Shuffled frog leaping algorithm

Wykonał Mateusz Kolasa i Tomasz Kerber

Data:31.03.2020

O algorytmie

- Algorytm powstał na podstawie obserwacji biologicznych zachowań płazów
- Inspiracją do algorytmu były żaby, które skaczą po bagnie, dla których najważniejszym celem jest znalezienie pożywienia, które znajdowało się na kamieniu
- Każda z grup żab (memplex), posiada swoją pulę genetyczną dzięki której posiada pewne cechy(skoczność, długość kończyn)
- Dzięki wymianie genetycznej między różnymi gatunkami żab na bagnie, żaby zdobywały nowe cechy, które przyczyniały się do tego że potrafiły się w końcu dostać na kamień
- Twórcami algorytmu, który powstał w 2003 roku są Eusuff i Lansey
- Algorytm ten rozszerza działanie algorytmów SCE i PSO

Ważne terminy

- Meme jest to odpowiednik genu ale do informacji
- Memplex jest grupa obiektów(w tym przypadku jest to grupa żab) o wspólnych cechach, posiadające zdolność wymiany tych cech
- Meme jest też cechą, która może być przekazywana komuś innemu
- Przykładami mem-ów są piosenki, pomysły, moda na ubrania i sposoby robienia garnków .
- Różnica między genami a memami to że geny przekazywane są z pokolenia na pokolenia a memy są przekazywane przez wymianę informacji
- Przestrzeń wykonywalna to przestrzeń, która spełnia określone ograniczenia

Sposób działania algorytmu

- Inicjalizacja
- Generowanie wirtualnej populacji
- Stworzenie rankingu żab
- Podział żab na mempleksy
- Ewolucja w obrębie każdego mempleksu
- Mieszanie mempleksów
- Sprawdzenie warunków stopu

Inicjalizacja i generowanie wirtualnej populacji

- Na samym początku algorytmu ustalamy liczbę m mempleksów i n ilość żab w każdym mempleksie. W związku z tym całkowita ilość próbek wyniesie F=m*n.
- Generowanie populacji F żab - każda żaba będzie reprezentowana przez wektor U wartości zmiennych decyzyjnych $U(i) = (U_i^1, U_i^2, \ldots, U_i^d)$, gdzie d jest ilością zmiennych decyzyjnych
- Następnie obliczamy wartość funkcji wydajności f(i) dla każdej żaby , specyficznej dla danego problemu

Stworzenie rankingu żab

- Następnie tworzymy ranking żab, sortując je malejąco względem funkcji f(i), to znaczy że żaba z indeksem 1 będzie posiadała najlepszą wartość tej funkcji
- Zapisujemy je do tablicy X ={U(i),f(i),i =1,...,F} , gdzie U(i) oznacza pojedynczą żabę, f(i) wartość funkcji wydajności dla danej żaby, i to numer indeksu, a F rozmiar takich żab ze skojarzonymi z nimi wartościami funkcji wydajności
- Następnie do zmiennej Px zapisujemy wartość żaby z najlepszym rankingiem, czyli Px=U(1)

Podział żab na mempleksy

- Dzielimy tablicę X na m mempleksów Y , każdy zawiera po n żab tak że

$Yk = [U(j)^k, f(j)^k | U(j)^k = U(k+m(j-1)), f(j)^k = f(k+m(j-1)),\ j = 1,\ldots,n\ ], k = 1,\ldots,m;$

Przykład

Dla m=3 i n =2, wygląda to w ten sposób, że do mempleksu 1 należą żaby

k+m(j-1), 1+3(1-1)=1+3(0)=1 oraz 1+3(2-1)=4

Ewolucja w obrębie każdego mempleksu

- Krok 0 - Ustawiamy im=0, które będziemy porównywali z całkowitą liczbą mempleksów m

  Ustawiamy iN, liczące ilość kroków ewolucyjnych i będziemy porównywali je z maksymalną liczbą kroków N do wykonania w każdym mempleksie

- Krok 1 – im=im+1
- Krok 2 – iN=iN+1
- Krok 3 – Tworzymy submempleks. Żabom wewnątrz mempleksu przydzielamy wagi według trójkątnego rozkładu prawdopodobieństwa pj =2(n+1−j)/n(n+1). Z tego wynika że żaba z największą funkcją wydajności ma największą wagę, a najmniejsza najmniejszą. Losujemy używając wag q losowych żab z pośród n żab każdego memplesku i tworzymy tablicę Z. Tę tablicę sortujemy malejąco wg funkcji wydajności. Do Pb zapisujemy najlepszą pozycję żaby z submempleksu a Pw najgorszą

Ewolucja w obrębie każdego mempleksu cd.

- Krok 4 – Poprawiamy pozycję najgorszej żaby dla pozytywnego kroku wg wzoru

  $S=\min\{int[rand(PB - PW)], Smax\}$

  a dla negatywnego kroku

  $S=\max\{int[rand(PB - PW)], -Smax\}$

- gdzie rand to wartość losowa od 0 do 1, a Smax maksymalną wielkością kroku i dodatkowo Smax ma rozmiar d. Nowa pozycja żaby będzie obliczana ze wzoru $U(q) = PW + S$. Jeśli obliczona pozycja jest w przestrzeni wykonywalnej to obliczamy wartość funkcji wydajności, w przeciwnym razie przechodzimy do kroku 5. Jeśli nowa wartość jest lepsza od starej to ją zamieniamy i idziemy do kroku 7 a w przeciwnym do 5.

Ewolucja w obrębie każdego mempleksu cd.

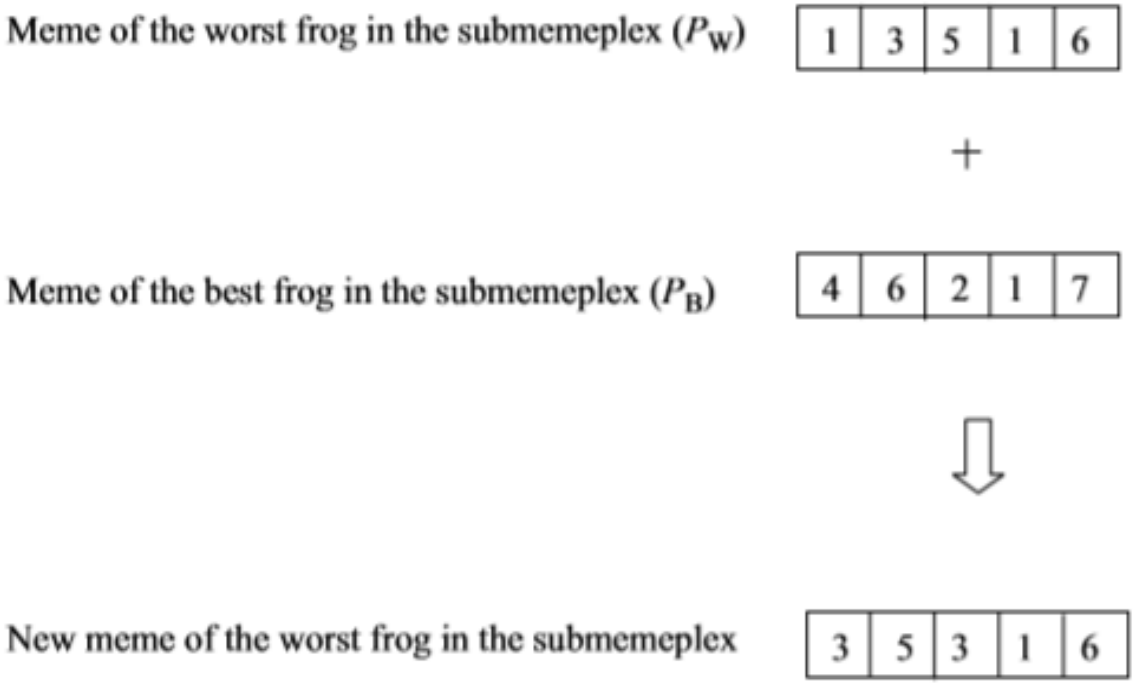


Figure 2. Illustration of the memetic evolution in a submemeplex. Each frog carries one meme and each meme consists of five memotypes. For example, if random number rand is 0.7 and the components of S_{max} are all 3, the first memotype is changed according to $(1 + \min\{\text{int}[\text{rand}(4 - 1)], 3\} = 3)$, and the third memotype is changed according to $(5 + \max\{\text{int}[\text{rand}(2 - 5)], -3\} = 3)$.

Ewolucja w obrębie każdego mempleksu cd.

- Krok 5 – Jeśli krok 4 nie przyniósł rezultatu to obliczamy nową pozycję ze wzoru

  S=min{int[rand(Px −Pw)],Smax}

  a dla negatywnego kroku

  S==max{int[rand(Px−Pw)],−Smax}

  gdzie rand to wartość losowa od 0 do 1, a Smax jest maksymalną wielkością kroku i Smax ma rozmiar d. Nowa pozycja żaby będzie obliczana ze wzoru U(q) =PW +S. Jeśli obliczona pozycja jest w przestrzeni wykonywalnej to obliczamy wartość funkcji wydajności, w przeciwnym razie przechodzimy do kroku 6.

Ewolucja w obrębie każdego mempleksu cd.

- Krok 6 – Jak nowa pozycja nie jest lepsza od starej, to losujemy nową żabę i obliczamy dla niej nową funkcję
- Krok 7 – Submempleksy wstawiamy w oryginalne pozycję Y i Y znowu sortujemy
- Krok 8 – Jeśli iN<N, to idziemy do kroku 2
- Krok 9 – Jeśli im <m to idziemy do kroku 1 , w przeciwnym razie kończymy ewolucję w obrębie każdego mempleksu

Mieszanie mempleksów i sprawdzenie warunków stopu

- Wektory $Y^1,...,Y^{mz}$ mempleksów zamieniamy z powrotem na X
- Sortujemy X i aktualizujemy Px.
- Jeśli kryteria stopu są spełnione należy przerwać algorytm w przeciwnym razie wracamy do podziału żab na mempleksy. Warunkiem stopu może być określona ilość iteracji bez zmiany najlepszej globalnej żaby. Alternatywnie można z góry narzucić liczbę iteracji.

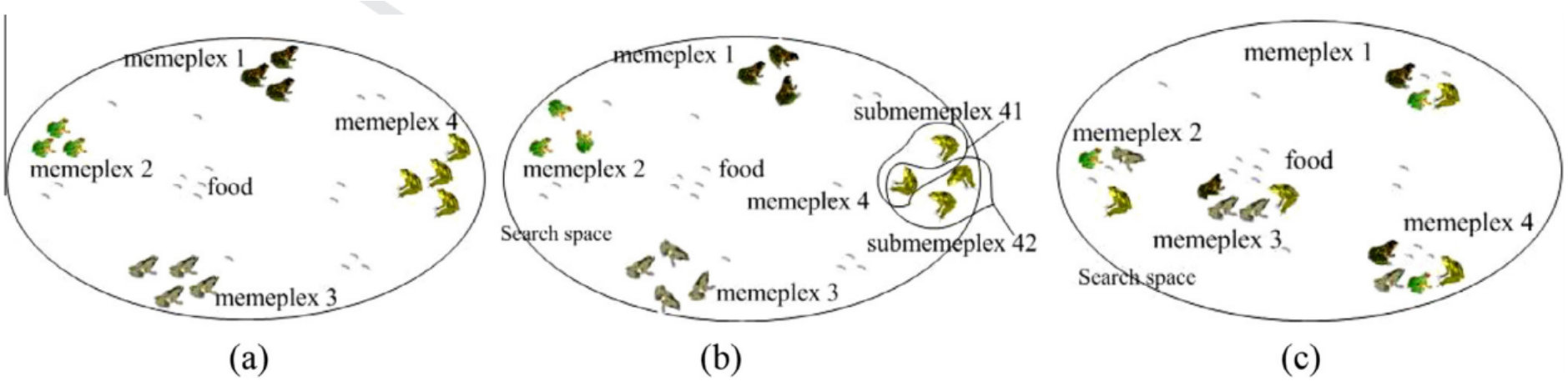


Fig. 2. The illustration of SFLA. (In the figure, the frogs with different behavior can transfer the information among different frogs by mutual imitation. For example, the frog in memeplex 1 leapt towards memeplex 4 and then has the same behavior and ideas as the frog in memeplex 4 through imitating the frog in memeplex 4.)

Pseudokod

```
U = U0
sort(U)
F = f(U)^T
I = [1 : F]^T
X = [U F I]
Px = min(X)
cnt = 0

while cnt < maxIterations
    Y = partition(X)

    for y in Y
        evolution(y)

    X = merge(Y)
    Px+1 = min(X)

    if Px+1 == Px
        cnt = cnt + 1
    else
        Px = Px+1
        cnt = 0
```

Pseudokod cd.

```
evolution (Y)
   in = 0

   while (in < N)
      in = in+1
      W = weight(Y)
      Z = Y[random(W, q)]
      Pb = min(Z)
      Pw = max(Z)

      S = step(Pb, Pw, Smax)
      Uq = Pw+S

      if (f(Uq) < f(Pw))
         Pw = Uq
      else
         S = step(Px, Pw, Smax)
         Uq = Pw + S

         if (f(Uq) < f(Pw))
            Pw = Uq
         else
            Pw = Ur()

      sort(Y)
```

Kluczowe parametry algorytmu

- W algorytmie SFLA jest 5 kluczowych parametrów, które mogą wpływać na przebieg algorytmu :liczba m mempleksów, liczba n żab, liczba q żab w submempleksach, liczbę N ewolucji pomiędzy 2 mieszaniami i wielkość maksymalnego kroku Smax. Wielkość F która jest zależna od m i n jest najważniejszą zmienną wraz ze wzrostem umożliwia lepsze zlokalizowanie globalnego optimum, ale wiążę się to także z większą liczbą ocen funkcji. Jeżeli żab n jest za mało to traci się korzyść ze strategii ewolucji pamięciowej. Parametr q gdy jest za mały w submempleksach powoduje to że czas rozwiązywania rośnie. Jeśli N w algorytmie będzie za małe, to mieszanie będzie odbywało się często, co zmniejszy zmiany tylko do lokalnego minimum. Parametr Smax służy do kontroli globalnych możliwości eksploracji SFLA. Jeżeli jest mały to alogrytm kończy się na wyszukiwaniu lokalnym, a duży może skutkować brakiem rzeczywistej optymalizacji.

Kluczowe parametry algorytmu

Table 2. Test results on the sensitivity of the SFLA performance to different parameters when applied to DeJong's F_5 function (Fn5). Base values are $m = 20, n = 20, q = 5, N = 15$ and $S_{max} = 45\%$ of the variable bounds.

Sensitivity to numbers m of memeplexes				Sensitivity to number n of frogs in a memeplex			
	Number of evaluations required				Number of evaluations required		
m	One optima[†]	All optima[‡]	Comments	n	One optima	All optima	Comments
5	Not found	Not found	Best[§] = 1.99	5	2714	4 558	Success = 70%
10	2283	13 506	Success* = 80%	10	3507	9 844	Success = 80%
20	3143	21 484	Success = 90%	20	3143	21 484	Success = 90%
25	2641	30 692	Success = 90%	25	2664	28 292	Success = 90%
30	1720	41 706	Success = 100%	30	8200	52 631	Success = 100%

[†]One optima indicates the event when at least one frog is infected by the global optimal idea.
[‡]All optima indicate the event when all the frogs are infected by the global optimal idea.
[§]The global optimal (0.998) was not found; rather the local best was determined.
*The success rate was measured on the basis of the number of global optimal solutions found in ten runs.

Kluczowe parametry algorytmu

Table 3. Test results on the sensitivity of the SFLA performance to different parameters for DeJong's F_5 function. Base values are $m = 20$, $n = 20$, $q = 15$, $N = 15$ and $S_{max} = 45\%$ of the variable bounds.

Sensitivity to number q of frogs in a submemeplex				Sensitivity to number N of evolution steps in a memeplex			
	Number of evaluations required				Number of evaluations required		
q	One optima[†]	All optima[‡]	Comments	N	One optima	All optima	Comments
5	3143	21 484	Success[§] = 90%	5	3391	9 297	Success = 100%
10	2103	11 317	Success = 100%	10	3995	11 034	Success = 100%
15	1070	8 491	Success = 100%	15	1070	8 491	Success = 100%
20	5237	10 466	Success = 100%	20	3950	11 098	Success = 100%

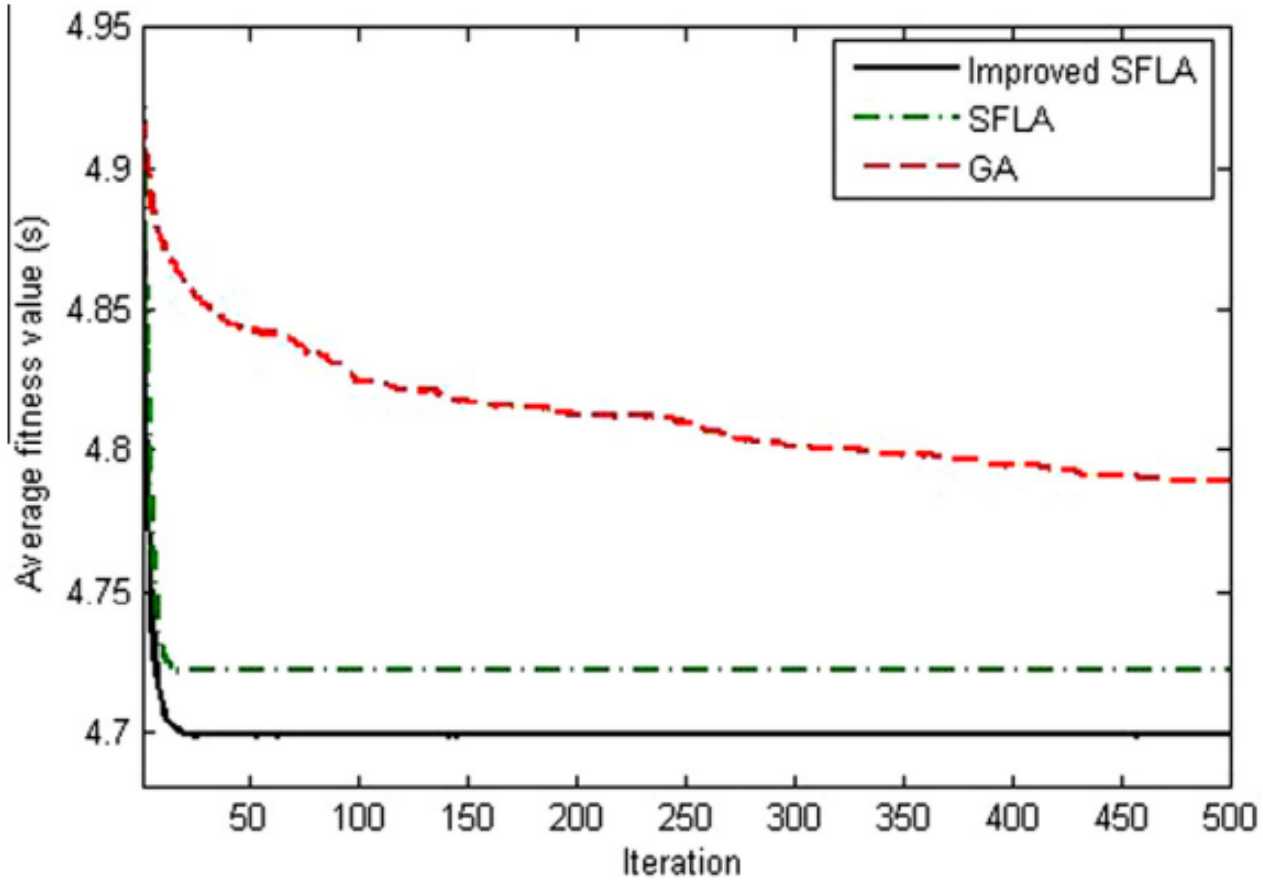
Zastosowanie algorytmu

- Algorytm ten używany jest do rozwiązywania kombinatoryjnych problemów optymalizacji
- Był używany w optymalizacji sieci wodociągowych gdzie zastosowano go do znajdowania rury o najbardziej optymalnej wielkości

Acknowledgments

The writers acknowledge financial support provided by the following agencies: Tucson Water, City of Tucson, Arizona, Sanitation Districts of Los Angeles County, American Water Works Association Research Foundation, City of Phoenix, Ariz., Subregional Operators group (Phoenix-area cities), Water Replenishment District of Southern California, City of Riverside, Calif., and City of Los Angeles Dept. of Water and Power.

Porównanie algorytmu SFLA z algorytmem genetycznym



Bibliografia

- Eusuff, M., Lansey, K., & Pasha, F. (2006). *Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. Engineering Optimization, 38(2), 129-154.*
- Eusuff, M. M., & Lansey, K. E. (2003). *Optimization of Water Distribution Network Design Using the Shuffled Frog Leaping Algorithm. Journal of Water Resources Planning and Management, 129(3)*
- Zhu, G.-Y., & Zhang, W. B. An improved Shuffled Frog-leaping Algorithm to optimize component pick-and-place sequencing optimization problem. Expert Systems with Applications (2014)

Dziękuje za uwagę