Proyecto: Algoritmo Memético Aplicado a Optimización Continua

Juan Gerardo Fuentes Almeida

Abstract—En este proyecto se implementa un algoritmo memético para optimización continua, aplicado a un problema de optimización de parámetros de un mecanismo de 4 brazos.

1 Introducción

Se dice que la cultura humana está conformada por unidades de conocimiento denominadas *memes*. Un meme es un bloque de conocimiento que puede ser duplicado en el cerebro humano y ser modificado o combinado con otros memes para generar nuevos. Dentro de un comunidad, algunos memes sencillamente no son interesantes y se extinguen en un periodo corto de tiempo; otros son más bien resistentes y se propagan entre la comunidad entera, como si fuera una infección. También pueden experimentar ligeras modificaciones o combinarse entre ellos y asi generar nuevos memes con características mas resistentes y ser mas durables y propensos a propagarse.

Esta interpretación de la cultura humana inspiró a Moscato y Norman a finales de los 80's para definir los Algoritmos Meméticos (MA's) como una modificación de los Algoritmos Genéticos (GA's) incluyendo una búsqueda local.

2 Teoría

El Algoritmo Memético Estándar se compone de los siguientes operadores:

1) Selección de padres.
2) Combinación de los padres para generar descendencia.
3) Mejoramiento local de la descendencia.
4) Actualización de la población.

La estructura básica del Algoritmo Memético se muestra en el Algoritmo 1. Los procedimientos *Cooperate*, *Improve* y *Compete* constituyen el núcleo principal del algoritmo, la función *Cooperate* involucra a todos los individuos de la población para generar una descendencia que puede poseer rasgos ya sea de uno o varios padres. La función *Improve* ejecuta una búsqueda local dentro de la nueva población generada, con el propósito de mejorar localmente la descendencia generada, mientras que la función *Compete* realiza un proceso de selección entre la población nueva y la anterior para determinar los individuos que pasarán a la siguiente generación.

```
function BasicMA (in P: Problem, in par: Parameters):
Solution;
begin
    pop ← Initialize(par, P);
    repeat
        newpop_1 ← Cooperate(pop, par, P);
        newpop_2 ← Improve(newpop_1, par, P);
        pop ← Compete (pop, newpop_2);
        if Converged(pop) then
            pop ← Restart(pop, par);
        end
    until TerminationCriterion(par);
    return GetNthBest(pop, 1);
end
```

Algorithm 1: A basic memetic algorithm

El procedimiento de Inicialización, descrito en el Algoritmo 2, es el responsable de producir el conjunto inicial de $|pop|$ soluciones. Esto puede realizarse de manera aleatoria, pero típicamente en un Algoritmo Memético se intenta producir soluciones de alta calidad desde el inicio, ya sea aplicando alguna heurística constructiva o una búsqueda local para mejorar las soluciones aleatorias.

```
function Initialize(in par: Parameters, in P: Problem):
Bag{Solution};
begin
    pop ← ∅;
    for j ← 1 to par.popsize do
        i ← RandomSolution(P);
        i ← LocalSearch (i, par, P);
        pop ← pop ∪ {i};
    end
    return pop;
end
```

Algorithm 2: Injecting high-quality solutions in the initial population.

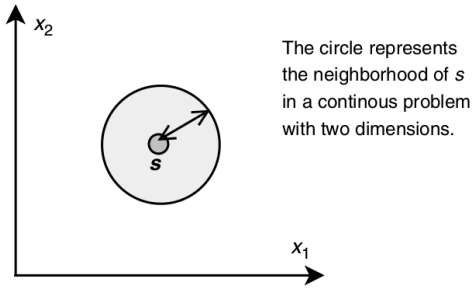
La función *Cooperate* (Algoritmo 3) parte de la utilización de dos operadores de selección de individuos de entre la población y recombinación entre ellos, pero también ser fácilmente extendido al uso de una colección mas grande de operadores de variación.

```
function Cooperate (in pop: Bag{Solution}, in par:
Parameters, in P: Problem): Bag{Solution};
begin
    lastpop ← pop;
    for j ← 1 to par.numop do
        newpop ← ∅;
        for k ← 1 to par.numapps^j do
            parents ← Select (lastpop, par.arityin^j);
            newpop ← newpop ∪ ApplyOperator (par.op^j,
            parents, P);
        end
        lastpop ← newpop;
    end
    return newpop;
end
```

Algorithm 3: The pipelined Cooperate procedure.

El procedimiento de mejora incorpora la aplicacion de una busqueda local entre los individuos de la poblacion, teniendo en cuenta que para optimizacion continua, la vecindad $N(s)$ de una solucion s es una hiperesfera con centro en s y radio igual a ϵ con $\epsilon > 0$.

Por tanto, se tiene que $N(s) = \{s' \in \Re^n : \| s' - s \| < \epsilon\}$ donde $\| s' - s \|$ es la norma Euclidiana.



Vecindad de una solución

Despues de realizar una serie de operadores, la poblacion generada $newpop2$ "compite" contra la poblacion anterior pop, construyendo asi una nueva poblacion a partir de estas dos. Existen dos posibles maneras de realizar este procedimiento, a saber, la estrategia *Plus* y la estrategia *Comma*:

1) Estrategia Plus: La nueva población es construida a partir de las mejores $popsize$ configuraciones de $pop \cup newpop$.

2) Estrategia Comma: Las mejores $popsize$ configuraciones se toman solamente de $newpop$. En estos casos, se requiere que $|newpop| > popsize$ para incrementar la presión de selección del proceso.

En cuanto al procedimiento de reinicio, se debe decidir si la población se ha degradado o no, utilizando alguna medida de diversidad, tal como la desviación estándar en el caso de optimización continua o la entropía de Shannon en el caso discreto. Una estrategia típica de reemplazo consiste en conservar una fracción de la población actual y generar soluciones nuevas (de forma aleatoria o heurística) para completar el resto.

```
function Restart (in pop: Bag{Solution}, in par: Parameters,
in P: Problem): Bag{Solution};
begin
    newpop ← ∅;
    for j ← 1 to par.preserved do
        i ← GetNthBest(pop, j);
        newpop ← {i};
    end
    for j ← par.preserved + 1 to par.popsize do
        i ← RandomSolution(P);
        i ← LocalSearch (i, par, P);
        newpop ← {i};
    end
    return newpop;
end
```

Algorithm 4: The restart procedure.

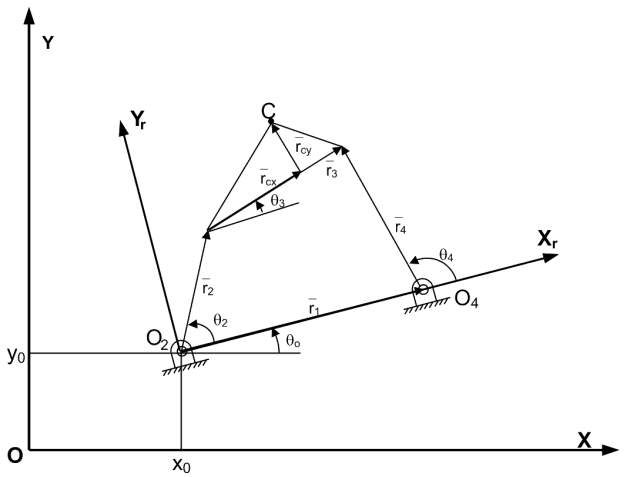
El criterio de paro puede elegirse de entre cualquiera de estas opciones:

1) Límite en el numero total de iteraciones.
2) Alcanzar un numero máximo de iteraciones sin ninguna mejora.
3) Haber ejecutado un cierto numero de reinicios.
4) Haber alcanzado un cierto valor de fitness.

3 Descripción del Problema

3.1 Mecanismo de cuatro brazos

El objetivo es que la terminal C alcance los mas posible a un conjunto de puntos de precision $\{C_d^i\}$. El par motriz está asociado a θ_2 mientras que $r_1, r_2, r_3, r_4, r_{cx}, r_{cy}, x_0, y_0$ y θ_0 son parámetros de diseñ a ser optimizados. Dados estos parámetros, es posible calcular las coordenadas de la terminal C.



Con respecto al sistema coordenado en x_0, y_0 y θ_0:

$$\widehat{C}_r = \widehat{r}_2 + \widehat{r}_{cx} + \widehat{r}_{cy}$$
$$C_{xr} = r_2\cos\theta_2 + r_{cx}\cos\theta_3 - r_{cy}\sin\theta_3$$
$$C_{yr} = r_2\sin\theta_2 + r_{cx}\sin\theta_3 + r_{cy}\cos\theta_3$$

Transformando al sistema coordenado **O**xy:

$$\begin{bmatrix} C_x \\ C_y \end{bmatrix} = \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 \\ \sin\theta_0 & \cos\theta_0 \end{bmatrix} \begin{bmatrix} C_{xr} \\ C_{yr} \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

Ecuación de lazo cerrado del mecanismo:

$$\widehat{r}_1 + \widehat{r}_4 = \widehat{r}_2 + \widehat{r}_3$$
$$r_2\cos\theta_2 + r_3\cos\theta_3 = r_1 + r_4\cos\theta_4$$
$$r_2\sin\theta_2 + r_3\sin\theta_3 = r_4\sin\theta_4$$

Elevando al cuadrado ambas ecuaciones:

$$r_4^2\cos^2\theta_4 = (r_2\cos\theta_2 + r_3\cos\theta_3 - r_1)^2$$
$$r_4^2\sin^2\theta_4 = (r_2\sin\theta_2 + r_3\sin\theta_3)^2$$

Sumando y aplicando identidades trigonométricas obtenemos:

$$r_4^2 = r_1^2+r_2^2+r_3^2+2r_2r_3\cos(\theta_2-\theta_3)-2r_1r_3\cos\theta_3-2r_1r_2\cos\theta_2$$

Reacomodando y redefiniendo los términos, formulamos la expresión conocida como *Ecuación de Freudenstein*:

$$k_1\cos\theta_3 + k_2\theta_2 + k_3 = \cos(\theta_2 - \theta_3)$$

Donde:

$$k_1 = \frac{r_1}{r_2};\quad k_2 = \frac{r_1}{r_3};\quad k_3 = \frac{r_4^2 - r_1^2 - r_2^2 - r_3^2}{2r_2r_3}$$

Sea $\varphi = tan(\frac{\theta_3}{2})$, entonces

$$\sin\theta_3 = \frac{2\varphi}{1+\varphi^2};\quad \cos\theta_3 = \frac{1-\varphi^2}{1+\varphi^2}$$

Sustituyendo en la ecuación de Freudenstein y reacomodando términos:

$$k_1(\frac{1-\varphi^2}{1+\varphi^2}) + k_2\cos\theta_2 + k_3 = (\frac{1-\varphi^2}{1+\varphi^2})\cos\theta_2 + \frac{2\varphi}{1+\varphi^2}\sin\theta_2$$

$$\varphi^2[k_3+(k_2+1)\cos\theta_2-k_1]+\varphi(-2\cos\theta_2)+[k_1+(k_2-1)\cos\theta_2+k_3] = 0$$

Aplicando la formula cuadrática:

$$\Rightarrow \varphi = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Donde:

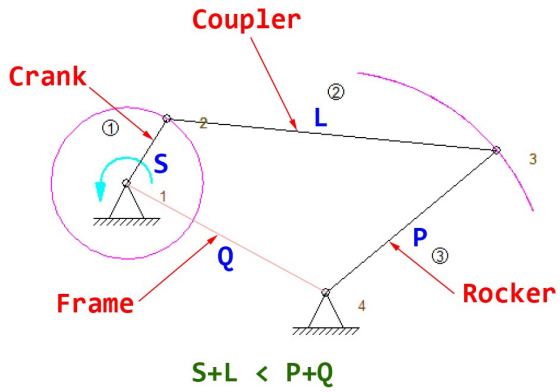
$$a = k_3 + (k_2+1)\cos\theta_2 - k_1$$
$$b = -2\cos\theta_2$$
$$c = k_1 + (k_2-1)\cos\theta_2 + k_3$$

Luego, θ_3 puede ser calculado como $\theta_3 = 2tan^{-1}\varphi$

Si $\varphi \notin \Re$, el mecanismo no puede construirse con las configuraciones dadas.

3.2 Restricciones

1) Los ángulos de entrada θ_2^i deben ser ordenados de modo que sean consecutivos en una circunferencia de 2π radianes.
2) Todas las magnitudes de lo brazos deben ser positivas: $r_1, r_2, r_3, r_4, r_{cx}, r_{cy} > 0$.
3) Se impone la condición de Grashof para formar una configuración *Crank-Rocker*, esto es $r_1+r_2 < r_3+r_4$, con $r_2 < r_3, r_4 < r_1$, y el brazo más corto conectado a la base.



4 Implementación

4.1 Función Objetivo

Con base en la definición del problema explicada anteriormente, el algoritmo de optimización se implementa considerando las restricciones 1), 2) y 3) al asignar valores a las variables de diseño, es decir, se procura que los valores que toman las variables desde el inicio cumplan con estas condiciones, asimismo las restricciones se incluyen como penalizaciones dentro de la función objetivo de la siguiente manera:

$$f(X) = \sum_{i=0}^{N} \parallel C_d^i(X) - C^i(X) \parallel^2 + M_1h_1(X) + M_2h_2(X)$$

donde:

$\parallel C_d^i(X) - C^i(X) \parallel \rightarrow$ Norma euclidiana
$X = [r_1, r_2, r_3, r_4, r_{cx}, r_{cy}, \theta_0, \theta_2^1, \theta_2^2, ..., \theta_2^N]$
$h_1(X) \in \{0, 1\} \rightarrow$ Condición Grashof verdadera/falsa.
$h_2(X) \in \{0, 1\} \rightarrow$ Condición de ordenamiento para θ_2 verdadera/falsa.

y M_1, M_2 son constantes de valor muy alto que penalizan la función objetivo cuando se violan las restricciones asociadas.

4.2 Operadores

4.2.1 Recombinación

La recombinación puede deninirse como un proceso en el cual un conjunto S_{par} de n configuraciones (padres) se manipula para crear otro conjunto $S_{desc} \subseteq sol_P(x)$ de m

r1	r2	r3	r4	rcx	rcy	x0	y0	theta0	Fmin
39.477382	15.330624	30.282259	58.593938	16.349212	23.587588	18.353512	5.246827	2.406239	0.176197
52.255772	12.963156	38.822416	46.179199	29.733312	0	22.765213	2.971741	0.73407	0.290153
37.58737	11.062279	24.698863	31.426664	26.280493	19.671253	0.431168	59.828429	3.811697	0.492721
29.388636	9.146775	14.192141	29.077303	10.920854	10.500886	26.540753	22.017716	0.231578	0.545703
59.952574	7.946658	31.275002	52.954867	17.482117	41.322055	52.04054	2.727492	2.327016	0.645628
19.026636	11.280449	45.859712	58.950268	12.232058	20.928936	8.947765	12.766181	2.007242	0.701864
60	7.104025	14.192972	57.760989	0	19.866334	32.935928	22.402985	2.295407	0.793279
29.295827	4.937699	60	36.732168	40.997202	15.821049	58.382148	13.845604	1.901042	0.870534
58.17197	9.922682	35.633674	52.388571	20.629705	37.075086	45.021406	0	2.279864	0.908264
21.480291	11.844333	43.228705	53.456131	15.758176	22.164237	9.759329	8.283996	1.928707	1.009136
22.881636	14.286364	29.883071	42.044654	15.672735	26.264395	17.629955	4.438734	2.010615	1.038692
45.7642	10.540563	31.458489	38.551879	18.450884	42.582394	51.806786	4.90329	2.188763	1.05932
42.862629	29.577117	37.683674	41.577883	36.155558	0.745587	20.520085	50.527588	-0.423013	1.530678
51.784809	15.479952	25.102741	43.122107	7.514159	20.860808	0.593934	52.626693	3.450889	1.551127
44.461371	8.072742	24.328193	60	23.047117	4.245846	5.96068	15.801855	2.76915	1.618656
10.006353	50.767216	60	18.46964	58.277171	14.421535	8.419757	33.968239	0.681897	1.696476
48.218725	11.167331	15.685003	44.794415	6.319886	11.824522	21.753671	21.450487	6.188749	1.747255
60	15.80322	24.284867	56.266316	18.145474	11.413215	10.324454	47.682454	3.362731	1.755067

Tabla 1. Resultados preliminares.

r1	r2	r3	r4	rcx	rcy	x0	y0	theta0	Fmin
55.996129	11.860317	26.684302	56.960179	30.735369	9.883953	3.142643	60	3.639357	0.0078
59.875522	10.727668	24.577835	58.406998	26.280596	12.157331	5.141042	56.566359	3.560392	0.019058
21.780566	16.606229	23.301379	35.007264	31.545113	8.443519	1.877289	17.435899	2.131829	0.056205
48.925425	9.471383	14.60938	60	14.385883	3.037092	10.834641	23.031599	2.919015	0.057158
59.357222	15.341739	31.193179	53.203131	31.443674	10.591183	10.075585	60	3.631323	0.068617
54.545358	12.828405	30.386007	60	22.220154	3.507703	22.268048	54.857281	5.834672	0.123284
58.653971	12.605048	23.248806	59.017517	25.818171	2.040868	17.393864	6.96745	2.719125	0.127332
48.205104	11.643272	22.048389	60	21.951581	7.189605	10.931092	12.262803	2.706078	0.135521
50.17406	9.547689	14.303315	48.069709	11.681369	12.891571	27.696188	19.179331	0.099839	0.136301
18.452372	12.747353	11.731494	26.301019	19.475191	8.622404	0.262667	47.633093	3.815419	0.141773
52.293613	5.36955	15.977059	57.319538	18.300585	23.233571	44.530481	20.244287	6.298891	0.151417
47.218007	7.353142	17.068789	46.920148	14.20552	23.388206	39.890086	18.257959	6.40881	0.165234
43.488833	11.53675	23.21043	53.100963	23.014422	6.763177	11.933853	10.399746	2.636528	0.167028
59.991858	11.549794	23.951936	57.012862	24.562784	8.267317	11.411802	56.637731	3.527898	0.173476
59.857459	14.703776	18.057195	59.966778	17.321549	6.453634	13.04751	45.753718	3.383988	0.1749
39.824146	11.749458	32.012771	60	27.747639	13.609629	5.615179	5.188327	2.492471	0.175453
32.777872	10.658131	21.95618	53.432359	15.250522	9.850886	8.52869	23.64741	2.712092	0.202074
20.110768	7.897732	32.785112	35.031658	20.60465	9.357022	35.098083	20.165608	0.763898	0.205023
29.810541	8.889684	50.570899	41.56882	34.653848	2.929273	42.69272	7.769873	1.333422	0.210038
46.149813	11.615341	20.813982	39.843859	12.552755	17.613575	4.851433	49.275176	3.519957	0.210232

Tabla 2. Resultados mejorados.

nuevas configuraciones (descendientes).

La generación de estos descendientes involucra la identificación y combinación de características extraídas de los padres y transmitidas a la descendencia. En esta implemetación se utiliza un método de recombinación denominado *Dynastically Optimal Recombination* (DOR), en el cual cada recombinación posible es explorada para encontrar la configuración con mejor fitness, la consiguración obtenida será cuando menos tan buena como el mejor padre.

Además, estamos considerando una recombinación parcial, esto es, sólo se consideran ciertos bloques de información en la recombinación de individuos; a saber, los conjuntos $\{r_1, r_2, r_3, r_4\}$, $\{r_{cx}, r_{cy}\}$, $\{x_0, y_0, \theta_0\}$ y $\{\theta_2^1, \theta_2^2, ..., \theta_2^N\}$

4.2.2 *Mutatción*

En la mutación, se crea un individuo x_i' a partir de cada padre $x_i, \forall i \in \{1, 2, ..., popsize\}$ por medio de

$$x_i'(j) = x_i(j) + \epsilon N_j(0, 1)$$

donde j es un entero aleatorio $\in \{1, 2, ..., popsize\}$, y $N_j(0, 1)$ es un valor aleatorio generado a partir de una distribución normal estándar. La constante ϵ es el radio de búsqueda local ya mencionado anteriormente.

5 RESULTADOS

5.1 Preliminares

La Tabla 1 muestra los mejores resultados obtenidos después de una prueba preliminar de 100 ejecuciones del algoritmo, se consideran las siguientes especificaciones:

Puntos de precisión:

$$C_d^i = \{(20, 20), (20, 25), (20, 30), (20, 35), (20, 40), (20, 45)\}$$

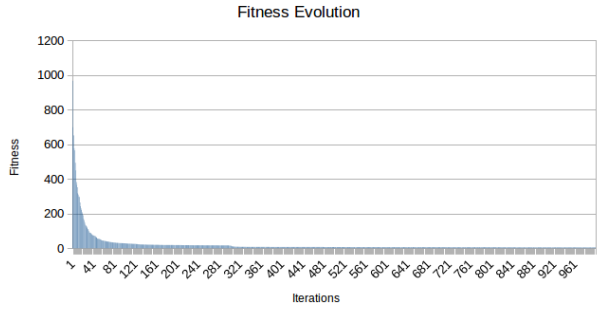


Figura 1. Evolución del fitness del algoritmo con recombinación DOR.

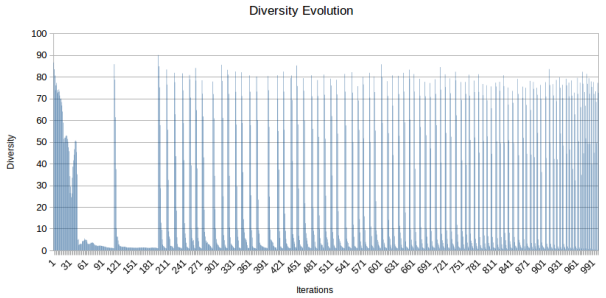


Figura 2. Evolución de la desviación estándar de la población con recombinación DOR.

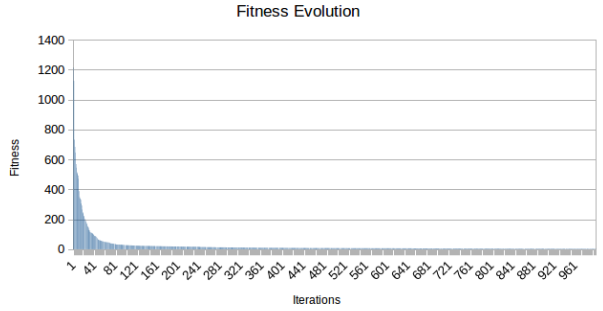


Figura 3. Evolución del fitness del algoritmo con recombinación aleatoria.

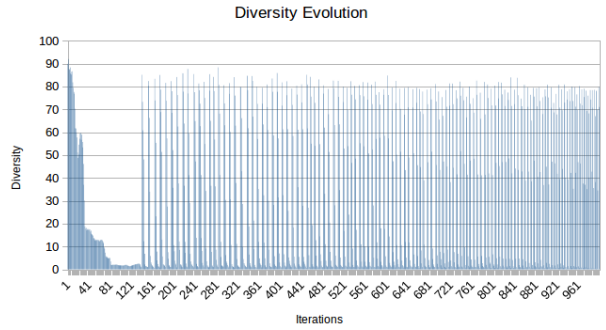


Figura 4. Evolución de la desviación estándar de la población con recombinación aleatoria.

Límites: $r_1, r_2, r_3, r_4, r_{cx}, r_{cy}, x_0, y_0 \in [0, 60]$
$\theta_0, \theta_2^1, \theta_2^2, \theta_2^3, \theta_2^4, \theta_2^5, \theta_2^6 \in [0, 2\pi]$
Parámetros: $popsize = 100$, $MaxIte = 1000$, $\epsilon_r = 10$, $\epsilon_\theta = 0.1$

5.2 Mejoras

Se hicieron dos modificaciones principales al algoritmo originalmente implementado:

1) Se implementó un esquema adaptativo en la búsqueda local, el cual consiste en iniciar con un valor máximo de ϵ y conforme aumenta el numero de iteraciones, este valor se va decrementando, la idea es lograr una reducción del radio de búsqueda para incrementar la intensificación en la ultimas evaluaciones del algoritmo.
2) Se combinaron dos esquemas de recombinación para cada mitad del numero total de iteraciones, durante la primera mitad se sigue un esquema aleatorio, esto es, se elige aleatoriamente una recombinación de todas las posibles; este esquema se incluyó con el propósito de mantener la diversidad durante las primeras etapas del algoritmo. Después de transcurrida la primera mitad de las iteraciones del algoritmo, se cambia este operador a un esquema dinástico, tal como se definió anteriormente.

Las figuras 1-4 muestran el comportamiento del algoritmo considerando una recombinación DOR y una recombinación mixta, en la parte de la evolución del fitness, se muestra como al utilizar recombinación aleatoria al inicio, retrasa la necesidad del primer reinicio de la población, la cual toma lugar cuando la desviación estándar de la población cae por debajo de 1.

La Tabla 2 muestra los resultados mejorados después de implementar estas modificaciones. Puesto que para el caso de prueba implementado, el artículo (referencia [1]) en el que nos estamos basando solamente proporciona el valor mínimo que se obtuvo (0.0267), sólo podemos concluir que nuestra implmentación fue capaz de producir un mejor resultado mínimo que el algoritmo propuesto en [1]. A continuación se muestra también una estadística que se obtuvo durante un total de 282 ejecuciones del programa:

Results	
Samples	282
Mean	6.50940473
Median	2.735167
Min	0.0078
Max	58.293134
StdDev	8.535354325

Estadísticas de las ejecuciones realizadas

6 Conclusiones

Primero, se observó durante la práctica que no era posible utilizar un esquema demasiado enfocado en diversidad, puesto que se deseaba llegar al mínimo en un limite de 1000 iteraciones, por tanto, convenía contar con un esquema más *greedy* que nos permitiera alcanzar un mínimo en el menor

tiempo posible.

También se pudo percibir durante la implmentación la diferencia que radica en aplicar una metaheurística a un problema de optimización continua, sobretodo aplicado a un problema donde se presentan muchas restricciones, y se observo que resulta ser una buena estrategia implementar una búsqueda local con alcance variable, ya que nos permite irnos enfocando en la intensificación de la solución a medida que el algoritmo avanza hacia un mínimo.

7 Compilación-Ejecución

Los programas implementados incluyen un makefile para compilarse, que soporta los comandos *make*, *make run* y *make clean*, así como un script de ejecución en el cluster.

7.1 Programa de Cálculos

El programa de cálculos recibe como parámetros el tamaño N de la población, el numero máximo de iteraciones $MaxIte$, el archivo de entrada, que debe contener el numero de puntos de precision que se van a optimizar y las coordenadas (x,y) de esos puntos, y por último, el archivo de salida donde se va a guardar el resultado.

```
almeida@almeida-X501A1: ~/Mecanismo/Calculos
almeida@almeida-X501A1:~$ cd /home/almeida/Mecanismo/Calculos
almeida@almeida-X501A1:~/Mecanismo/Calculos$ make
g++              -std=c++11 -fopenmp -c  src/function.cpp -o obj/function.o
g++              -std=c++11 -fopenmp -c  src/main.cpp -o obj/main.o
g++              -std=c++11 -fopenmp -c  src/memo.cpp -o obj/memo.o
g++              -std=c++11      -o bin/ejecutable obj/function.o obj/main.o obj/memo.o -lgomp
almeida@almeida-X501A1:~/Mecanismo/Calculos$ make run args="100 1000 input.dat output.dat"
```

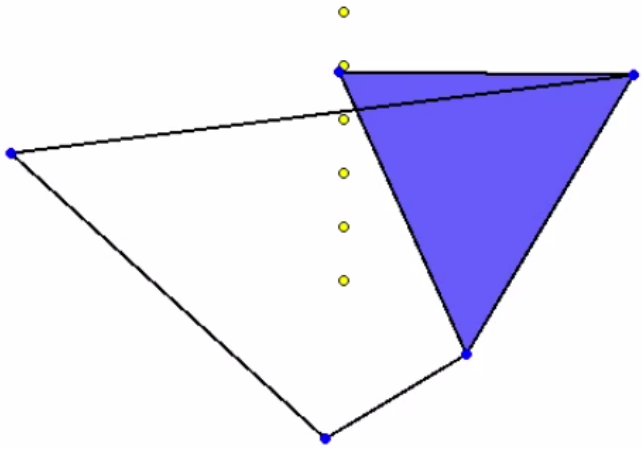
Compilación y ejecución del programa de cálculos.

7.2 Programa de Simulación

El programa de simulación lee el archivo de entrada y el archivo de salida generado por el programa de cálculos para correr la simulación del mecanismo (Se requiere WxWidgets). Recibe como parámetros el directorio de estos archivos (Se incluyen visualizaciones de algunos resultados obtenidos).

```
almeida@almeida-X501A1: ~/Mecanismo
almeida@almeida-X501A1:~$ cd /home/almeida/Mecanismo
almeida@almeida-X501A1:~/Mecanismo$ make
g++ -Wno-unused-result `wx-config --cflags`  -O2 -c main.cpp
g++ -Wno-unused-result `wx-config --cflags`  -O2 -c shell.cpp
g++ -Wno-unused-result `wx-config --cflags`  -O2 -c function.cpp
g++ -O2 -lm `wx-config --libs`  -lstdc++ -lfftw3 -lfftw3f  main.o shell.o function.o -o cshell -lm
`wx-config --libs`  -lstdc++ -lfftw3 -lfftw3f
almeida@almeida-X501A1:~/Mecanismo$ make run args="Calculos/input.dat Calculos/output.dat"
./cshell Calculos/input.dat Calculos/output.dat
almeida@almeida-X501A1:~/Mecanismo$
```

Compilación y ejecución del simulador.



Simulación del Mecanismo.

8 Referencias

1 *J.A. Cabrera & A. Simon, M. Prado* (2002). **Optimal synthesis of mechanisms with genetic algorithms.** Journal of Mechanism and Machine Theory 37 (2002) 1165-1177.

2 *El-Ghazali Talbi* (2009). **Metaheuristics: form Design to Implementation.** John Wiley & Sons, Inc.

3 *Pablo Moscato & Carlos Cotta* (2003). **A Gentle Introduction to Memetic Algorithms.** Handbook of Metaheuristics, Kluwer Academic Publishers, pp. 105-144

4 *Kumar Chellapilla* (1998). **Combining Mutation Operators in Evolutionary Programming.** IEEE Transactions on Evolutionary Computation, Vol. 2, No. 3.

5 *Ferrante Neri & Carlos Cotta* (2012). **Memetic algorithms and memetic computing optimization: A literature review.** Swarm and Evolutionary Computation 2, 1–14.