Metaheurísticas (2017-2018)
Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

Galactic Swarm Optimization:

A new global optimization metaheuristic inspired by galactic motion

Francisco Solano López Rodríguez

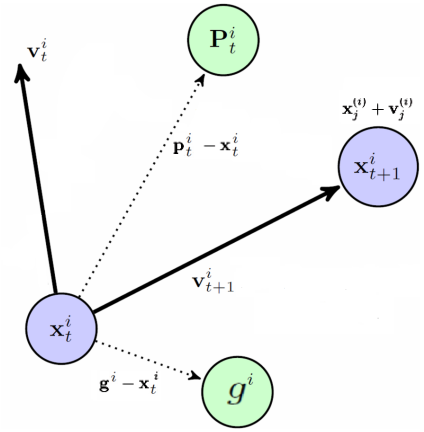
1. Introducción

La metaheurística GSO esta inspirada en el movimiento que describen las estrellas en el interior de las galaxias y además a una mayor escala en el movimiento de las galaxias pertenecientes a un cúmulo. Siguiendo esta idea se podría seguir aumentando la escala moviendo también los cúmulos, conjuntos de cúmulos, etc. Pero GSO se queda en el segundo nivel, es decir movimiento de estrellas y galaxias.

La idea del GSO está basada en el algoritmo PSO (Particle Swar Optimization) el cual está inspirado en el movimiento de una nube de partículas. El movimiento de cada partícula dentro de la nube viene dado por la siguiente fórmula:

$$v_j^{(i)} \leftarrow w_1 v_j^{(i)} + c_1 r_1 (p_j^{(i)} - x_j^{(i)}) + c_2 r_2 (g^{(i)} - x_j^{(i)})$$
$$x_j^{(i)} \leftarrow x_j^{(i)} + v_j^{(i)}$$

Donde p_j se corresponde con la mejor solución personal encontrada por la partícula j-ésima, v_j la velocidad de la partícula y g la mejor solución global encontrada. c_i son valores constantes llamados coeficientes de aceleración y w_i números aleatorios entre 0 y 1.



El algoritmo GSO sigue esta misma idea, pero dividida en dos niveles, Para ello crea un conjunto de M galaxias y en cada galaxia genera de forma aleatoria un conjunto de N estrellas (cada estrella debe entenderse como una solución).

En el primer nivel se moverán las estrellas dentro de cada galaxia de la misma forma que en el PSO, donde el movimiento vendrá dado de acuerdo a su velocidad actual, mejor posición personal encontrada y la mejor solución obtenida por la galaxia en la que se encuentra.

En el segundo nivel se tomarán las mejores soluciones de cada galaxia y se creará una población de soluciones con estas. De nuevo esta población buscará soluciones en el espacio moviendose de nuevo como se hacía en el PSO, donde ahora la velocidad se calculará con el mejor personal de cada solución, la velocidad actual y la mejor solución global encontrada hasta el momento.

2. Algoritmo

A continuación se muestra el pseudocódigo de este algoritmo.

Level 1 Initialization: $\mathbf{x}_j^{(i)}, \mathbf{v}_j^{(i)}, \mathbf{p}_j^{(i)}, \mathbf{g}^{(i)}$ within $[x_{min}, x_{max}]^D$ randomly.
Level 2 Initialization: $\mathbf{v}^{(i)}, \mathbf{p}^{(i)}, \mathbf{g}$ within $[x_{min}, x_{max}]^D$ randomly.
for $EP \leftarrow 1$ **to** EP_{max}
do
- Begin PSO: Level 1
- **for** $i \leftarrow 1$ **to** M
  - **do**
    - **for** $k \leftarrow 0$ **to** L_1
      - **do**
        - **for** $j \leftarrow 1$ **to** N
          - **do**
            - $\mathbf{v}_j^{(i)} \leftarrow \omega_1 \mathbf{v}_j^{(i)} + c_1 r_1 (\mathbf{p}_j^{(i)} - \mathbf{x}_j^{(i)}) + c_2 r_2 (\mathbf{g}^{(i)} - \mathbf{x}_j^{(i)})$;
            - $\mathbf{x}_j^{(i)} \leftarrow \mathbf{x}_j^{(i)} + \mathbf{v}_j^{(i)}$;
            - **if** $f(\mathbf{x}_j^{(i)}) < f(\mathbf{p}_j^{(i)})$
              - **then**
                - $\mathbf{p}_j^{(i)} \leftarrow \mathbf{x}_j^{(i)}$;
                - **if** $f(\mathbf{p}_j^{(i)}) < f(\mathbf{g}^{(i)})$
                  - **then**
                    - $\mathbf{g}^{(i)} \leftarrow \mathbf{p}_j^{(i)}$;
                    - **if** $f(\mathbf{g}^{(i)}) < f(\mathbf{g})$
                      - **then** $\mathbf{g} \leftarrow \mathbf{g}^{(i)}$;
- Begin PSO: Level 2
- Initialize Swarm $\mathbf{y}^{(i)} = \mathbf{g}^{(i)} : i = 1, 2, \cdots, M$;
- **for** $k \leftarrow 0$ **to** L_2
  - **do**
    - **for** $i \leftarrow 1$ **to** M
      - **do**
        - $\mathbf{v}^{(i)} \leftarrow \omega_2 \mathbf{v}^{(i)} + c_3 r_3 (\mathbf{p}^{(i)} - \mathbf{y}^{(i)}) + c_4 r_4 (\mathbf{g} - \mathbf{y}^{(i)})$;
        - $\mathbf{y}^{(i)} \leftarrow \mathbf{y}^{(i)} + \mathbf{v}^{(i)}$;
        - **if** $f(\mathbf{y}^{(i)}) < f(\mathbf{p}^{(i)})$
          - **then**
            - $\mathbf{p}^{(i)} \leftarrow \mathbf{y}^{(i)}$;
            - **if** $f(\mathbf{p}^{(i)}) < f(\mathbf{g})$
              - **then** $\mathbf{g} \leftarrow \mathbf{p}^{(i)}$;

Return $\mathbf{g}, f(\mathbf{g})$

Podemos distinguir los dos niveles comentados anteriormente. En el primero recorremos cada subpoblación y movemos cada solución dentro de esta. En el segundo creamos una nueva población compuesta por las mejores soluciones obtenidas en cada subpoblación. Este proceso se repite y vamos alternando entre uno y otro. El primer nivel se corresponde con la parte de exploración del algoritmo, y el segundo nivel, al considerar solo las mejores soluciones, se correspondería con la parte de explotación. Este algoritmo busca un equilibrio entre estas dos partes, aunque se puede modificar aumentando el valor de N y disminuyendo el de M y viceversa.

3. Código

En esta sección se va a describir la implementación del código utilizado.

Lo primero es la generación aleatoria de las poblaciones, para ello se ha aprovechado una función, del código aportado por Daniel Molina, con la cual podemos crear una solución aleatoria dentro de dominio del problema.

```
void getInitRandom(Random *random, DomainRealPtr domain,
                           tChromosomeReal &crom) {
   tReal min, max;

   for(unsigned i = 0; i < crom.size(); ++i){
      domain->getValues(i, &min, &max, true);
      crom[i] = random->randreal(min, max);
   }

}
```

A continuación se muestra la generación aleatoria de las poblaciones y subpobaciones.

```
for(int i = 0; i < M; i++){
   for(int j = 0; j < N; j++){
      getInitRandom(m_random, domain, x[i][j]);
      getInitRandom(m_random, domain, v1[i][j]);
      getInitRandom(m_random, domain, p1[i][j]);
      p1_value[i][j] = m_eval->eval(p1[i][j]);
   }
}

for(int i = 0; i < M; i++){
   getInitRandom(m_random, domain, g[i]);
   getInitRandom(m_random, domain, v2[i]);
   getInitRandom(m_random, domain, p2[i]);
   g_value[i] = m_eval->eval(g[i]);
   p2_value[i] = m_eval->eval(p2[i]);
}

getInitRandom(m_random, domain, gbest);
gbest_value = m_eval->eval(gbest);
```

Ahora pasamos al código del algoritmo, primero mostraremos el nivel 1.

```
\\ NIVEL 1
for(int i = 0; i < M; i++){
  for(int k = 0; k <= L1; k++){
     double w1 = 1.0-k*L1_div;
     for(int j = 0; j < N; j++){
        double r1 = m_random->rand();
        double r2 = m_random->rand();

        for(int d = 0; d < D; d++){
```

```
            v1[i][j][d] = w1*v1[i][j][d]+c1*r1*(p1[i][j][d]-x[i][j][d])
            + c2*r2*(g[i][d]-x[i][j][d]);
            x[i][j][d] = x[i][j][d] + v1[i][j][d];
         }

         domain->clip(x[i][j]);
         double new_value = m_eval->eval(x[i][j]);
         eval++;

         if(new_value < p1_value[i][j]){
            for(int d = 0; d < D; d++)
               p1[i][j][d] = x[i][j][d];
            p1_value[i][j] = new_value;

            if(new_value < g_value[i]){
               for(int d = 0; d < D; d++)
                  g[i][d] = p1[i][j][d];
               g_value[i] = new_value;

               if(new_value < gbest_value){
                  for(int d = 0; d < D; d++)
                     gbest[d] = g[i][d];
                  gbest_value = new_value;
               }
            }
         }
      }
   }
}
```

EL siguiente código se corresponde con el nivel 2.

```
\\ NIVEL 2
vector< vector<double> > y(g);

for(int k = 0; k <= L2; k++){
   double w2 = 1.0-k*1.0*L2_div;
   for(int i = 0; i < M; i++){
      double r3 =  m_random->rand();
      double r4 =  m_random->rand();

      for(int d = 0; d < D; d++){
         v2[i][d] = w2*v2[i][d] + c3*r3*(p2[i][d] - y[i][d])
               + c4*r4*(gbest[d]-y[i][d]);
         y[i][d] = y[i][d] + v2[i][d];
      }

      domain->clip(y[i]);
      double new_value = m_eval->eval(y[i]);
      eval++;
```

```
      if(new_value < p2_value[i]){
         for(int d = 0; d < D; d++)
            p2[i][d] = y[i][d];
         p2_value[i] = new_value;

         if(new_value < gbest_value){
            for(int d = 0; d < D; d++)
               gbest[d] = p2[i][d];
            gbest_value = new_value;
         }
      }
   }
}
```

El siguiente se corresponde con el bucle principal del algoritmo el cual va intercalando entre nivel 1 y nivel 2, para primero explorar y después pasar a la fase de explotación e ir intercalando ambas fases.

```
for(int ep = 0; ep < EP_max; ep++){
   \\ NIVEL 1
   GSO_nivel_1();
   \\ Nivel 2
   GSO_nivel_2();
}
```

4. Local Search

En esta sección veremos las modificaciones realizadas al código anterior, para añadirle una búsqueda local. La búsqueda local utilizada ha sido CMAES, puesto que ha sido la que mejores resultados ha dado, tras haber hecho pruebas con las 3 LS proporcionadas.

Para hacer uso de CMAES se ha modificado el nivel 2 de algoritmo. En este nivel se tomaban las mejores soluciones de cada galaxia y se ejecutaba un PSO con ellas. Se ha seguido la misma idea de tomar las mejores soluciones de cada galaxia, pero en vez de usar PSO, se ha ejecutado CMAES sobre cada una de estas soluciones. EL código se muestra a continuación.

```
// NIVEL 2

vector< vector<double> > y(g);

for(int i = 0;  i < M; i++){
   ILocalSearch *ls;
   ILSParameters *ls_options;
   CMAESHansen *cmaes = new CMAESHansen("cmaesinit.par");
   cmaes->searchRange(0.1);
   ls = cmaes;
   ls->setProblem(m_problem);
   ls->setRandom(m_random);
   ls_options = ls->getInitOptions(sol);
   unsigned evals = ls->apply(ls_options, y[i], p2_value[i], 1000);
   eval += evals;

   for(int d = 0; d < D; d++)
      p2[i][d] = y[i][d];

   if(p2_value[i] < gbest_value){
      for(int d = 0; d < D; d++)
         gbest[d] = p2[i][d];
      gbest_value = p2_value[i];
   }
}
```

Además al final de algoritmo se ha ejecutado CMAES sobre la mejor solución encontrada en el GSO.

```
ILocalSearch *ls;
ILSParameters *ls_options;
//ls = new SimplexDim();
CMAESHansen *cmaes = new CMAESHansen("cmaesinit.par");
cmaes->searchRange(0.1);
ls = cmaes;
ls->setProblem(m_problem);
ls->setRandom(m_random);
ls_options = ls->getInitOptions(sol);
unsigned evals = ls->apply(ls_options, sol, fitness, 22000);

eval += evals;
```

5. Resultados

5.1. GSO

Los parámetros utlizados han sido los siguientes.

Dimensión	M	N	L_1	L_2	EP_{max}	$c_1 = c_2$	$c_3 = c_4$
10	12	8	80	1000	5	2.05	0.5
30	20	5	280	1500	5	2.05	1.05

Dimensión 10				**Dimensión 30**		
Función	**Media**	**Desv.**	**Mediana**	**Media**	**Desv.**	**Mediana**
F1	665525	986620	319094	3.35383e+07	2.37511e+07	2.35493e+07
F2	7111.75	4933.42	9863.47	6.55879e+08	9.03877e+08	2.52627e+08
F3	267.021	233.547	245.913	9155.07	2956.95	8944.79
F4	36.0719	17.0868	34.7803	253.41	83.345	230.306
F5	20.1538	0.0925186	20.1551	20.5305	0.122447	20.5437
F6	6.25161	1.21448	6.06837	30.3963	3.29772	31.0341
F7	2.96936	2.12357	2.25073	6.90277	7.9903	3.08737
F8	23.0069	8.45772	19.9005	135.669	33.6428	135.777
F9	28.5642	10.3621	29.0956	175.089	37.589	175.021
F10	654.016	232.524	670.208	4588.56	636.641	4600.59
F11	855.566	297.636	862.308	4814.08	598.692	4944.39
F12	0.42005	0.153718	0.426314	0.990066	0.249225	0.98602
F13	0.410964	0.137787	0.351583	0.626694	0.198228	0.596704
F14	0.58442	0.370606	0.396934	1.59153	3.31706	0.45672
F15	2.58898	1.63657	2.15886	111.04	185.005	46.8188
F16	3.31785	0.197635	3.39961	12.383	0.454295	12.4798
F17	697.934	306.897	646.263	2.4189e+06	3.55183e+06	840912
F18	73.4638	54.7024	58.7193	2.62698e+07	9.0391e+07	13865.6
F19	5.71442	1.95925	5.47976	37.2742	32.5489	20.0341
F20	88.692	55.3184	67.9788	1779.72	1557.75	1165.51

Tras muchos ajustes de parámetros estos son los mejores resultados que he podido obtener con GSO. Comentar que con los parámetros recomendados en el artículo de este algoritmo, obtuve resultados mucho peores, creo que debido a que tenía gran capacidad de exploración, pero una explotación muy débil. Debido a ello he aumentado el valor de aquellos parámetros con los que

pudiera obtener mayor explotación, a pesar de ello los resultados no han sido tan buenos como podría esperarse. También se puede apreciar una desviación bastante grande, algo que no es muy deseable.

DE dimensión 10

	CoDE	D-SHADE	EPSDE	JADE	L-SHADE	OP-aCM	SHADE11	SaDE	dynNP-jDE	iCMAES-ILS
F1	0	0	0	0	0	0	0	2,5523186	2,1693E-07	0
F2	0	0	0	0	0	0	0	0	0	0
F3	0	0	0	0,00617	0	0	0	0	0	0
F4	10,4826	30,773486	0	27,6187	29,409553	2,81989	29,49456	18,08504	3,32292061	14,3852921
F5	18,449	17,726874	20,04996	17,2677	14,145598	18,0702	18,00618	15,784068	15,9500416	14,6543978
F6	1,65E-06	0	3,041562	0,17552	0,0175401	0,33053	0	0	0	0
F7	0,03759	0,0053139	0,017574	0,01186	0,0030429	0	0,009782	0,0072429	0,00496919	0
F8	0	0	0	0	0	3,69725	0	0	0	0,25365771
F9	3,88229	3,0826834	3,688733	3,50709	2,3445977	0,32622	3,140748	3,5837731	3,8578531	0,09754876
F10	0,03551	0,0489839	0,044085	0,00612	0,0085722	91,6179	0,012246	0,0195936	0,00244919	122,040112
F11	75,9703	54,934451	323,1317	83,6965	32,055826	116,833	63,1802	196,4226	136,2142	8,58508098
F12	0,04338	0,0529084	0,32105	0,25014	0,0681671	0,01008	0,136408	0,435019	0,31112176	0,06500938
F13	0,0798	0,048919	0,122368	0,08397	0,051562	0,01089	0,073997	0,1252054	0,11896589	0,00911486
F14	0,10715	0,0900969	0,136322	0,11054	0,0813617	0,28243	0,105517	0,1857701	0,13522187	0,15455911
F15	0,65232	0,4027667	0,753867	0,57825	0,3660992	0,54672	0,505169	0,7903287	0,7815061	0,72333112
F16	1,12919	1,3390289	2,541299	1,65084	1,2407968	2,52952	1,55714	1,9669672	1,59448403	1,90705025
F17	2,66213	3,3813451	53,28958	30,911	0,9766638	38,8965	1,557691	28,333598	2,62282136	21,0348886
F18	0,43056	0,4749765	1,197126	0,23878	0,2440928	3,57667	0,236954	1,6451056	0,44095956	0,52591997
F19	0,07447	0,2050415	1,431944	0,25492	0,0773	0,8277	0,191691	0,0668896	0,12183151	0,70769573
F20	0,02391	0,2733571	0,165033	0,32407	0,1848826	1,31679	0,243349	0,1076009	0,04147881	0,80409414

DE dimensión 30

	CoDE	D-SHADE	EPSDE	JADE	L-SHADE	OP-aCM	SHADE11	SaDE	ynNP-jD	MAES-ILS
F1	26331,9	0,00504	24161,6	447,82	0	0	481,345	298971	46510,2	0
F2	0	0	0	0	0	0	0	0	0	0
F3	0	0	0	0,00056	0	0	0	14,2579	0	0
F4	2,51743	5,03E-09	3,21266	0	0	0	0	37,184	2,03512	0
F5	20,0639	20,0142	20,3465	20,2871	20,1147	20,5171	20,1011	20,5357	20,2913	20
F6	1,98906	0,05924	18,8933	9,42289	1,38E-07	0,71355	0,52891	5,45994	1,1962	0,004
F7	0,00015	0	0,00208	0	0	0	0,00048	0,01233	0	0
F8	0	0	0	0	0	9,97799	0	0,07804	0	2,41701
F9	40,3709	8,70331	44,3622	26,166	6,78488	3,24113	15,8318	38,1205	33,9323	2,56503
F10	0,50019	0,03511	0,2014	0,00531	0,01633	636,003	0,01266	0,26919	0,00408	145,006
F11	1951,29	1303,97	3564,63	1639,94	1229,48	731,495	1396,94	3147,46	1953,74	73,8476
F12	0,05999	0,09665	0,52516	0,27113	0,16058	0,01322	0,16227	0,79419	0,36209	0,02829
F13	0,2313	0,13435	0,24299	0,2203	0,12412	0,03891	0,20401	0,25159	0,2531	0,02951
F14	0,23889	0,23173	0,27812	0,23399	0,2417	0,32782	0,22468	0,22853	0,26566	0,16982
F15	3,17523	1,89185	5,66927	3,09797	2,14637	2,13582	2,56413	4,141	4,75585	2,51102
F16	9,26248	8,51852	11,1465	9,36984	8,49901	10,6241	9,14776	10,9109	9,22422	10,8702
F17	1452,99	210,32	46053,2	9673,39	187,508	852,206	1058,91	11530,7	957,701	1046,75
F18	13,4426	10,3642	331,882	358,052	5,91007	115,353	49,8746	443,834	21,0204	96,0895
F19	2,70413	3,52751	13,3001	4,43732	3,68178	5,69994	4,30572	4,0013	3,9067	6,45644
F20	10,9131	4,20164	50,0425	2885,37	3,08186	24,0393	12,6424	124,543	8,52817	33,5459

Si se comparar con los algoritmos DE, los resultados obtenidos han sido notablemente peores.

El algoritmo GSO creo que es muy bueno para encontrar zonas prometedoras, pero se ve demasiado influenciado por las mejores soluciones obtenidas lo cual puede provocar una convergencia demasiado rápida en óptimos locales. Lo bueno es que su capacidad de encontrar zonas prometedoras puede combinarse con otros algoritmos y llegar a dar resultados realmente buenos. Esto es lo que vamos a ver en el apartado siguiente.

5.2. GSO con local search (CMAES)

Dimensión	M	N	L_1	L_2	EP_{max}	$c_1 = c_2$	$c_3 = c_4$
10	11	4	150	800	5	2.05	1.05
30	20	5	280	1500	5	2.05	1.05

Dimensión 10				**Dimensión 30**		
Función	**Media**	**Desv.**	**Mediana**	**Media**	**Desv.**	**Mediana**
F1	3.41061e-15	6.06922e-15	0	89299.5	47915.4	83088.5
F2	3.41061e-15	9.23596e-15	0	2.6148e-14	7.71062e-15	2.84217e-14
F3	6.82121e-15	1.84719e-14	0	705.366	804.641	393.686
F4	0.159463	0.781207	0	11.4824	1.69733	11.2217
F5	20.1306	0.126057	20.1421	20.6958	0.073157	20.7063
F6	0.102493	0.1131	0.0165172	2.35257	1.66983	1.88441
F7	0.00915841	0.00955326	0.0073960	0.000690428	0.00236724	1.13687e-13
F8	6.15269	1.72933	5.96975	54.1655	16.9623	49.7479
F9	6.08911	1.33469	6.07561	53.1307	15.5595	54.7226
F10	387.025	103.025	381.999	2869.31	845.37	2803.99
F11	315.376	102.658	283.342	3122.64	940.959	2967.59
F12	0.220911	0.212019	0.141145	0.692645	0.647705	0.909875
F13	0.0908851	0.0382812	0.0877907	0.302646	0.0817274	0.295432
F14	0.238046	0.0518777	0.246368	0.342666	0.0543367	0.332612
F15	1.06398	0.295651	1.03589	3.35017	0.785337	3.17071
F16	3.24118	0.180191	3.25163	12.6987	0.245135	12.7228
F17	619.505	271.402	582.308	2645.98	899.895	2609.8
F18	101.77	41.0501	91.4863	736.895	261.758	653.919
F19	1.79736	0.300453	1.65395	14.262	2.04533	14.6611
F20	88.0418	46.0177	83.0554	1246.51	771.103	1081.76

Al meter CMAES en el algoritmo, los resultados se han visto mejorados de forma increíble, sobre todo en aquellas funciones que requerían de una mayor explotación como es la función F1. La desviación típica es mucho menor, lo cual indica que en cada ejecución del algoritmo sobre la misma función se han obtenido resultados similares.

5.3. Comparando GSO con y sin CMAES

Dimensión 10			Dimensión 30	
Función	**GSO**	**GSO-CMAES**	**GSO**	**GSO-CMAES**
F1	665525	3.41061e-15	3.35383e+07	89299.5
F2	7111.75	3.41061e-15	6.55879e+08	2.6148e-14
F3	267.021	6.82121e-15	9155.07	705.366
F4	36.0719	0.159463	253.41	11.4824
F5	20.1538	20.1306	20.5305	20.6958
F6	6.25161	0.102493	30.3963	2.35257
F7	2.96936	0.00915841	6.90277	0.000690428
F8	23.0069	6.15269	135.669	54.1655
F9	28.5642	6.08911	175.089	53.1307
F10	654.016	387.025	4588.56	2869.31
F11	855.566	315.376	4814.08	3122.64
F12	0.42005	0.220911	0.990066	0.692645
F13	0.410964	0.0908851	0.626694	0.302646
F14	0.58442	0.238046	1.59153	0.342666
F15	2.58898	1.06398	111.04	3.35017
F16	3.31785	3.24118	12.383	12.6987
F17	697.934	619.505	2.4189e+06	2645.98
F18	73.4638	101.77	2.62698e+07	736.895
F19	5.71442	1.79736	37.2742	14.262
F20	88.692	88.0418	1779.72	1246.51

Los resultados hablan por si solos, haber incorporado CMAES ha provocado una mejora espectacular en practicamente todas las funciones, y en aquellas funciones que no ha conseguido mejorar ha obtenido resultados muy similares. Los resultados obtenidos por GSO con CMAES ya podrían incluso competir con los algoritmos DE, cuyas tablas se han mostrado antes. los resultados en general son parecidos a los algoritmos DE y seguramente si siguieramos ajustando parámetros podríamos haber quedado de los primeros en dicha tabla.