
Grado en Ingeniería Informática



METAHEURÍSTICA

METAHEURÍSTICAS BASADAS EN POBLACIONES

(Algoritmo Genético Generacional)

Alumno 1: Sergio Perea de la Casa.

DNI: 77433569-K.

Correo: spc00033@red.ujaen.es

Alumno 2: Jaime Rubio López.

DNI: 77646295-C

Correo: jrl00039@red.ujaen.es

Profesor: Ángel Miguel García Vico.

Grupo de prácticas: Grupo 4. **Identificación equipo:** 11.

Horario de prácticas: Viernes (12:30 - 14:30).

Descripción del problema.	2
Consideraciones.	3
Parámetros de configuración.	4
Explicación de las clases de objetos.	5
Clase Algoritmo Genético Generacional	5
Torneo binario.	6
Cruces (Método general)	6
Cruce en 2 puntos.	7
ALGORITMO	7
REPARACIÓN	7
Cruce MPX.	8
ALGORITMO	8
REPARACIÓN	8
Mutación.	9
Clase Poblacion	10
Clase Individuo	10
Análisis de resultados.	11
Parámetros para la obtención de los siguientes resultados	11
Desviación típica del cruce en 2 puntos.	11
Cruce 2 puntos. Élite 2.	11
Cruce 2 puntos. Élite 3.	12
Desviación típica del cruce en MPX.	12
Observación del cruce MPX con Élite 2:	12
Observación del cruce MPX con Élite 3:	13
Élite 2. Operadores de cruce en 2 puntos vs MPX.	13
Observación de resultados	14
Élite 3. Operadores de cruce en 2 puntos vs MPX.	14
Observación de resultados	15
Finalistas. Operador de cruce MPX con élite 2 vs élite 3.	16
Observación de resultados	16
Conclusiones.	17
Conclusión del ganador.	18
Conclusión respecto al análisis de la práctica anterior.	19
Conclusión final	20

Enlace para consultar los **archivos logs** de las ejecuciones de los algoritmos sobre los correspondientes archivos.

<https://drive.google.com/drive/folders/1BXwA5GFIUJ1AymMbcQoefJYBSYSoXazI?usp=sharing>

Descripción del problema.

El problema que debemos afrontar es el de la Máxima Diversidad; es decir, comúnmente expresado como MDP. Este problema NP-Completo mantiene un coste computacional elevado por lo que debemos de solucionar dicho coste mediante algoritmos que optimicen la eficiencia computacional y el coste final obtenido.

Para entender mejor el problema planteado hay que saber que dado un **conjunto inicial N** compuesto de **n** elementos, se selecciona un **subconjunto M** de **m** elementos con el objetivo de maximizar la diversidad entre los elementos del conjunto M. Es trivial que ningún elemento del conjunto N debe ser repetido y que por lo tanto su subconjunto M tampoco tendrá elementos repetidos. Cada fichero para plantear el problema nos proporciona la *n* y *m* correspondiente a los tamaños de su conjunto N y su subconjunto solución M. Además, nos proporciona una matriz regular de distancias $D=(d_{ij})$.

Las restricciones del problema según la estructura tomada para su resolución son las siguientes:

1. El orden de los elementos no importa.
2. En el subconjunto solución M debe de haber siempre m elementos.
3. No debe de haber elementos repetidos (Controlado mediante HashSet).

Por lo tanto, debemos de solucionar el problema MDP de forma que su función objetivo sea el siguiente:

$$\text{Max } C(S) = \sum_{i=1}^{n-1} \sum_{j=i+1}^n (d_{ij} \times e_i \times e_j)$$

$$\text{s.a. } \sum_{j=i+1}^n e_i = m \quad e_i = \{0, 1\} \quad 1 \leq i \leq n.$$

Consideraciones.

Para la resolución del problema hemos tenido en cuenta las siguientes consideraciones:

➤ **Algoritmo Genético Generacional.**

Es el núcleo de la práctica. Indica el algoritmo en sí a realizarse, dependiendo de la configuración llevada a cabo en el config.txt. En dicho algoritmo tenemos encapsulado cada uno de los pasos que debe de aparecer en el algoritmo genético propuesto.

➤ **Clase Población.**

Los objetos que se creen a partir de dicho tipo de clase tendrán asociado un contenedor de individuos los cuales son ya inicializados con sus cromosomas generados aleatoriamente.

➤ **Clase Individuo.**

Dicha clase contiene un contenedor llamado cromosoma que nos indica una solución factible al objetivo de dicha práctica y, por lo tanto, tendrá en la clase el coste fitness asociado a dicho cromosoma.

➤ **Otras consideraciones.**

La representación de la solución y la función objetivo son idénticas a las usadas en la práctica anterior. Por lo que no tendremos en cuenta en el informe una explicación detallada de ello.

Por otro lado, la clase Configurator y ArchivoDatos son los encargados de recopilar la información del problema para su posible funcionamiento.

También tener en cuenta que la ejecución del algoritmo usando diferentes operadores de cruce y diferente número de elitismo será manualmente indicado por el usuario desde el config.txt. El programa está ejecutado de forma que se realicen con programación multihilo todas las semillas por archivo correspondientes a la configuración que se desee por el usuario desde config.txt.

Parámetros de configuración.

En el archivo de configuración “*config.txt*” están los diferentes parámetros que se usarán en los diferentes algoritmos. Estos serán rellenados de dicho archivo con los siguientes atributos estáticos a usar:

- **ArrayList<String> *archivos***. Contenedor de diferentes archivos a explotar.
- **ArrayList<Long> *semillas***. Contenedor de las semillas que se usarán para las diferentes ejecuciones de un mismo algoritmo en un mismo archivo.
En nuestra configuración usaremos {77433569, 74335697, 43356977, 33569774, 35697743}.
- **Integer *MAX_ITERACIONES***. Número máximo de evaluaciones que se hará en los algoritmos; es decir, por cada generación, en la población, tenemos un número NUM_INDIVIDUOS a evaluar y por tanto se añadirá tantas veces hasta que sea menor a dicho MAX_ITERACIONES.
- **Integer *NUM_INDIVIDUOS***. Número de individuos que contendrá la población.
- **Float *PROB_CRUCE***. Probabilidad de que 2 individuos de una población se crucen para generar 2 nuevos individuos hijos a partir de dicho cruce. Por defecto tenemos puesto 0.70.
- **Float *PROB_GEN_MUTE***. Probabilidad de que un gen de un individuo mute. Por defecto tenemos puesto 0.05.
- **Float *PORCENTAJE_MPX***. Porcentaje asociado a la cantidad de genes, porcentualmente hablando, respecto a la totalidad de genes de un individuo que se van a coger para proceder a cruzarlos con todos los genes de otro individuo. El motivo de haber hecho este porcentaje como un parámetro estático es que si lo configuras de cierta forma dará unos resultados mejores a otros. Por lo tanto, en nuestro caso, hemos comprobado que la mejor solución obtenida es con el porcentaje al máximo (0.80).
- **Integer *NUM_ELITE_INDIVIDUOS***. Número de k individuos que no serán reemplazados en la antigua población por la población que ha sufrido el proceso evolutivo en dicha generación. Por defecto será $k = 2$ ó 3 .
- **String *TIPO_CRUCE***. Tipo de cruce que se realiza. 2 opciones a elegir: “2puntos” y “MPX”.

Explicación de las clases de objetos.

Clase Algoritmo Genético Generacional

Como se ha explicado anteriormente, se ha creado una única clase desde donde se va a realizar el análisis del algoritmo a desarrollar con sus diferentes configuraciones de cruce y élite.

Los atributos de dicha clase son los siguientes :

- **config**. Objeto que controla los parámetros modificables del config.txt.
- **archivo**. Objeto que mantiene la estructura de los datos del fichero correspondiente (Tamaño de la matriz, Matriz de datos, etc.).
- **random**. Random usado en diferentes ocasiones, ya sea en la propia clase o pasado como parámetro en la creación de algún objeto.
- **MEJOR_INDIVIDUO**. Conjunto de elementos que representan la mejor solución encontrada.
- **MEJOR_FITNESS**. Coste fitness que proporciona el individuo MEJOR_INDIVIDUO.
- **NUM_ELEMENTOS**. Corresponde a n; es decir, el tamaño de elementos que deben poder ser partícipes de la solución o no.
- **NUM_CANDIDATOS**. Corresponde a m; es decir, el tamaño de la solución.
- **poblacion**. Población actual del problema.
- **evaluaciones**. Controla las evaluaciones que se han ido llevando a cabo.

La estructura general del algoritmo es la siguiente:

```

1  #Algoritmo_Genetico_Generacional
2
3  Variables:
4  -Poblacion poblacion --> Población actual del problema.
5  -Poblacion nuevaPoblacion --> Servirá para seleccionar de la población padre para generar la nueva población
6  -Poblacion poblacionCruzada --> Población después de hacer el cruce
7  -Poblacion poblacionMutada --> Población después de hacer la mutación
8  -int evaluaciones --> Controla las evaluaciones que se han ido llevando a cabo
9  -int NUM_ELITE --> numero de elite de los individuos
10
11
12
13
14  ejecucionAGG(){
15      evaluacion( poblacion );
16      MIENTRAS( evaluaciones < Max_Iteraciones ){
17          nuevaPoblacion <- torneoBinario();
18          poblacionCruzada <- cruce( nuevaPoblacion );
19          poblacionMutada <- mutacion( poblacionCruzada );
20          evaluacion( poblacionCruzada );
21          poblacion <- reemplazo( NUM_ELITE, poblacionMutada );
22      }
23  }
```

Los métodos que influyen en el anterior esquema y que creemos que son importantes de mostrar con pseudocódigo son los siguientes:

Torneo binario.

```
#Torneo_Binario

Variables:
-Poblacion p --> Servirá para seleccionar de la población padre para generar la nueva población
-int i --> Primer individuo elegido al azar de la población //Entre 0 y NUM_INDIVIDUOS
-int pos --> Segundo individuo elegido al azar de la población, distinto del primero //Entre 0 y NUM_INDIVIDUOS

torneoBinario(){
    Poblacion p;
    MIENTRAS( p<-size() < NUM_INDIVIDUOS ){

        asignarAleatorios( i, pos );

        SI( poblacion.Individuo(i).getFitness() >= poblacion.Individuo(pos).getFitness() ){
            p.vPoblacion.add( poblacion.vPoblacion.get(i) );
        }SINO{
            p.vPoblacion.add( poblacion.vPoblacion.get(POS) );
        }
    }
    DEVOLVER p;
}
```

Cruces (Método general)

```
#CRUCE
Poblacion cruce(Poblacion origen){
    inicializarVariables(Poblacion destinatario,ArrayList<Individuo> cruzados);
    POR CADA 2 INDIVIDUOS en i{
        SI (ALEATORIO_CRUZAR == TRUE) {
            SI (CRUCE == "2puntos") {
                cruzados = cruce2puntos(origen.INDIVIDUO[i], origen.INDIVIDUO[i+1]);
            } else {
                cruzados = cruceMPX(origen.INDIVIDUO[i], origen.INDIVIDUO[i+1]);
            }
            destinatario <- add(cruzados[Hijo1],cruzados[Hijo2]);
        } else {
            destinatario <- origen.INDIVIDUO[i];
            destinatario <- origen.INDIVIDUO[i+1];
        }
    }
    }HASTA i == TAM_POBLACION
    DEVOLVER destinatario;
}
```

Cruce en 2 puntos.

ALGORITMO

```
#Cruce_2_Puntos
Variables:
-int punto1 --> Primer punto generado para cruzar //Entre 1 y NUM_CANDIDATOS
-int punto2 --> Segundo punto generado para cruzar //Entre 1 y NUM_CANDIDATOS
-Individuo padre1 --> Primer individuo que intervendrá en el cruce con el segundo individuo
-Individuo padre2 --> Segundo individuo que intervendrá en el cruce con el primer individuo
-int aux --> Nos servirá para hacer el intercambio
-Individuo hijo1, hijo2 --> Hijos generados con el cruce
-ArrayList<Individuo> cruzados --> ArrayList donde devolveremos los dos hijos cruzados

cruce2puntos( _padre1, _padre2 ){
    asignarAleatorios( punto1, punto2 );

    j <- 0;
    REPETIR{
        hijo1 <- add(padre1.genes,padre2.genes,j,punto1,punto2); //SEGÚN j (punto1,punto2) coge de padre1 o padre2
        hijo2 <- add(padre2.genes,padre1.genes,j,punto1,punto2); //SEGÚN j (punto1,punto2) coge de padre2 o padre1
    }MIENTRAS( j < padre1.size() ); //AMBOS PADRES DEBEN TENER MISMO TAMAÑO.

    SI (comprobarSiRepararHijos(hijo1,hijo2)){
        RepararHijos( hijo1, hijo2);
    }
    cruzados<-add( hijo1 );
    cruzados<-add( hijo2 );
    DEVOLVER cruzados;
}
```

REPARACIÓN

```
repara2puntos(Individuo ind){
    REPETIR{
        j = 0;
        REPETIR{
            punto <- buscarMasAporta_Individuo(N,ind.cromosoma);
        }MIENTRAS ( j < N.size());
        ind.add(punto);
    }MIENTRAS (ind.cromosoma.size() < NUM_CANDIDATOS)
}
```


Cruce MPX.

ALGORITMO

```
#Cruce_MPX
Variables:
-HashSet<int> hijo1, hijo2 --> Hijos generados con la mutacion
-Individuo padre1 --> Primer individuo que intervendrá en el cruce con el segundo individuo
-Individuo padre2 --> Segundo individuo que intervendrá en el cruce con el primer individuo
-ArrayList<Individuo> cruzados --> ArrayList donde devolveremos los dos hijos cruzados
-Integer numAleatorios_padre1 --> Número de elementos a coger aleatoriamente del padre1 para cruzar.

cruceMPX( _padre1, _padre2 ){
    MPXobtenerHijo1( _padre1, _padre2, hijo1 );
    MPXobtenerHijo2( _padre2, _padre1, hijo2 );
    SI (comprobarSiRepararHijos( hijo1, hijo2 ))
        RepararHijos(hijo1,hijo2);

    cruzados<-add( hijo1 );
    cruzados<-add( hijo2 );
    DEVOLVER cruzados;
}

MPXobtenerHijo1(padre1, padre2, hijo){
    hijo <- padre2; //HECHO CON HASHSET, POR ELLO LO HACEMOS DE ESTA FORMA.
    j = 0;
    REPETIR{
        punto <- randomNOREPETIDO(0, padre1.size()-1);
        hijo <- add(padre1.get(punto));
    }MIENTRAS( j++ < numAleatorios_padre1);
}
```

REPARACIÓN

```
reparaMPX (Individuo ind){
    ordenarSegunAporte(ind.genesCromosoma);
    sobrantes <- (ind.cromosomas.size() - NUM_CANDIDATOS);
    ind.remove( 0, sobrantes );
}
```

Mutación.

```
#Mutacion
Variables:
-Poblacion origen --> Poblacion de origen con la que se realizara la mutacion
-Poblacion destino --> Nueva poblacion con los individuos mutados o no mutados
-int cambiar --> //Entre 0 y NUM_ELEMENTOS-1
-boolean mutado --> Identifica si el cromosoma ha sido mutado o no
-ArrayList<int> cromosoma_mutado --> Lista de los cromosomas que han sido mutados
-Individuo indiv --> Individuo que ha sido mutado

mutacion( origen ){
    i <- 0;
    REPETIR{
        inicializarVariables(cromosoma_mutado, indiv);
        j <- 0;
        REPETIR{
            SI( RANDOM(0,1) < PROBABILIDAD_MUTAR ){
                cromosoma_mutado.eliminar(j);
                HACER{
                    asignarAleatorios( cambiar );
                }MIENTRAS( cromosoma_mutado.contiene( cambiar ) );
                cromosoma_mutado <- add( j, cambiar );
            }
        }HASTA( j < cromosoma_mutado.size() );

        indiv.asignar(cromosoma_mutado);
        destino <- add(indiv);
    }HASTA( i < origen.size() );
    DEVOLVER destino;
}
```

Reemplazo.

```
#Reemplazo
Variables:
-int k_elitismo --> Para hacer reemplazo a partir de élite "k"
-Poblacion p --> Poblacion mutada
-Poblacion destino --> Poblacion ya reemplazada
-ArrayList<Pair<Individuo, Double>> antigua_ordenada --> Poblacion a ser reemplazada, ordenada segun fitness
-ArrayList<Pair<Individuo, Double>> nueva_ordenada --> Poblacion ya reemplazada, ordenada segun fitness
-int mayorAporte --> Posición del individuo de mayor aporte

reemplazo(){
    inicializarVariables( destino, antigua_ordenada <- poblacion.ordenarSegunFitness(), nueva_ordenada <- p.ordenarSegunFitness() );
    i <- 0;
    REPETIR{
        SI( i < k_elitismo ){
            destino <- add( i, antigua_ordenada<get(mayorAporte) );
            SI (comprobar_MEJORSOLUCION(MEJOR_INDIVIDUO,antigua_ordenada<get(mayorAporte) ))
                MEJOR_INDIVIDUO <- antigua_ordenada<get(mayorAporte);
            mayorAporte--;
        }SINO{
            destino.vPoblacion() <- add( i, nueva_ordenada.get(i) );
            SI (comprobar_MEJORSOLUCION(MEJOR_INDIVIDUO,nueva_ordenada.get(i) ))
                MEJOR_INDIVIDUO <- nueva_ordenada.get(i);
        }
    }HASTA( i < NUM_INDIVIDUOS );
    DEVOLVER destino;
}
```

Clase Poblacion

Y por último, en la **clase Poblacion** tenemos el método siguiente, a parte de los getters correspondientes, útil para gran parte de la resolución del algoritmo:

- ❑ **`ArrayList<Pair<Individuo,Double>> ordenacionSegunFitness()`**:

Ordena de menor a mayor la estructura que mantiene al par `<Individuo, costeFitness>` el cual hace referencia al coste fitness que lleva asociado cada uno de los individuos de la población.

- ❑ **`void mostrarPoblacion()`**:

Método que muestra por consola cada uno de los individuos que la componen de forma que muestra el cromosoma y su coste fitness asociado.

Clase Individuo

- ❑ **`double costeFitness(float[][] matrizDatos)`**:

Sumatoria del coste proporcionado por el subconjunto M a partir de una función de evaluación.

- ❑ **`double distanciasElemento(Integer elem, float[][] matrizDatos)`**:

Distancias de un elemento respecto a los demás de la solución M.

- ❑ **`void ordenacionMenorAporte (ArrayList<Pair<Integer, Double>> v_distancias, float[][] matrizDatos) :`**

Ordena de menor a mayor la estructura que mantiene al par `<Integer, Double>` el cual hace referencia al aporte de un alelo respecto a los demás del cromosoma.

Análisis de resultados.

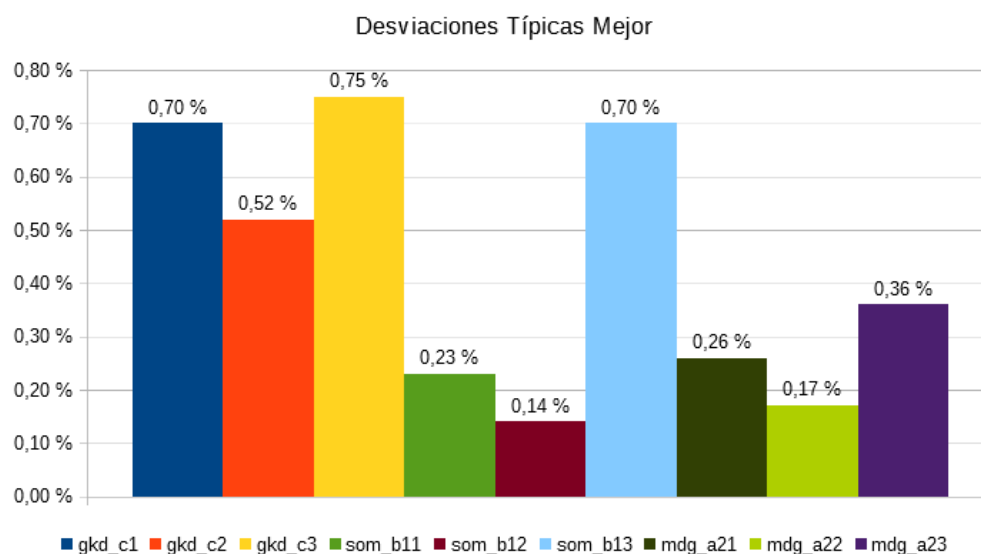
Parámetros para la obtención de los siguientes resultados

Los parámetros que se han llevado a cabo para la realización de la práctica han sido los siguientes:

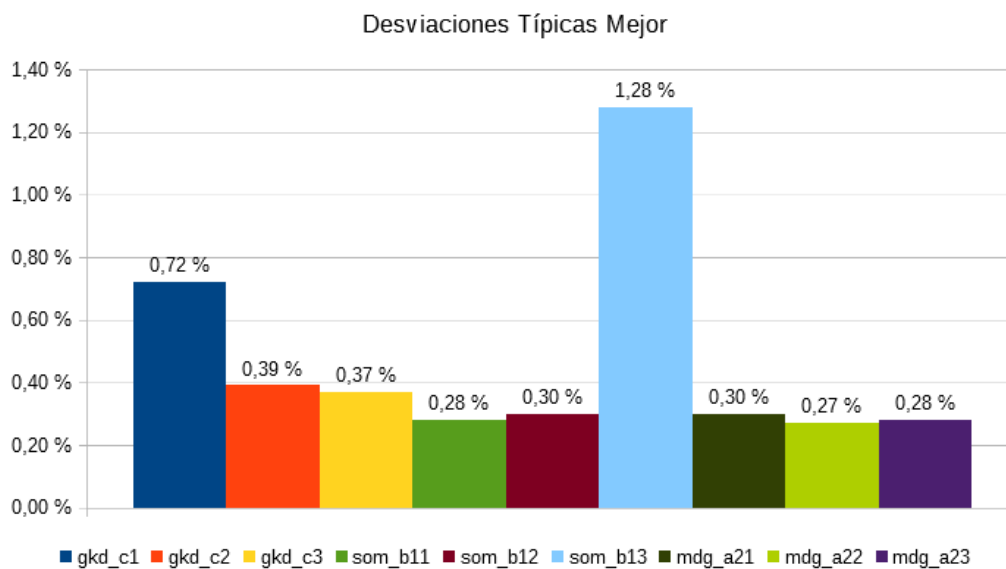
- 50 individuos para la población.
- 50.000 evaluaciones como condición de parada.
- Algoritmo de selección: torneo binario (torneo, $k = 2$).
- Probabilidad de cruce del 70% y probabilidad de mutación del 5%.
- Semillas aplicadas secuencialmente según ejecución: 77433569 con traslación de 1 número hacia derechas por ejecución.
- Si se ha aplicado el cruce MPX, entonces tiene un porcentaje del 80% para la cantidad aleatoria de elementos cogidos de uno de los padres para cruzar. Dato que hemos considerado estático debido a los buenos resultados que suponía mantenerlo de forma estática y en un porcentaje alto.

Desviación típica del cruce en 2 puntos.

Cruce 2 puntos. Élite 2.

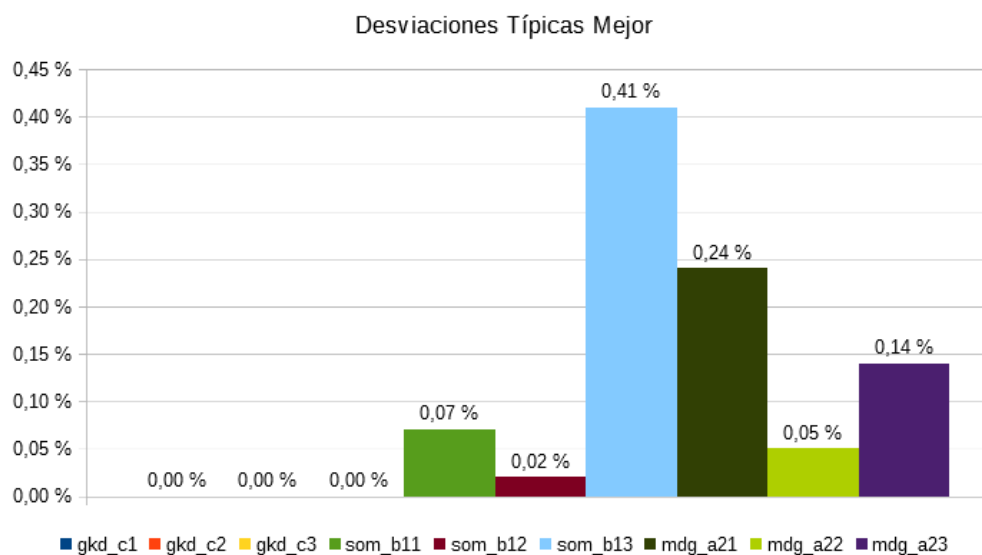


Cruce 2 puntos. Élite 3.

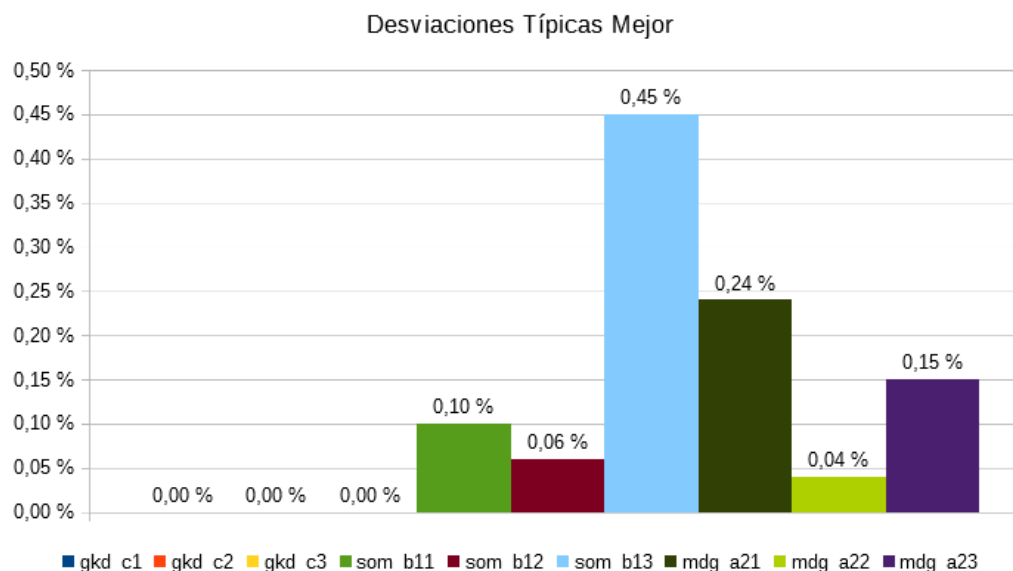


Desviación típica del cruce en MPX.

Observación del cruce MPX con Élite 2:



Observación del cruce MPX con Élite 3:

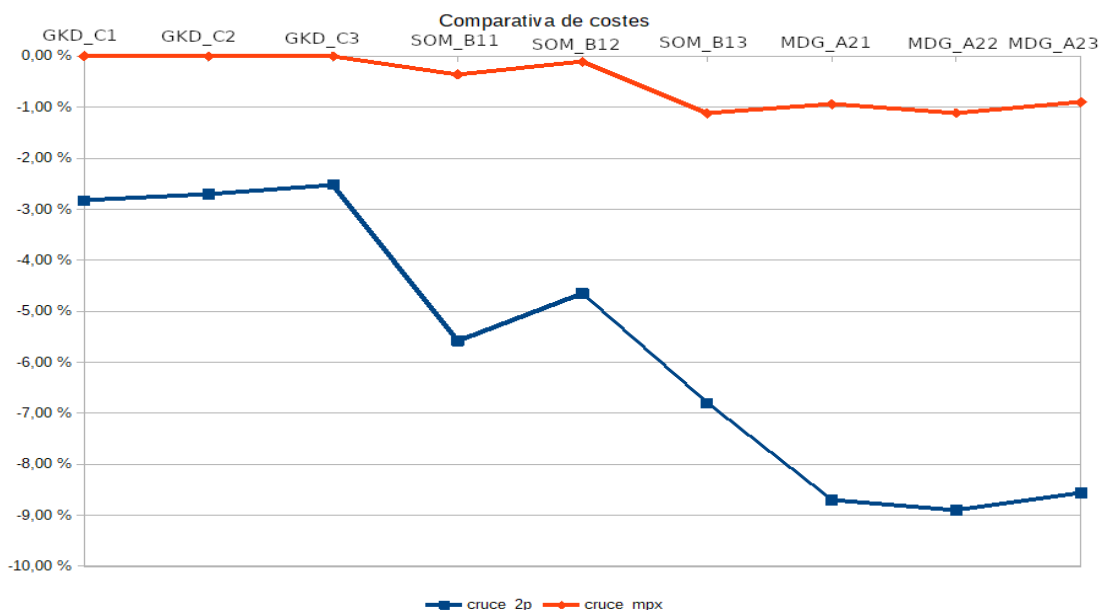


Élite 2. Operadores de cruce en 2 puntos vs MPX.

Cruce 2 puntos. Élite 2	Tamaño 20		Tamaño 20		Tamaño 30		Tamaño 30		Tamaño 20		Tamaño 21		Tamaño 22		Tamaño 25		Tamaño 25	
	Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste	
	GKD-c 1 n500 m5		GKD-c 2 n500 m50		GKD-c 3 n500 m5		SOM-b 11 n300 m		SOM-b 12 n300 m		SOM-b 13 n400 m		MDG-a 21 n2000		MDG-a 22 n2000		MDG-a 23 n2000	
	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time
Ejecución 1	18758.63	10177.00	19133.98	9000.00	18819.47	8878.00	19521.00	14678.00	34177.00	22902.00	4361.00	6135.00	104748.00	381883.00	104049.00	383480.00	103864.00	397554.00
Ejecución 2	18854.11	10163.00	19259.84	9103.00	19001.60	9134.00	19625.00	14642.00	34262.00	23239.00	4330.00	6066.00	103980.00	352551.00	104165.00	380123.00	104313.00	382845.00
Ejecución 3	19112.88	10231.00	19234.01	8942.00	19132.93	9235.00	19559.00	14696.00	34267.00	22918.00	4289.00	6029.00	104319.00	3633325.00	103908.00	373511.00	104931.00	392295.00
Ejecución 4	19002.07	10244.00	19211.87	9000.00	19157.23	9088.00	19584.00	14682.00	34193.00	23081.00	4358.00	6082.00	104107.00	365408.00	104262.00	375755.00	104095.00	393508.00
Ejecución 5	18936.43	10417.00	19004.88	8716.00	19163.76	8975.00	19640.00	14717.00	34164.00	23301.00	4368.00	6048.00	104437.00	3633945.00	104397.00	377186.00	104558.00	401118.00
Media	-2.83 %	10246.40	-2.70 %	8952.20	-2.52 %	9062.00	-5.58 %	14683.00	-4.65 %	23088.20	-6.80 %	6072.00	-8.70 %	365422.40	-8.90 %	378011.00	-8.56 %	393464.00
Desv. típica	0.70 %	101.39	0.52 %	144.23	0.75 %	138.94	0.23 %	27.53	0.14 %	181.46	0.70 %	40.40	0.26 %	10526.23	0.17 %	3885.57	0.36 %	6881.92

Cruce MPX. Élite 2	Tamaño 20		Tamaño 20		Tamaño 30		Tamaño 30		Tamaño 20		Tamaño 21		Tamaño 22		Tamaño 25		Tamaño 25	
	Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste	
	GKD-c 1 n500 m5		GKD-c 2 n500 m50		GKD-c 3 n500 m5		SOM-b 11 n300 m		SOM-b 12 n300 m		SOM-b 13 n400 m		MDG-a 21 n2000		MDG-a 22 n2000		MDG-a 23 n2000	
	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time
Ejecución 1	19485.19	4023.00	19701.53	4837.00	19547.21	5080.00	20652.00	10469.00	35835.00	17685.00	4625.00	4204.00	113285.00	85933.00	113129.00	82108.00	113178.00	83820.00
Ejecución 2	19485.19	3970.00	19701.53	4959.00	19547.21	5037.00	20690.00	10664.00	35842.00	17697.00	4575.00	4193.00	113538.00	85255.00	113019.00	81605.00	113015.00	83042.00
Ejecución 3	19484.75	4037.00	19701.53	4811.00	19547.21	5007.00	20672.00	10860.00	35854.00	17494.00	4618.00	4170.00	112826.00	87445.00	113123.00	82424.00	112917.00	84326.00
Ejecución 4	19485.19	3994.00	19701.53	4929.00	19547.21	5037.00	20669.00	10810.00	35834.00	17672.00	4602.00	4169.00	113038.00	86072.00	113032.00	81707.00	113327.00	80940.00
Ejecución 5	19485.19	4033.00	19701.53	4801.00	19547.21	5040.00	20660.00	10717.00	35836.00	17597.00	4608.00	4096.00	113265.00	86668.00	113012.00	81794.00	113062.00	84048.00
Media	0.00 %	4011.40	0.00 %	4867.40	0.00 %	5040.20	-0.36 %	10704.00	-0.11 %	17629.00	-1.12 %	4166.40	-0.94 %	86274.60	-1.11 %	81927.60	-0.90 %	83235.20
Desv. típica	0.00 %	28.61	0.00 %	71.94	0.00 %	26.01	0.07 %	152.12	0.02 %	84.94	0.41 %	42.12	0.24 %	824.91	0.05 %	335.24	0.14 %	1369.10

	GKD-c 1 n500 m5		GKD-c 2 n500 m50		GKD-c 3 n500 m5		SOM-b 11 n300 m		SOM-b 12 n300 m		SOM-b 13 n400 m		MDG-a 21 n2000		MDG-a 22 n2000		MDG-a 23 n2000		MEDIA	
	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time
Cruce 2	-2.83 %	10246.40	-2.70 %	8952.20	-2.52 %	9062.00	-5.58 %	14683.00	-4.65 %	23088.20	-6.80 %	6072.00	-8.70 %	365422.40	-8.90 %	378011.00	-8.56 %	393464.00	-5.69 %	134333.47
Cruce MPX	0.00 %	4011.40	0.00 %	4867.40	0.00 %	5040.20	-0.36 %	10704.00	-0.11 %	17629.00	-1.12 %	4166.40	-0.94 %	86274.60	-1.11 %	81927.60	-0.90 %	83235.20	-0.50 %	33095.09



Observación de resultados

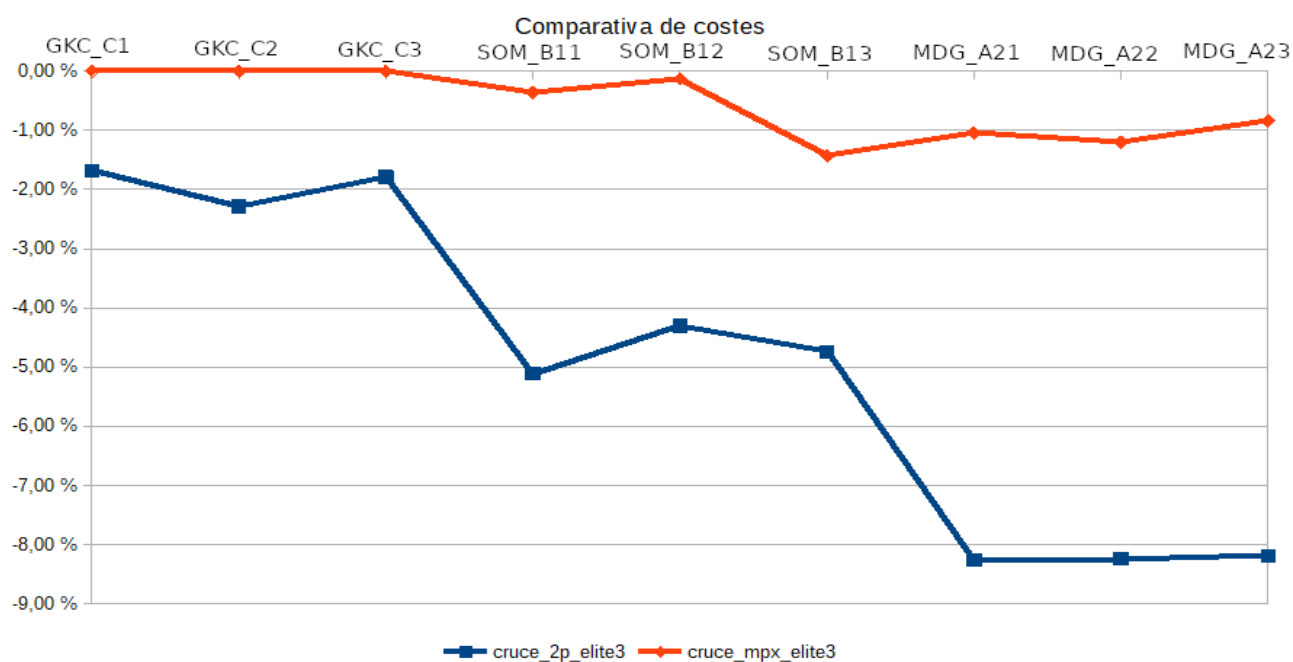
La primera comparación entre 2 algoritmos para la resolución del evolutivo nos da un claro ganador; es decir, el Algoritmo Genético Generacional con cruce MPX y élite 2. No hay punto de comparación entre lo obtenido tanto en tiempo como en optimización a la maximización del coste que se obtiene en el cruce MPX respecto al cruce en 2 puntos.

Élite 3. Operadores de cruce en 2 puntos vs MPX.

Cruce 2 puntos. Élite 3	Tamaño 20		Tamaño 20		Tamaño 30		Tamaño 30		Tamaño 20		Tamaño 21		Tamaño 22		Tamaño 25	
	Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste	
	19485,1875		19701,5377		19547,2070		20743		35881		4658		114259		114327	
	GKD-c 1	n500 m5	GKD-c 2	n500 m50	GKD-c 3	n500 m5	SOM-b 11	n300 m	SOM-b 12	n300 m	SOM-b 13	n400 m	MDG-a 21	n2000	MDG-a 22	n2000
	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time
Ejecución 1	19059.26	9899.00	19278.84	10196.00	19114.09	10337.00	19663.00	15340.00	34243.00	24011.00	4441.00	7027.00	104861.00	464242.00	105212.00	443646.00
Ejecución 2	18999.45	10038.00	19366.15	10201.00	19278.95	9874.00	19768.00	15429.00	34324.00	23785.00	4363.00	6741.00	104490.00	466464.00	104684.00	431800.00
Ejecución 3	19184.62	10121.00	19166.63	10246.00	19213.79	10047.00	19709.00	15345.00	34363.00	24397.00	4485.00	6786.00	104596.00	471938.00	104701.00	431665.00
Ejecución 4	19366.60	10264.00	19198.57	10150.00	19253.24	10337.00	19652.00	15024.00	34511.00	24299.00	4394.00	6880.00	104765.00	448177.00	105279.00	429051.00
Ejecución 5	19180.86	10240.00	19255.66	10346.00	19131.20	10176.00	19618.00	15224.00	34256.00	23558.00	4504.00	7062.00	105384.00	480157.00	104657.00	436777.00
Media	-1.68 %	10112.40	-2.28 %	10227.80	-1.79 %	10154.20	-5.11 %	15272.40	-4.30 %	24010.00	-4.74 %	6899.20	-8.26 %	466195.60	-8.24 %	434587.80
Desv. típica	0.72 %	150.34	0.39 %	74.30	0.37 %	198.31	0.28 %	156.85	0.30 %	349.36	1.28 %	142.34	0.30 %	11794.64	0.27 %	5783.91

Cruce MPX. Élite 3	Tamaño 20		Tamaño 20		Tamaño 30		Tamaño 30		Tamaño 20		Tamaño 21		Tamaño 22		Tamaño 25	
	Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste		Mejor coste	
	19485,1875		19701,5377		19547,2070		20743		35881		4658		114259		114327	
	GKD-c 1	n500 m5	GKD-c 2	n500 m50	GKD-c 3	n500 m5	SOM-b 11	n300 m	SOM-b 12	n300 m	SOM-b 13	n400 m	MDG-a 21	n2000	MDG-a 22	n2000
	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time
Ejecución 1	19485.19	3987.00	19701.53	3592.00	19547.21	4192.00	20634.00	10309.00	35796.00	15894.00	4557.00	3522.00	113031.00	79366.00	112955.00	92713.00
Ejecución 2	19485.19	3963.00	19701.53	3564.00	19547.21	4176.00	20678.00	10359.00	35826.00	16047.00	4612.00	3510.00	113199.00	78009.00	112989.00	91855.00
Ejecución 3	19485.19	4026.00	19701.53	3528.00	19547.21	4107.00	20687.00	10492.00	35849.00	15886.00	4591.00	3636.00	112790.00	79190.00	112891.00	93363.00
Ejecución 4	19485.19	4005.00	19701.53	3537.00	19547.21	4143.00	20664.00	10448.00	35847.00	15824.00	4602.00	3592.00	112845.00	78572.00	112947.00	92599.00
Ejecución 5	19485.19	3992.00	19701.53	3591.00	19547.21	4207.00	20675.00	10432.00	35850.00	16113.00	4594.00	3565.00	113474.00	77990.00	112992.00	91103.00
Media	0.00 %	3994.60	0.00 %	3562.40	0.00 %	4165.00	-0.36 %	10408.00	-0.13 %	15952.80	-1.43 %	3565.00	-1.04 %	78625.40	-1.20 %	92326.60
Desv. típica	0.00 %	23.22	0.00 %	29.69	0.00 %	40.19	0.10 %	73.20	0.06 %	121.50	0.45 %	51.58	0.24 %	643.00	0.04 %	868.51

	GKD-c 1 n500 m5		GKD-c 2 n500 m50		GKD-c 3 n500 m5		SOM-b 11 n300 m		SOM-b 12 n300 m		SOM-b 13 n400 m		MDG-a 21 n2000		MDG-a 22 n2000		MDG-a 23 n2000		MEDIA	
	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time
Cruce 2	-1,68 %	10112,40	-2,28 %	10227,80	-1,79 %	10154,20	-5,11 %	15272,40	-4,30 %	24010,00	-4,74 %	6899,20	-8,26 %	466195,60	-8,24 %	434587,80	-8,19 %	428398,80	-4,95 %	156206,47
Cruce MPX	0,00 %	3994,60	0,00 %	3562,40	0,00 %	4165,00	-0,36 %	10408,00	-0,13 %	15952,80	-1,43 %	3565,00	-1,04 %	78625,40	-1,20 %	92326,60	-0,84 %	89710,20	-0,56 %	33590,00

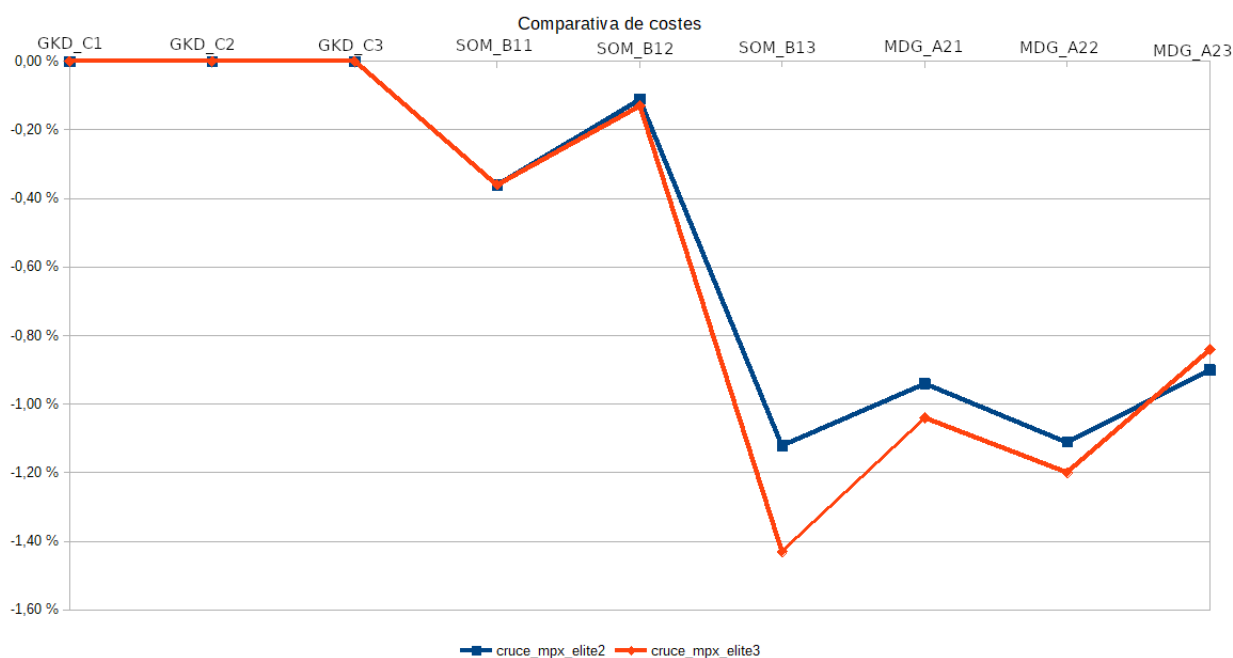


Observación de resultados

Una vez más, el enfrentamiento entre el cruce en 2 puntos y el cruce MPX provoca que haya mucha diferenciación entre la obtención de un buen resultado tanto en tiempos como en maximización del coste por archivo en el cruce MPX respecto al cruce 2 puntos.

Finalistas. Operador de cruce MPX con élite 2 vs élite 3.

	GKD-c 1 n500 m5		GKD-c 2 n500 m50		GKD-c 3 n500 m5		SOM-b 11 n300 m		SOM-b 12 n300 m		SOM-b 13 n400 m		MDG-a 21 n2000		MDG-a 22 n2000		MDG-a 23 n2000		MEDIA	
	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time	C	Time
MPX																				
ELITE 2	0.00 %	4011.40	0.00 %	4867.40	0.00 %	5040.20	-0.36 %	10704.00	-0.11 %	17629.00	-1.12 %	4166.40	-0.94 %	86274.60	-1.11 %	81927.60	-0.90 %	83235.20	-0.50 %	33095.09
ELITE 3	0.00 %	3994.60	0.00 %	3562.40	0.00 %	4165.00	-0.36 %	10408.00	-0.13 %	15952.80	-1.43 %	3565.00	-1.04 %	78625.40	-1.20 %	92326.60	-0.84 %	89710.20	-0.56 %	33590.00



Observación de resultados

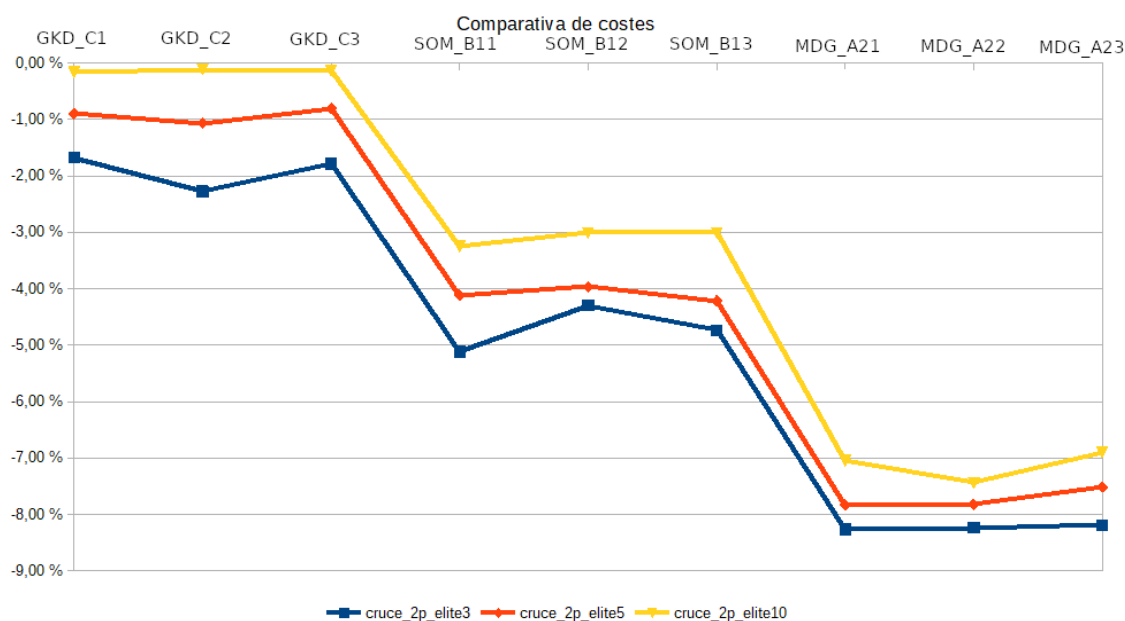
Como finalistas en el análisis de resultados, tenemos para el mismo cruce MPX el elitismo con $k = 2$ y $k = 3$. Si lo comparamos a nivel de tiempos, prácticamente son iguales por lo que su comparación en este aspecto es irrelevante para la conclusión. En términos de resultados de maximización de los costes por archivo podemos ver gráficamente que el cruce MPX con elitismo de $k = 2$ es ligeramente mejor en todos los archivos exceptuando el último (también es unos *ms* más rápido, pero sin llegar a destacar). Por lo tanto, a nivel de resultados tenemos un ganador: **Algoritmo Genético Generacional con cruce MPX y élite 2.**


Conclusiones.

Antes de llegar a conclusiones específicas hay que dar una explicación general de cómo se ha llevado a cabo algunos aspectos. Hemos hecho un algoritmo genético generacional el cual una vez se hace reemplazamiento debemos de quedarnos con todos los individuos nuevos exceptuando que los únicos sobrevivientes de la población anterior dependen del elitismo $k = 2$ o 3. Aún así, mantenemos este criterio sean mejor o no estos individuos antiguos por lo que es posible que en el algoritmo para cruce 2 puntos sea clave esta decisión (gráfica extra añadida sobre ello).

Como hemos visto en los resultados, **el cruce en 2 puntos ha supuesto una cantidad mayor en tiempos respecto al cruce MPX**. Esto es posible, sobretodo, debido a la reparación que supone el MPX respecto a la reparación de un cruce en 2 puntos. No es lo mismo una previa ordenación del aporte de cada gen respecto a los demás y eliminar los sobrantes (caso de la reparación MPX) a tener que buscar entre todos los no seleccionados en N cuál es el que más aporta respecto a los que ya están (caso de la reparación en 2 puntos) por cada elemento faltante hasta completar la solución.

Por otro lado, igualmente **el cruce MPX ha supuesto un acercamiento a los óptimos globales mayor que el cruce en 2 puntos**. Es cierto que, según la configuración que se le aplique al elitismo con un cruce en 2 puntos se puede obtener una mejora algo notable si se aumentaba el k del elitismo en el reemplazo. La siguiente gráfica demuestra este estudio que hemos realizado para comprobar si mejoraba o no modificando el parámetro del elitismo:





Como podemos observar, se mejoran los valores obtenidos en todos los archivos cada vez que se aumenta el elitismo. Esto nos ha llevado a pensar que el cruce en 2 puntos lleva a una exploración demasiado brusca de forma que no se llegue a centrar algo más de lo necesario en la intensificación de sus mejores individuos. Quizás al aumentar el k elitista estamos también afectando a que la mutación no provoque un alto salto de exploración en los cromosomas y por tanto en las generaciones de la población.

Una posible alternativa a ello sería también partir de un elitismo neutral ($k = 5$, quizás) y añadir la funcionalidad siguiente: Tener en cuenta si los individuos antiguos que van a mantenerse en la población son mejores en fitness a los que van a ser eliminados de la nueva población. Con esto estaríamos intentando aproximar el algoritmo a una intensificación mejor de los buenos resultados.

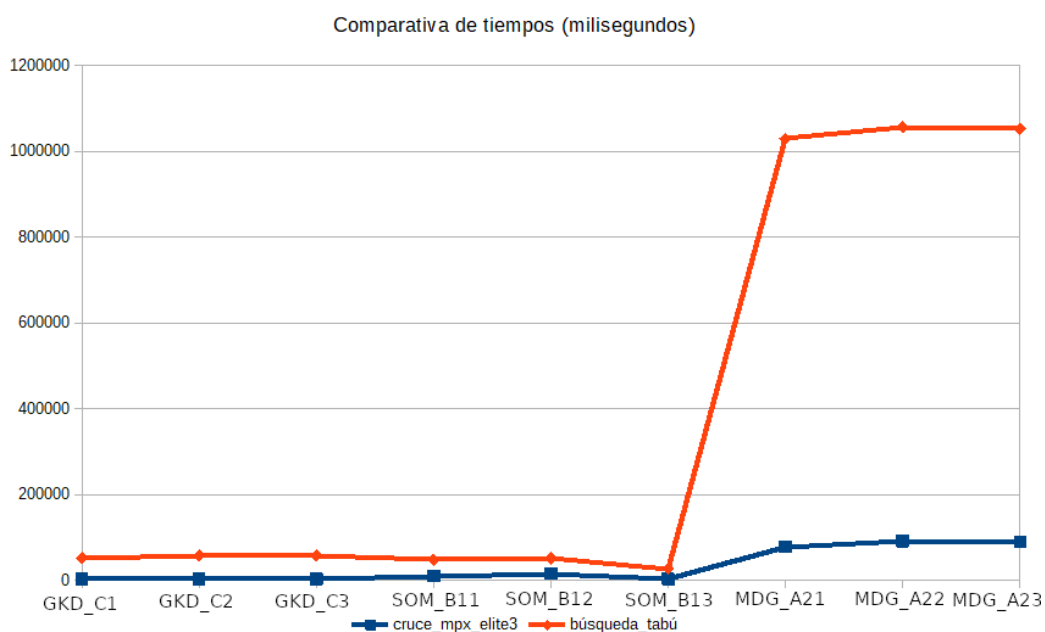
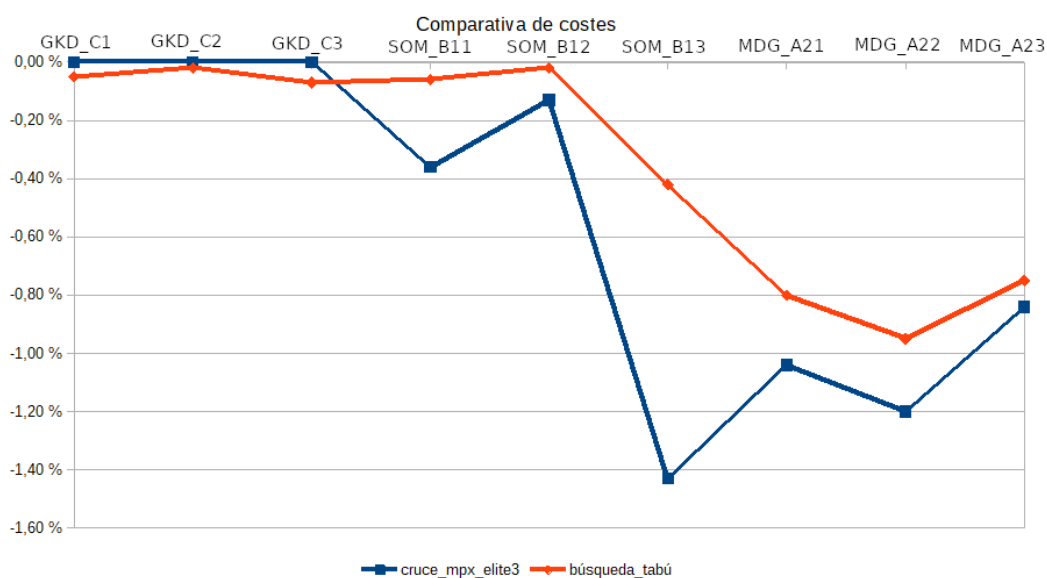
Conclusión del ganador.

El algoritmo genético generacional más prometedor para estos archivos ha sido con cruce MPX y un elitismo de $k = 2$. El tiempo de ejecución global de los archivos en comparación con los demás ha sido mejor a ellos, de tal forma que contra el cruce en 2 puntos se ha obtenido hasta 3 veces menor tiempo. Por otro lado, el cruce MPX supone una alta convergencia prematura de que todos los individuos mantengan unos cromosomas iguales o parecidos a los demás. Esto lleva a el problema de la “pérdida de evaluaciones” en los casos en los que a partir de cierto número de evaluaciones su mejor coste y solución encontrada varía insignificadamente. Es cierto que, dentro de lo que cabe y viendo los resultados obtenidos, a nivel general en todos los archivos no supone tanto problema como podría llegar a serlo en otro tipo de problema a optimizar.

El resultado general obtenido por el ganador nos ha llevado a plantear de nuevo cuál sería el algoritmo que se debería de escoger si se quiere una optimización maximizada de los costes por archivo. Si retomamos los resultados obtenidos en la práctica anterior, habíamos elegido como ganador el algoritmo de búsqueda tabú pero sacrificando el tiempo que ello suponía. En el siguiente apartado explicamos una comparación de resultados entre nuestro algoritmo genético generacional ganador y el ganador de nuestros algoritmos de búsquedas de trayectorias; es decir, sobre nuestra búsqueda tabú.

Conclusión respecto al análisis de la práctica anterior.

Las siguientes gráficas muestran la comparativa tanto en acercamiento a óptimos globales de costes obtenidos por archivo y los tiempos que ha supuesto los dos mejores algoritmos obtenidos en la práctica anterior y en esta:



Conclusión final

Tras ver los resultados gráficamente obtenidos en el apartado anterior, llegamos a la conclusión de que hemos conseguido en esta práctica obtener un algoritmo que solventa el problema que nos suponía el algoritmo ganador de la práctica anterior; es decir, obtener unos buenos resultados de acercamiento a los óptimos globales (en la mayoría de archivos) con unos tiempos de ejecución realmente buenos.

Por lo tanto, aunque se obtengan insignificantes resultados peores en algunos archivos, elegiremos como **algoritmo que mejor obtención de resultados nos da, en relación coste/tiempo para ambas prácticas, es el algoritmo genético generacional con cruce MPX y élite 2.**

Quizás mantener el equilibrio que supone los buenos resultados de la búsqueda tabú y los acercamientos a esos resultados de un algoritmo genético generacional en poco tiempo nos lleva a pensar que **la hibridación de ellos puede ser también un aspecto bueno a considerar.**