UNIVERSIDAD DE GRANADA

Práctica alternativa al examen: Soccer Game Optimization (SGO)

Alberto Lejárraga Rubio

DNI: 50618389N

Email: alejarragar@correo.ugr.es

Granada, 22 de junio de 2018

1. Índice

2. Breve explicación del algoritmo original

El algoritmo Soccer Game Optimization (SGO) es un algoritmo de optimización de variables continuas que basa su funcionamiento en una imitación a lo que podría ser un partido de fútbol. En este algoritmo aparecen, por tanto, términos heredados del mundo del fútbol que podrían resumirse básicamente en los siguientes:

- Jugador: solución generada por el algoritmo, es lo que sería un cromosoma en un algoritmo genético
- Equipo: es un conjunto de jugadores o soluciones que se puede relacionar con la población de algoritmos genéticos
- Poseedor del balón: identifica al jugador que ha encontrado una mejor solución durante la ejecución del algoritmo.
- Jugador titular: es cada uno de los jugadores del equipo que realizan movimientos en el terreno de juego (espacio de soluciones) para ir encontrando y valorando soluciones.
- Jugador suplente: es una serie de jugadores que no realiza movimientos, sino que "observan" a los titulares y van almacenando sus mejores posiciones. Estos podrán pasar a ser titulares en base a una probabilidad para reemplazar a titulares con un rendimiento pobre.

El funcionamiento del algoritmo se basa en realizar, tras generar el equipo inicial creando las listas de jugadores titulares(los mejores) y suplentes (los peores entre los generados), un número determinado de iteraciones en las que, en cada una de ellas, cada jugador titular realiza movimientos para encontrar nuevas soluciones. En estos movimientos es donde aparece tanto la exploración como la explotación del algoritmo principalmente, aunque hay otros factores que después explicaré. Hay dos tipos:

- *Move forward*: es el que aporta una mayor explotación teniendo en cuenta para su ejecución, la posición anterior del jugador, la mejor solución encontrada por este y la posición del poseedor del balón. Su pseudocódigo es el siguiente (para cada jugador titular):

```
Función Move Forward:

    Declarar una variable flotante nuevoValor
    Para cada atributo i del problema (dimensión del problema):
        nuevoValor = posición anterior del jugador
        nuevoValor += mejor posición encontrada por el jugador
        nuevoValor += posición actual del poseedor del balón
        Actualizar el atributo i del jugador actual con nuevoValor/3
```

- Move off: trata de llevar a cabo una mayor exploración pero de manera controlada, de una forma aleatoria no uniforme basada en la siguiente fórmula:

$X^i_t = \begin{cases} Xi + \Delta(t, LS - Xi) \; si \; r < 0{,}5 \\ Xi - \Delta(t, Xi - LI) \; en \; otro \; caso \end{cases}$, siendo LS el límite superior y LI el límite inferior del atributo i

Y la función $\Delta(t,y) = y * (1 - r^{\left(1 - \frac{t}{T}\right)^{\alpha}})$ Siendo r un número aleatorio, t la iteración actual, T el máximo número de iteraciones, y α el parámetro de no uniformidad

Introduciendo en la fórmula las iteraciones actuales se consigue que la aleatoriedad del movimiento sea mayor al principio y vaya decayendo según se llega al final. Por otro lado, el parámetro α hace que se pueda decidir si se desea una mayor o menor aleatoriedad, ya que si es 0 el movimiento será completamente aleatorio, mientras que si es 1 tendrá una importancia muy grande el número de iteraciones realizadas hasta el momento.

Dicho esto, el pseudocódigo es el siguiente:

```
Función Move Off:
      Para cada atributo i del problema (dimensión del problema):
            Min=valor mínimo del atributo i
            Max=valor máximo del atributo i
            Aleatorio1 y aleatorio2=números aleatorios entre 0 y 1
            Crear la variable nuevoValor y darle el valor de la posición actual del jugador
            Si aleatorio1 < 0.5:
                  Aplicar la fórmula para r<0.5 con Xi=nuevoValor
            En otro caso:
                  Aplicar la fórmula para r>=0.5 con Xi=nuevoValor
            Actualizar el atributo i del jugador actual con el resultado
```

Primera fórmula explicada arriba

Como se ha comentado, los jugadores suplentes podrán pasar a ser titulares para reemplazar a aquellos que no tengan un gran rendimiento en función de la valoración de su solución. Por esto, en cada iteración, tras realizar los movimientos, se realizarán las sustituciones. El método para hacerlo se basa en ir comparando el mejor suplente con el peor titular para sustituirlo si así lo indica un número aleatorio. Si no se sustituye se pasará a revisar el segundo peor titular. Y así sucesivamente hasta que se introduzca al jugador suplente en el campo sustituyendo a un titular, o bien el jugador titular con el que se compara el suplente ya sea mejor que este último.

Cuando una de estas dos opciones ocurra se pasará a revisar el segundo mejor suplente del mismo modo, hasta que se compruebe si entran todos los suplentes. El pseudocódigo es el siguiente, teniendo en cuenta que se utiliza una lista ordenada tanto de titulares como de suplentes, estando en la posición 0 el mejor jugador:

```
Función realizarSustituciones:
      Declarar variable indiceSup inicializada a 0
      Declarar variable indiceTit inicializada a numeroDeTitulares-1
      Mientras(indiceSup < numeroDeTitulares e índiceTit>=0):
            Si es mejor el suplente indiceSup que el titular indiceTit:
                  Si <numeroAleatorio> <= k:
                        Intercambiar los jugadores en las listas de titulares y suplentes
                        indiceSup++
                        indiceTit = numeroDeTitulares-1
                  En otro caso:
                        indiceTit—
            En otro caso:
                  indiceSup++
                  indiceTit = numTitulares-1
```

Se hace el intercambio, por lo que se pasa a comprobar el siguiente suplente desde el peor titular

No se hace y se pasa a comprobar el siguiente titular para el mismo suplente

El suplente ya no es mejor que el titular estudiado, se desecha ese suplente

Hecho esto, ya se pasaría a actualizar el conocimiento de los jugadores tanto titulares (modificando su mejor posición encontrada si fuera necesario) como suplentes (cogiendo las mejores posiciones encontradas por los titulares si mejoran a las que ya tenían). Por tanto, el pseudocódigo completo del algoritmo sería el siguiente:

```
Generar equipo inicial e introducirlo en arrays jugTitulares y jugSuplentes
Ordenar los arrays de jugadores //para que el mejor sea el primero
Declarar poseedorDelBalón=0
Declarar evaluaciones e inicializarla a numeroDeTitulares+numeroDeSuplentes
Mientras evaluaciones < evaluacionesTotales:
      Para cada jugador titular:
            Si <numeroAleatorio> <= m:
                  Hacer Move Off a jugador
                  Si <numeroAleatorio> <=l:
                        Hacer Move Forward a jugador
            En otro caso:
                  Hacer Move Forward a jugador
            Evaluar jugador y actualizar su "fitness"
            evaluaciones++;
      ordenar los arrays de jugadores
      Realizar sustituciones
      Ordenar los arrays de jugadores
      Actualizar el conocimiento de los jugadores titulares
      Actualizar el conocimiento de los jugadores suplentes
Devolver el jugador titular en la posición del poseedorDelBalón, 0
```

3. Mejoras al algoritmo

Para mejorar el algoritmo he probado varios métodos, que podría resumir en los siguientes:

- Hacer adaptativo el parámetro m, que discrimina entre hacer "Move Off" o "Move Forward"
- Hacer adaptativo el parámetro l, que determina si se hará un "Move Forward" después de hacer un "Move Off"
- Hacer el movimiento "Move Off" completamente aleatorio
- Reinicializar los parámetros l y/o m si la búsqueda quedaba estancada
- Reinicializar todos los jugadores menos los 1,2 o 3 mejores si la búsqueda quedaba estancada
- Aplicar los distintos tipos de búsqueda local disponibles
- Aumentar la comunicación entre jugadores, de forma que si al hacer "Move Forward" un jugador quedaba muy cerca (utilizaba una distancia umbral y comparaba las distancias euclídeas) de otro, se le realizara un movimiento "Move Off" al primero para alejarlo de este
- Probar distintos parámetros del algoritmo, en función del tipo de función (unimodal o multimodal, como indicaba el "paper" original)
- Aplicar pesos a cada uno de los sumandos implicados en el movimiento "Move Forward"

Tras hacer un gran número de pruebas probando todas estas opciones finalmente he decidido que en la implementación que entrego aparezcan las que más convenían al algoritmo y con las que se obtenían mejores resultados.

Por tanto, las modificaciones que he realizado al algoritmo básico explicado en la sección anterior son:

Mismos parámetros para todas las funciones y algunos adaptativos

En los "papers" publicados por los autores en los que se explica el algoritmo se aconsejaba utilizar valores de los parámetros distintos según la función fuera unimodal o multimodal:

Parámetro	Valor unimodal	Valor multimodal
M	0.05-0.1	0.2-0.4
L	0.9	0.1-0.3
K	0.05-0.2	0.05-0.2
Alfa	0.2	0.2
Nº de titulares	0.5-2 * dimensión del problema	0.5-2 * dimensión del problema
Nº de suplentes	5-15% de la dimensión del problema	10-40% de la dimensión del problema

En cuanto a los valores de los parámetros, se han realizado pruebas aplicando a cada función sus propios parámetros según fueran unimodales(1-3), multimodales(4-16) o hibridas(17-20), haciéndolos también adaptativos. Finalmente he decidido aplicar a todas las funciones el mismo rango de parámetros, al encontrar que no hay mejora entre los que yo he utilizado y los que se proponían. Estos son por tanto, los siguientes:

Parámetro	Valor
Mini	0.8
Mfin	0.4
Lini	0.1
Lfin	0.3
K	0.15
Alfa	0.2
Nº de titulares	1.1*dimensión del problema
Nº de suplentes	40% de la dimensión del problema

Los parámetros L y M tienen una forma de adaptarse muy similar, con la diferencia de que M disminuye su valor según avanzan las iteraciones (se realizan cada vez más "Move Forward") y L aumenta (se realizan cada vez más "Move Forward" después de "Move Off").

En principio, probé con que ambos parámetros se adaptaran de manera lineal desde su valor "ini" hasta su valor "fin", aunque al final introduje a la hora de modificarlos un factor que hace que la diferencia entre una iteración y la siguiente vaya siendo cada vez más pequeña, aunque nunca se llegue el valor indicado en el valor "fin".

Simplemente, se crea una variable reducción (para el parámetro m) y aumento (para l) al principio del algoritmo en función del número de evaluaciones a realizar. Después, al final de cada iteración se suma o resta este valor al parámetro correspondiente multiplicado por el factor <evaluaciones totales>/<evaluaciones realizadas> (T/t):

Parametro	Formula de incremento	Aplicación del incremento
L	(lFin-lIni)/T	$L_{anterior}$ += (T/t)*<incremento>
M	(mIni-mFin)/T	$M_{anterior}$ -= (T/t)*<incremento>

Búsqueda local CMAES con el 30% de las evaluaciones

La búsqueda local CMAES la aplico tras ejecutar el algoritmo SGO con el 70% de las evaluaciones máximas. Al aplicar esta búsqueda local al final consigo que aunque se haga una gran exploración en el algoritmo (con los parámetros explicados dirigidos hacia ello) en esta parte se explote bastante la mejor solución encontrada. Es decir, el algoritmo dirigirá la búsqueda hacia una solución potencialmente buena y será la búsqueda local CMAES la que mejore mucho más esta solución.

En cuanto a la decisión de utilizar CMAES y no los otros modelos de búsqueda local se debe a que en las pruebas realizadas esta es la que daba un mejor rendimiento en media en todas las funciones.

Movimiento "Move Off" aleatorio no uniforme

Con respecto a este movimiento "Move Off", he utilizado el explicado en el primer apartado de este documento. En los distintos "papers" publicados no dejaba claro cuál era el mejor modelo de "Move Off", si debía ser completamente aleatorio en el espacio de búsqueda o si debería ser guiado de alguna forma tal como se ha explicado.

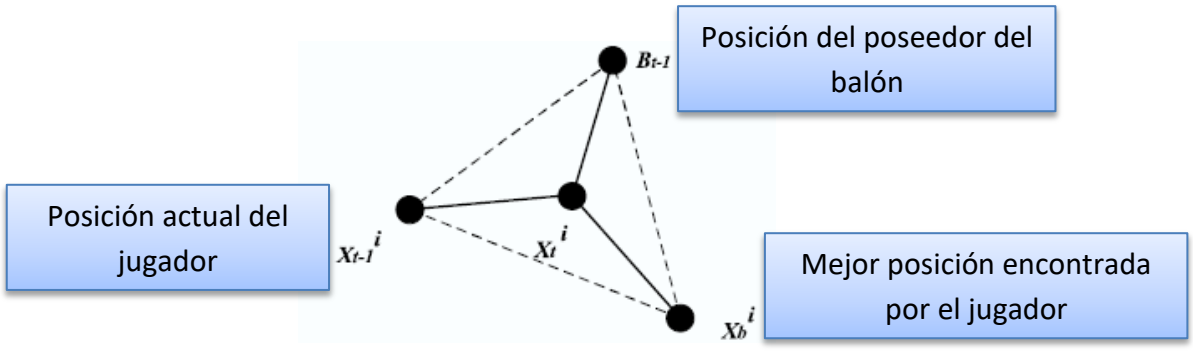
En las pruebas que he realizado he observado que se comporta mucho mejor este movimiento "Move Off", probablemente porque al utilizar unos valores de m y l que llevan al algoritmo a hacer muchas veces este movimiento, de no estar mínimamente guiado, la búsqueda se convertiría bastante en una búsqueda aleatoria.

Movimiento "Move Forward" uniforme

De nuevo, como con el movimiento anterior en cada uno de los "papers" publicados se explicaban dos versiones distintas de este movimiento. En uno de ellos, se recomendaba usar pesos para dar una mayor importancia al poseedor del balón, pues sus valores recomendados eran los siguientes:

Sumando	Peso
Posición actual del jugador	0.191
Mejor posición encontrada por el jugador	0.191
Posición del poseedor del balón	0.618

En otro de los informes se comentaba que este movimiento debía hacerse siguiendo el "Principio del Centro de Masas", teniendo todos los sumandos el mismo peso de 1/3:



4. Resultados obtenidos y análisis

En la siguiente tabla, se muestra de izquierda a derecha la calidad de la solución obtenida, la distancia al óptimo, la diferencia con respecto a la media, la diferencia con respecto a la mejor solución encontrada y el número de algoritmos que han encontrado una solución mejor para las 20 primeras funciones del CEC2014, habiéndose ejecutado el algoritmo para cada función 25 veces y realizada la media. Como se puede observar, para dimensión 10 el algoritmo quedaría en décimo quinta posición con una distancia media con respecto al óptimo de 79378.66:

SGOBasico	Error obtenido	diferencia media	diferencia min	posicion
859093,00	858993,00	-258285,38	858993,00	**12,00**
108892,00	108692,00	-23524672,56	108692,00	**13,00**
1231,30	931,30	-574,35	931,30	**10,00**
454,23	54,23	37,90	54,23	**16,00**
520,35	20,35	1,82	6,70	**15,00**
604,86	4,86	3,67	4,86	**15,00**
702,86	2,86	2,41	2,86	**15,00**
827,38	27,38	21,66	27,38	**16,00**
927,96	27,96	17,83	26,30	**15,00**
2038,49	1038,49	913,47	1038,48	**16,00**
2222,81	1122,81	781,35	1102,68	**16,00**
1201,14	1,14	0,91	1,14	**16,00**
1300,45	0,45	0,31	0,44	**16,00**
1400,34	0,34	0,08	0,26	**13,00**
1504,70	4,70	-0,43	4,33	**13,00**
1603,05	3,05	0,93	2,00	**15,00**
9017,75	7317,75	-215107,61	7316,77	**12,00**
10916,90	9116,90	-21157,97	9116,68	**13,00**
1903,93	3,93	2,89	3,85	**16,00**
2248,08	248,08	-26725,41	247,90	**11,00**
		-1202236,92	49378,66	**14,20**

Puede comprobarse como en ninguna de las funciones se alcanza el óptimo y, de hecho, en ninguna se colocaría entre las 10 mejores metaheurísticas. En la última fila, se puede ver el promedio de los valores de su columna correspondiente. En la de la diferencia con respecto a la media, puede verse como aunque se mejora bastante a la media esto se debe a que hay funciones en las que otros algoritmos se comportan muy mal y dan unas soluciones muy alejadas, aunque obviamente este algoritmo sigue dando soluciones bastante malas.

A continuación muestro esta misma tabla pero ya una vez realizadas las modificaciones explicadas:

SGOadapt+LSCMAES	Error obtenido	diferencia media	diferencia min	posicion
100,00	0,000	-1117278,38	0,00	**0,00**
200,00	0,000	-23633364,56	0,00	**0,00**
300,00	0,000	-1505,65	0,00	**0,00**
400,32	0,319	-16,01	0,32	**4,00**
518,53	18,532	0,01	4,88	**5,00**
600,60	0,597	-0,60	0,60	**9,00**
700,01	0,009	-0,43	0,01	**5,00**
813,74	13,737	8,02	13,74	**13,00**
912,01	12,011	1,88	10,35	**12,00**
1628,62	628,620	503,60	628,61	**16,00**
1776,62	676,620	335,16	656,49	**14,00**
1200,42	0,420	0,19	0,42	**13,00**
1300,11	0,110	-0,03	0,10	**8,00**
1400,30	0,300	0,04	0,22	**12,00**
1501,26	1,260	-3,87	0,89	**12,00**
1603,17	3,170	1,05	2,12	**15,00**
2296,06	596,060	-221829,30	595,08	**10,00**
1849,85	49,850	-30225,02	49,63	**10,00**
1902,62	2,620	1,58	2,54	**16,00**
2060,03	60,030	-26913,46	59,85	**10,00**
		-1251514,29	101,29	**9,20**

Como puede observarse, tras aplicar las modificaciones al algoritmo, este comienza a comportarse algo mejor, de media, para las funciones del CEC2014. En esta ocasión puede verse como se alcanza el óptimo en las primeras tres funciones (unimodales) y que se obtienen resultados no tan malos como los obtenidos antes. Esto puede comprobarse en la distancia al óptimo, que de media es de 101.29 frente a los 49378,66 obtenidos en la versión básica.

Además, en esta ocasión ya se colocaría entre los diez primeros puestos de media, mientras que antes quedaba entre el último y el penúltimo puesto.

Estas dos tablas muestran los resultados obtenidos para las funciones con dimensión 10 y las que muestro a continuación son los resultados para dimensión 30:

SGOBasico	Error obtenido	diferencia media	diferencia min	posicion
2,57E+07	25653300,00	4005501,19	25653300,00	**13,00**
1,91E+08	190821800,00	-168150163,44	190821800,00	**13,00**
24716,1	24416,10	21171,79	24416,10	**16,00**
621,921	221,92	154,69	221,92	**15,00**
520,546	20,55	0,33	0,55	**14,00**
631,797	31,80	23,30	31,80	**16,00**
703,723	3,72	-0,17	3,72	**13,00**

913,52	113,52	81,04	113,52	**15,00**
1034,18	134,18	71,25	131,99	**14,00**
4622,43	3622,43	2559,63	3622,41	**16,00**
5348,08	4248,08	1626,26	3018,60	**15,00**
1201,33	1,33	0,90	1,33	**14,00**
1300,35	0,35	0,03	0,30	**11,00**
1400,16	0,16	-1,08	-0,03	**0,00**
1568,59	68,59	-2837,87	66,44	**13,00**
1612,17	12,17	1,43	3,67	**14,00**
550976	549276,00	-1744349,59	549088,49	**13,00**
1564,31	235,69	-6313340,61	229,78	**10,00**
1931,16	31,16	9,05	28,08	**14,00**
6307,09	4307,09	-2184,53	4304,01	**11,00**
		-8609083,82	10853019,13	**13,00**

En este caso el comportamiento es bastante similar en cuanto a su posición con respecto a las demás metaheurísticas aunque cabe destacar que se coloca como el mejor algoritmo para la décimo cuarta función.

Una vez realizadas las modificaciones, la metaheurística mejora bastante:

SGOadapt+LSCMAES	Error obtenido	diferencia media	diferencia min	posicion
100,00	0,000	-21647798,81	0,00	**0,00**
200,00	0,000	-358971963,44	0,00	**0,00**
300,00	0,000	-3244,31	0,00	**0,00**
400,00	0,000	-67,23	0,00	**0,00**
520,58	20,583	0,37	0,58	**14,00**
605,05	5,045	-3,45	5,04	**8,00**
700,00	0,002	-3,89	0,00	**6,00**
849,67	49,668	17,18	49,67	**11,00**
953,13	53,131	-9,80	50,95	**7,00**
3712,39	2712,390	1649,59	2712,37	**15,00**
4048,47	2948,470	326,65	1718,99	**11,00**
1200,70	0,700	0,27	0,70	**12,00**
1300,27	0,270	-0,05	0,22	**6,00**
1400,32	0,320	-0,92	0,13	**12,00**
1503,45	3,450	-2903,01	1,30	**6,00**
1612,31	12,310	1,57	3,81	**14,00**
3416,48	1716,480	-2291909,11	1528,97	**8,00**
1928,81	128,810	-6313447,49	122,90	**9,00**
1909,00	9,000	-13,11	5,92	**9,00**
2517,17	517,170	-5974,45	514,09	**9,00**
			335,78	**7,85**

Se puede observar como en este caso se mejoran bastante los resultados, volviendo a entrar dentro de los diez primeros puestos y encontrando el valor óptimo en las primeras cuatro funciones.

Con respecto a este aumento de dimensión, cabe destacar que la reinicialización sí que mejoraría algo los resultados, pues en una de las ejecuciones se conseguía hacer que el algoritmo entrase entre los cinco mejores. Esto ocurre porque al haber más ejecuciones disponibles, es más posible que la búsqueda quede estancada y el reinicializar los jugadores puede aportar una exploración extra que mejore bastante el resultado.

Aun así, la versión que entrego no incluye esta característica por no responder igual de bien para la dimensión 10.

5. Manual de uso del código

El código proporcionado del algoritmo es la versión final de este, codificado en C++ y probado en Windows 10. Para ejecutarlo en este sistema operativo se deben realizar la siguiente secuencia de pasos situado en la carpeta raíz “localsearch”.

1. cmake -G "CodeBlocks - MinGW Makefiles"
2. mingw32-make
3. example_ls.exe <dimension>

El primer paso es ejecutar el “cmake”, yo lo ejecuto con el modificador –G y la versión indicada arriba aunque puede depender del software que se tenga instalado. Con cmake –G se muestran todas las opciones para este modificador.

El segundo es ejecutar el make, que en mi caso, usando el compilador mingw se hace con el comando mostrado.

Por último, una vez generado el ejecutable habrá que lanzarlo acompañado de la dimensión del problema, 10 o 30, aunque por defecto si no se indica nada se lanzará la versión de dimensión 10.

Antes de ejecutar todos estos comandos habrá que asegurarse de que está instalado todo el software necesario, cmake y el compilador que permita hacer make.

6. Conclusión

En primer lugar, destacaría la alta influencia que tiene el tipo de función en esta clase de algoritmos, ya sea unimodal o multimodal. Obviamente, con el primer tipo es mucho más sencillo encontrar el óptimo ya que solo hay uno en todo el espacio de búsqueda. En cambio, las funciones multimodales, al tener varios óptimos locales pueden ser más complicadas por el hecho de que el algoritmo converja hacia estos óptimos. De hecho, llama la atención que por ejemplo, en la función número 5 todos los algoritmos parecen converger hacia óptimos parecidos que no son globales, ya que la evaluación de la

función objetivo siempre se queda en torno a 520 frente a los 500 del óptimo y todos los algoritmos quedan atrapados alrededor de ellos.

Por otro lado, he podido comprobar que las características que se han estudiado durante el curso en las distintas metaheurísticas pueden ser muy válidas para algunos de estos algoritmos pero no así para otros. De hecho, como muestro al principio del tercer apartado, he intentado hacer varias mejoras que no han surgido efecto y que se utilizan en otros algoritmos de forma correcta.

Por último, ha quedado muy clara la necesidad de realizar un correcto equilibrio entre exploración y explotación a lo largo del algoritmo. En el caso del SGO, al estudiar los "papers" de los autores podía parecer que este equilibrio estaba bastante bien conseguido con los dos tipos de movimientos posibles, "Move Off" y "Move Forward" y con la estrategia de sustituciones. Aun así, al implementar el algoritmo y analizar los resultados quedó bastante claro que este equilibrio era bastante mejorable.

Por ello, las modificaciones que he realizado se han basado en intentar aumentar bastante la exploración durante la ejecución del algoritmo en sí, aunque controlando también una correcta explotación. Al acabar esta parte del algoritmo, comprobé que ejecutar una búsqueda local sobre la mejor solución encontrada mejoraba enormemente el resultado, lo que da muestra de que la intensificación que realizaba el algoritmo no era del todo buena.

7. Bibliografía

Para la realización de la práctica he utilizado las diapositivas de la asignatura, tanto de la parte teórica como de la parte práctica a la hora de obtener ideas que aplicar al algoritmo. Por otro lado, también me ha sido útil lo aprendido tanto en las prácticas realizadas a lo largo del cuatrimestre, como lo estudiado en clase y las presentaciones de otros compañeros sobre sus metaheurísticas.

Para analizar las funciones del CEC2014 obtuve el siguiente documento:

http://web.mysites.ntu.edu.sg/epnsugan/PublicSite/Shared%20Documents/CEC-2014/Definitions%20of%20%20CEC2014%20benchmark%20suite%20Part%20A.pdf

Para estudiar el algoritmo, utilice los siguientes recursos:

http://jsiskom.undip.ac.id/index.php/jsk/article/download/63/40

https://ac.els-cdn.com/S0377042715000205/1-s2.0-S0377042715000205-main.pdf?_tid=bd98e584-9de3-4c68-a6a6-a3fff6872d04&acdnat=1527327856_3b73e886ded77e4e0e95b54fcdbfe273

http://ojs.atmajaya.ac.id/index.php/metris/article/view/71/32

Por último, en uno de los documentos anteriores se hace referencia al siguiente a la hora de analizar los resultados:

http://embeddedlab.csuohio.edu/BBO/BBO_Papers/HaipingMa2.pdf