柔性計算期末報告
Modified Grey Wolf Optimizer Based on Stochastic Fractal Search

R10546020 工工所碩一 華方綾

January 17, 2022

1 Introduction

- Objectives:
  本研究的目標爲結合 Modified Grey Wolf Optimizer (GWO) 與 Stochastic Fractal Search (SFS)，透過增強 exploration 與 exploitation，以更有效、更準確地求解連續優化問題（尤其在高維度的清況下）。

- Methodologies:
  作者提出的演算法 GWO-SFS 可分爲兩個階段：(1) Modified-GWO; (2) SFS。
  首先，透過 GWO 模擬灰狼領導服從以及捕獵的行爲，更新灰狼群的位置。接著，modified-GWO 引入 Genetic Algorithm (GA) 裡面的交配 (crossover) 跟突變 (mutation) 機制，增加 exploration 與 exploitation。第二階段則利用 SFS 的 diffusion process 做 local search，找到更好的位置。

- Scope: Metaheuristic algorithms, evolutionary algorithm, Grey Wolf Optimization, random fractals, Genetic algorithm, exploration, exploitation

2 Methods and Problems

Methods

總的來說，GWO-SFS 創新的方法爲透過指數方式更新參數 a，使得演算法有更多機會可以探索 search space；透過 GA 的交配與突變機制增加 exploitation 與 exploration；再透過 SFS 的 diffusion process 做 local search。透過這樣的改良將有效的提升 GWO 搜尋最佳解的能力，平衡 exploitation 與 exploration。

- 初始化：

  演算法最初將問題的可行解用灰狼的位置 $\vec{G}_i(i = 1, 2, ..., n)$ 作爲代表，設定幾隻灰狼 (n) 作爲族群的大小則因題目大小與性質不同而異。由於我們的目標是解有限制條件的連續優化問題，故依據問題的維度，以及每個維度下變數的值域限制，我們可以在限制條件之下隨機初始化一個二維陣列來儲存最初灰狼所在的位置。另外，在初始化的階段亦要隨機產生向量 $\vec{a}, \vec{A}, \vec{C}$，作爲後續更新位置所需的參數。最後，則對每一個初始化的灰狼位置算 fitness function，並求出此時的最佳解、次佳解，以及第三佳的解 $(\vec{G_\alpha}, \vec{G_\beta}, \vec{G_\delta})$。

- **Modified Grey Wolf Optimizer:**

(1) 位置更新：對於灰狼群內每一隻灰狼做更新。依據灰狼群的行爲，群內具有領導能力者的話語權較高，具有支配下者的能力，故更新的方式參照 $(\vec{G_\alpha}, \vec{G_\beta}, \vec{G_\delta})$ 目前所在的位置而移動到距離獵物（最佳解）較近的位置。以數學式來表達如下：

$$\begin{aligned}
\vec{D_\alpha} &= |\vec{C_1} \cdot \vec{G_\alpha} - \vec{G}| \\
\vec{D_\beta} &= |\vec{C_2} \cdot \vec{G_\beta} - \vec{G}| \\
\vec{D_\delta} &= |\vec{C_3} \cdot \vec{G_\delta} - \vec{G}| \\
\vec{G_1} &= \vec{G_\alpha} - \vec{A_1} \cdot \vec{D_\alpha} \\
\vec{G_2} &= \vec{G_\beta} - \vec{A_2} \cdot \vec{D_\beta} \\
\vec{G_3} &= \vec{G_\delta} - \vec{A_3} \cdot \vec{D_\delta} \\
\vec{G}(t+1) &= \frac{\vec{G_1} + \vec{G_2} + \vec{G_3}}{3}
\end{aligned}$$

其中，$\vec{A}, \vec{C}$ 爲係數向量，計算方式如下：

$$\begin{aligned}
\vec{A} &= 2 \cdot a \cdot r_1 - a \\
\vec{C} &= 2 \cdot r_2
\end{aligned}$$

r_1, r_2 分別爲 $[0, 1]$ 間的隨機變數。

(2) 交配與突變

交配的方式使用 GA 裡面的 one-point cut，隨機找一個點 $cp_i, i = 0, ..., N-1$ 當做切點。因此，offspring 的計算方式如下：

$$\begin{aligned}
\text{offspring} = [&\vec{G_1}(section < cp_i)] + \vec{G_2}(cp_i > section), \\
&\vec{G_2}(section < cp_i)] + \vec{G_3}(cp_i > section), \\
&\vec{G_3}(section < cp_i)] + \vec{G_1}(cp_i > section)]
\end{aligned}$$

$$(\vec{G_1'}, \vec{G_2'}, \vec{G_3'}) = Mutation(\text{offspring})$$

其中，$\vec{G_1'}, \vec{G_2'}, \vec{G_3'}$ 分別代表經過交配過後的灰狼。而 Mutation 這個 operator 的實踐方法跟 Genetic Algorithm 一樣，隨機選擇一個或多個元素突變。
令 $positions[i][j]$ 爲第 i 隻灰狼所在位置的第 j 個方向位置，突變的方式爲：

$$positions[i][j] = positions[i][j] + r \cdot (upperBounds[i] - positions[i][j]) \cdot (1 - \frac{iterationID}{iterationLimit})^2$$

其中，r 爲 $[0, 1]$ 裡的隨機變數。

- **Stochastic Fractal Search – Diffusion Process**

  SFS 利用 Diffusion Limited Aggregation (DLA) 來模擬碎形 (fractal) 生長的過程，透過已知群中最佳解位置 G_α，在其附近搜尋更好的位置，是 Diffusion Process 的核心，也是 GWO-SFS 增加其 exploitation 功能的重要機制。在這個過程中，我們利用 Gaussian distribution 來模擬 random walk，式子如下：

  $$\vec{G'_{\alpha_i}} = Gaussian(\mu_{\vec{G_\alpha}}, \sigma) + (\eta \times \vec{G_\alpha} - \eta' \times \vec{P_i})$$

  其中，參數 $\eta, \eta' \in [0, 1]$，P_i 爲群中第 i 隻灰狼的位置。

- 參數更新

  $$a = 2(1 - (\frac{t}{M_t})^2)$$

  其中，t 爲目前迭代次數，M_t 爲迭代次數上限。注意，這裡對 a 的更新與原本的 GWO 使用線性遞減的方法不一樣。透過上式，a 的值將隨著 t 的增加成指數遞減，而這麼做的好處是 exploration 的次數將比原本的 GWO 多次，也就是有更多機會在搜尋空間探索更好的解。

Problems

GWO-SFS 適合求解高維度問題，報告第四部份將展示其效能，並跟其他知名的啓發式優化演算法 (GA, Particle Swarm Optimization) 做比較。
以下爲使用的標竿問題，從低維 (2) 到高維 (500) 的「連續非線性優化問題」:

- Ackley(2),(10),(20),(30)
- DixonPrice(2),(10),(20),(30)
- Girewank(2),(10),(20),(30)
- HyperEllipsoid(10),(20),(30)
- Powell(10),(20),(30)
- Rastrigin(2),(10),(20),(30)
- Rosenbrock(2),(10),(20),(30)
- Schwefel(2),(10),(20),(30)
- Sphere(2),(10),(20),(30),(500)
- SumPowers(2),(10),(20),(30),(500)
- SumSquares(2),(10),(20),(30),(500)
- Zakharov(2),(10),(20),(30),(500)

3 Requirements and Functionality Implementation of the System

- **Data structures**

(1) Grey Wolf Optimizer

– Base class: GreyWolfOptimizer

// data fields
protected int populationSize = 20; // size n (number of wolves in a pack)
protected int numberOfVariables;
protected double[ ][ ] positions; // 0: alpha (best) 1: beta (2nd best) 2: delta (3rd best) 3 N-1: omega (others)
protected double[] positionOfPrey; // the position of prey
protected int secondBestID = 0; // the index of G_β
protected int thirdBestID = 0; // the index of G_δ
protected double secondBestObjective;
protected double thirdBestObjective;
protected double[ ] lowerBounds, upperBounds; // the lower bound and upper bound of each variable
protected double[ ][ ] A; // coefficient vectors
protected double[ ][ ] C; // coefficient vectors
protected double a; // a scalar that decreases linearly from 2 to 0 throughout iterations
protected double[ ][ ] r1; // random vectors whose components are R.V. in [0,1]
protected double[ ][ ] r2; // random vectors whose components are R.V. in [0,1]

protected double[ ] objectiveValues;
protected double soFarTheBestObjectiveValue;
protected double[ ] soFarTheBestPosition;
protected double iterationAverageObjective;
protected double iterationBestObjective;
protected int iterationBestID;
protected int iterationID;
protected int iterationLimit = 200;
protected COPObjectiveFunction theObjectiveFunction; // delegate
protected Random rnd = new Random();

// public interfacing functions
public void Reset()
public virtual void RunOneIteration()
public void RunToEnd()

// helping functions
protected void UpdateSoFarTheBestPosition()
protected virtual void UpdatePositions()
protected virtual void UpdateParameters()
protected void FindSecondAndThirdBest()

– Child class: GreyWolfOptimizerBasedOnSFS

// data fields
int maximumDiffusionNumber = 1;
double[ ] G1;
double[ ] G2;
double[ ] G3;
double[ ] G1prime;
double[ ] G2prime;
double[ ] G3prime;

// public interfacing functions
public override void RunOneIteration()
public void RunToEnd()

// helping functions
protected override void UpdatePositions()
protected override void UpdateParameters()

(2) Stochastic Fractal Search

– Class: StochasticFractalSearch

// data fields
int numberOfVariables;
int numberOfParticles = 100;
int maximumDiffusionNumber = 1;
GaussianType gaussianType = GaussianType.GaussianWalkWithBestPosition;

double[ ][ ] positions; double[ ] GaussianWalk; // temporary storage for the particle generated in the diffusion process
double[ ] bestParticlePosition; // the best particle generated in the diffusion process
int[ ] indexArr; // index array to store the indices of the positions

double[ ] lowerBounds, upperBounds; // the lower bound and upper bound of each variable

double[ ] objectiveValues;
double soFarTheBestObjectiveValue;
double[ ] soFarTheBestPosition;
double iterationAverageObjective;
double iterationBestObjective;

int iterationID;
int iterationLimit = 200;
COPObjectiveFunction theObjectiveFunction; // delegate

```
Random rnd = new Random();

// public interfacing functions
public void Reset()
public void RunOneIteration()

// helping functions
private void Diffuse()
private void GaussianWalk1(int i)
private void GaussianWalk2(int i)
private void UpdatePositions()
private void UpdateSoFarTheBestPosition()
```

- **Algorithms**

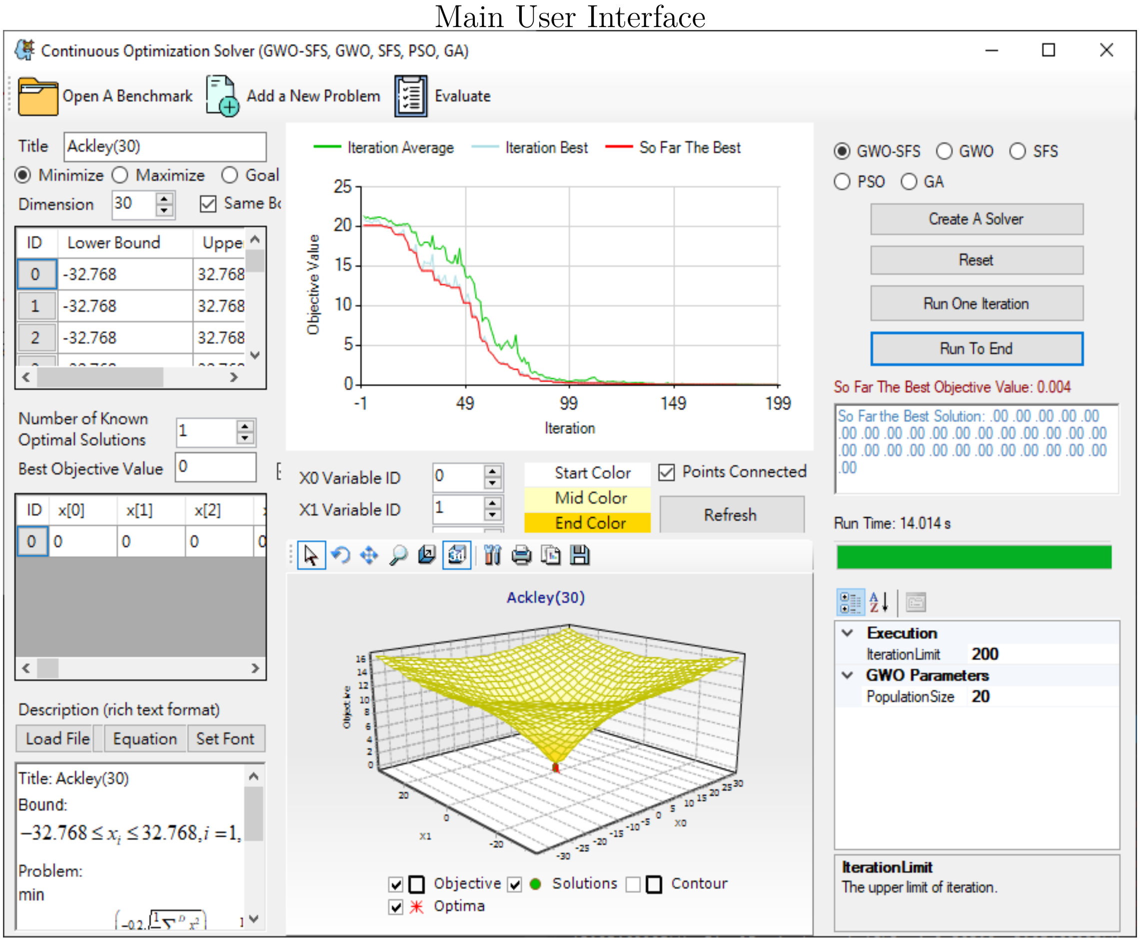
Initialize GWO-SFS population $G_i(i = \vec{1}, 2, ..., n)$ and parameters $(a, \vec{A}, \vec{C})$
Set $t \leftarrow 1$
Calculate the fitness function for each G_i
Find best, second and third best individuals as $\vec{G_\alpha}, \vec{G_\beta}, \vec{G_\delta}$
while $t < M_t$ **do**
    for (i=1:i<n+1) **do**
        Calculate $\vec{D_\alpha} = |\vec{C_1} \cdot \vec{G_\alpha} - \vec{G}|$
        Calculate $\vec{D_\beta} = |\vec{C_2} \cdot \vec{G_\beta} - \vec{G}|$
        Calculate $\vec{D_\delta} = |\vec{C_3} \cdot \vec{G_\delta} - \vec{G}$
        Calculate $\vec{G_1} = \vec{G_\alpha} - \vec{A_1} \cdot \vec{D_\alpha}$
        Calculate $\vec{G_2} = \vec{G_\beta} - \vec{A_2} \cdot \vec{D_\beta}$
        Calculate $\vec{G_3} = \vec{G_\delta} - \vec{A_3} \cdot \vec{D_\delta}$
        Calculate $\vec{G}(t+1) = \frac{\vec{G_1}+\vec{G_2}+\vec{G_3}}{3}$
        Apply Crossover
        Apply Mutation
    end for
    for (i=1:i<n+1) **do**
        Apply Diffusion Process to get $\vec{G'_{\alpha_i}} = Gaussian(\mu_{\vec{G_\alpha}}, \sigma) + (\eta \times \vec{G_\alpha} - \eta' \times \vec{P_i})$
    end for
    Update a by the exponential form of $a = 2(1 - (\frac{t}{M_t})^2)$
    Calculate the fitness function for each G_i
    Update $\vec{G_\alpha}, \vec{G_\beta}, \vec{G_\delta}$
    Set $t \leftarrow t + 1$
end while
return $\vec{G_\alpha}$

- **Interface design and functionalities**

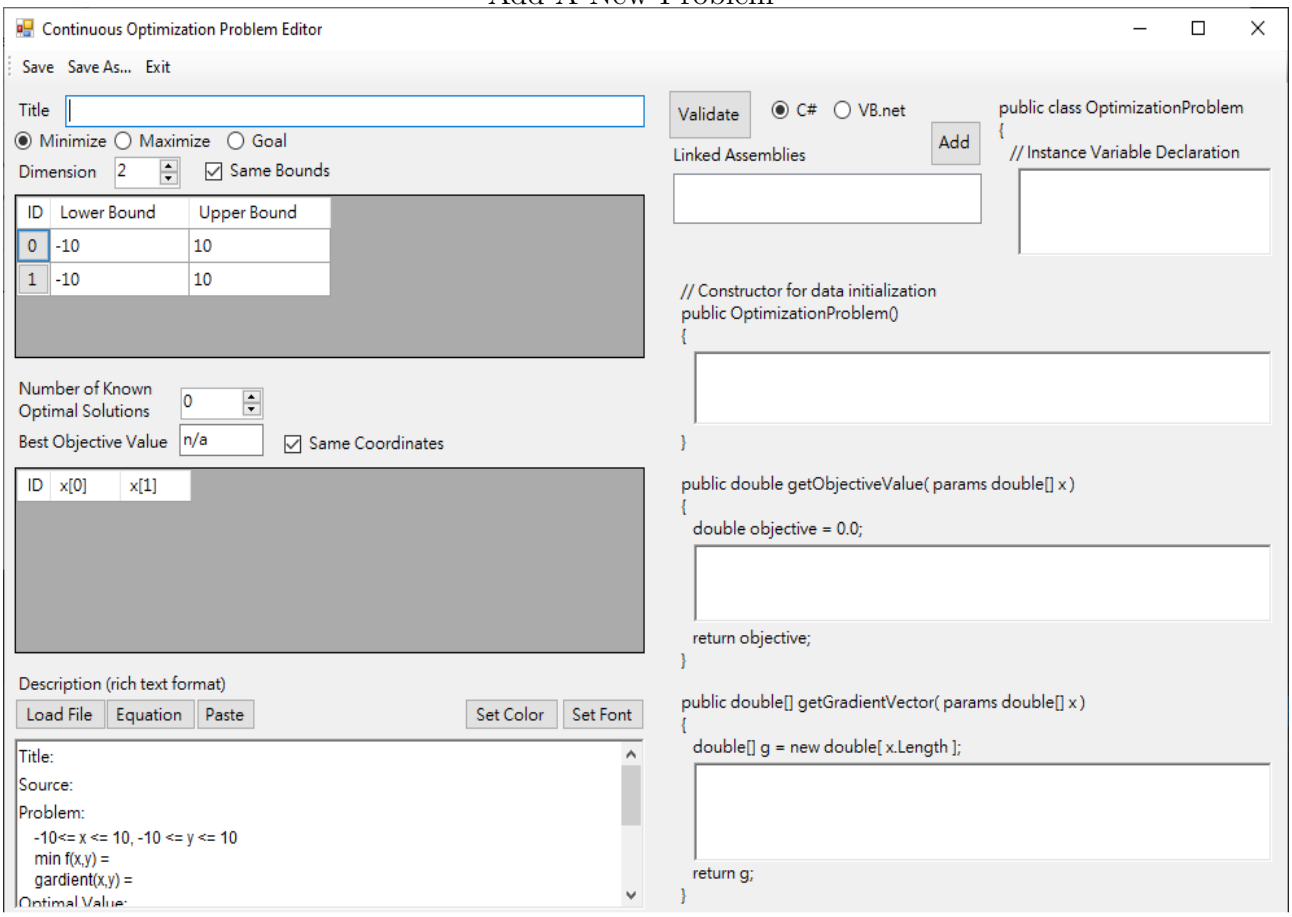
  主使用者介面：

  - 解連續優化問題：
    Open A Benchmark -> Create A Solver -> Reset -> Run To End
  - 新增問題：
    Add A New Problem
  - 比較演算法：
    Evaluate -> Select Benchmark (一次選取多個標竿問題) -> (選取想要比較的演算法，預設爲全選；設定實驗參數)
    (1) 一次對不同的演算法評估比較: -> Evaluate (點下去之後可以看到右邊跑出結果的表格，欄位分別是演算法名稱、標問問題名稱、問題的維度、平均值、標準差)
    (2) 畫收斂圖：左下角的介面可以讓使用者選擇要使用某個已匯入的標竿問題，對已勾選的演算法畫收斂圖 (註: 此收斂圖只有跑一次，亦即 Number of Runs=1)
    (3) 匯出表格：Export -> Table -> Result

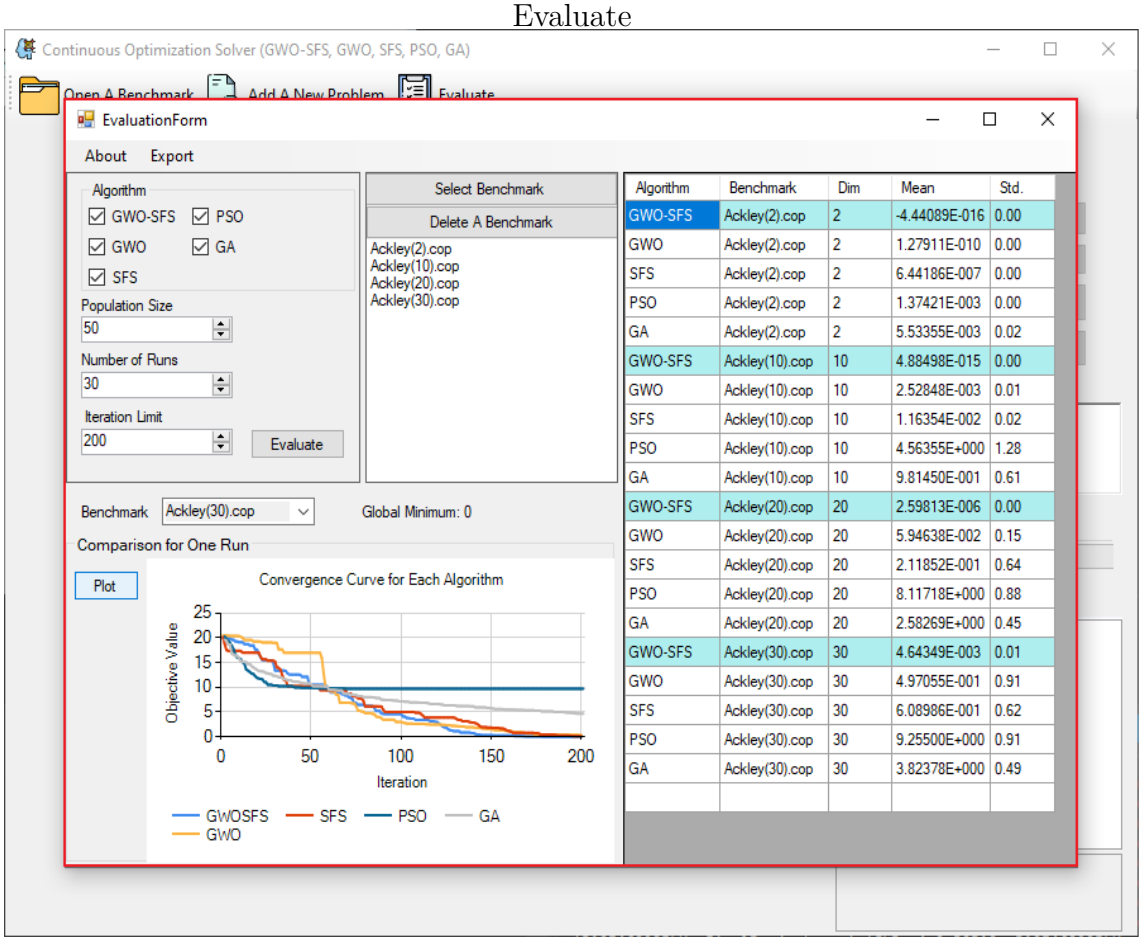
Main User Interface



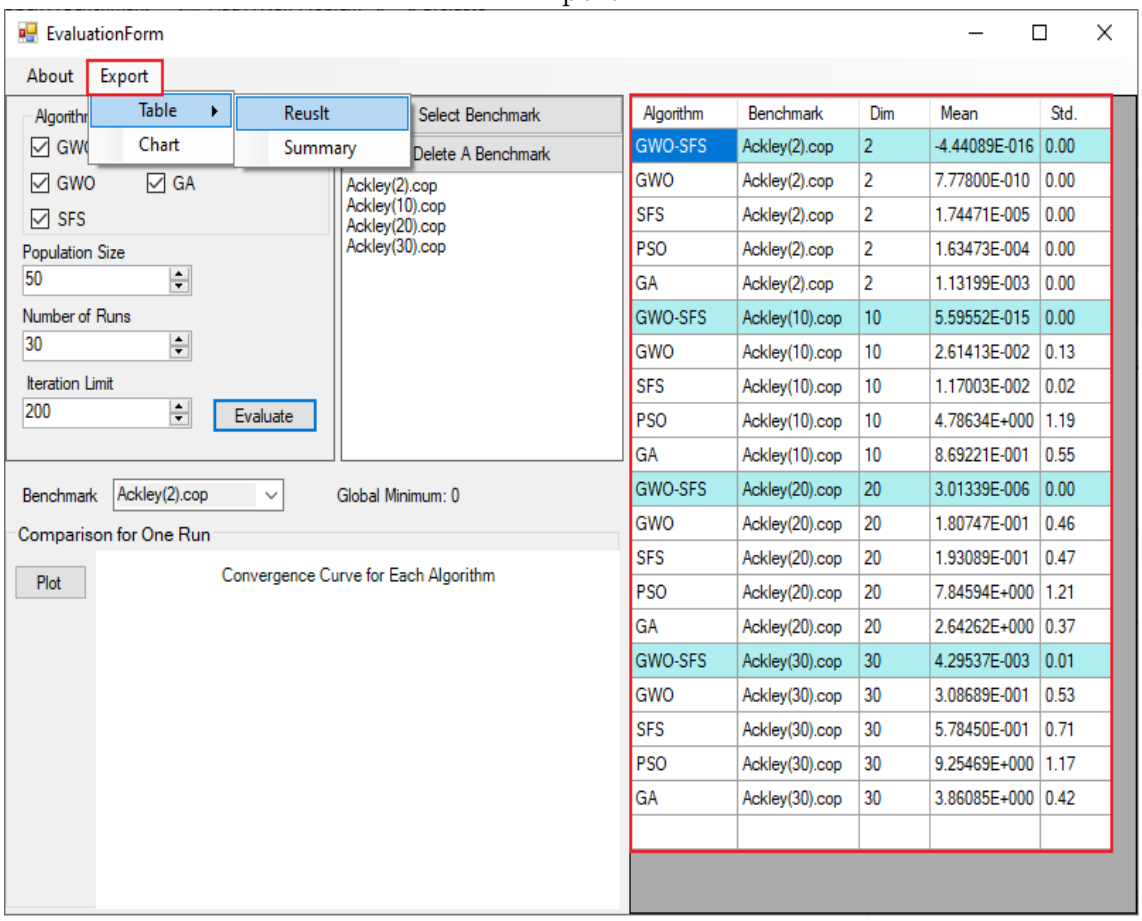
Add A New Problem



Evaluate



Export



Algorithm	Benchmark	Dim	Mean	Std.
GWO-SFS	Ackley(2).cop	2	-4.44089E-016	0.00
GWO	Ackley(2).cop	2	7.77800E-010	0.00
SFS	Ackley(2).cop	2	1.74471E-005	0.00
PSO	Ackley(2).cop	2	1.63473E-004	0.00
GA	Ackley(2).cop	2	1.13199E-003	0.00
GWO-SFS	Ackley(10).cop	10	5.59552E-015	0.00
GWO	Ackley(10).cop	10	2.61413E-002	0.13
SFS	Ackley(10).cop	10	1.17003E-002	0.02
PSO	Ackley(10).cop	10	4.78634E+000	1.19
GA	Ackley(10).cop	10	8.69221E-001	0.55
GWO-SFS	Ackley(20).cop	20	3.01339E-006	0.00
GWO	Ackley(20).cop	20	1.80747E-001	0.46
SFS	Ackley(20).cop	20	1.93089E-001	0.47
PSO	Ackley(20).cop	20	7.84594E+000	1.21
GA	Ackley(20).cop	20	2.64262E+000	0.37
GWO-SFS	Ackley(30).cop	30	4.29537E-003	0.01
GWO	Ackley(30).cop	30	3.08689E-001	0.53
SFS	Ackley(30).cop	30	5.78450E-001	0.71
PSO	Ackley(30).cop	30	9.25469E+000	1.17
GA	Ackley(30).cop	30	3.86085E+000	0.42

4 Numerical Tests

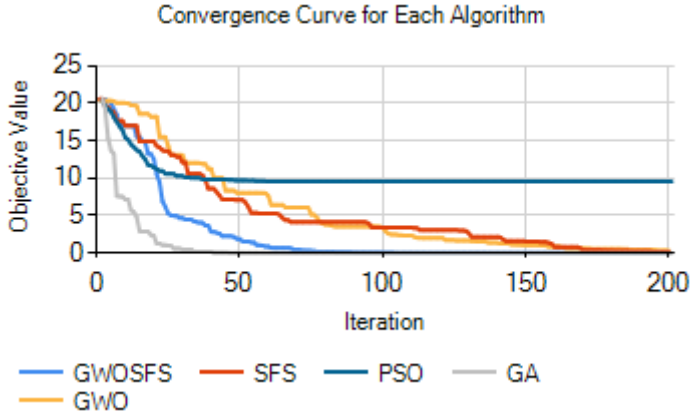
- 實驗設定：
  - Functions: 詳見報告第 2 部分，2 維共 10 個、10 維共 12 個、20 維共 12 個、30 維共 12 個、500 維共 4 個。
  - Iterations: 200
  - Number of Runs: 30
- 演算法參數設定：
  - GWO-SFS: Maximum Diffusion Number (MDN) 設定爲 1; Population Size 50。
  - GWO: Population Size 50。
  - SFS: Population Size 50。
  - PSO: self factor 0.8; social factor 0.2; Population Size 50。
  - GA: crossover rate 0.8, mutation rate 0.05; crossover type = linear; mutation type = gene number based; selection type = deterministic; Population Size 28 *。

*註：GA 在每次迭代的過程中會有新子代交配出來，導致 population size 增加的數量較其他演算法多，故需特別小心其 population size 的設定。舉例來說，程式預設 GA 的 crossover rate 爲 0.8，因此 population size 應設定爲 28 ($28 + 28 \cdot 0.8 \approx 51$)

以 Ackley 爲例，比較不同維度下不同演算法的解的集中與離散程度，最後一行的排名係依據演算法收斂情況與平均跟標準差爲衡量指標。

Algorithm	Benchmark	Dimension	Mean	Std	Rank
GWO-SFS	Ackley(2).cop	2	-4.44E-16	0	1
GWO	Ackley(2).cop	2	2.73E-09	0	4
SFS	Ackley(2).cop	2	1.74E-06	0	3
PSO	Ackley(2).cop	2	9.95E-04	0	5
GA	Ackley(2).cop	2	-4.44E-16	0	2
GWO-SFS	Ackley(10).cop	10	7.14E-15	0	2
GWO	Ackley(10).cop	10	8.58E-02	0.46	4
SFS	Ackley(10).cop	10	5.96E-03	0.01	3
PSO	Ackley(10).cop	10	4.83E+00	1.39	5
GA	Ackley(10).cop	10	4.27E-14	0	1
GWO-SFS	Ackley(20).cop	20	5.85E-06	0	2
GWO	Ackley(20).cop	20	6.25E-02	0.16	3
SFS	Ackley(20).cop	20	1.32E-01	0.2	4
PSO	Ackley(20).cop	20	8.02E+00	1.01	5
GA	Ackley(20).cop	20	8.38E-10	0	1
GWO-SFS	Ackley(30).cop	30	4.71E-03	0.01	2
GWO	Ackley(30).cop	30	6.59E-01	0.91	4
SFS	Ackley(30).cop	30	6.56E-01	0.7	3
PSO	Ackley(30).cop	30	9.59E+00	0.83	5
GA	Ackley(30).cop	30	3.75E-07	0	1

以 Ackley(30).cop 爲例，下圖爲比較 5 種演算法在跑一輪之後的收斂圖



跑過所有的 benchmark problem 後，依據上述的衡量標準爲演算法做排名，結果如下：
(表格裡面的數字爲名次，名次後面的括弧代表在該維度的所有 bencmark problem 裡面表現最佳的次數)

Algorithm/Dimension	2	10	20	30	500
GWO-SFS	1(8)	1(8)	2(4)	2(5)	2(1)
GWO	4	4	4	4	4
SFS	3	3	3	3	3
PSO	5	5	5	5	5
GA	2(2)	2(4)	1(8)	1(12)	1(3)

5 Discussion and Conclusion

討論

SFS 演算法特點與數學性質

1. Consistent: 在不同尺度下都是同樣的模式在生長
2. 融合 BFS 跟 DFS: 模仿 DLA (Diffusion Limited Aggregation) growth，演算法一開始採取先廣後深的策略，但在每次的迭代過程中透過 fitness function 更新、篩選合適的點，有效控制點的成長，並使得候選人收斂到相對較好的解。想像在一個平面上切成九塊在每塊都有效的做 local search 不適合的路徑就會被砍掉且最後會收斂 (by contraction)
3. Statistically self-similar: 一開始的集合 F 需要滿足 $S_i(F) \subset F$ 這個條件，可以想說這類型的 fractal 就是不斷的往內縮小製造出來的，所以這個條件蠻合理的 (要求縮小之後會落在自己裡面)。而定理是說滿足這條件的任何一個 F 去不斷地作迭代，最後都會得到同樣一個圖形: 就是我們要的 fractal，在演算法優化裡面就是搜尋解的路徑圖。
4. 唯一性：最後的集合 independent of 一開始的選取 (有點像是作 optimization 的時候，起始點可能不一樣，但經過迭代後最後都會收斂到唯一的 global minimum)

GWO 演算法更新後的特性

1. 對 a 的更新與原本的 GWO 使用線性遞減的方法不一樣。透過上式，a 的值將隨著 t 的增加成指數遞減，而這麼做的好處是 exploration 的次數將比原本的 GWO 多次，也就是有更多機會在搜尋空間探索更好的解。
2. 加入 GA 的 crossover 與 mutation，增加最佳解的搜尋能力。

總的來說，GWO-SFS 綜合 SFS、GA 與一些數學性質，使得演算法能夠在平衡 exploitation 與 exploration 的同時，加強找尋最佳解的能力。