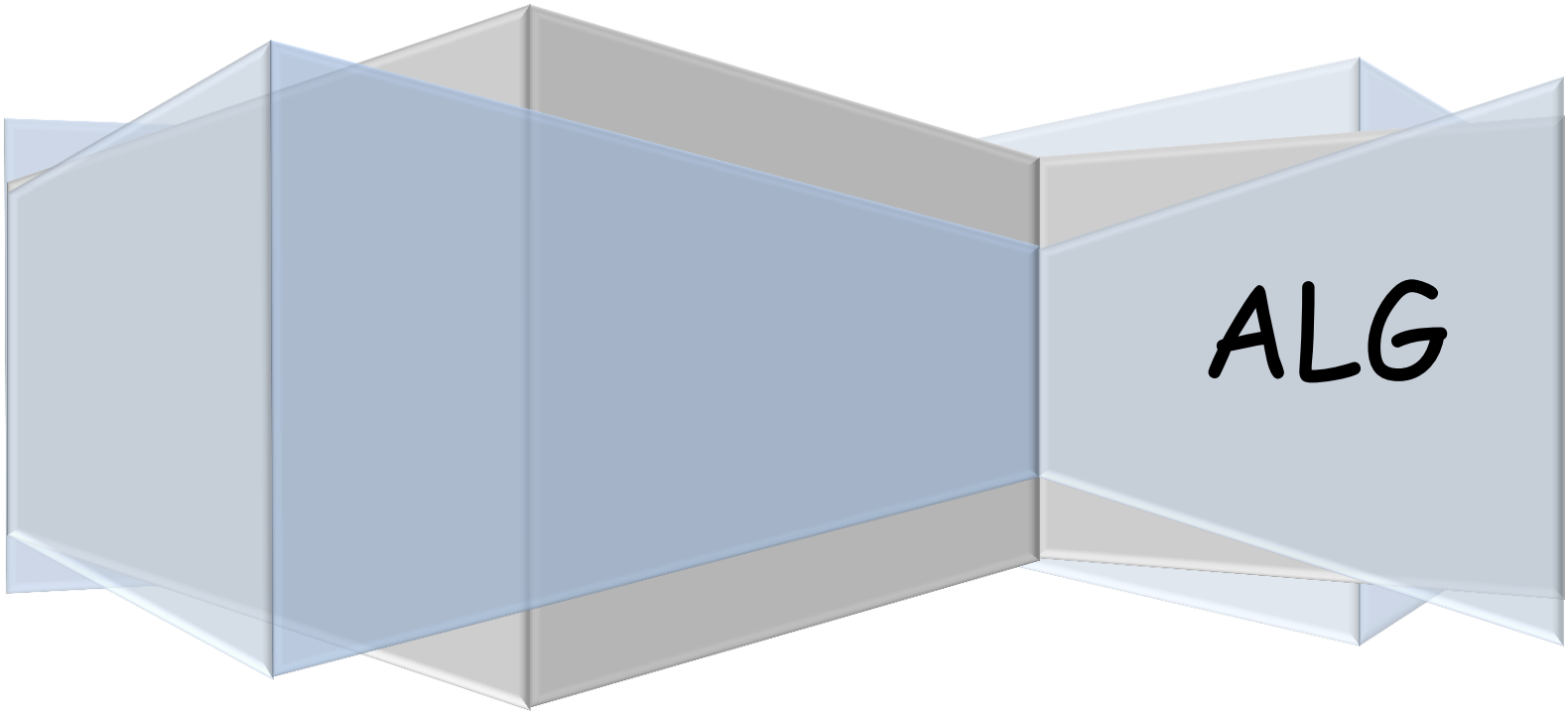
Práctica 3

Algoritmos Genéticos e Hibridaciones

Carlos Jesús Fernández Basso 75927137-C

Email: karloos@correo.ugr.es
Grupo 3 (miércoles de 10:00 a 12:00)
Profesor: Rafael Alcalá Fernández



ÍNDICE

1. Descripción y formulación del problema

1. Descripción del problema

   Tenemos un conjunto de datos, el tamaño de un subconjunto de estos y la diversidad que existe entre cada par de elementos del mismo. Debemos seleccionar el subconjunto que tenga la máxima diversidad entre todos sus elementos.

   La diversidad empleada suele ser la distancia entre dos puntos, empleándose generalmente la distancia Euclídea.

2. Formulación del problema

   Diversidad(X) -> esta función nos mara la Diversidad del conjunto X

   cardinal(ConjuntoDatos)=N
   cardinal (Subconjunto)=M } Donde M<N
   MatrizDiversidad[][]
   Obtener MAXIMO(Diversidad(Subcojunto))

3. Aplicaciones
   El problema de la diversidad máxima tiene por ejemplo aplicaciones en:

   a. problemas sociales
   b. preservación ecológica
   c. control de la contaminación
   d. diseño de productos
   e. inversión de capital
   f. gestión de la mano de obra
   g. diseño de currículos
   h. ingeniería genética

2. Descripción de la aplicación al problema de los algoritmos empleados

a) Representación.

Represento la solución de dos maneras diferentes:

1. Una solución es una lista binaria si la posición es 1 el elemento esta seleccionado y si es 0 no lo esta.
2. Mediante dos listas una en la que se encuentran las posiciones que están escogidas para el subconjunto(Lista1) y otro donde están las que no(Lista0). Esta es la representación que utilizo para realizar los algoritmos.

b) Función objetivo

Lo que realiza esta función es la suma de la diversidad lo los elementos escogidos en el subconjunto, contando solo 1 vez la diversidad entre dos elementos.

c) Descripción en pseudocódigo de la función objetivo.

```
FunObjetivo (){
        Diversidad=0
        Desde i=0 hasta i=M{
                Desde j=i hasta i=M{
                        Diversidad=Matriz[Lista1[i],Lista1[j]]+Diversidad
                }
        }
}
```

d) Descripción del algoritmo de búsqueda local

```
BL(boolean Flag1, int semilla,cota):
        Si(Flag1==true){
                LimpiaSolucion()
                SolucionAleatoria()
                Random(semilla)
        }
        maxGlobal=FunObjetivo(Lista1,Lista0)
        Auxiliar1=Lista1.clone()
        Auxiliar0=Lista0.clone()
```

```
mientras(mejora==True OR x<cota){
        mejora=False
        barajar(self.Lista1)

        desde i hasta i=Lista1.size and mejora==False{
                desde i hasta i=Lista1.size and mejora==False{
                        Lista1.insertar(Lista0[j])
                        Lista0.borrar(Lista0[j])
                        Lista0.insertar(Lista1[i])
                        Lista1.borrar(Lista1[i])
                        max=RecalculaDiversidad()
                        x++;
                        if(maxGlobal<max):
                                mejora=True
                                Auxiliar1=Lista1.clone()
                                Auxiliar0=Lista0.clone()
                                maxGlobal=max
                        else:
                                Lista1=Auxiliar1.clone()
                                Lista0=Auxiliar0.clone()
                }
        }
}
```

3. Pseudocódigo

A. *ALG Genético Generacional*

```
ALG_Genetico_Generacional( TamPoblacion, semilla,mutar, cruzar, Tammutar) {

        desde  0  hasta TamPoblacion{
                x=SolucionAleatoria();
                población.Añadir(x)
        }
        desde  desde  TamPoblacion  hasta 100000{
                Padres=Torneo(Poblacion)
                Desde i=0 hasta tamPoblacion/2{
                        Si(Aleatorio <0.7){
                                Hijos=Cruce(padre[1],padre[2])
                        }else{
                                Si(Aleatorio < 0.1){
                                        Padre[1].Mutar(0.1)
                                }
                                Si(Aleatorio < 0.1){
                                        Padre[1].Mutar(0.1)
                                }
                        }
                        Población.Borrar(padres)
                        Poblacion.Añadir(Hijos)
                        GuargarMejorSolucion(Poblacion)
                }

        }
        Devolver maximo
}
```

B. ALG Genético Estacionario

```
ALG_Genetico_Estacionario( TamPoblacion, semilla,mutar, cruzar, Tammutar) {

        desde  0  hasta TamPoblacion{
                x=SolucionAleatoria();
                población.Añadir(x)
        }
        desde  TamPoblacion  hasta 100000{
                desde 0 hasta 2{
                        Padres=Torneo(Poblacion)
                        Hijos=Cruce(padre[1],padre[2])
                        Si(Hijos mejores Peores(población)){
                                Población.BorrarPeores()
                                Poblacion.Añadir(Hijos,poblacion)
                        }
                }
        }
        Devolver MejorSolucion.maximo
}
```

Utilizo las siguientes funciones en los dos algoritmos Genéticos, en los memeticos, y en el búsqueda diversa(en este el torneo no)

Torneo

```
Cruce(poblacion  ) {
        Num1=Aleatorio (1,población.size())
        Num2=Aleatorio (1, población.size())
        Num3=Aleatorio (1, población.size())
        Num4=Aleatorio (1, población.size())

        Padres.Añadir(EscogerMejor(población[Num1],población[Num2]))
        Padres.Añadir(EscogerMejor(población[Num3],población[Num4]))

        Devolver Padres
```

}

Cruce

```
Cruce(Solucion S1,Solucion S2  ) {
        Punto1=Aleatorio (1,n)
        Punto2=Aleatorio(punto1,n)
        Solucion1
        Solucion2
        desde i=1 hasta n{
                if (i<Punto1){
                        Solucion1[i]=S2[i]
                        Solucion2[i]=S1[i]
                }
                if(i > punto1 AND i < punto2){
                        Solucion1[i]=S1[i]
                        Solucion2[i]=S2[i]
                }
                else{
                        Solucion1[i]=S2[i]
                        Solucion2[i]=S1[i]

                }
        }
        Validar(Solucion1)
        Validar(Solucion2)
        If(Aleatorio<0.1){
                Solucion1.Mutacion(0.1)
        }
        If(Aleatorio<0.1){
                Solucion2.Mutacion(0.1)
        }
        Hijos.Añadir(Solucion1,Solucion2)
        Devolver Hijos ;
}
```

Validar

```
Cruce(Solucion S1) {
        Diferencia=unos.size()-m
        if (0<diferencia){
                desde 0 hasta diferencia{
                        uno=Aleatorio(Unos.size())
                        Unos.Borrar(uno)
```

```
                    Ceros.Añadir(uno)
            }
        }
        if (0>diferencia){
            desde 0 hasta diferencia{
                    cero=Aleatorio(Ceros.size())
                    Ceros.Borrar(cero)
                    Unos.Añadir(cero)
            }
        }
```

Mutación

```
Mutacion( s ) {
    Desde  0 hasta  s {
        numeroUno = r.nextInt(unos.tamaño());
        numeroCero = r.nextInt(ceros.tamaño());
        Unoscam.añadir(numeroUno);
        Ceroscam.añadir(numeroCero);
        unos.borrar(numeroUno);
        ceros.borrar(numeroCero);
    }
    Desde  0 hasta s{
        unos.añadir(Unoscam.borrar(recore));
         ceros.añadir(Ceroscam.borrar(recore));
    }
    Devolver FuncionObjetivo();
    }
```

C. Algoritmo Memetico

```
ALG_Memetico( TamPoblacion, semilla,mutar, cruzar, Tammutar ,periodo, Aplica) {

        desde  0  hasta TamPoblacion{
                x=SolucionAleatoria();
                población.Añadir(x)
        }
        desde  TamPoblacion  hasta 100000{
                Padres=Torneo(Poblacion)
                Desde i=0 hasta tamPoblacion/2{
                        Si(Aleatorio <0.7){
                                Hijos=Cruce(padre[1],padre[2])
                        }else{
                                Si(Aleatorio < 0.1){
                                        Padre[1].Mutar(0.1)
                                }
                                Si(Aleatorio < 0.1){
                                        Padre[1].Mutar(0.1)
                                }
                        }
                        Población.Borrar(padres)
                        Poblacion.Añadir(Hijos)
                        GuargarMejorSolucion(Poblacion)
                }
                si (con % periodo == 0) {
                        for (int y = 0; y < aplica; y++) {
                                poblacion.get(y).Busquedalocal(evaluaciones,1000);
                        }
                   }
                GuargarMejorSolucion(Poblacion)
        }
        Devolver maximo
}
```

D. Busqueda dispersa

```
Busqueda_Dispersa(){
        desde  0  hasta TamPoblacion{
                x=SolucionDiversa();
                población.Añadir(x)
        }
        desde  TamPoblacion  hasta 100000{
                CalcularDistancias(poblacion)
                InsertarMejores(R1)
                InsertarMasDistantes(R2)
                HijosTotal=Cruzar(R1[0],R1[0])
                Hijos= Cruzar(R1[0],R1[0])
                HijosTotal.Añadir(Hijos[0])
                HijosTotal=Cruzar(R1[1],R1[2])
                Hijos= Cruzar(R1[1],R1[2])
                HijosTotal.Añadir(Hijos[0])
                HijosTotal=Cruzar(R1[0],R1[2])
                Hijos= Cruzar(R1[0],R1[2])
                HijosTotal.Añadir(Hijos[0])

                HijosTotal=Cruzar(R2[0],R2[0])
                HijosTotal=Cruzar(R2[1],R2[2])
                HijosTotal=Cruzar(R2[0],R2[2])

                Desde i=0 hasta 3{
                        Desde j=0 hasta 3{
                                HijosTotal.añadir(Cruzar(R2[i],R2[j]))
                        }
                }
                GuargarMejorSolucion(Poblacion)
        }
Devolver Mejor;
}
```

Eleccion Diversa

```
EleccionDiversa(Ci,TamPoblacion)
{
        x = 0, i = 0;
        mientras (x < m) {
           int numero = Aleatorio (ceros.size());
           if ((1 - ((Ci[numero] / TamPoblacion))) > Aleatorio(0,1)) {
              cero = ceros.borrar(numero);
              unos.añadir(cero);
              sol[cero]=1
              x++;
              x = x % n;
           }
           i++;
        }
        this.FuncionObjetivo();
        this.BL(1000);
        devolver Solucion;
}
```

Calcular distancia

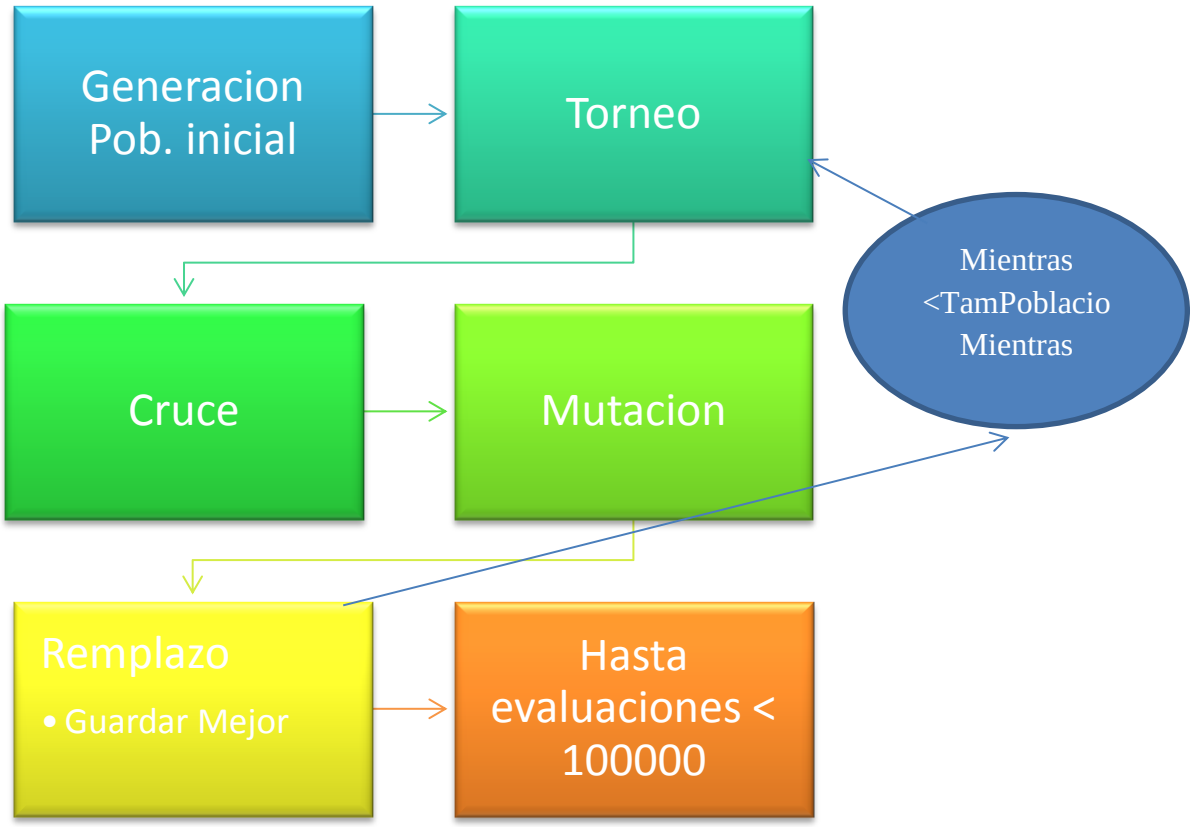
Calculo la distancia entre dos soluciones:

```
distancia(Solucion s1, Solucion s2) {
        d = 0;
        desde i =0 hasta n{
                si (s1[i] diferente s2[i]) {
                        d++;
                }
        }
        return d;
}
```

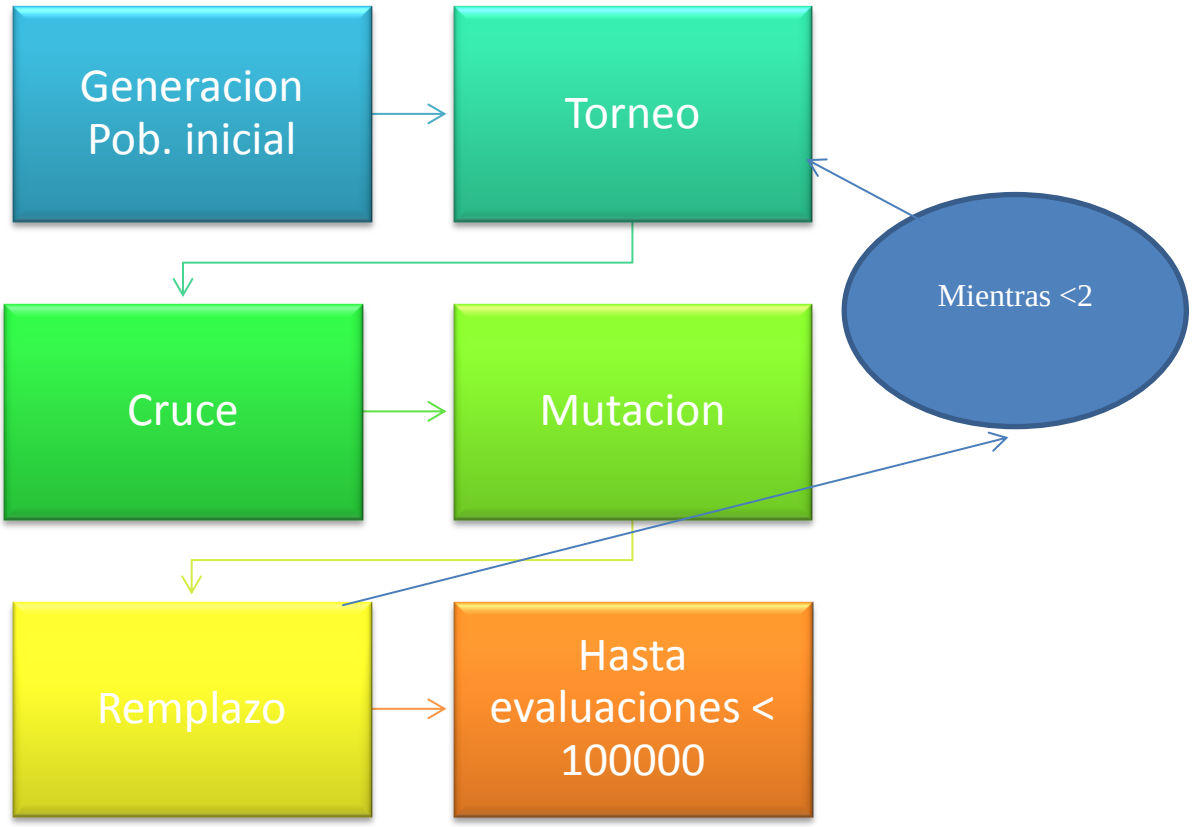
Y me ayudo de esta función anterior para calcular la de una solución y una poblacion

```
CacularDistancia(Solucion S1,Poblacion)
{
        Cont=0
        Desde i=0 hasta población.size(){
                Cont=Cont+Distancia(S1,población[i])
        }
        devolver Solucion;
}
```

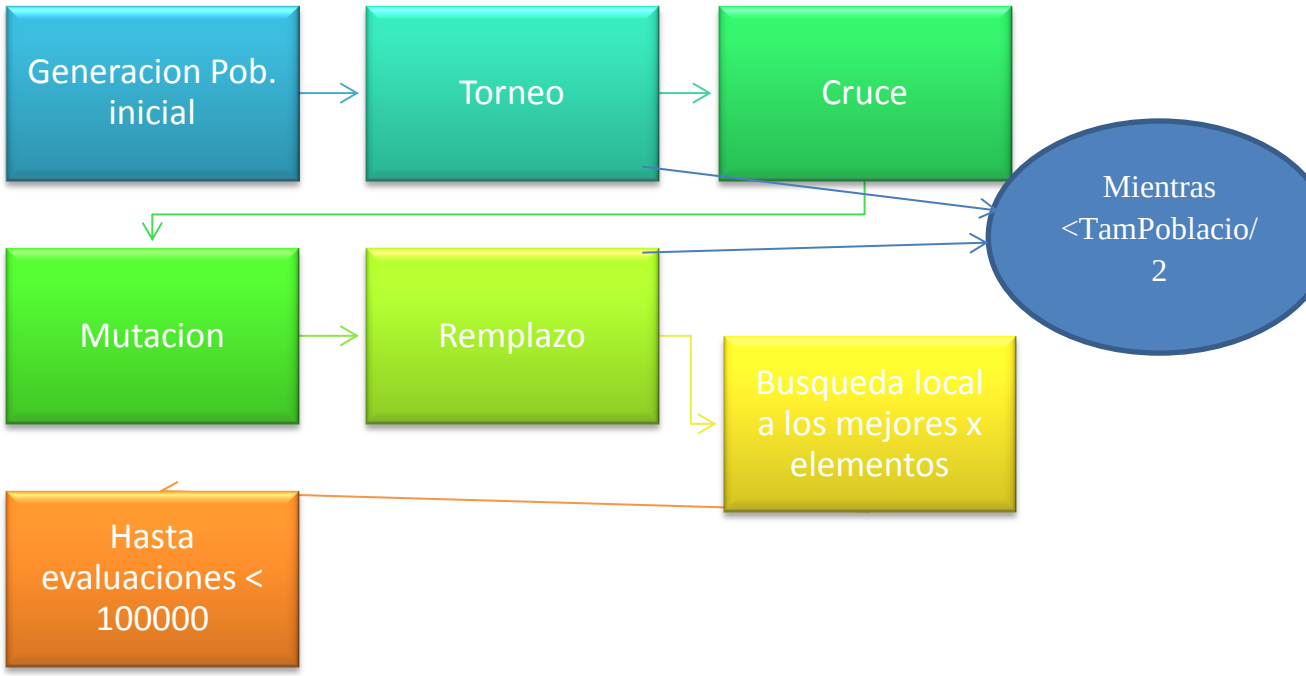
Algoritmo Genético Generacional



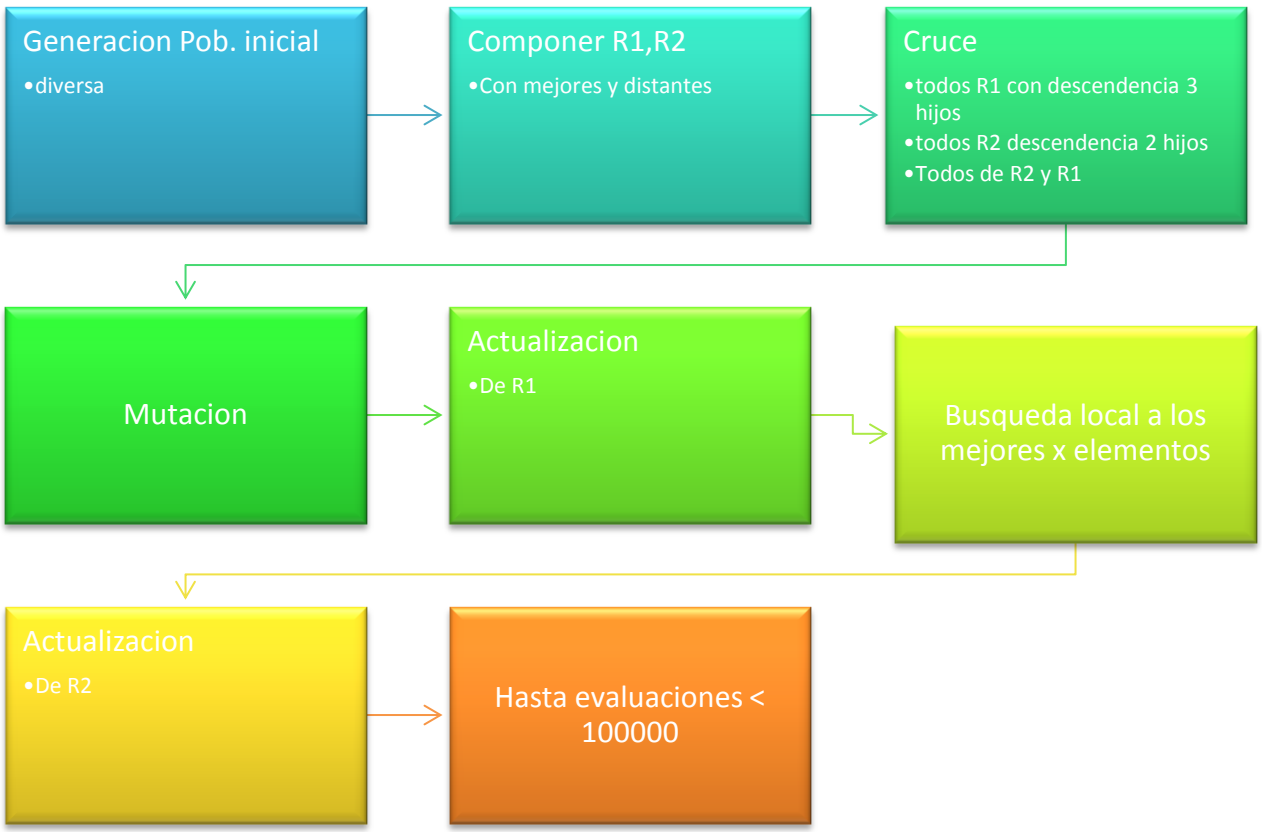
Algoritmo Genético Estacionario



Algoritmo Memetico



Algoritmo Busqueda dispersa



4. Experimentos y análisis de resultados

a) Instancias del problema estudiadas y valores de los parámetros considerados en las ejecuciones de cada algoritmo.

✓ **Semillas utilizadas:**

Ejecución	Semilla
1	12345678
2	23456781
3	34567812
4	45678123
5	56781234
6	67812345
7	78123456
8	81234567
9	87654321
10	18765432

✓ **Instancias del problema estudiadas:**

Las tres instancias del problema que he usado son:

- SOM-b-10: Número de elementos 300, m=60.
- MDG-b-1: Número de elementos 500, m=50.
- SOM-b-20: Número de elementos 500, m=200.
- GKD-c_20: Número de elementos 500, m= 50.

.

b) Resultados para cada problema.

1. Alg genetico

	SOM-b-10	MDG-b-1	SOM-b-20	GKD-c_20
Greedy	9620	760737,59	97263	19604,84356
BL-Limitada	8459	652772,230999999	90633	16676,3846789999
Optimo	9689	778030,625	97344	19604,84375

Análisis de los resultados:

Los algoritmos geneticos .

2. Alg Genético Generacional

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	Generacional	Generacional	Generacional	Generacional
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	94307	737862,400000000	9388	19314,58439
23456781	94463	737529,880000000	9394	19422,22624
34567812	94714	718980,159999999	9343	19448,77938
45678123	94962	725797,000000000	9292	19429,44491
56781234	94322	730246,190000000	9443	19458,14339
67812345	95104	720794,520000000	9487	19515,66807
78123456	94854	722588,170000001	9158	19346,69844
81234567	95189	731060,780000000	9292	19344,51291
87654321	93625	746260,400000001	9322	19562,01777
18765432	94258	731244,989999999	9227	19316,91611
Optimo	97344	778030,625	9689	19604,84375

Análisis de resultados

Este algoritmo da muy bueno resultados por que explora y explota muy bien además no converge demasiado

3. Alg genético estacionario

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	Estacionario	Estacionario	Estacionario	Estacionario
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	92348	686142,64	9065	18457,62525
23456781	92849	689678,05	8908	18665,99523
34567812	93124	709647,02	8973	18364,91793
45678123	92883	693401,37	8958	18825,54057
56781234	93005	705833,7	8957	18756,19618
67812345	92562	692368,2	9040	18756,8497
78123456	93342	699897,01	9000	18358,92096
81234567	93124	704072,39	8954	18446,4407
87654321	92958	696519,54	8889	18681,51141
18765432	92484	693553,93	9161	18492,7809
Optimo	97344	778030,625	9689	19604,84375

Análisis de resultados

4. Alg Memetico(1,2)

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	Memetico	Memetico	Memetico	Memetico
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	93295	686235,70000000	8997	17732,899280000
23456781	92774	701579,02000000	8949	18169,578170000
34567812	92223	709726,32000000	9041	18469,505340000
45678123	93250	704405,89000000	9064	18415,381540000
56781234	92633	705283,66000000	9051	17463,706370000
67812345	92672	709902,90000000	9036	18150,801730000
78123456	93280	700911,89000000	9004	18221,242890000
81234567	93238	707817,55000000	9183	18045,434380000
87654321	93314	694017,70000000	9039	18163,818110000
18765432	92831	698803,14000000	8722	17791,798020000
Optimo	97344	778030,625	9689	19604,84375

Análisis de resultados

En todos los algoritmos memeticos utilizamos una búsqueda local acotada para poder observar como explota y explora las soluciones aunque no consigue profundizar demasiado en la explotación aunque en algunas soluciones si consigue profundizar más.

5. Alg Memetico(1,10)

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	Memetico	Memetico	Memetico	Memetico
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	92395	699974,45000	8936	17462,876450000
23456781	92912	715553,02000	9077	17636,068580000
34567812	92700	707854,78000	9076	17574,315740000
45678123	92467	692827,37000	8898	17715,772530000
56781234	92244	697744,13000	8996	17597,134100000
67812345	92874	717202,07000	9056	17533,931930000
78123456	92100	708333,83000	8996	17695,823320000
81234567	92208	707179,79000	9007	17624,443890000
87654321	92022	695532,55000	8971	17607,998400000
18765432	92628	714532,04000	9113	17558,281590000
Optimo	97344	778030,625	9689	19604,84375

Análisis de resultados

En este otro caso es lento aunque consigue explorar mucho gracias a las búsquedas locales aunque perdemos diversidad.

6. Alg Memetico(10,2)

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	Memetico	Memetico	Memetico	Memetico
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	92920	711664,37000	9124	18291,899800000
23456781	93207	733842,82000	9026	17733,230520000
34567812	92678	710227,82000	9022	17973,459120000
45678123	92610	703226,84000	9020	18369,581230000
56781234	93076	705404,97000	9108	18614,873900000
67812345	92830	702071,00000	9072	17610,797050000
78123456	93252	705076,55000	8899	18308,724340000
81234567	92923	708845,22000	8995	17868,370730000
87654321	92854	717824,58000	8962	18307,310630000
18765432	93331	681947,32000	8937	18162,056650000
Optimo	97344	778030,625	9689	19604,84375

Análisis de resultados

Este algoritmo no consigue explotar demasiado la población de soluciones.

7. Alg Memetico(10,10)

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	Memetico	Memetico	Memetico	Memetico
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	92363	709324,58000	8977	18431,19571000
23456781	92652	706115,73000	9007	18163,86759000
34567812	93285	702526,49000	9126	18090,41519000
45678123	92740	699396,34000	8926	18001,30206000
56781234	92703	710695,30000	8978	18172,77914000
67812345	93001	706102,64000	9098	18292,81981000
78123456	92953	706163,01000	8968	18211,28520000
81234567	93045	708747,33000	9087	18211,51854000
87654321	92776	705912,36000	8908	18451,48289000
18765432	92805	712809,72000	9003	18113,06476000
Optimo	97344	778030,625	9689	19604,84375

Análisis de resultados

Esta búsqueda es más lenta y los resultados son muy parecidos a los de las otras y converges más.

8. Alg Búsqueda Dispersa

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	B. Dispersa	B. Dispersa	B. Dispersa	B. Dispersa
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	92643	705887,79000	8900	18012,88355000
23456781	92448	704796,74000	8979	18162,18984000
34567812	92669	701111,50000	9092	18127,70520000
45678123	92927	703913,39000	9072	17958,14424000
56781234	92655	711443,52000	9033	18064,53592000
67812345	92738	693539,24000	9064	18331,27622000
78123456	93080	699251,12000	9057	17974,62020000
81234567	92599	703955,87000	8917	18230,06212000
87654321	92526	708048,45000	8928	17939,11701000
18765432	92706	708513,38000	9034	18220,92356000
Optimo	97344	778030,625	9689	19604,84375

Análisis de resultados

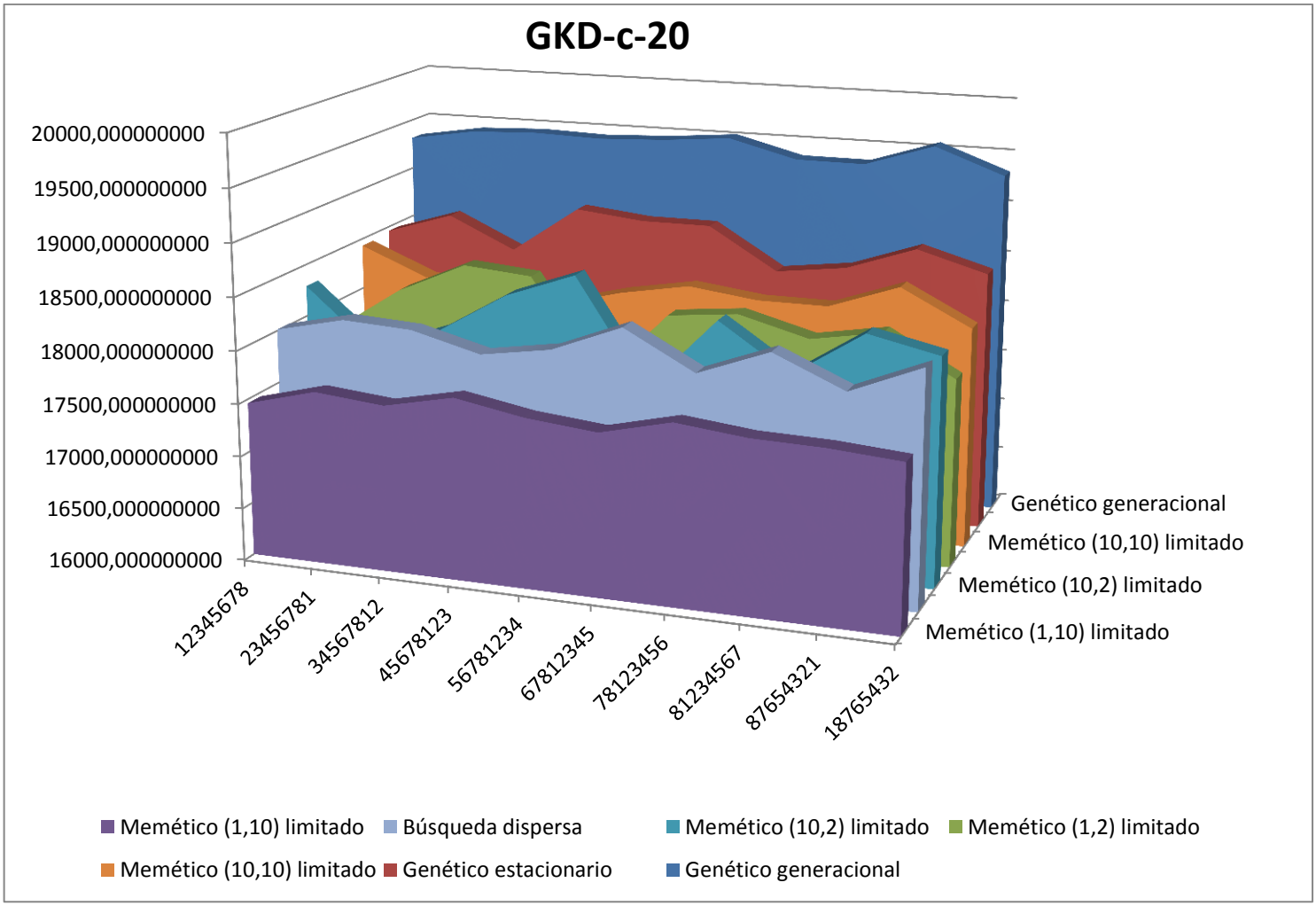
Esta búsqueda consigue explorar y explotar pero no explora demasiado el espacio de búsqueda.

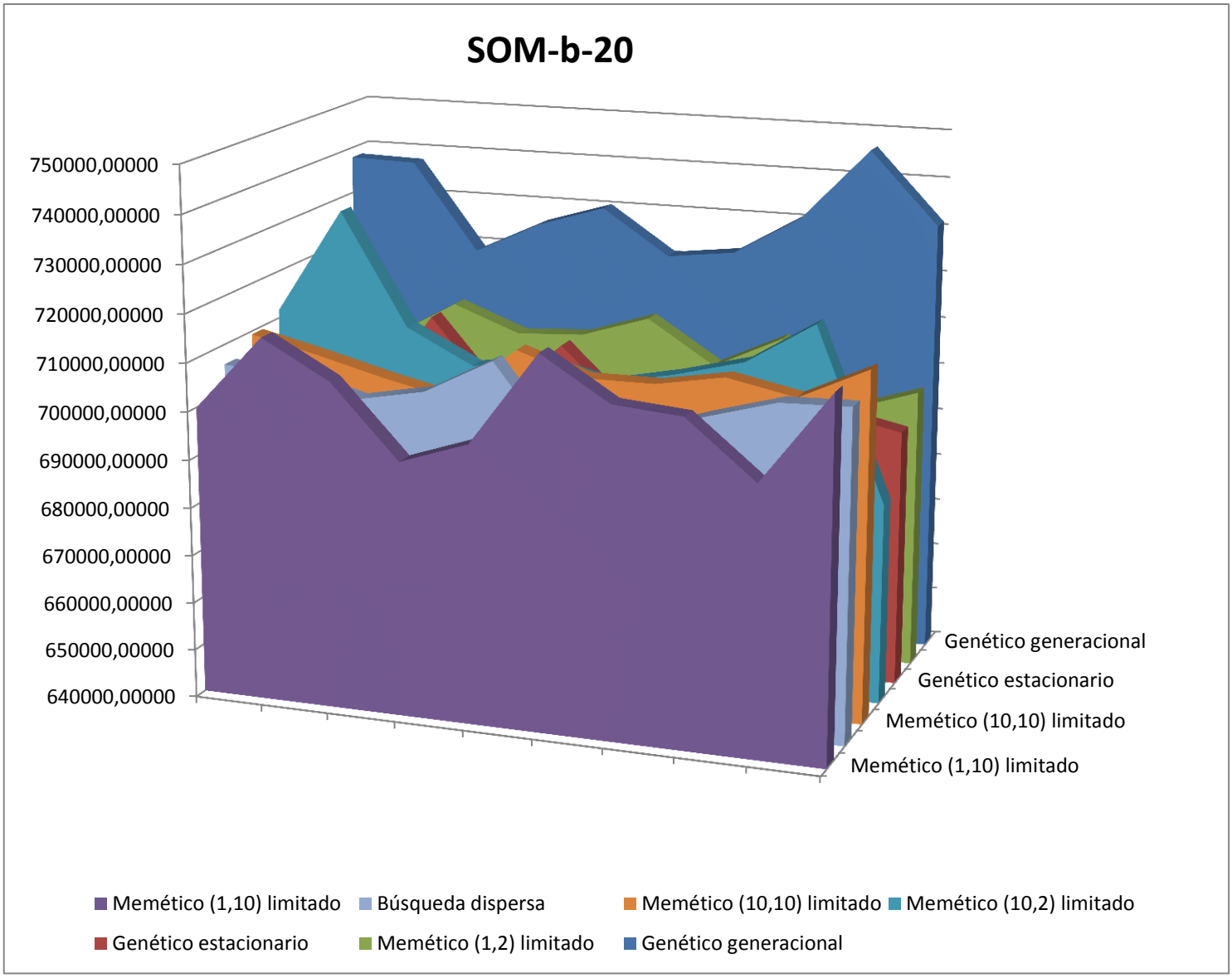
c) Resultados Globales obtenidos

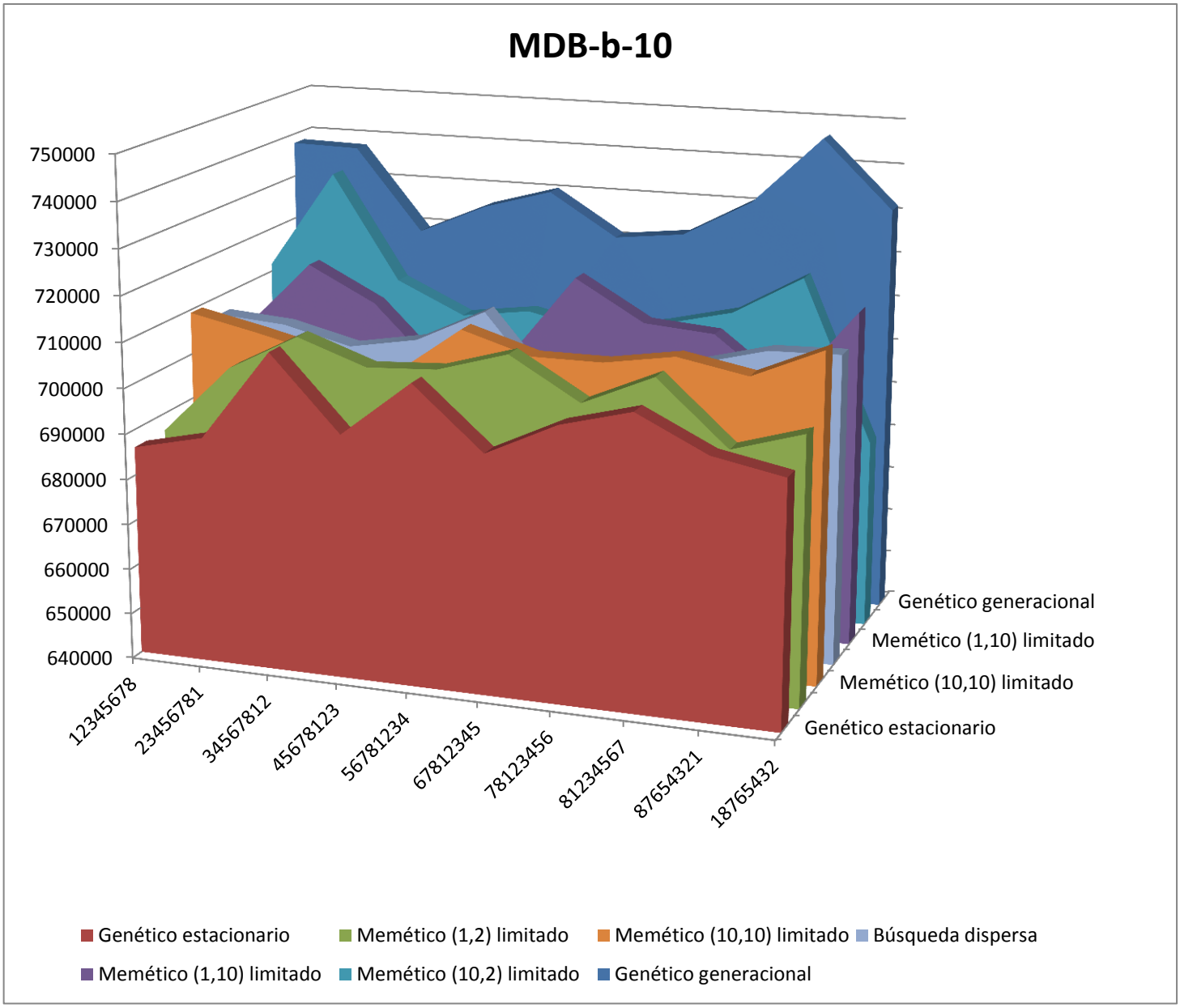
	SOM-b-20				MDG-b-1			
Metodo	mejor	media	peor	σ	m	x	p	σ
Optimo	-	97344	-	-	-	778030,625	-	-
Greedy	-	96499	-	-	-	754034,96	-	-
BL(1000)	-	90633,5	-	-	-	652772,2310	-	-
AG Generacional	95189	94579,80	93625	453,310446	746260,4000000010	730236,44900000	718980,1599999990	8147,833981
AG Estacionario	93342	92867,90	92348	299,11451	709647,0200000010	697111,3849999990	686142,6400000000	7171,274914
AM-(1,2)	93314	92951,00	92223	358,440232	709902,9000000010	701868,3769999990	686235,6999999990	7044,179227
AM-(1,10)	92912	92455,00	92022	299,524957	717202,0699999990	705673,4030000000	692827,3699999990	8284,363721
AM-(10,2)	93331	92968,10	92610	230,183166	733842,819999998	708013,148999999	681947,319999999	12391,27172
AM-(10,10)	93285	92832,30	92363	239,814533	712809,719999999	706779,350000000	699396,340000001	3696,981163
BD	93080	92699,10	92448	175,390108	711443,519999999	704046,099999999	693539,240000000	4864,936506

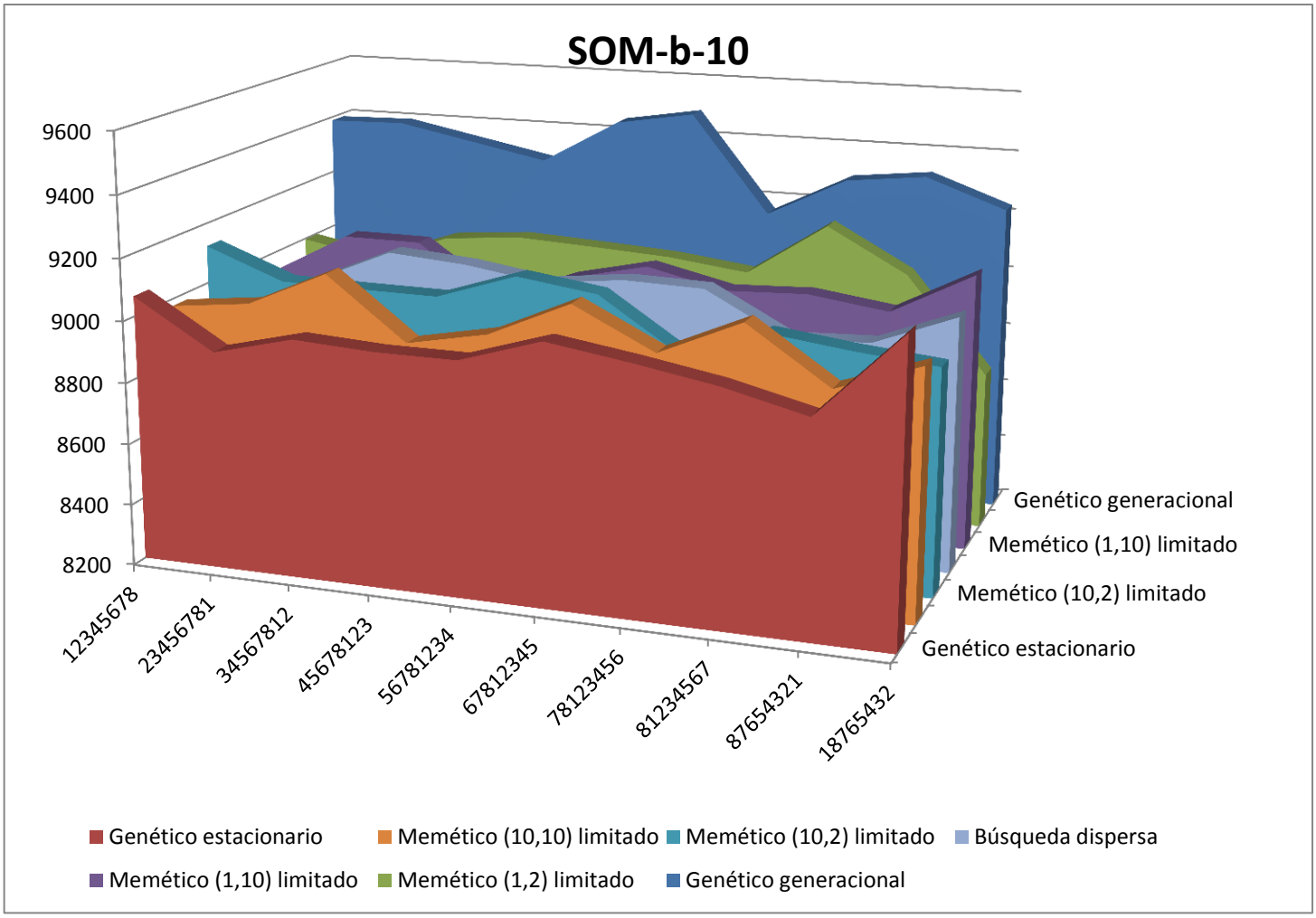
SOM-b-10				GKD-c-20			
m	x	p	σ	m	x	p	σ
-	9689	-	-	-	19604,84375	-	-
-	9523	-	-	-	19604,84356	-	-
-	8459.8	-	-	-	16676,384679	-	-
9487	9334,6	9158	93,79786778	19562,0177700000	19415,8991610000	19314,5843900000	80,21268594
9161	8990,5	8889	76,21449993	18825,5405700000	18580,6778830000	18358,9209600000	166,0913329
9183	9008,6	8722	111,2898917	18469,5053399999	18062,4165830000	17463,7063700000	297,6016627
9113	9012,6	8898	64,3773252	17715,7725299999	17600,6646529999	17462,8764499999	70,79517534
9124	9016,5	8899	68,33776408	18614,8739000000	18124,0303970000	17610,7970500000	300,452787
9126	9007,8	8908	69,624421	18451,4828899999	18213,9730889999	18001,3020600000	135,9088452
9126	9007,6	8900	67,21190371	18331,276220000	18102,145786000	17939,117010000	126,5661593

d) Graficas de comparación.




SOM-b-20
750000,00000
740000,00000
730000,00000
720000,00000
710000,00000
700000,00000
690000,00000
680000,00000
670000,00000
660000,00000
650000,00000
640000,00000
Genético generacional
Genético estacionario
Memético (10,10) limitado
Memético (1,10) limitado
Memético (1,10) limitado
Búsqueda dispersa
Memético (10,10) limitado
Memético (10,2) limitado
Genético estacionario
Memético (1,2) limitado
Genético generacional


MDB-b-10
750000
740000
730000
720000
710000
700000
690000
680000
670000
660000
650000
640000
12345678
23456781
34567812
45678123
56781234
67812345
78123456
81234567
87654321
18765432
Genético generacional
Memético (1,10) limitado
Memético (10,10) limitado
Genético estacionario
Genético estacionario
Memético (1,2) limitado
Memético (10,10) limitado
Búsqueda dispersa
Memético (1,10) limitado
Memético (10,2) limitado
Genético generacional


SOM-b-10
9600
9400
9200
9000
8800
8600
8400
8200
12345678
23456781
34567812
45678123
56781234
67812345
78123456
81234567
87654321
18765432
Genético generacional
Memético (1,10) limitado
Memético (10,2) limitado
Genético estacionario
Genético estacionario
Memético (10,10) limitado
Memético (10,2) limitado
Búsqueda dispersa
Memético (1,10) limitado
Memético (1,2) limitado
Genético generacional

5. Referencias bibliográficas

- Russell, S.; Norvig, P. (2003): Inteligencia Artificial — Un Enfoque Moderno (2ª ed.). Prentice Hall Hispanoamericana
- Apuntes asignatura Algorítmica 4º Ingeniería informática (2012)

6. Manual de usuario

El programa se compila mediante netsBeans, pulsando Mayusculas+F11 y dentro de una carpeta llamada dist estará nuestro punto jar.

El programa funciona leyendo de un fichero llamado datos.txt, en el cual debemos poner Numero del algoritmo:0 AG Generacinal,1 AG Estacionario,2 Memetico(1,2) ,3 Memetico(1,10),4 Memetico(10,2),5 Memetico(10,10),6 Memetico(1,2) Limitado,7 Memetico(1,10) Limitado,8 Memetico(10,2) Limitado,9 Memetico(10,10) Limitado,10 Busqueda dispersa, y después podremos espacio, numero de semillas a utilizar , y espacio y número de ficheros , y después las semillas con un \n al final y los nombres de archivo igual. Nos generara un fichero con las soluciones.