Metaheurística
Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

Práctica 1
Metaheurísticas novedosas: Brain Storm Optimization

Iván Sevillano García

DNI: 77187364-P

E-mail: ivansevillanogarcia@correo.ugr.es

7 de junio de 2018

Índice

1. Presentación de la metaheurística. Problema que soluciona.

Esta metaheurística la comenzó a desarrollar Yuhui Shi en la universidad de Liverpool[1]. Está basada en la idea de que el desarrollo de soluciones para problemas de distinta índole se basa en el desarrollo de una lluvia de ideas, las cuales se pueden ir mejorando con distintos mecanismo.

Esta idea surgió porque las metaheurísticas bio-inspiradas han dado buenos resultados en distintos campos. Las que se basan en poblaciones, por ejemplo, se basan en paralelismos con peces, hormigas, pájaros, peces... Y siendo el ser humano un ser vivo más (y, según el paper, el más inteligente), ¿Por qué no utilizar su forma de llegar a soluciones óptimas a problemas para resolver problemas de optimización?

1.1. Proceso humano de la lluvia de ideas. Modelo abstracto.

El mecanismo de lluvia de ideas para generar ideas para un proyecto es muy conocido. En él se genera entre todos los individuos tantas ideas como se pueda para luego trabajar sobre ellas e intentar encaminar la búsqueda de la mejor solución. El algoritmo que describen en el paper sería el siguiente:

1. Generamos ideas de distinta índole sin importar lo buenas o malas que sean a priori.
2. Sobre estas ideas, seleccionamos unas pocas ideas buenas que sean representativas del resto de ideas. Estas ideas tendrán más probabilidad de ser seleccionadas para ser mejoradas.
3. Se intenta potenciar las ideas seleccionadas con más probabilidad tanto mezclando con otras como profundizando en la misma.
4. Con suerte, se habrá generado una idea lo suficientemente buena. Se puede repetir el proceso de optimización de ideas tantas veces se quiera o se pueda.

Cuando utilizamos la idea de idea representante y le damos más prioridad la utilizamos para que se utilice esta idea como un punto atractor del resto. Puesto que tiene mejor puntuación, debe de ser una idea entorno a la cuál profundizar la búsqueda de soluciones óptimas. También, como diferenciamos entre distintos representantes, actúan como distintos puntos atractores, manteniendo la búsqueda diversificada.
La creación de ideas sigue unas reglas descritas en el paper referidas a las regalse descritas en [2], las cuales se detallan a continuación:

- Ninguna idea, a priori, es mala
- Todo lo que se ocurra será productivo para la generación de la idea optimizada.

- La generación de nuevas ideas deben generarse en base a ideas previas generadas.
- Prioriza la cantidad, la calidad vendrá a sola.

Estas reglas se crean para generar tantas ideas como podamos y para que estas tengan mucha diversidad. Cuantas más ideas tengamos, más posibilidad tendremos de encontrar mejores soluciones.

2. Breve descripción de los algorítmos utilizados.

Tras la introducción abstracta de la lluvia de ideas, veamos cómo formalizar este procedimiento para la optimización de parámetros. Para empezar, desarrollaremos los paralelismos:

- **Idea**. El concepto de idea lo transportamos al de solución, que para la optimización de parámetros será un vector de tantos elementos reales como dimensión del problema.
- **Generación de ideas aleatorias.** Generaremos ideas generando vectores aleatorios dentro del conjunto donde buscamos.
- **Ideas semejantes.** Para separar ideas por "semejanza", utilizaremos algoritmos de clustering en base a la distancia euclidea de los vectores que representan nuestra solución. En el paper se plantea el algoritmo específico de las k-medias.
- **Idea representante.** Una idea representante será la mejor idea de cada cluster.
- **Generación de ideas en base a ideas ya generadas.** Una generación de idea en base a una ya generada podría ser una alteración pequeña de la idea base. Mezclar dos o más ideas para generar una nueva lo haremos a través de cualquier algoritmo de cruce de soluciones.

El paper propone una mutación específica:

$$X^d_{new} < -X^d_{selected} + \psi \times n(0,1)$$

$$\psi = logsig\left(\frac{\frac{Max_{iter}}{2} - Curr_{iter}}{k}\right) \times random()$$

donde la función $logsig(x) = \frac{1}{1+exp(-x)}$ es un sigmoide, la función n es una variable aleatoria normal y $random$ una variable aleatoria uniforme en (0,1). Lo que pretende el autor con esta función de mutación específica es que, al principio del algoritmo, las mutaciones consigan diversidad en las soluciones buscando en lugares lejanos a la solución actual. También consigue que, conforme avance el algoritmo, las mutaciones busquen optimizar la solución que muta.

Aunque el paper no propone una función de cruce en específico, para las pruebas se ha utilizado el cruce utilizado con Differential-Evolution con parámetro F = 0,5:

$$X_{new} = X_1 + F \times (X_2 - X_3)$$

En el algoritmo se propone que sólo se sustituya la nueva idea por la primera idea escogida(si es mejor). Nosotros la sustituiremos por la peor de ellas(si es mejor).

En base a esto, pasamos a definir el algoritmo propuesto en el paper:

1. Generamos n ideas de forma aleatoria.
2. Reunir por el algoritmo de clustering las n ideas generadas en m clusters.
3. De cada cluster, guarda la idea que tenga mejor puntuación de la función de coste. Esta será la idea representante.
4. Con probabilidad p_{muta}, se selecciona un representante de cluster y se cambia por una idea generada de forma aleatoria.
5. Generamos nuevos individuos. Con una probabilidad $p_{profundiza}$:
   - Selecciona un cluster específico:
     a) Con una probabilidad $p_{representante}$, mutas al representante de cluster.
     b) Si no, mutas un individuo aleatorio del cluster.
   - Selecciona varios cluster(en el paper solo 2):
     a) Con un probabilidad $p_{representantes}$, generas una idea en base a los representantes de los clusters seleccionados.
     b) Si no, se seleccionan de cada cluster una idea aleatoria y se genera una nueva idea en base a las ideas seleccionadas.
6. Si la nueva idea generada supera a su predecesora, esta es sustituida y la nueva idea es aceptada.
7. Mientras no se hayan generado y aceptado n nuevos individuos, se vuelve al paso 5, generar nuevos individuos.
8. Si se han generado ya el máximo de individuos, termina el algoritmo. Si no, vuelve al paso 3, realizar clustering sobre las ideas generadas.

3. Pseudocódigos y explicaciones.

El algoritmo esencial se describe a continuación:

```
Generamos n ideas aleatorias: Ideas[n]

Mientras no hayamos sobrepasado el limite de evaluaciones:
  Hacemos K clusters de Ideas: Clusters[K]

  Mientras no hayamos modificado con exito n nuevas ideas:
    Seleccionamos con probabilidad c_p
    Con probabilidad c_m, cambiamos el centro de cluster por una
     solucion aleatoria.

    Con probabilidad c_e:
      Escogemos un cluster con probabilidad c_p
      Con probabilidad c_r:
        Mutamos el representante del cluster seleccionado
      Si no:
        Mutamos una idea aleatoria del cluster
    Si no:
      Escogemos 3 clusters
      Con probabilidad c_c:
        Producimos una idea a traves de los 3 centros de cluster y
        la cambiamos por el primer representante de cluster.
      Si no:
        Producimos una idea a traves de 3 ideas, una de cada cluster
        seleccinado, y la cambiamos por la primera de ellas.

  Si el individuo creado es mejor que la idea que sustituye, se realiza
  con exito la modificaci n. Si no, se mantiene la anterior idea.
```

Cada una de las probabilidades antes mencionadas se describen a continuación:

- c_p. Es la probabilidad con la que queremos escoger un cluster. Esta probabilidad será proporcional al número de ideas que contenga este cluster.
- c_m. Esta probabilidad define cuánto queremos introducir nuevas ideas de cero.
- c_e. Es la probabilidad con la que vamos a intensificar la búsqueda local por mutación de alguna idea en concreto. La complementaria es la probabilidad con la que vamos a cruzar varias ideas.

- c_r**.** Es la probabilidad con la que, dentro de la intensificación de la búsqueda, vamos a aplicar la intensificación en el representante de cluster.
- c_c**.** Es la probabilidad con la que, dentro del cruce de ideas, vamos a cruzar los representantes de cluster.

En el paper se proponen los siguientes valores de los parámetros:

Parámetro	Valor
n	100
K	5
c_m	0.2
c_e	0.8
c_r	0.4
c_c	0.5

Tabla 3.1: Datos obtenidos para el algoritmo BL

4. Demo del algortimo: Prueba con Rastrigin en dimensión 2.

Para ver el comportamiento del algoritmo, hemos probado a minimizar la función de coste Rastrigin para dimensión 2:

$$f(X) = \sum_{i=1}^{2}(X_i^2 - 10cos(2\pi X_i) + 10)$$

Los gráficos donde se muestran los distintos clusters, representantes de cluster y avanze en cada generación, están en el directorio $./graficos/$. Aquí sólo mostraremos la primera generación, una intermedia y la última:

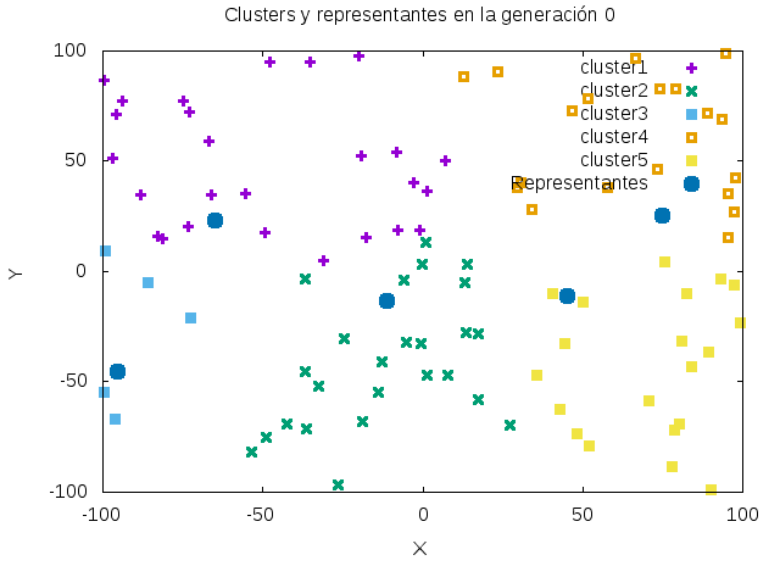

Figura 4.1: Coste mínimo: 32.063385819971245

En esta primera generación de ideas aleatorias se tienen separados por colores cada uno de los clusters anteriormente dichos. Cada uno de ellos tiene un punto azul, el cuál es su representante de clase. Es la idea que, dentro de su cluster, tiene menor coste. Esos puntos será donde centraremos nuestra búsqueda y actuarán como punto de atracción.

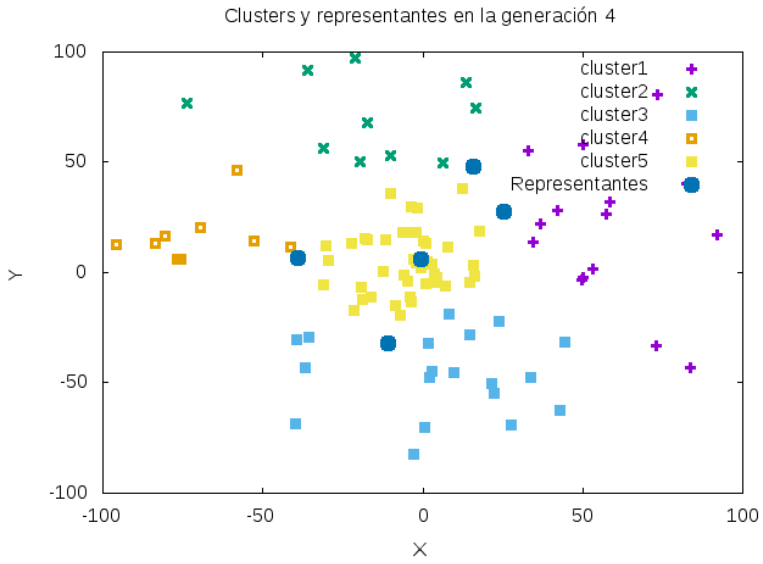

Figura 4.2: Coste mínimo: 10.868608361185013

Tras unas cuantas generaciones vemos que se han agrupado ligeramente en el centro unos cuantos puntos (que es donde sabemos que la función de coste tiene un mínimo). Por último, en la última generación:

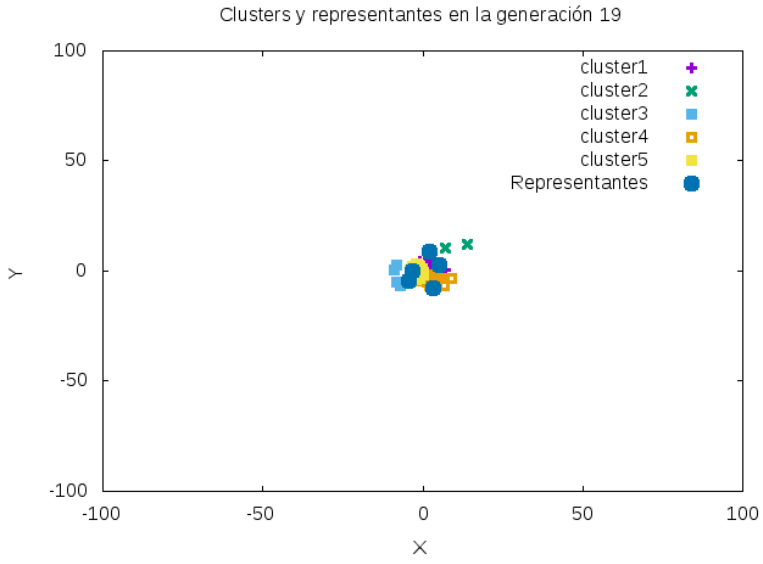

Figura 4.3: Coste mínimo: 0.20025825415018517

se observa que se han agrupado todavía más en el centro.

5. Experimento y análisis de resultados.

Para probar la bondad de este algoritmo vamos a compararlos con los resultados de la competición cec2014. Para empezar, aquí tenemos los resultados de nuestro algoritmo para las distintas dimensiones:

Dimensión 10

Función	MejorCoste
1	1183474.3628089789
2	68864449.62907574
3	12052.281460498438
4	69.51174869048253
5	20.248590712702935
6	7.753179595914844
7	3.037059227484292
8	37.275446377989056
9	43.98963821630082
10	1000.5484210292489
11	1113.7294002188828
12	0.6969247915296819
13	0.307090955059266
14	0.48214180040963583
15	7.174151442472294
16	3.337923901207205
17	5020.6204587966295
18	1390.9914594921079
19	4.2737342582693145
20	418.4109712910604

Tabla 5.1: Datos obtenidos para el algoritmo BL

5.1. Análisis de resultados

5.2. Posibles mejoras de los algoritmos

Referencias

[1] 2011 - BSO (IJSIR) -Brain Storm Optimization Algorithm

[2] Smith, R.: The 7 Levels of Change, 2nd edn. Tapeslry Press (2002)