

Metaheurística Battle Royale

Autor: Andrés Millán Muñoz

Repositorio:

github.com/asmilex/cec2017real





**¿Qué es un battle royale
y por qué es interesante?**



Los videojuegos del tipo battle royale

- Juegos competitivos individuales o por equipos.
- Exploración del mapa y movimientos forzados por los desarrolladores mediante anillos, de forma que se acorralan a los equipos.
- Gana el equipo que quede en pie.


Ejemplo: **Apex Legends**.



Inicio del juego. Los equipos empiezan en la zona del mapa deseada



Conforme avanzan las rondas, el terreno se reduce.



Ideando una metaheurística



Primera idea: segmentar el espacio de búsqueda

Podemos dividir el espacio de búsqueda en sectores, y hacer que combatan las soluciones que se encuentren en un mismo sector.

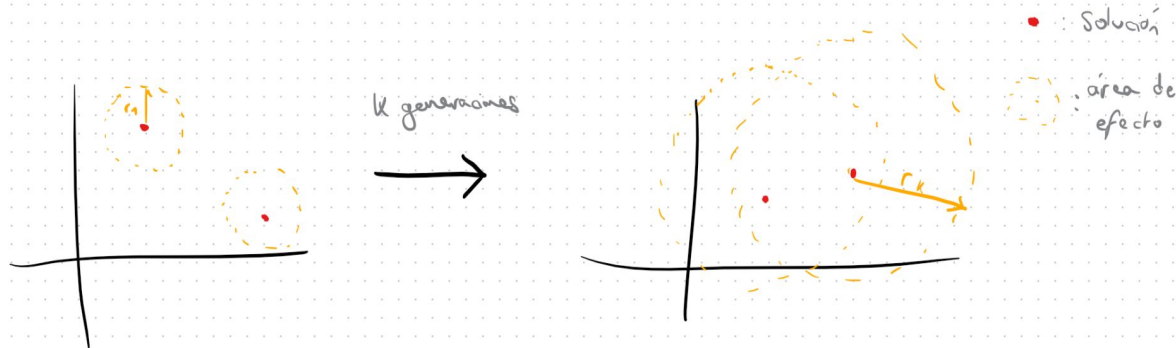
- **Problema:** complejidad exponencial.
 - Dimensión = 2, dividir en 2 zonas/dimensión => 4 sectores (matriz de 2x2).
 - Dimensión = 3, dividir en 2 zonas/dimensión => 8 sectores (8 cubos dentro del espacio).
- **Solución:** anillo dinámicos dependientes de la solución.

Anillos alrededor de las soluciones

Cada solución tendrá una bola de cierto radio alrededor suya. Conforme avancen las generaciones, aumentará.

Cuando dos soluciones se encuentren mutuamente, combatirán. La peor, morirá.

Ejemplo: dimensión = 2





Consideraciones y parámetros

Problema

- ¿Cómo explorar?
- ¿Aumento del radio?
- ¿Balance población-BL?

Solución

- Aplicar Solis-Wets.
- Primera versión: radio + constante.
- Población inicial: 20, 100 evaluaciones/pasada BL.



Primera versión

Enlace al pseudocódigo: <https://kutt.it/eqyLVB>

Inconvenientes:

- **No rinde bien.**
- **Las soluciones se quedan estancadas.**
- **No vence ni a PSO ni a DE.**

Dimensión 10

Porcentaje	DE	PSO	mh-battle-royale
5%	7	10	13
20%	23	6	1
50%	21	7	2
100%	20	8	2

Dimensión 30

Porcentaje	DE	PSO	mh-battle-royal
5%	8	3	19
20%	17	2	11
50%	23	2	5
100%	22	3	5

Dimensión 50

Porcentaje	DE	PSO	mh-battle-royale
5%	8	4	18
20%	14	1	15
50%	17	2	11
100%	21	2	7



Necesitamos mejorar



Segunda versión: partir de mejores soluciones iniciales

- Intentar empezar con soluciones más prometedoras.
- Idea: generar cierto número de vecinos y quedarnos con los mejores.
- Gastamos evaluaciones al inicio, pero nos las ahorramos de la búsqueda local.



Generación de vecinos mejorada

```
generar_poblacion_inicial(elementos_poblacion, aleatorios_a_generar, dim) {  
    for (_ in 0 .. aleatorios_a_generar) {  
        poblacion.push(  
            generar_nueva_solucion(dim)  
        )  
    }  
  
    poblacion.sort()  
    poblacion.erase(poblacion.begin() + elementos_poblacion, poblacion.end())  
  
    return poblacion  
}
```

Parámetros:

- aleatorios_a_generar = 100




Tercera versión: mutación de la ganadora

Nos quedamos estancados en las últimas generaciones.

Para arreglarlo, cuando quede un único elemento, lo mutaremos aleatoriamente varias veces y reintroduciremos los resultantes en la población.

Además, haremos retroceder al radio para hacer la batalla más justa.



```
incrementar_radio (radio_antiguo) {  
    return radio_antiguo * 1.1 + k  // k = 1 en esta versión  
}
```



Este esquema de aumento del radio balancea la exploración en las primeras fases con las batallas de las últimas.

```

mutar (original) {
    s = original.clone()

    inicio_segmento = aleatorio en [0, s.size())
    tamano_segmento = aleatorio en (-100, 100)

    i = inicio_segmento
    copias = 0

    while (copias < tamano_segmento) {
        s[i] = original[i]

        i = (i+1) % s.size()

        copias++
    }

    while (true) {
        s[i] = aleatorio en (-100, 100)

        i = (i+1) % s.size()

        if (i == inicio_segmento) {
            break
        }
    }

    return s
}

```

Funcionamiento:

- Se genera un segmento de tamaño y posición aleatoria, y se traspasa al hijo.
- Los genes restantes se generan aleatoriamente en el espacio.

Para hacer la batalla más justa cuando se reintroduzcan los hijos, reduciremos el radio a $\text{radio}/5$.

Adicionalmente, aumentaremos el tamaño la población inicial a 25.



Resultados finales



Dimensión 10

Evaluaciones	DE	MH <u>Battle Royale</u>	PSO
1%	3	20	7
2%	4	16	10
3%	4	17	9
5%	8	14	8
10%	19	5	6
20%	22	2	6
30%	21	3	6
40%	22	2	6
50%	21	2	7
60%	19	3	8
70%	17	5	8
80%	17	5	8
90%	17	7	7
100%	17	9	5

Dimensión 30

Evaluaciones	DE	MH <u>Battle Royale</u>	PSO
1%	3	16	11
2%	2	17	11
3%	2	19	9
5%	10	15	5
10%	16	11	3
20%	19	8	3
30%	20	8	2
40%	20	8	2
50%	21	8	1
60%	23	6	1
70%	22	7	1
80%	20	9	1
90%	22	7	1
100%	21	8	1



Mejoras obtenidas en los resultados

- Se mejoran las primeras evaluaciones, debido al sistema de generación de vecinos.
- Mutaciones añaden diversidad en la parte final, lo que produce una mejora en las últimas evaluaciones.
- Importante aumento en las medias de la dimensión 30.
- **El algoritmo sigue siendo algo mediocre.**



Los obstáculos se repiten

- Seguimos atascados en las últimas fases. La mutación no ha sido suficiente.
- El aumento del radio podría ser más granular.
- Seguimos atascados en óptimos locales.
- Nuestro algoritmo se parece mucho a un memético, aún habiendo partido de los juegos battle royale. ¿Estamos traspasando demasiadas ideas de otros algoritmos? ¿Inspiración natural limitada?
- Solo comparamos con DE y PSO. Son dos algoritmos débiles generalmente.