研究所
Email: max@xoo.tw

曾明性*
中山醫學大學應用資訊科學學系
*Email: mht@csmu.edu.tw

摘要

基因演算法是以達爾文進化論「自然選擇」及「適者生存」的概念設計，業經證實可有效率地求解 NP-Hard 問題並可有效能地搜尋複雜非線性求解問題空間。在全域最佳化問題的求解上，常需面臨的挑戰是：當求解維度愈高或局部較佳解愈多時，演算法愈容易陷入局部較佳解。為了改良傳統實數編碼基因演算法的計算效能，本研究整合「適應性突變」、「實驗設計」與「多精英保留」的運算機制，藉以提高全域搜尋能力並降低搜尋時間。經 12 個複雜測試函數演算結果顯示，即使在維度高達 100 的情形下，本文提出之改良型實數編碼基因演算法確比前人諸多方法更具有優異的最佳化效能表現以及能最最短的時間內可達到有效的收斂。

關鍵詞：實數編碼基因演算法、實驗設計、適應性突變、多精英保留

一、緒論

在最佳化求解的問題中，如何以最短的時間，得到一個最高的準確率，一直是軟式計算領域關注的課題。基因演算法是一種有效的最佳化搜尋方法，而且被廣泛的應用於搜尋各類問題的最佳解。基因演算法的運作包含六個模組：選擇(Selection)、交配(crossover)、突變(mutation)、複製(replacement)、評估(evaluation)、精英(elite) [2-4 , 9-14]。但過程中的交配及突變運算，是以隨機的方式改變群體內的基因，雖然使群體當中保有豐富的多樣性，但隨機的方式也包含太多無法掌握的狀況。如接近全域最佳解時，隨機的搜尋反而無法找到全域最佳解，並且而容易造成系統的振盪甚至發散，甚至花費了大量的時間，進行系統的疊代演化，卻無法在求解的問題上得到更好的收斂狀況。這樣的缺陷在高維度問題求解上更是嚴重，所以這一部份的改善將是本文研究的重點。

本文共分成五大部份。第二部份主要是介紹基因演算法的文獻回顧，第三部份則說明本文的研究方法，包含實數編碼基因演算法的原理、實驗設計方法、適應性突變運算與多精英保留機制以改進計算效能。針對本文所提的改進方法，第四部份是以連續／不連續、單極點／多極點、或其混合型式函數的極值搜尋問題之測試實驗。最後總結本文的結論於第五部份。。

二、文獻探討

基因演算法(Genetic Algorithms, GA)是由學者 John Holland 於 1975 年所提出，其原理主要參照達爾文進化論—「物競天擇，適者生存」而發展出此方法，演算過程主要藉由仿效生物演化特性，讓母體群隨著世代進行演化，經由競爭之過程與控制—複製/選擇(Reproduction/Selection)、交配(Crossover)、突變(Mutation)之方法來保留較佳之染色體，淘汰較差之染色體。應用於工程計

算時，以疊代方式隨機產生多組解進行演算，逐步求得最佳解。

基因演算法是屬於隨機的(Stochastic)最佳化搜尋技術，其演算原理係以適合度函數(Fitness Function)來控制求解問題之目標值，並藉由複製/選擇、交配和突變的演算程序搜尋問題之最佳解。由於基因演算法是以多個不同的點所組成的母體來搜尋多個不同的解，因而能夠涵蓋較大的搜尋空間，所得的解會較穩定且接近最佳解。其具有以下優於傳統最佳化之特性：容易使用、適用範圍廣、具多點搜尋之特性、適合處理複雜的問題、有較高的機率求得全域最佳解。

自 Holland 成功提出基因演算法理論之後，廣泛應用於最佳化控制器(Optimal Controller)、機械學習(Machine Learning)、圖形辨識(Pattern Recognition)與 NP-Hard 等需要在廣大搜尋空間尋找最佳解的搜尋解(Search for Solutions)問題上，基因演算法已被證實是用以解決搜尋與最佳化問題的有效理論。近年來有許多學者引用許多的改良方法，嘗試改善傳統 GA 演算法普遍存在的收斂緩慢與僅搜尋局部極值等缺點[1-4, 11-14]。

傳統的基因演算法是將基因形式的染色體以二位元編碼之基因演算法(Binary-coded Genetic Algorithm, BGA) [8-9,11]。二位元編碼的基因演算法運算時須經過以下幾個轉換與運算工作，包含編碼、解碼、產生初始族群、計算適應度、選擇、複製、交配和突變。由於染色體之位元數目與求解的精確度與搜尋速度息息相關，當利用 BGA 求解連續型的問題時，欲提高求解精度則需加大染色體的位元數，將導致計算時間的加長，因而另一種直接利用實數編碼之基因演算法(Real-coded Genetic Algorithm, RGA)[8-9,11]遂孕運而生。雖然前人研究大都使用二元編碼基因演算法進行相關最佳化問題求解，但實數編碼基因演算法因不需編碼及解碼等運算單元故執行效能較佳，加之其求解精度亦不會受限於字串位元數，因而本文採用實數編碼基因演算法進行後續探討。

三、研究方法

(一) 改良型實數編碼基因演算法

為了改良傳統實數編碼基因演算法的計算效能，本研究整合「實驗設計」、「適應性突變」與「多精英保留」的運算機制，藉以提高全域搜尋能力並降低搜尋時間。其中「多精英保留」係針對選擇(Selection)運算進行改良，「適應性突變」則針對突變(Mutation)運算進行改良，而「實驗設計」乃針對交配(CrossOver)運算進行改良。本文提出之改良型實數編碼基因演算法之步驟可概述如下：

步驟 1：群族初始化，並以亂數產生出初代族群隨機產生 N 個染色體為實數解。

步驟 2：帶入適應函數取得適應值，比較全部適應值有無優於最佳解，如果有則取代為最佳解。

步驟 3：否則以最佳解依照精英保留率保留相對數量的染色體，為「多精英保留機制」。

步驟 4：以輪盤總值法進行挑選染色體進交配池，並依交配機率為 0.8 的規則以「實驗設計」的方法進行新子代的產生。

步驟 5：以突變率為 0.05 的機率進行突變產生子代，且突變方式依「適應性突變法」產生更優良子代。

步驟 6：以子代為新一代的母代染色體。

步驟 7：檢查疊代次數是否已達到、函數是否已達全域最佳解或適應值是否持續維持 200 代 ，則終止演算，否則重覆步驟 3 至步驟 6。

(二) 多精英運算多精英運算(Multi-Elitisms Operators, MEO)

在基因演算法中，複製是指在為數眾多的母代個體中選取其一進行複製，複製出來的子代個體可以跟母代個體完全一樣，也可以有部份的不同。其目的就是讓母代個體的優良基因保留到子代個體，以免隨著時間的演化，個體中優良的基因反而消失不見了。由於複製是挑選表現較佳的個體來取代較差的個體，因此這個動作也被稱為「選擇」(Selection)，而選擇數量的多寡，則是

由「選擇率」來決定[8]。

在傳統的基因演算法中，通常在每代的演化過程中會實行精英保留機制，即保留演化過程中的最佳解，且在系統每次產生一新子代時便隨機將最佳解取代子代中的一個個體。精英保留策略，是影響優良的子代基因是否可以在系統中保有，並逐漸演化為更優秀子代基因的重要因素 。太少的精英子代，在為數眾多的基因中無法被選擇到，而喪失了繼續演化的機會。但太多的子代基因，又會降低的基因的多樣式，阻礙了優良基因產生的機會，而使系統提早收斂而無法達到全域最佳解。Ho et al. [8]研究指出，選擇率設定以 0.2 為最佳，也就是精英個體的個數應為群體數的 20%為最佳精英個數。

(三) 實驗設計(Design of Experiment, DOE)

實驗設計法一般分為二階段式實驗法，第一階段實驗目的是要利用直交表(Orthogonal Array)找出顯著因子，再藉由每一個因素分析出的主效果（Main Effect）推論每一個因素對於該實驗結果的優劣。第二階段實驗最主要目的是針對顯著因子找出其工作區間。並將實驗的數據是利用變異數分析，統計運算求出各因子的貢獻度，便可由因子貢獻度，並進行多特性分析，其重點在於兩特性間做一個較為有利的選擇[5,8-9,12]。

而在科學實驗的過程當中，通常會利用許多假設來減少實驗結果的因素個數，以節省時間上的浪費，而為了解決此問題，應用直交實驗設計也因此被提出。實驗設計的應用上常被使用的有直交表有全因素實驗設計法(Full Factorial Design, FFD)及田口實驗設計法(Taguchi methods) 。

全因素實驗設計法[5,9]，因素的效應定義成因素階次改變時對應實驗結果產生變化。實驗的主要因素效應稱做主效應（main effect）。一因素應不同階次而其他的因素變動的情況下目標值隨之而改變稱之為因素間的交互作用（interaction）。並以符號“－”表示某一因素的低階次（low level），“＋”表示高階次（high level）。系統中以最重要實驗情形，k 個因素，每一因素有兩個階次。則全部實驗次數需要 2×2×……×2=2k，將稱做 2k 因素設計。2k 設計對於許多因素的初期實驗工作特別有用，因為其可提供 k 個因素，的取少實驗次數來篩選其中較中重要的實驗程序變數。

表 1 為 2 階次 3 因素實驗設計即 2^3 設計方陣,以”＋”、”－”分別代表高階次及低階次符號，因此設計方陣可簡化為下表以＋、－為符號之設計表。透過直交表來作因素分析，來了解到底哪些因素對此事件影響的效果為何，稱為因素分析或主效果評估。主效果即為表 1 中的 F，是計算該因素的某一水準在不同實驗中，對目標函數的貢獻程度多少[5,9]。

表 1 2^3全因素實驗設計直交表

No	Orthogonal table of FFD							
	A	B	AB	C	AC	BC	ABC	Output
1	－	－	＋	－	＋	＋	－	F1
2	＋	－	－	－	－	＋	＋	F2
3	－	＋	－	－	＋	－	＋	F3
4	＋	＋	＋	－	－	－	－	F4
5	－	－	＋	＋	－	－	＋	F5
6	＋	－	－	＋	＋	－	－	F6
7	－	＋	－	＋	－	＋	－	F7
8	＋	＋	＋	＋	＋	＋	＋	F8

表 2 $L_8(2^7)$田口直交表

No	Orthogonal table of Taguchi							
	1	2	3	4	5	6	7	Output
1	1	1	1	1	1	1	1	F1
2	1	1	1	2	2	2	2	F2
3	1	2	2	1	1	2	2	F3
4	1	2	2	2	2	1	1	F4
5	2	1	2	1	2	1	2	F5
6	2	1	2	2	1	2	1	F6
7	2	2	1	1	2	2	1	F7
8	2	2	1	2	1	1	2	F8

田口實驗設計法[8,12]是利用簡單的直交表實驗設計與簡潔的變異數分析，以少量的實驗數據進行分析，得到有用的資訊。雖不如全因子法真正找出確切的最佳化位置，但能以少數實驗便

能指出最佳化趨勢。

表2即為$L_8(2^7)$直交表，表中本體內之1 和2數字分別表示因子的水準一和水準二。直交表中的每一行代表實驗中的某一個特定因子的變化情況。「行」的編號，可供因子或交互作用配置其上之用； L_8直交表上共有7 行，代表最多能配置七個因子。列數等於直交表的實驗次數，L_8直交表的實驗次數為8，故實驗編號由1 至8。在田口法實驗設計法中的主效果即為表2中的F，是計算該因素的某一水準在不同實驗中，對目標函數的貢獻程度有多少，表示直交表實驗中第j次實驗的評估函數值[8,12]。

(四)適應性突變(Adaptive Mutation Operators, AMO)

傳統基因演算法的突變方法是以隨機的方式，在求解的區域範圍內進行在突變[2-4, 10-11]，隨機即包含很多不確定性，無法掌握突變方式的正確性，所以搜尋的結果，在同一條件之下也會有很大的差異，無法確保在突變時均朝著正確的方向進行收斂。

適應性突變[1, 10]的目的是要避免因交配運算造成群體中，或是因任意突變使系統變成隨意搜尋的反效應。所以適應性突變的方法將依循模擬梯度或反梯度的方向，使得突變能依據搜尋空問地形變化的資訊，瞭解族群變動趨勢對適應值變化的影響程度，並據以適度調整突變的策略。有關梯度(或反梯度)方向的模擬是根據演化時，群體中最佳個體適應值的變量決定。定義g(t-1)及g(t)分別是連續三個世代(t-2, t-1, t)最佳個體適應值的變量：

$$g(t) = f\left(x^{best}(t)\right) - f\left(x^{best}(t-1)\right)$$
$$g(t-1) = f\left(x^{best}(t-1)\right) - f\left(x^{best}(t-2)\right) \quad (1)$$

其中$x^{best}(t)$是第t世代的最佳個體。但由於個體中的每個基因的變動，皆會影響適應值的改變，所以突變的運算除了以(1)式為依據外，也必須根據個體基因值變動的情形作為突變方向的參考。且定義$\Delta x(t-1)$及$\Delta x(t)$是任一個體的第k維度在連續三個世代的變化量為：

$$\Delta x_k(t) = x_k(t) - x_k(t-1)$$
$$\Delta x_k(t-1) = x_k(t-1) - x_k(t-2) \quad (2)$$

結合(1)式和(2)式的結果，個體基因的變動將以疊代的方式更新：

$$x_k = x_k \pm Pm * \Delta x_k \quad (3)$$

其中Pm是根據個體的適應值依[10]的方式所得。適應值較差的個體有較大的幅度變動，而±的決定是依個體適應值及基因的改變，且±的改變有助於系統跳脫區域解。

四、實驗結果

為了與前人的研究結果[8]進行比較，本文使用 文獻上 12 個常見用來測試最佳化問題的標準測試函數 (benchmark function) ，如表3所列，作為本文所研提各改善方法的效能評估比較之基準。其中所有函數皆具有延展性，意指可以設定變數個數，適合用來測試大量連續型參數最佳化問題設計。函數性質包括 unimodal 、multimodal 及 step function 等複雜性，可以充分測試出各種方法針對各種型態連續型變數問題的演化能力。

所有的測試過程本文使用的 PC 設備規格為 Core 2 Quad Q8200 2.33GMhz CPU 和 2 giga-byte RAM ，且所有的測試方法皆由 Microsoft.NET framework 所開發完成。

本研究首先以傳統實數型基因演算法(TRGA)為基礎，並加入直交表實驗設計(FFD OA / Taguchi OA)後，進行第一階段之改良測試，接者以改良式直交表實驗設計(N-FFD OA /N-Taguchi OA)後，進行第二階段之改良測試。繼之分別加入多精英運算(MEO)或適應性突變(AMO) 進行第三階段之改良測試，最後整合實驗設計、多精英運算(MEO)與適應性突變(AMO)進行第四階段之改良測試。

為了與 Ho et al. [8]的研究結果進行比較，演算過程所有參數值設定均比照 Ho et al. [8]中所設：群體數量 N=30，交配機率為 0.8，突變率為 0.05 及選擇率為 0.2，演算終於條件為(1)總疊代次數為 12000 次(2)或在演化過程中若連續經過 200 代未產生更好的子代便提早終止(3)以收斂至函數最佳解，並重複 30 次的實驗。最後取平均做為最後的實驗結果，且分別以中維度(D=10)與高維度(D=100)之參數最佳化測試問題，進行計算效能之比較。

首先以疊代次數為 12000 代為終止條件，進行第一及第二階段之改良測試比較，結果如表 4 所示。繼之以適應值持續維持 200 代為終止條件，進行第三及第四階段之改良測試比較，結果如表 5、表 6 所示。

(一) 中維度實驗(D=10)

Ho et al. [8]提出的 IEA 演算法在中維度問題求解，其效能表現並不優異，而其中比較的七個演算法中以 BOA 演算法在 D=10 表現最好。所以本研究特將 IEA 與 BOA 演算法結果並列，並與本研究研擬的 11 種方法進行求解準精度比較，結果詳見表 5 所示。

如表 5 所示，在中維度時，本文自行開發的 TRGA 演算法加上直交表實驗設計方法後，於 12 個測試函數中，已有 11 個函數優於 Ho et al. [8]的 IEA 法及全部函數皆優於 BOA 法的表現，且演化次數皆在 3500 次內即有此結果，統計結果如表 7 所示。其可能原因初步推斷為 IEA 與 BOA 法採用二位元編碼基因演算法，而本研究採實數編碼基因演算法。

將本文研提的 11 個演算法加上 Ho et al. [8]中最優的 2 種演算法進行在中維度(D=10)搜尋 12 個測試函數的全域極值的求解能力進行排名，結果如表 5 之括弧內數字所示。本文並將之加以平均求取總體能力排名，如表 5 最後兩列所示。實驗證實本研究提出的 11 個演算法中，除 TRGA 外，在中維度(D=10)的全域搜尋能力皆優於 Ho et al. [8]的 IEA 或 BOA 法的表現，且所需演化次數少於其 1/4。

計算效能的比較如表 8 所示，本文研提的改良方法，在已能搜尋到全域極值及經連續演化 200 次未產生更佳解的函數，皆可提早收斂。而反應在疊代次數及時間也能相對的減少。在疊代次數及所需計算時間的表現，統計結果如表 8 所示。

表7 各演算法準確性(12種測試函數)

TRGA 改良方法	優於 BOA	優於 IEA		達極值數		最佳排序數	
	中	中	高	中	高	中	高
FFD	**12**	11	**12**	0	0	0	0
Taguchi	**12**	11	11	0	0	0	0
N-FFD	**12**	**12**	**12**	2	0	2	0
N-Taguchi	11	**12**	11	2	0	3	0
N-FFD+MEO	**12**	**12**	**12**	2	0	2	0
N-Taguchi+MEO	**12**	**12**	11	2	0	2	0
N-FFD+AMO	**12**	**12**	**12**	**4**	**4**	**7**	6
N-Taguchi+AMO	**12**	**12**	11	**4**	**4**	**7**	**10**
N-FFD+MEO +AMO	**12**	**12**	**12**	**4**	**4**	**7**	6
N-Taguchi+MEO +AMO	**12**	**12**	**12**	**4**	**4**	6	6

(二) 高維度實驗(D=100)

Ho et al. [8]提出的 IEA 演算法在高維度問題求解，其效能表現最為優異，所以本研究特將 IEA 法與本文所研提的 11 種方法進行高維度求解的準確性比較，結果詳見表 7 所示。

將本文研提的 11 個演算法加上 Ho et al. [8]中最優的 IEA 演算法進行在高維度(D=100)搜尋 12 個測試函數的全域極值的求解能力進行排名，結果如表 6 之括弧內數字所示。本文並將之加以平均求取總體能力排名，如表 6 最後兩列所示。實驗證實本研究提出 11 個演算法中，除 TRGA 外，在在高維度(D=100)的全域搜尋能力確能優於 Ho et al. [8]的 IEA 法的表現。

計算效能的比較如表 8 所示，本文研提的改良方法，在已能搜尋到全域極值及經連續演化 200 次未產生更佳解的函數，皆可提早收斂。而反應在疊代次數及時間也能相對的減少。在疊代次數及所需計算時間的表現，統計結果如表 8 所示。

五、結論與後續研究

經 12 個複雜測試函數演算結果顯示，本研究整合「直交表實驗設計」「適應性突變」與「多精英保留」的運算機制，確實可有效改良傳統實數編碼基因演算法的計算準確度及效能，即使在維度高達 100 的情形下亦具有優異的表現。其主

因乃當 TRGA 加入 MEO 後在演化的過程中，因增加了多個精英，提高了精英被選取到的機會，在演化的過程中也增加了收斂至全域極值的機會。而加入直交表實驗設計後，在交配運算中個體中優良的基因確實予以保留，不似傳統基因演算法中交配機制，會因隨機交配運算而流失優良基因。而加入 AMO 後系統會依其個體目前的適應值變化情形，在極值附近作小區域的搜尋，因而提昇整體搜尋至全域極值的能力。即使在高維度問題求解時，系統不但有擁有優良的全域搜尋能力，而且是在極短的時間內完成。

在疊代 12000 代的測試實驗中，本研究改良式直交表實驗設計方法(N-FFD/N-Taguchi)，與前人所提之直交表實驗設計[5,8-9,12]方法，不論全域搜尋能力及計算效能皆有更優良的表現。如表 4 所示。測試結果亦顯示利用全因素實驗設計法(FFD)進行交配運算的改良，確比田口實驗設計法(Taguchi)不論在中維度或高維度問題求解更具成效。

在連續演化持續 200 代便停止演化的實驗中，中維度(D=10)時，除 TRGA 以外，搜尋能力的優劣程度以 TRGA+N-FFD+AMO 最佳。上述除 TRGA 以外的 11 種方法求解準確度皆優於 BOA 和 IEA[8]的方法。在高維度(D=100)時，搜尋能力的優劣程度，也是以 TRGA+N-FFD+AMO 為最佳，。且上述除 TRGA 外的 11 種方法求解準確度亦皆優於 IEA 方法。

在計算效能的比較　　上，在中維度(D=10)時，疊代次數及演算時間的表現，以準確度最高的 TRGA+N-FFD+AMO 法相對的疊代次數較少且演算時間也最少。在高維度(D=100)時，不管是疊代次數或演算時間的表現，都是以 TRGA+N-Taguchi 法為最佳。

本研究改善了基因演算法中三個主要步驟中的[選擇][交配][突變]的演化方式，並經實驗驗證得知其具優異的全域搜尋能力與快速收斂的特性。後續研究擬將其延伸運用在資料探勘相關研究上。

六、致謝

本研究部分成果承蒙國科會專題研究計畫 NSC-98 -2211-E- 040-011 之經費補助，特此感謝。

七、參考文獻

[1] 陳木松、廖鴻翰,「適應性突變運算及其運用」,大業學報，第七卷，第一期，第 91-101 頁,1998。

[2] 陳聖哲、曾明性,「調適型基因演算法於知識規則探勘之研究」,第十一屆人工智慧與應用研討會，高雄應用科技大學, 2006。

[3] 陳聖哲、曾明性,「基於細胞分裂之智慧型雙重基因演算法在知識發掘之研究」,第十二資訊管理暨實務研討會,虎尾科技大學, 2006。

[4] 湯秉宏、曾明性,「適應性突變及多精英保留於高維度參數最佳化之研究」,第十四屆人工智慧與應用研討會，朝陽科技大學, 2009。

[5] Bhote, K. R., World Class Quality: Using Design of Experiments to make it Happen, American Management Association Press, New York, 1991.

[6] Coley, D. A., An Introduction to Genetic Algorithms for Scientists and Engineers, World Scientific, London, 1998.

[7] Goldberg, D. E., Richardson, J., “Genetic algorithms with sharing for multimodal function optimization. In:editors (Ed.)”, Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, Cambridge, Massachusetts, United States, pp. 41–49,1987.

[8] Ho, S. Y., Shu L.S., and Chen, J.-H."Intelligent Evolutionary Algorithms for Large Parameter Optimization Problems," IEEE Trans. Evolutionary Computation, vol. 8, no. 6, pp. 522-541, 2004.

[9] Liu, J.L.,"Intelligent Genetic Algorithm and Its Application to Aerodynamic Optimization of Airplanes," AIAA Journal, vol. 43, no. 3, pp. 530-538, 2005.

[10] Srinivas, M., Patnaik, L.M., "Adaptive probabilities of crossover and mutation in genetic algorithms",IEEE Transactions on Systems, Man and Cybernetics, Vol. 24 No.4, pp.656-67,1994.

[11] Tang, P. H. and Tseng, M. H. “Medical data mining using BGA and RGA for weighting of features in fuzzy k-NN classification.”IEEE,the International Conference on Machine Learning and Cybernetics (ICMLC 2009), Hebei, China,2009.

[12] Tsai, J. T., T. K. Liu, and J. H. Chou, “Hybrid Taguchi-Genetic Algorithm for Global Numerical Optimization,” IEEE Trans. On Evolutionary Computation, Vol. 8, pp. 365-377, 2004.

[13] Tseng, M. H., Chen, S. J., Hwang, G. H., Shen, M. Y. “A genetic algorithm rule-based approach for land-cover classification”, ISPRS Journal of Photogrammetry and Remote Sensing, Vol 63,.pp. 202-212, 2008.

[14] Tseng, M. H., Liao H. L., “The genetic algorithm for breast tumor diagnosis- The case of DNA viruses”, Applied Soft Computing, Vol 9, pp. 703-710, 2009.

表 3 測試函數表

Test Function	
$f_1 = \sum_{i=1}^{D} \left\lvert \frac{\sin(10x_i\pi)}{10x_i\pi} \right\rvert$ x_i domain：$[-0.5,0.5]$ Optimum ：0(min)	$f_7 = -\sum_{i=1}^{D} \left[\sin(x_i) + \sin\left(\frac{2x_i}{3}\right) \right]$ x_i domain：$[3,13]$ Optimum：≈1.21598D(max)
$f_2 = \sum_{i=1}^{D-1} \left[\sin(x_i + x_{i+1}) + \sin\left(\frac{2x_i x_{i+1}}{3}\right) \right]$ x_i domain： $[3,13]$ Optimum ：≈2D(max)	$f_8 = 20 + e - 20e^{-0.2\sqrt{\frac{\sum_{i=1}^{D} {x_i}^2}{D}}} - e^{\frac{\sum_{i=1}^{D} \cos(2\pi x_i)}{D}}$ x_i domain：$[-30,30]$ Optimum ：≈0(min)
$f_3 = \sum_{i=1}^{D} \lfloor x_i + 0.5 \rfloor^2$ x_i domain：$[-100,100]$ Optimum ：0(min)	$f_9 = 418.9828N - \sum_{i=1}^{D} x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$ x_i domain：$[-500,500]$ Optimum ：0(min)
$f_4 = \sum_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i) + 10]$ x_i domain：$[-5.12,5.12]$ Optimum ：0(min)	$f_{10} = \sum_{i=1}^{D-1} [100({x_{i+1}} - {x_i}^2)^2 - (x_i - 1)^2]$ x_i domain：$[-5.12,5.12]$ Optimum ：0(min)
$f_5 = \sum_{i=1}^{D} {x_i}^2$ x_i domain：$[-5.12,5.12]$ Optimum ：0(min)	$f_{11} = 6D + \sum_{i=1}^{D} \lfloor x_i \rfloor$ x_i domain：$[-5.12,5.12]$ Optimum ：0(min)
$f_6 = \sum_{i=1}^{D} \left(x_i \sin(10\pi x_i)\right)$ x_i domain：$[-1.0,2.0]$ Optimum ：≈1.85D(max)	$f_{12} = \frac{1}{4000} \sum_{i=1}^{D} {x_i}^2 - \prod_{i=1}^{D} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$ x_i domain：$[-600,600]$ Optimum ：0(min)

表 4 直交表實驗設計改良前後比較表(疊代 12000 代)

n = 100		IEA[8]	TRGA		TRGA	
			FFD	Taguchi	N-FFD	N-Taguchi
f_1 (min)	Fitness	0.65(3)	0.0026(2)	1.2013(4)	0.0018(1)	1.5756(5)
	Iteration	X	12000(1)	12000(1)	12000(1)	12000(1)
	CPU Time(s)	X	895.62(2)	1175.89(3)	894.44(1)	1204.85(4)
f_2 (max)	Fitness	153.15(5)	186.8877(3)	186.7815(4)	188.1909(2)	188.6635(1)
	Iteration	X	12000(1)	12000(1)	12000(1)	12000(1)
	CPU Time(s)	X	998.3(2)	1030.45(4)	981.47(1)	1026.26(3)
f_3 (min)	Fitness	621(5)	0(1)	0(1)	0(1)	0(1)
	Iteration	X	815(4)	777(3)	328(2)	317(1)
	CPU Time(s)	X	72.93(3)	74.74(4)	29.8(1)	30.81(2)
f_4 (min)	Fitness	213.46(5)	0.0176(4)	0.0163(3)	0.0079(2)	0.0043(1)
	Iteration	X	12000(1)	12000(1)	12000(1)	12000(1)
	CPU Time(s)	X	1105.3(1)	1130.27(3)	1107.04(2)	1160.67(4)
f_5 (min)	Fitness	1.6(5)	9.88E-05(3)	1.5 E-05(1)	6.28 E-05(2)	1.34 E-04(4)
	Iteration	X	12000(1)	12000(1)	12000(1)	12000(1)
	CPU Time(s)	X	1033.73(1)	1065.05(3)	1033.98(2)	1082.87(4)
f_6 (max)	Fitness	131.31(5)	185.0219(3)	185.021(4)	185.0252(1)	185.0249(2)
	Iteration	X	12000(1)	12000(1)	12000(1)	12000(1)
	CPU Time(s)	X	878.92(1)	891.72(4)	889.89(3)	885.36(2)
f_7 (max)	Fitness	120.44(5)	121.5981(1)	121.598(3)	121.5981(1)	121.598(3)
	Iteration	X	12000(1)	12000(1)	12000(1)	12000(1)
	CPU Time(s)	X	1001.69(1)	1018.31(3)	1004.22(2)	1029.67(4)
f_8 (min)	Fitness	3.69(5)	0.1971(2)	0.1803(1)	0.6328(4)	0.5833(3)
	Iteration	X	12000(1)	12000(1)	12000(1)	12000(1)
	CPU Time(s)	X	1102.99(1)	1151.38(3)	1108.42(2)	1157.02(4)
f_9 (min)	Fitness	8011(5)	0.0903(4)	0.0857(3)	0.034(2)	0.0291(1)
	Iteration	X	12000(1)	12000(1)	12000(1)	12000(1)
	CPU Time(s)	X	896.44(2)	916.16(3)	895.28(1)	919.4(4)
f_{10} (min)	Fitness	2081(5)	89.8287(3)	116.2926(4)	68.7907(2)	31.4564(1)
	Iteration	X	12000(1)	12000(1)	12000(1)	12000(1)
	CPU Time(s)	X	1470.34(2)	1537.7(3)	1453.07(1)	1557.77(4)
f_{11} (min)	Fitness	43.94(5)	0(1)	0(1)	0(1)	0(1)
	Iteration	X	692(4)	589(3)	264(2)	240(1)
	CPU Time(s)	X	50.32(4)	47.16(3)	19.18(2)	18.09(1)
f_{12} (min)	Fitness	32.86(5)	0.173(1)	0.186(2)	0.2284(3)	0.2336(4)
	Iteration	X	12000(1)	12000(1)	12000(1)	12000(1)
	CPU Time(s)	X	1149.23(2)	1163.59(3)	1115.9(1)	1181.93(4)
Averaged rank	Fitness	4.83	2.33	2.58	1.83	2.25
	Iteration	X	1.5	1.33	1.17	1
	CPU Time(s)	X	1.83	3.25	1.58	3.33
Final rank	Fitness	5	3	4	**1**	2
	Iteration	X	4	3	2	1
	CPU Time(s)	X	2	3	**1**	4

表 5 中維度(D=10)實驗結果(連續演化200代未產生更好解即終止演算)

n = 10		IEA*[8]	BOA*[8]	TRGA	TRGA		TRGA	
					FFD	Taguchi	N-FFD	N-Taguchi
f_1 (min)	Fitness	0.054(12)	0.019(9)	0.1495(13)	0.0018(6)	0.0143(8)	7.48E-04(3)	0.0489(11)
	Iteration	X	X	892(4)	1108(6)	1112(7)	1154(9)	1073(5)
	CPU Time(s)	X	X	2.22(3)	4.23(5)	4.66(8)	4.4(6)	4.63(7)
f_2 (max)	Fitness	15.32(11)	12.29(13)	13.8952(12)	17.15385(9)	17.0365(10)	17.4884(4)	17.6207(1)
	Iteration	X	X	548(1)	1267(3)	1265(2)	1479(6)	1400(4)
	CPU Time(s)	X	X	1.39(1)	5.13(3)	5.1(2)	6.04(5)	5.87(4)
f_3 (min)	Fitness	5.13(12)	0.77(11)	101.6667(13)	0.2(10)	0.1(9)	0(1)	0(1)
	Iteration	X	X	627(11)	295(10)	259(9)	88(5)	102(7)
	CPU Time(s)	X	X	1.61(11)	1.23(10)	1.13(9)	0.36(5)	0.39(6)
f_4 (min)	Fitness	15.42(12)	5.32(11)	24.6499(13)	0.1899(10)	0.1577(9)	0.016(7)	0.0117(5)
	Iteration	X	X	533(5)	964(6)	1036(7)	1116(9)	1056(8)
	CPU Time(s)	X	X	1.39(1)	4.1(6)	4.22(7)	4.64(9)	4.37(8)
f_5 (min)	Fitness	0.0003(9)	0.0077(12)	0.2629(13)	6.97E-04(11)	6.42E-04(10)	8.99E-05(6)	5.63E-05(5)
	Iteration	X	X	697(1)	1153(8)	1085(3)	1109(5)	1099(4)
	CPU Time(s)	X	X	1.72(1)	4.52(6)	4.25(2)	4.44(3)	4.49(5)
f_6 (max)	Fitness	14.6(12)	18.1(11)	14.2597(13)	18.45513(9)	18.4446(10)	18.494(7)	18.4976(5)
	Iteration	X	X	1012(3)	1104(9)	1075(7)	1094(8)	1028(4)
	CPU Time(s)	X	X	2.46(1)	4.29(5)	4.09(2)	4.37(7)	4.34(6)
f_7 (max)	Fitness	12.116(12)	12.151(11)	11.9569(13)	12.1595(9)	12.15946(10)	12.15978(6)	12.1598(3)
	Iteration	X	X	643(1)	1083(10)	937(6)	1105(11)	1053(9)
	CPU Time(s)	X	X	1.71(1)	4.4(9)	4.07(6)	462(11)	4.35(8)
f_8 (min)	Fitness	1(12)	0.93(11)	5.9605(13)	0.3016(9)	0.3364(10)	0.1314(6)	0.1284(5)
	Iteration	X	X	565(1)	1117(7)	1124(9)	1108(5)	1087(2)
	CPU Time(s)	X	X	1.49(1)	4.39(2)	4.68(8)	4.5(3)	4.57(5)
f_9 (min)	Fitness	667.4(12)	8.4(11)	915.8229(13)	0.452(9)	0.8119(10)	0.1033(7)	0.0774(5)
	Iteration	X	X	870(2)	1192(11)	1030(6)	1093(9)	1087(8)
	CPU Time(s)	X	X	2.16(1)	4.35(10)	3.92(6)	4.04(7)	4.25(9)
f_{10} (min)	Fitness	116.44(13)	8.93(11)	30.4121(12)	2.3253(8)	2.8886(10)	0.7908(4)	1.1469(5)
	Iteration	X	X	676(1)	1585(7)	1503(5)	1181(2)	1476(4)
	CPU Time(s)	X	X	1.77(1)	6.96(6)	6.56(4)	5.33(2)	6.73(5)
f_{11} (min)	Fitness	0.338(11)	10.067(12)	10.4667(13)	0.1333(10)	0.1(9)	0(1)	0(1)
	Iteration	X	X	503(11)	225(9)	232(10)	75(6)	66(5)
	CPU Time(s)	X	X	1.28(11)	0.9(9)	0.93(10)	0.28(7)	0.26(5)
f_{12} (min)	Fitness	0.999(11)	1.008(12)	2.1926(13)	0.2598(9)	0.2696(10)	0.2091(8)	0.1401(5)
	Iteration	X	X	511(1)	1216(10)	1164(9)	1056(6)	1277(11)
	CPU Time(s)	X	X	1.34(1)	4.93(10)	4.53(7)	4.38(6)	5.29(11)
Averaged rank	Fitness	11.58	11.25	12.83	9.08	9.58	5	4.33
	Iteration	X	X	3.5	8	6.67	6.75	5.92
	CPU Time(s)	X	X	2.83	6.75	5.92	5.92	6.58
Final rank	Fitness	12	11	13	9	10	6	5
	Iteration	X	X	**1**	11	7	8	4
	CPU Time(s)	X	X	**1**	9	4	4	8

* IEA 及 BOA 演算終止條件為疊代 12000 代

表 5 中維度(D=10)實驗結果(連續演化200代未產生更好解即終止演算)(續)

n = 10		TRGA+MEO		TRGA+AMO		TRGA++MEO+AMO	
		N-FFD	N-Taguchi	N-FFD	N-Taguchi	N-FFD	N-Taguchi
f_1 (min)	Fitness	8.23E-04(4)	0.0107(7)	3.9E-16(1)	0.0357(10)	3.9E-16(1)	0.0016(5)
	Iteration	1208(11)	1193(10)	285(2)	1120(8)	276(1)	624(3)
	CPU Time(s)	4.73(9)	5.17(10)	1.06(1)	5.17(10)	1.08(2)	2.84(4)
f_2 (max)	Fitness	17.43915(5)	17.4138(7)	17.5666(2)	17.4177(6)	17.3722(8)	17.5073(3)
	Iteration	1563(7)	1427(5)	2784(11)	2378(9)	2288(8)	2597(10)
	CPU Time(s)	6.53(7)	6.08(6)	11.44(10)	10.51(9)	9.62(8)	11.64(11)
f_3 (min)	Fitness	0(1)	0(1)	0(1)	0(1)	0(1)	0(1)
	Iteration	95(6)	116(8)	45(1)	49(2)	49(2)	53(4)
	CPU Time(s)	0.39(6)	0.45(8)	0.17(1)	0.21(3)	0.2(2)	0.23(4)
f_4 (min)	Fitness	0.0155(6)	0.0195(8)	0(1)	0(1)	0(1)	0(1)
	Iteration	1284(11)	1203(10)	448(1)	490(3)	471(2)	502(4)
	CPU Time(s)	5.16(11)	5(10)	1.72(2)	2.15(4)	1.99(3)	2.39(5)
f_5 (min)	Fitness	1.19E-04(8)	1.03E-04(7)	0(1)	0(1)	0(1)	0(1)
	Iteration	1129(6)	1073(2)	1249(9)	1395(11)	1141(7)	1274(10)
	CPU Time(s)	4.55(7)	4.45(4)	5.1(8)	6.22(11)	5.35(9)	5.96(10)
f_6 (max)	Fitness	18.4953(6)	18.49346(8)	18.5015(1)	18.50148(2)	18.5007(4)	18.5012(3)
	Iteration	1174(11)	1137(10)	991(1)	1007(2)	1034(5)	1065(6)
	CPU Time(s)	4.89(11)	4.83(10)	4.14(3)	4.22(4)	4.58(9)	4.56(8)
f_7 (max)	Fitness	12.15976(8)	12.15978(6)	12.15981(2)	12.1598(3)	12.15979(5)	12.159814(1)
	Iteration	997(7)	1040(8)	803(2)	846(3)	857(4)	927(5)
	CPU Time(s)	4.24(7)	4.4(9)	3.32(3)	3.31(2)	3.85(4)	3.87(5)
f_8 (min)	Fitness	0.1755(8)	0.1695(7)	-3.24E-07(1)	-3.24E-07(1)	-3.24E-07(1)	-3.24E-07(1)
	Iteration	1094(3)	1121(8)	1114(6)	1260(11)	1096(4)	1161(10)
	CPU Time(s)	4.62(7)	4.71(9)	4.56(4)	4.99(11)	4.61(6)	4.78(10)
f_9 (min)	Fitness	0.1431(8)	0.0815(6)	0.0511(4)	0.0285(1)	0.0492(3)	0.04(2)
	Iteration	1125(10)	1069(7)	844(1)	965(5)	925(4)	916(3)
	CPU Time(s)	4.5(11)	4.2(8)	3.44(2)	3.74(3)	3.85(5)	3.83(4)
f_{10} (min)	Fitness	1.2389(6)	2.3837(9)	0.763(2)	0.7631(3)	0.6997(1)	1.2674(7)
	Iteration	1407(3)	1571(6)	3257(11)	2845(10)	1662(8)	2794(9)
	CPU Time(s)	6.31(3)	6.98(7)	14.93(11)	11.89(10)	7.81(8)	11.25(9)
f_{11} (min)	Fitness	0(1)	0(1)	0(1)	0(1)	0(1)	0(1)
	Iteration	77(7)	81(8)	33(2)	35(3)	32(1)	36(4)
	CPU Time(s)	0.27(6)	0.31(8)	0.13(2)	0.14(3)	0.12(1)	0.14(3)
f_{12} (min)	Fitness	0.1721(6)	0.1849(7)	0.0277(2)	0.019(1)	0.0339(4)	0.0327(3)
	Iteration	1116(7)	1124(8)	617(2)	751(5)	620(3)	655(4)
	CPU Time(s)	4.64(8)	4.71(9)	2.51(2)	3(5)	2.61(4)	2.53(3)
Averaged rank	Fitness	5.58	6.17	1.58	2.58	2.58	2.42
	Iteration	7.42	7.5	4.08	6	4.08	6
	CPU Time(s)	7.75	8.17	4.08	6.25	5.08	6.33
Final rank	Fitness	7	8	**1**	3	3	2
	Iteration	9	10	2	5	2	5
	CPU Time(s)	10	11	2	6	3	7

表 6 高維度(D=100)實驗結果(連續演化200代未產生更好解即終止演算)

n = 100		IEA*[8]	TRGA	TRGA		TRGA	
				FFD	Taguchi	N-FFD	N-Taguchi
f_1 (min)	Fitness	0.65(7)	4.0276(12)	0.0056(3)	2.279(9)	0.0071(4)	2.95415(10)
	Iteration	X	2261(3)	5677(11)	2762(6)	4146(9)	2426(4)
	CPU Time(s)	X	21.53(1)	427.39(11)	268.58(6)	304(7)	236.07(4)
f_2 (max)	Fitness	153.15(11)	81.5926(12)	186.5417(6)	186.852(5)	185.51305(8)	185.7785(7)
	Iteration	X	1246(1)	10636(7)	8928(6)	2935(4)	1605(2)
	CPU Time(s)	X	12.27(1)	870.05(7)	753.18(6)	240.71(4)	133.46(2)
f_3 (min)	Fitness	621(11)	50993.54(12)	0.6(8)	0.1333(5)	1.4333(9)	6.7(10)
	Iteration	X	680(9)	720(11)	703(10)	325(6)	288(5)
	CPU Time(s)	X	6.74(2)	64.88(10)	66.68(11)	29.33(7)	26.55(6)
f_4 (min)	Fitness	213.46(11)	867.1523(12)	0.229(5)	0.2986(7)	0.4486(9)	0.5382(10)
	Iteration	X	1005(5)	3888(8)	4160(10)	3717(7)	2398(6)
	CPU Time(s)	X	9.75(1)	358.5(8)	393.3(10)	337.96(7)	218.31(6)
f_5 (min)	Fitness	1.6(11)	130.8696(12)	0.0011(5)	0.0014(6)	0.01385(10)	0.0049(9)
	Iteration	X	785(1)	3816(10)	3469(9)	3317(8)	1843(2)
	CPU Time(s)	X	7.57(1)	336.44(11)	308.93(9)	284.04(8)	159.6(2)
f_6 (max)	Fitness	131.31(11)	78.0294(12)	184.9553(6)	184.9666(5)	184.88492(8)	184.8557(9)
	Iteration	X	2286(2)	3826(5)	4230(7)	3173(3)	2197(1)
	CPU Time(s)	X	21.47(1)	287.03(5)	325.03(7)	238.4(3)	163.86(2)
f_7 (max)	Fitness	120.44(11)	84.6326(12)	121.5975(5)	121.5973(6)	121.59714(8)	121.5959(9)
	Iteration	X	889(1)	3208(11)	3016(9)	2284(8)	1753(2)
	CPU Time(s)	X	8.98(1)	275.83(11)	261.51(10)	192.88(7)	148.37(2)
f_8 (min)	Fitness	3.69(11)	15.3168(12)	1.6506(6)	1.5293(5)	2.372834(8)	2.273(7)
	Iteration	X	801(1)	2650(10)	3046(11)	1859(5)	1812(4)
	CPU Time(s)	X	8.06(1)	256.04(10)	301.4(11)	173.43(5)	170.23(4)
f_9 (min)	Fitness	8011(11)	21384.04(12)	1.1917(6)	1.3683(9)	0.76109(5)	7.2858(10)
	Iteration	X	1957(1)	4265(11)	3567(8)	3531(7)	2065(2)
	CPU Time(s)	X	17.77(1)	327.851(11)	274.73(9)	266.79(7)	155.88(2)
f_{10} (min)	Fitness	2081(11)	63762.02(12)	106.8198(6)	84.1844(5)	109.93173(7)	171.5293(10)
	Iteration	X	660(1)	6846(7)	6084(6)	2225(3)	925(2)
	CPU Time(s)	X	6.99(1)	833.82(7)	773.69(6)	272.06(3)	113.61(2)
f_{11} (min)	Fitness	43.94(11)	225.7333(12)	0.6333(10)	0.5(9)	0.17(7)	0.1333(5)
	Iteration	X	867(11)	593(9)	630(10)	235(5)	274(6)
	CPU Time(s)	X	8.08(5)	44.69(10)	48.06(11)	18.38(6)	19.83(7)
f_{12} (min)	Fitness	32.86(11)	461.9481(12)	0.7854(7)	0.7446(6)	0.80125(8)	0.7382(5)
	Iteration	X	727(1)	2216(10)	2347(11)	1809(8)	1711(6)
	CPU Time(s)	X	7.23(1)	211.65(10)	230.5(11)	170.49(8)	162.91(6)
Averaged rank	Fitness	10.67	12	6.08	6.42	7.58	8.42
	Iteration	X	3.08	9.17	8.58	6.08	3.5
	CPU Time(s)	X	1.42	9.25	8.92	6	3.75
Final rank	Fitness	11	12	5	6	8	9
	Iteration	X	**1**	11	9	7	2
	CPU Time(s)	X	**1**	11	10	7	2

*IEA演算終止條件為疊代12000代

表 6 高維度(D=100)實驗結果(連續演化200代未產生更好解即終止演算)(續)

n = 100		TRGA+MEO		TRGA+AMO		TRGA++MEO+AMO	
		N-FFD	N-Taguchi	N-FFD	N-Taguchi	N-FFD	N-Taguchi
f_1 (min)	Fitness	0.0084(5)	1.3067(8)	3.58E-04(2)	3.0756(11)	3.27E-04(1)	0.4006(6)
	Iteration	4415(10)	3395(7)	629(2)	2443(5)	603(1)	3914(8)
	CPU Time(s)	327.37(8)	330.74(9)	47.72(3)	251.1(5)	46.68(2)	391.2(10)
f_2 (max)	Fitness	184.8029(9)	184.2642(10)	187.6898(3)	187.8815(1)	187.689(4)	187.7572(2)
	Iteration	4570(5)	2762(3)	11985(10)	11754(8)	12000(11)	11974(9)
	CPU Time(s)	381.37(5)	235.26(3)	1017.96(10)	1013.71(8)	1015.99(9)	1048.03(11)
f_3 (min)	Fitness	0.27(6)	0.367(7)	0(1)	0(1)	0(1)	0(1)
	Iteration	552(8)	536(7)	80(3)	70(1)	83(4)	78(2)
	CPU Time(s)	50.07(9)	49.52(8)	7.45(3)	6.55(1)	7.63(5)	7.53(4)
f_4 (min)	Fitness	0.3038(8)	0.289(6)	0(1)	0(1)	0(1)	0(1)
	Iteration	4320(11)	3957(9)	906(3)	988(4)	779(1)	809(2)
	CPU Time(s)	401.27(11)	368.63(9)	86.19(4)	93.99(5)	73.5(2)	77.68(3)
f_5 (min)	Fitness	0.0014(6)	0.0021(8)	0(1)	0(1)	0(1)	0(1)
	Iteration	3858(11)	2740(6)	2244(5)	2859(7)	1845(3)	2110(4)
	CPU Time(s)	334.52(10)	237.78(6)	216.56(5)	274.54(7)	179.46(3)	200.15(4)
f_6 (max)	Fitness	184.8876(7)	184.684(10)	185.09155(1)	185.0239(3)	185.0172(4)	185.02483(2)
	Iteration	4521(11)	3541(4)	4480(10)	4238(8)	4354(9)	4019(6)
	CPU Time(s)	337.4(10)	264.63(4)	344.21(11)	328.1(8)	332.59(9)	311.25(6)
f_7 (max)	Fitness	121.59723(7)	121.59167(10)	121.59783(3)	121.59798(1)	121.59773(4)	121.59798(1)
	Iteration	3083(10)	2159(3)	2215(6)	2192(5)	2234(7)	2172(4)
	CPU Time(s)	260.62(9)	183.43(3)	191.68(5)	190.33(4)	192.47(6)	194.73(8)
f_8 (min)	Fitness	3.4079(9)	3.5068(10)	1.78E-06(1)	1.78E-06(1)	1.78E-06(1)	1.78E-06(1)
	Iteration	2022(8)	1928(6)	1762(3)	2595(9)	1492(2)	1943(7)
	CPU Time(s)	188.28(8)	182.34(6)	164.81(3)	248.13(9)	140.64(2)	187.71(7)
f_9 (min)	Fitness	1.337526(8)	1.21969(7)	0.1254(2)	0.1013(1)	0.2327(4)	0.1715(3)
	Iteration	3813(10)	3660(9)	2932(5)	2909(4)	2934(6)	2832(3)
	CPU Time(s)	281.02(10)	274.28(8)	228.05(4)	228.22(5)	229.04(6)	222.43(3)
f_{10} (min)	Fitness	140.7014(8)	141.0401(9)	72.2137(4)	47.675(1)	55.8744(2)	58.7481(3)
	Iteration	3513(5)	2843(4)	11924(11)	10159(8)	11749(10)	11280(9)
	CPU Time(s)	423.85(5)	343.93(4)	1481.68(11)	1269.39(8)	1455.5(10)	1381.3(9)
f_{11} (min)	Fitness	0.1333(5)	0.2(8)	0(1)	0(1)	0(1)	0(1)
	Iteration	312(7)	362(8)	74(2)	70(1)	80(4)	75(3)
	CPU Time(s)	22.07(8)	26.24(9)	5.76(3)	5.35(1)	6.04(4)	5.7(2)
f_{12} (min)	Fitness	0.9289(9)	0.9297(10)	0.0013(2)	1.33E-06(1)	0.0136(4)	0.0054(3)
	Iteration	2108(9)	1756(7)	924(4)	1048(5)	796(2)	878(3)
	CPU Time(s)	198.76(9)	166.89(7)	92.44(4)	102.38(5)	77.36(2)	84.59(3)
Averaged rank	Fitness	7.25	8.58	1.83	2	2.33	2.08
	Iteration	8.75	6.08	5.33	5.42	5	5
	CPU Time(s)	8.5	6.33	5.5	5.5	5	5.83
Final rank	Fitness	7	10	**1**	2	4	3
	Iteration	10	7	5	6	3	3
	CPU Time(s)	9	8	4	4	3	6