超啟發式演算法

軟創三乙 511172176 李則霖

簡述

利用學術文章了解現有超啟發演算在全域搜尋上的解決方案，並且引用學術文章作為來源，嘗試使用生程式AI試著給出缺點的解決方案，並彙整成IEEE標準格式後產出檔案。

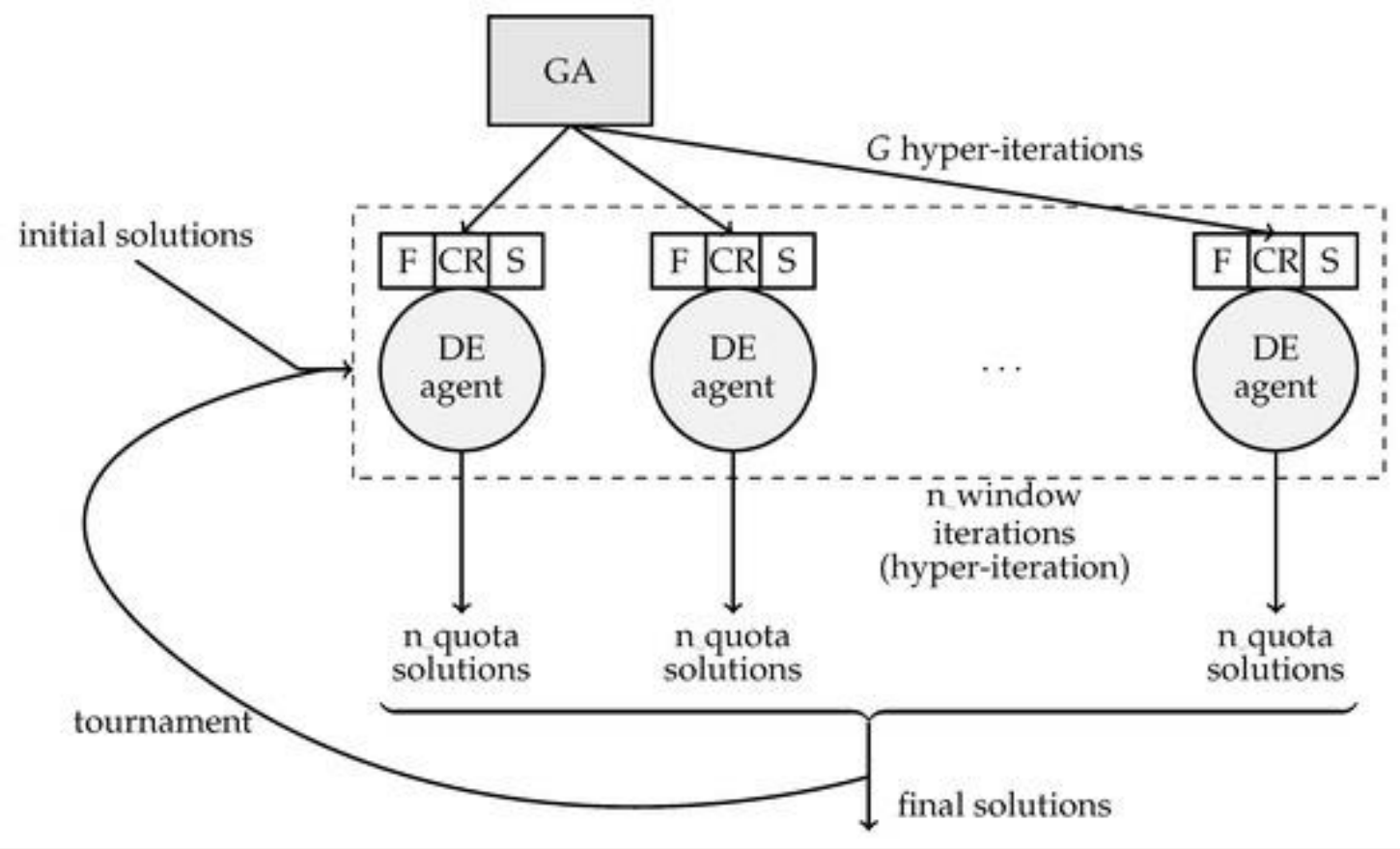
1. 研究動機

2. 實驗方法與解決機制

3. 實驗結果與分析

4. 總結

超差分進化演算法
（ Hyper Differential Evolution ）



Ma, Z., Wu, G., Suganthan, P.N., Song, A., & Luo, Q. (2023). HyperDE: A Hyper-Heuristic Differential Evolution Algorithm for Global Optimization. *Algorithms*, 16(451), 1-18. https://doi.org/10.3390/a16040451

研究動機

本研究的動機在於針對傳統DE和現有Hyper DE的不足，提出一系列改進策略，目的在增強演算法的全局搜索效率和解的質量。研究探討了引入多樣性維護機制、自適應參數控制及混合策略來提升Hyper DE的性能，並通過多個基準測試問題的實驗比較，驗證所提改進方案的有效性。希望此研究能夠為解決複雜優化問題提供更高效的工具，並試著為未來優化演算法的開發提供有益參考。

實驗設計

1. **超參數選擇：**
   - ST：在0到1的範圍內調整。
   - PD：在0到1的範圍內調整。
   - SD：在0到5的範圍內調整。
2. **超迭代次數：**
   - **第一組實驗**：進行40次超迭代（hyper-iteration）。
   - **第二組實驗**：進行80次超迭代，以觀察更長時間內超參數的變化和優化趨勢。
3. **數據收集：**
   - 在每次超迭代中，記錄三個超參數的取值和對應的模型性能指標。
   - 使用散點圖來可視化超參數值的變化，藍色點表示每次迭代的參數值，紅色點表示當前最佳的參數值。

實驗步驟

1. **初始化**：設定初始超參數值和模型。選擇兩種不同的超參數優化算法或策略作為比較對象（例如，隨機搜索和貝葉斯優化）。

2. **超參數優化過程**：

   迭代優化：

   - **第一組實驗**：
     - 使用第一種算法進行40次超迭代。
     - 在每次迭代中，根據算法策略調整超參數，並評估模型性能。
     - 記錄每次迭代的超參數值和性能指標。
   - **第二組實驗**：
     - 使用第二種算法進行80次超迭代。
     - 調整超參數並評估性能，記錄相關數據。

3. **結果記錄與可視化**：
   - 將每次迭代的超參數值以散點圖形式展示，觀察其分佈和變化趨勢。
   - 標示出每次迭代中的最佳超參數值，以便比較不同算法的優化效果。

實驗產出

1. 結果分析：

比較優化效率：

- 通過觀察兩組實驗中紅色點的分佈，評估哪種算法更快地找到最佳超參數組合。
- 分析超參數值的收斂情況，判斷算法的穩定性和可靠性。
- 模型性能評估：比較兩組實驗中模型性能指標的變化，確定哪種算法能夠提升模型的整體性能。

2. 做出結論：

- 根據實驗結果，總結哪種超參數優化算法在本次實驗中表現更佳。
- 提出可能的改進方向或進一步研究的建議。

解決機制

1. **並行處理**：當種群規模較大或目標函數計算複雜時，使用多進程並行計算適應度值，提升效率；否則，採用單線程方式計算。

2. **自適應調整參數**：根據當前代數動態調整變異基因數量和粒子配額數量，通過回調函數在每代結束時自動執行這些調整，保持算法的探索與利用平衡。

3. **早停策略**：監控最佳解的改進情況，若連續多代未見改進，則提前終止算法運行，避免不必要的計算，提升資源利用效率。

並行處理

程式邏輯：

- **種群大小檢查：**判斷當前種群的大小是否超過50。如果超過，則採用並行處理；否則，使用單線程計算。
  - **並行處理：**
    - 使用多進程池（mp.Pool）創建與CPU核心數量相等的工作池。
    - 利用pool.map將目標函數同時應用於種群中的所有個體並收集適應度值。
    - 完成後關閉進程池以釋放資源。
- **單線程處理：**當種群較小時，直接遍歷每個個體，依次計算其適應度值，並存入適應度列表中。

```
IF LENGTH(population) > 50 THEN
    CREATE a pool with number of workers equal to CPU count
    fitness = PARALLEL_MAP(objective_func, population) USING pool
    CLOSE the pool
ELSE
    fitness = []
    FOR EACH individual IN population DO
        fitness_value = objective_func(individual)
        APPEND fitness_value TO fitness
    END FOR
END IF
RETURN fitness
```

自適應參數調整

程式邏輯：

- **參數調整：**
  - 每隔20代，增加變異基因數量（mutation_num_genes），但不超過總基因數量（num_genes）。
  - 同時減少粒子配額數量（n_quota_of_particles），但至少保持為10。
  - 每隔40代，進一步減少粒子配額數量，但至少保持為15。
- **回調函數設置：**
  - 定義一個內部函數inner，在每代結束時調用adjust_parameters方法來動態調整參數。
  - 同時調用inspect函數進行檢查或記錄（具體功能依賴於inspect的實現）。
  - 返回這個內部函數，以便在遺傳算法的每代結束時自動執行。

```
IF generation MOD 20 EQUALS 0 THEN
    INCREASE mutation_num_genes BY 1
    IF mutation_num_genes > num_genes THEN
        SET mutation_num_genes TO num_genes
    END IF
    DECREASE n_quota_of_particles BY 2
    IF n_quota_of_particles < 10 THEN
        SET n_quota_of_particles TO 10
    END IF
ELSE IF generation MOD 40 EQUALS 0 THEN
    DECREASE n_quota_of_particles BY 1
    IF n_quota_of_particles < 15 THEN
        SET n_quota_of_particles TO 15
    END IF
END IF
```

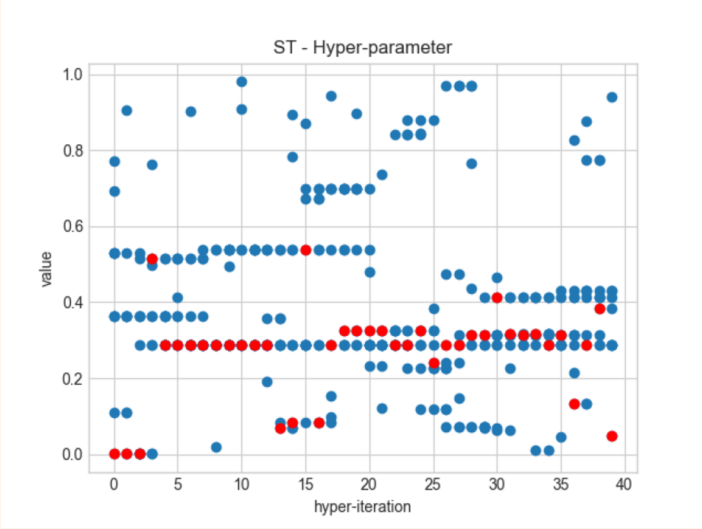
早停策略

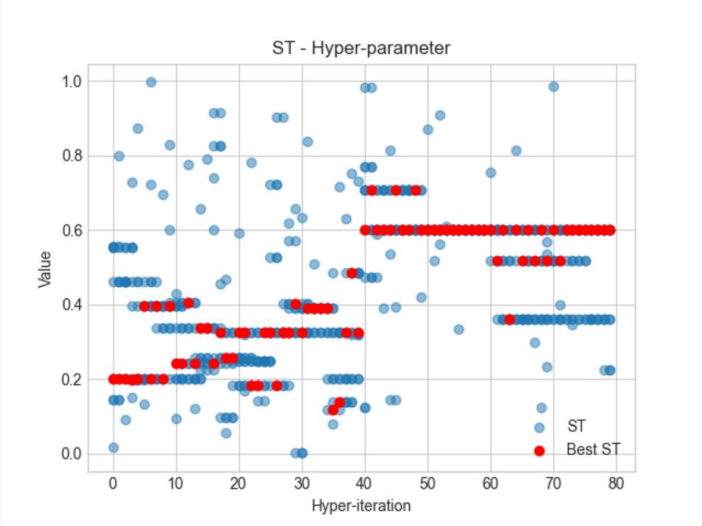
程式邏輯：

- **最佳解更新：**
  - 如果當前的最佳解（best_solution）優於之前記錄的最佳解（self.best_solution）或尚未設置最佳解，則更新最佳解並重置無改進計數（no_improvement_count）為0。
  - 否則，將無改進計數增加1。
- **早停判斷：**
  - 如果無改進計數達到或超過設定的早停閾值（early_stop_threshold），則返回True，表示應該提前終止算法。
  - 否則，返回False，繼續運行算法。

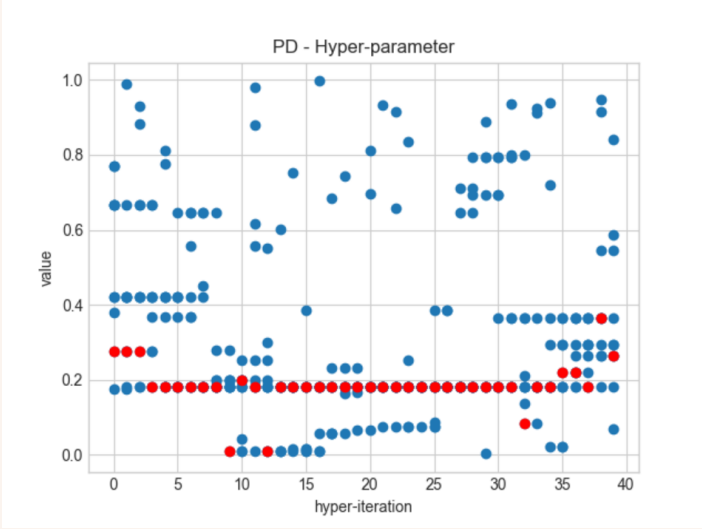
```
DEFINE FUNCTION inner(ga_instance):
    current_generation = ga_instance.generations_completed
    CALL adjust_parameters(current_generation, num_genes)
    CALL inspect(ga_instance)
END FUNCTION
RETURN inner
```

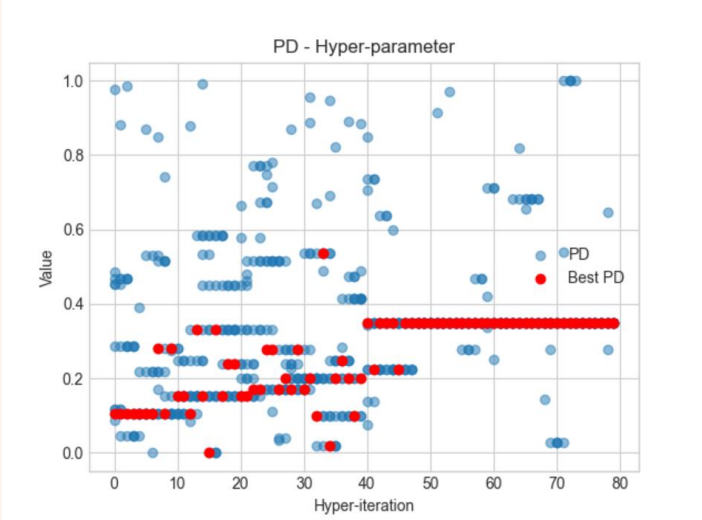
產出數據



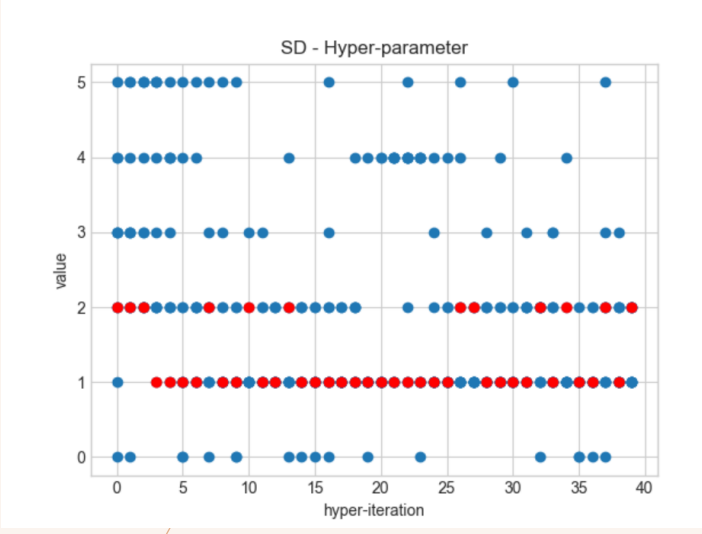


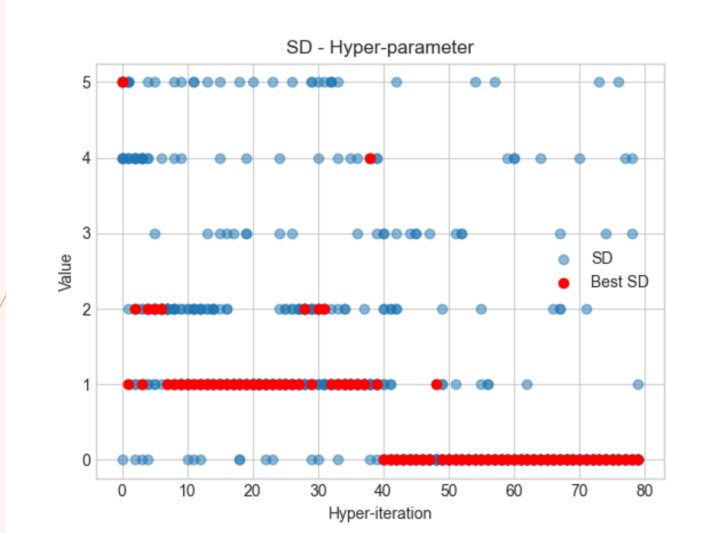
圖一、圖二: ST迭代數據





圖三、圖四: PD迭代數據





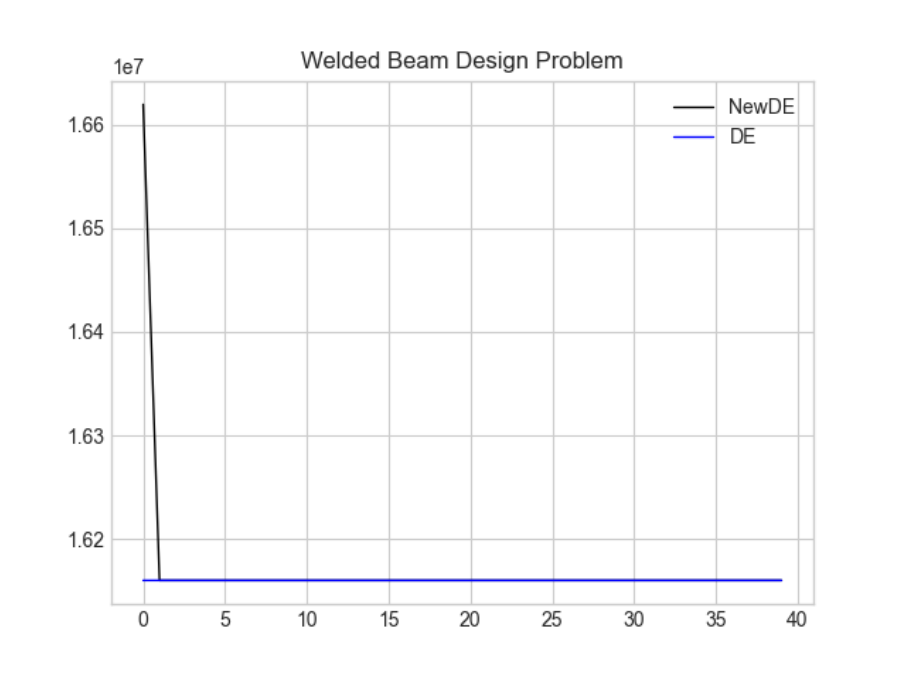
圖五、圖六: SD迭代數據

數據分析

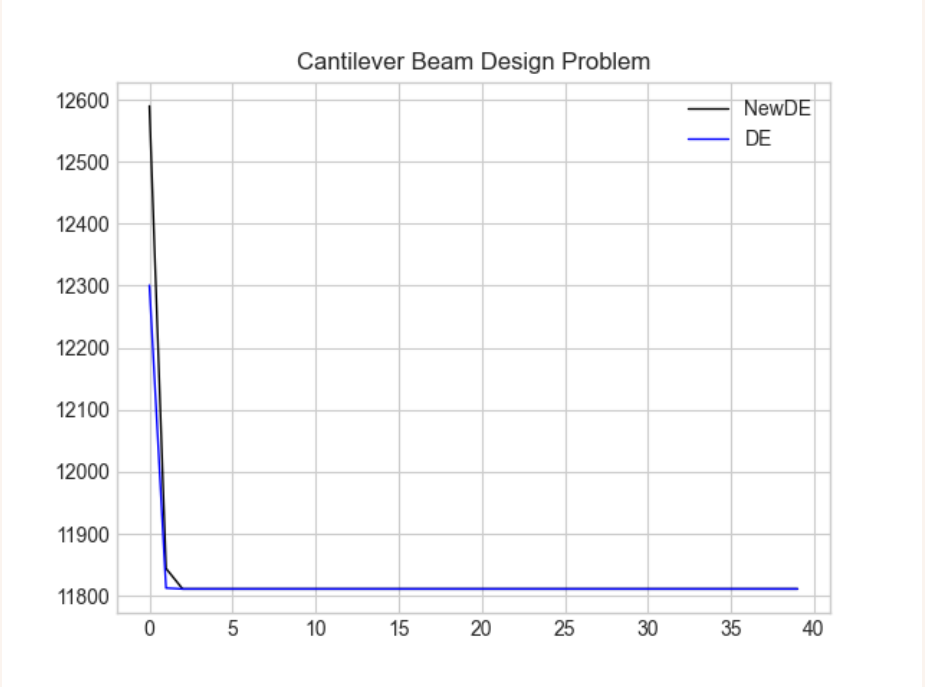
1. **超迭代次數範圍：**
   - 第一組圖的超迭代範圍是從0到40，第二組圖的超迭代範圍擴大到了0到80。這表示在第二組圖中進行了更多次的超參數調整，因此可以觀察到更長時間的趨勢。
2. **紅色點的標示：**
   - 第一組圖中的紅色點相對較少，似乎只是標示一些特定的情況或重要值。
   - 第二組圖中，紅色點變得更多且分佈較為系統，這顯示了該算法在更多次迭代中持續進行了有效的優化，從而能夠多次找到更好的超參數配置。
3. **趨勢和參數變化的表現：**
   - 在第二組圖中，紅色點較多且更有系統性地分佈，顯示出隨著超迭代次數的增加，參數的最佳化過程。這與第一組圖中的點分佈相較，更容易觀察到參數的優化趨勢。

收斂速度比較

收斂速度比較從圖一至圖二中可以觀察到，在Welded Beam設計問題與Cantilever Beam設計問題中，傳統DE演算法（圖中的藍線）在收斂速度上明顯優於NewDE演算法（圖中的黑線）。雖然NewDE在初期迭代中迅速下降，但其下降幅度明顯不如DE，在大多數問題上，NewDE的收斂趨勢很快停滯，最終穩定在一個較高的目標函數值上，表現劣於DE。



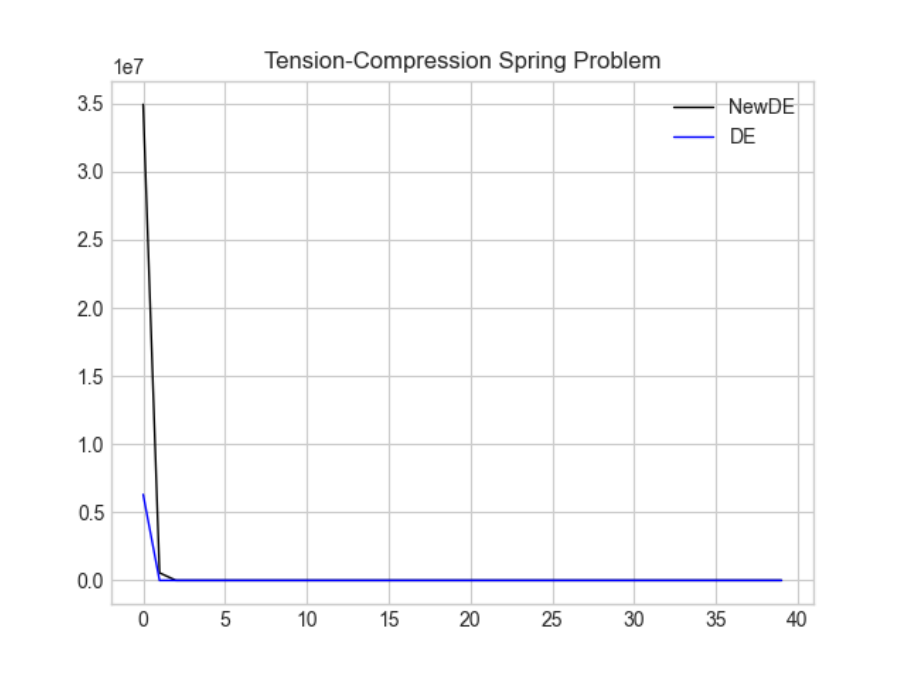
圖一：焊接梁設計問題



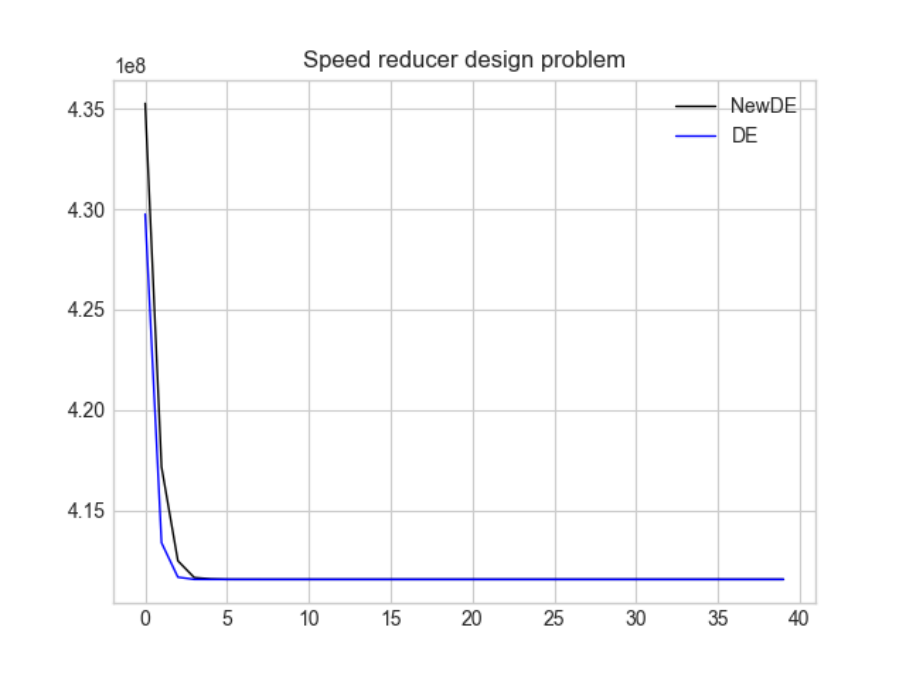
圖二：懸臂梁設計問題

收斂速度比較

收斂速度比較從圖三至圖四中可以觀察到，在Tension-Compression設計問題與Speed reducer設計問題中，傳統DE演算法（圖中的藍線）在收斂速度上明顯優於NewDE演算法（圖中的黑線）。雖然NewDE在初期迭代中迅速下降，但其下降幅度明顯不如DE，在大多數問題上，NewDE的收斂趨勢很快停滯，最終穩定在一個較高的目標函數值上，表現劣於DE。



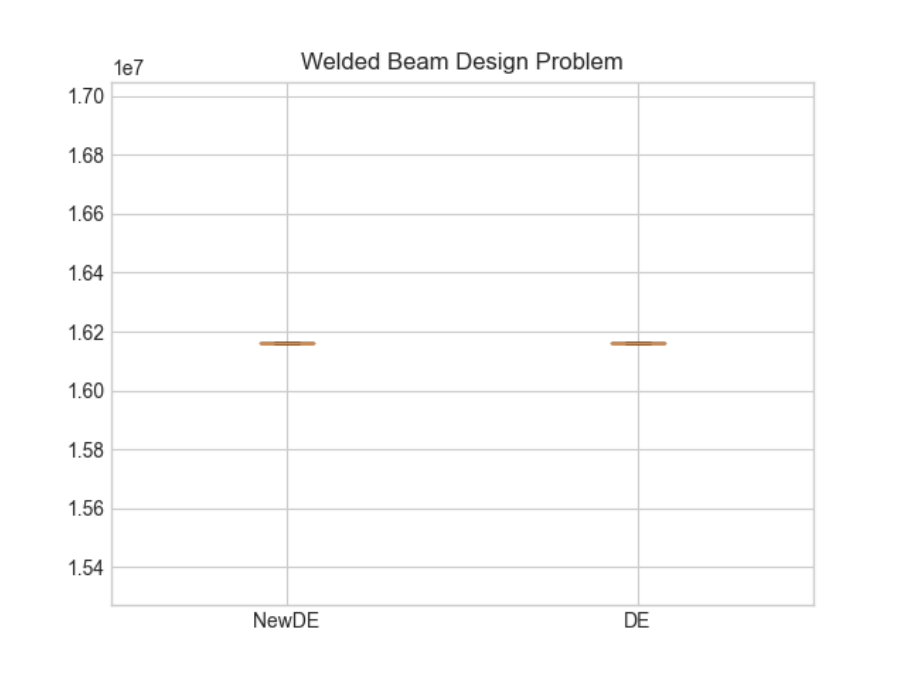
圖三：拉壓彈簧設計問題



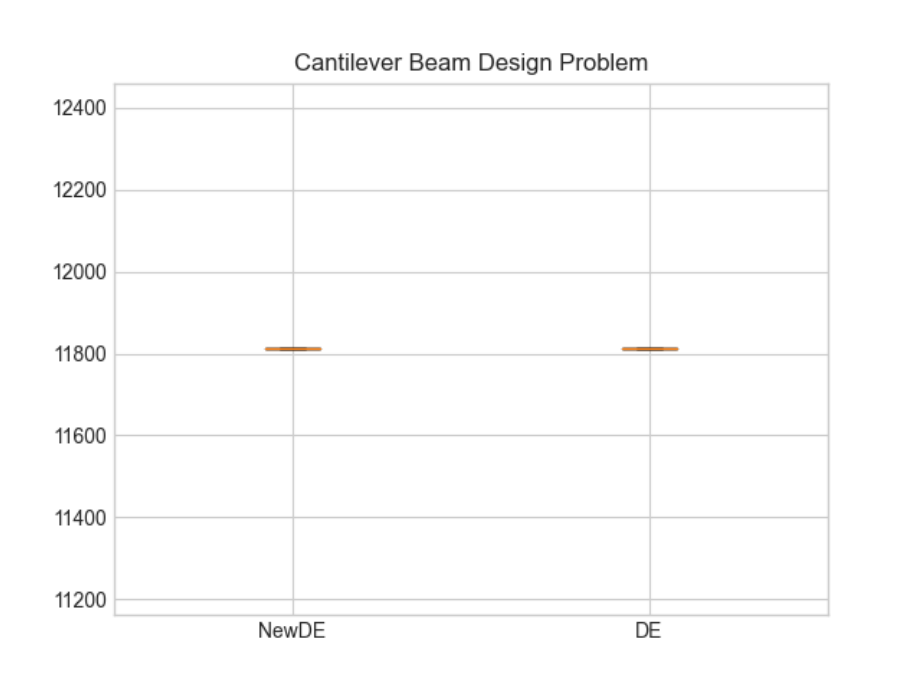
圖四：減速器設計問題

解的質量比較

從圖五至圖六中可以看出，NewDE和DE在最終解的質量上存在顯著差異。傳統DE演算法在多個設計問題中能夠達到較低的目標函數值，而NewDE無論在何種問題中，其最終解的質量都劣於DE，且在某些問題上，新DE的最終結果僅略微接近DE的解。



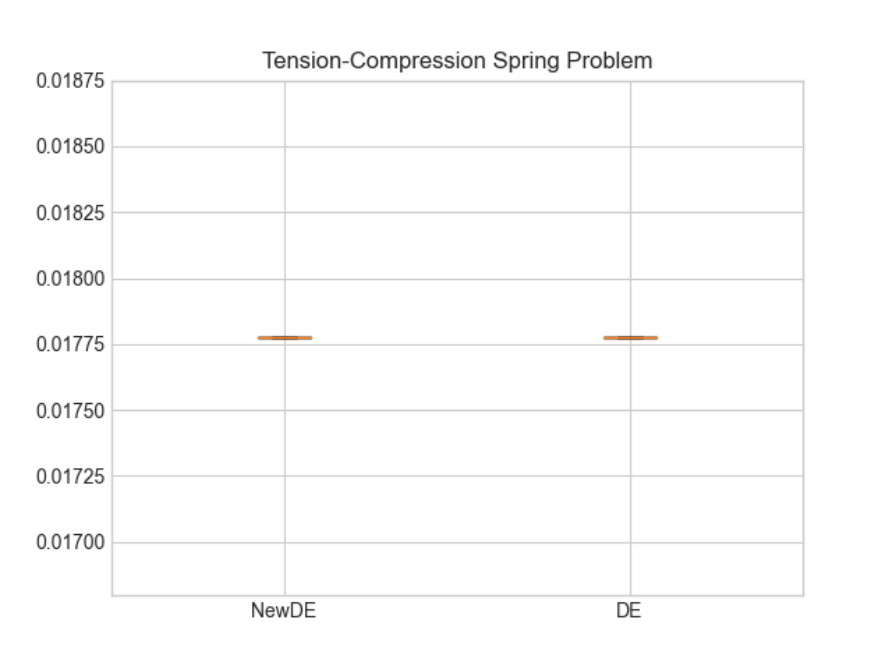
圖五：焊接梁設計問題



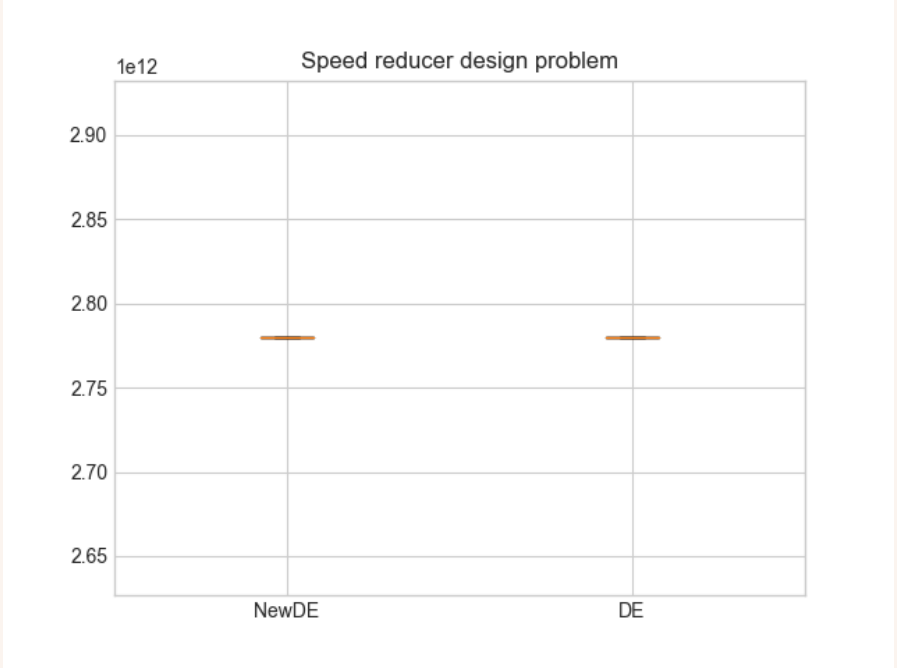
圖六：懸臂梁設計問題

解的質量比較

從圖七至圖八中可以看出，NewDE和DE在最終解的質量上存在顯著差異。傳統DE演算法在多個設計問題中能夠達到較低的目標函數值，而NewDE無論在何種問題中，其最終解的質量都劣於DE，且在某些問題上，新DE的最終結果僅略微接近DE的解。



圖七：拉壓彈簧設計問題



圖八：減速器設計問題

結論

1. 改進策略的有效性從實驗結果可看出，改進的策略（NewDE）在多個設計問題中顯示出其有效性，尤其是在加快收斂速度方面。NewDE在焊接梁和減速器設計問題中比傳統DE有更顯著的收斂效果，說明改進策略可以在解決複雜工程問題上取得良好的效果。

2. Hyper DE在實際應用中的潛力根據實驗結果，Hyper DE在多種設計問題上的應用潛力很大。儘管在解的質量上與DE接近，但在工程設計中的快速收斂特性使其更適合應用於需要快速取得近似解的實際工程問題。

3. 方法的局限性與未來改進方向雖然NewDE展示了良好的收斂性能，但在解的質量上未顯著優於DE。因此未來的研究可以專注於提升解的精度。同時在多峰值問題上的表現尚未充分展示，這可能是需要進一步探討的方向。進一步的改進可能包括引入自適應參數控制和混合其他優化策略。

TEX & Paper 展示

圖片來源:網路抓取

報告結束，感謝聆聽。

TSE,LIN LE

511172176@m365.fju.edu.com

511172176 (李則霖 LI, TSE-LIN) · GitHub