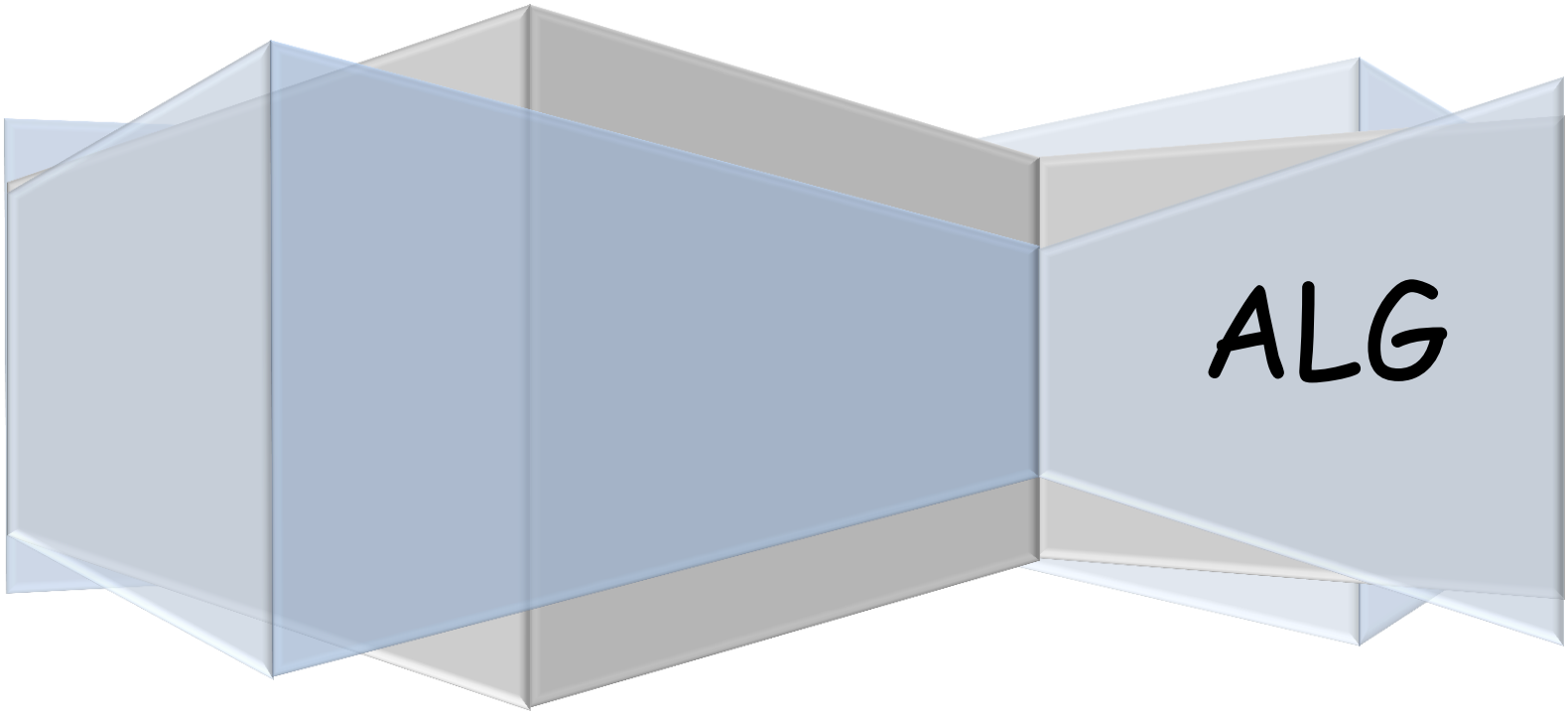
Práctica 2

MDP con Búsqueda Multiarranque, GRASP, GRASP extendido ILS VNS

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 _____ 2

2. Descripción de la aplicación al problema de los algoritmos empleados _____ 3

Representación _____ 3

Función objetivo _____ 3

Descripción en pseudocódigo de la función objetivo _____ 3

Descripción en pseudocódigo Generar Solución aleatoria _____ 4

Búsqueda Local_____ 4

3. Pseudocódigo _____ 5

4. Experimentos y análisis de resultados _____ 14

5. Referencias bibliográficas _____17

6. Manual de usuario _____18

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 en pseudocódigo Generar Solución aleatoria.

```
GeneraSolAleatria (){
        Desde i=0 hasta i=M{
                NumAleatorio=GeneraNumAleatorio(0,Lista0.size())
                Lista0.borrar(NumAleatrorio)
                Lista1.añadir (NumAleatrorio)
        }
}
```

e) Descripción del algoritmo de búsqueda local

```
BL(boolean Flag1, int semilla):
        Si(Flag1==true){
                LimpiaSolucion()
                SolucionAleatoria()
                Random(semilla)
        }
        maxGlobal=FunObjetivo(Lista1,Lista0)
        Auxiliar1=Lista1.clone()
        Auxiliar0=Lista0.clone()
        mientras(mejora==True){
                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()
                                if(maxGlobal<max):
                                        mejora=True
                                        Auxiliar1=Lista1.clone()
                                        Auxiliar0=Lista0.clone()
                                        maxGlobal=max
                                else:
                                        Lista1=Auxiliar1.clone()
                                        Lista0=Auxiliar0.clone()
                        }
                }
        }
```

3. Pseudocódigo

A. BMB

```
public double MBM(int semilla) {
        mejorCeros = ceros.clone();
         mejorUnos = unos.clone();
        maximo = 0;
        desde  0  hasta 25{
                SolucionAleatoria();
                diver = BL();
                si (diver > maximo) {
                        maximo = diver;
                        mejorCeros = ceros.clone();
                        mejorUnos = unos.clone();
                }
        }
        SolucionCeros=mejorCeros
        SolucionUnos=mejorUnos
        Devolver maximo
}
```

B. GRASP

```
GRAPS() {
        maximo = this.GredyND(rnd, 5);
        mejorUnos = unos.clone();
        mejorCeros = ceros.clone();
        desde 0 hasta 25 {
                BL();
                si(MAX > maximo) {
                        maximo = MAX;
                        mejorUnos = unos.clone();
                        mejorCeros = ceros.clone();
                }
                GredyND(rnd, 5);
        }
        SolucionUnos = mejorUnos.clone();
        SolucionCeros = mejorCeros.clone();
        return MAX;
}
```

Greedy Probabilístico

```
GredyND(Random a, l ) {
        siguiente = a.nextInt(ceros.size());
        unos.añadir(siguiente);
        ceros.borrar(siguiente);
        desde 1 hasta m{
                siguiente = this.BuscarProbablidad(l, a);
                cero = ceros.borrar(siguiente);
                unos.añadir(cero);
        }
        Devolver  FuncionObjetivo();
}


BuscarProbablidad(l, Random a) {
        max = 0;
        listaMax =[ ]
        pos = 0;
```

```
        AuxCeros= ceros.clone();
        AuxUnos = unos.clone();
        Desde  0 hasta  l {
                desde 0 hasta AuxCeros.tamaño{
                aux = 0;
                for (int s = 0; s < AuxUnos.tamaño(); s = s + 1) {
                        si (AuxCeros[i] < AuxUnos[s]) {
                        aux = Matriz[AuxCeros[i]][AuxUnos[s]] + aux;
                        }
                        si (AuxCeros[i] > AuxUnos[s]) {
                                aux = Matriz[AuxUnos[s]][AuxCeros[i]] + aux;
                        }
                }
                si (max < aux) {
                        listaMax.añadir(i);
                        AuxCeros.borrar(i);
                }
        }
}
      pos = listaMax.get(a.nextInt(listaMax.size()));
      return pos;
   }
```

C. GRASP-Extendido

```
GRAPS_Ex() {
        maximo = this.GredyND(rnd, 7);
        mejorUnos = (ArrayList<Integer>) unos.clone();
        mejorCeros = (ArrayList<Integer>) ceros.clone();
        Desde 0 hasta 5 {
                Desde 0 hasta 5 {
                        this.BL(false, 0);
                        if (this.MAX > maximo) {
                                maximo = this.MAX;
                                mejorUnos = (ArrayList<Integer>) unos.clone();
                                mejorCeros = (ArrayList<Integer>) ceros.clone();
                        }
                        s = 0.10 * m;
                        Mutacion(s, rnd);
                        Si (this.MAX > maximo) {
                                maximo = MAX;
                                mejorUnos = unos.clone();
                                mejorCeros = ceros.clone();
                        }
                }
                GredyND(rnd, 5);
        }
        unos = mejorUnos.clone();
        ceros = mejorCeros.clone();
        devolver maximo;
}
```

D. *Greedy + BL-Extendida*

```
GreedyBL_Ex(int semilla) {
        maximo = Gredy();
        mejorUnos = unos.clone();
        mejorCeros = ceros.clone();
        desde 0 hasta 5{
                s = (int) 0.1 * m;
                baraja(unos);
                Mutacion(s);
                BL();
                si (MAX > maximo) {
                        maximo = MAX;
                        mejorUnos = unos.clone();
                        mejorCeros = ceros.clone();
                }
                Gredy();
        }
        unos = mejorUnos;
        ceros = (ArrayList<Integer>) mejorCeros;
        Devolver  FuncionObjetivo()
}
```

Greedy Probabilístico

```
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();
}
```

E. ILS

```
public double ILS(int semilla) {
        s = (int) 0.10 * m;
        maximo = SolucionAleatoria(rnd);
        mejorUnos = (ArrayList<Integer>) unos.clone();
        mejorCeros = (ArrayList<Integer>) ceros.clone();
        BL(false, 0);
        Desde  0 hasta 24{
                Mutacion(s, rnd);
                BL(false, 0);
                si(MAX > maximo) {
                        maximo = MAX;
                        mejorUnos = unos.clone();
                        mejorCeros = ceros.clone();
                }
        }
        unos = mejorUnos;
        ceros = mejorCeros;
        Devolver FuncionObjetivo();
}
```

F. VNS

```
VNS() {
        maximo = this.SolucionAleatoria(rnd);
        UnosActual = unos.clone();
        CerosActual = ceros.clone();
        kmax = 5;
        s = 0;
        k = 1;
        bl = 0;
        Mientras (bl < 25) {
                si (k > kmax)  entonces k = 1
                si (k == 1) entonces s = 0.02 * m
                si (k == 2) entonces s = 0.04 * m
                si (k == 3) entonces s = 0.06 * m
                si (k == 4) entonces s = 0.08 * m
                si (k == 5) entonces s = 0.1 * m
                GenerarVecino(s, rnd);
                BL(false, 0);
                bl = bl + 1;
                if (MAX > maximo) {
                        UnosActual = (ArrayList<Integer>) unos;
                        CerosActual = (ArrayList<Integer>) ceros;
                        maximo = MAX;
                        k = 1;
                } else {
                        k = k + 1;
                }
        }
        unos = (ArrayList<Integer>) UnosActual;
        ceros = (ArrayList<Integer>) CerosActual;
        Devolver FuncionObjetivo();
}
```

Genera Vecino

```
GenerarVecino( s ) {
        Desde  0 hasta s {
                numeroUno = r.nextInt(unos.size())
                numeroCero = r.nextInt(ceros.size())
                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()
}
```

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. Greedy mas BL

	SOM-b-10	MDG-b-1	SOM-b-20	GKD-c_20
Greedy	9620	760737,59	97263	19604,84356
Optimo	9689	778030,625	97344	19604,84375

Análisis de los resultados:

Los algoritmos Greedy mas búsqueda local son generalmente bastante buenos aunque dependen de si al empezar en la solución greedy nos quedamos antes o más tarde un máximo local. A pesar de esto los resultados son muy buenos en casi todos los archivos.

2. BMB

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	BMB	BMB	BMB	BMB
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	97344	777890,2700000003	9667	19604,843560000027
23456781	97331	777606,5900000004	9665	19604,843560000016
34567812	97302	772781,8300000014	9655	19604,843560000023
45678123	97316	776273,7900000002	9688	19604,843560000023
56781234	97309	777890,2699999991	9684	19604,84356000002
67812345	97322	775903,3300000008	9680	19604,84356000003
78123456	97339	773727,1599999998	9675	19604,843560000023
81234567	97325	777670,8700000001	9658	19604,843560000012
87654321	97322	778030,570000002	9672	19604,843560000023
18765432	97325	777741,9400000004	9681	19604,843560000016
Optimo	97344	778030,625	9689	19604,84375

Análisis de resultados

Esta búsqueda da muy buenos resultados obteniendo entre ellos el óptimo, Esto tiene su lógica puesto que exploramos mucho con la BL y como en cada una de ellas inicializamos de nuevo exploramos de nuevo.

3. GRASP

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	GRASP	GRASP	GRASP	GRASP
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	97326	778030,5700000005	9676	19604,843560000016
23456781	97344	778030,5700000032	9685	19604,843560000023
34567812	97316	773986,8399999975	9648	19604,843560000016
45678123	97313	777606,5900000012	9686	19604,84356000002
56781234	97327	775251,6500000008	9685	19604,84356000002
67812345	97333	773912,5600000032	9683	19604,84356000002
78123456	97326	773756,4100000019	9675	19604,84356000002
81234567	97340	774869,1299999977	9683	19604,84356000003
87654321	97333	775852,7300000001	9678	19604,843560000023
18765432	97322	775071,8899999986	9681	19604.843560000023
Optimo	97344	778030,625	9689	19604,84375

Análisis de resultados

Esta búsqueda da muy buenos resultados aunque se queda estancada en máximos globales estos se aproximan bastante al óptimo.

4. GRASP Extendido

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	GRASP-EX	GRASP-EX	GRASP-EX	GRASP-EX
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	97291	771647,3600000003	9669	19604,8435600000
23456781	97322	775728,2600000002	9681	19604,8435600000
34567812	97313	771919,5699999998	9676	19604,8435600000
45678123	97321	772125,6599999989	9596	19604,8435600000
56781234	97257	773465,7400000006	9619	19604,8435600000
67812345	97308	770648,7299999996	9637	19604,8435600000
78123456	97269	773306,6900000002	9642	19604,8435600000
81234567	97255	767148,4999999999	9664	19604,4356000001
87654321	97292	772492,0099999997	9661	19604,8435600000
18765432	97322	771713,1599999999	9616	19604,8435600000
Optimo	97344	778030,625	9689	19604,84375

Análisis de resultados

Esta búsqueda da muy buenos resultados, se queda pronto en máximos locales que no son tan buenos como el GRASP, debería hacer un poco más de exploración.

5. Greedy mas BL Extendida

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	Greedy+BL-EX	Greedy+BL-EX	Greedy+BL-EX	Greedy+BL-EX
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	97258	771789,9400000006	9616	19604,84356000002
23456781	97310	773549,6600000012	9632	19604,843560000034
34567812	97278	772393,1400000011	9625	19604,843560000023
45678123	97257	772301,6900000001	9625	19604,843560000027
56781234	97301	773341,7200000002	9608	19604,843560000034
67812345	97305	770060,6500000008	9644	19604,843560000045
78123456	97312	776017,9299999995	9627	19604,843560000016
81234567	97298	773407,4799999997	9638	19604,84356000002
87654321	97305	771935,4500000005	9639	19604,843560000012
18765432	97290	766141,7600000013	9618	19604,84356000002
Optimo	97344	778030,625	9689	19604,84375

Análisis de resultados

Esta búsqueda da muy buenos resultados con una explotación y exploraciones buenas .

6. ILS

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	ILS	ILS	ILS	ILS
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	97112	772748,3199999998	9591	19604,843560000016
23456781	97259	772863,7700000007	9624	19604,843560000005
34567812	97313	774863,3000000003	9517	19604,843559999998
45678123	97161	774538,6600000008	9661	19604,843559999994
56781234	97164	772312,2000000002	9662	19604,84356
67812345	97217	764055,3799999999	9616	19604,843560000005
78123456	97311	755872,0299999999	9542	19604,843559999983
81234567	97281	759277,5100000013	9643	19604,84356
87654321	97124	771869,4699999994	9514	19604,843559999998
18765432	97262	760194,1099999995	9618	19604,843559999994
Optimo	97344	778030,625	9689	19604,84375

Análisis de resultados

Esta búsqueda da malos resultados porque no explota mucho cuando se produce la mutación.

7. VNS

	SOM-b-20	MDG-b-1	SOM-b-10	GKD-c_20
	ILS	ILS	ILS	ILS
Semillas	Resultados	Resultados	Resultados	Resultados
12345678	97054	757944,9000000004	9583	19442,42748000002
23456781	97255	756867,8700000009	9516	19442,52576000002
34567812	97012	768278,6600000004	9490	19446,13829999999
45678123	97001	770934,6500000008	9503	19454,145349999977
56781234	97046	765778,0799999997	9482	19445,484080000006
67812345	97200	767200,6100000006	9495	19435,011529999996
78123456	97190	761130,35	9458	19422,713969999997
81234567	97131	750328,6600000007	9490	19400,962730000003
87654321	97146	756601,060000001	9517	19433,01990000002
18765432	96997	754347,5499999984	9539	19415,8892
Optimo	97344	778030,625	9689	19604,84375

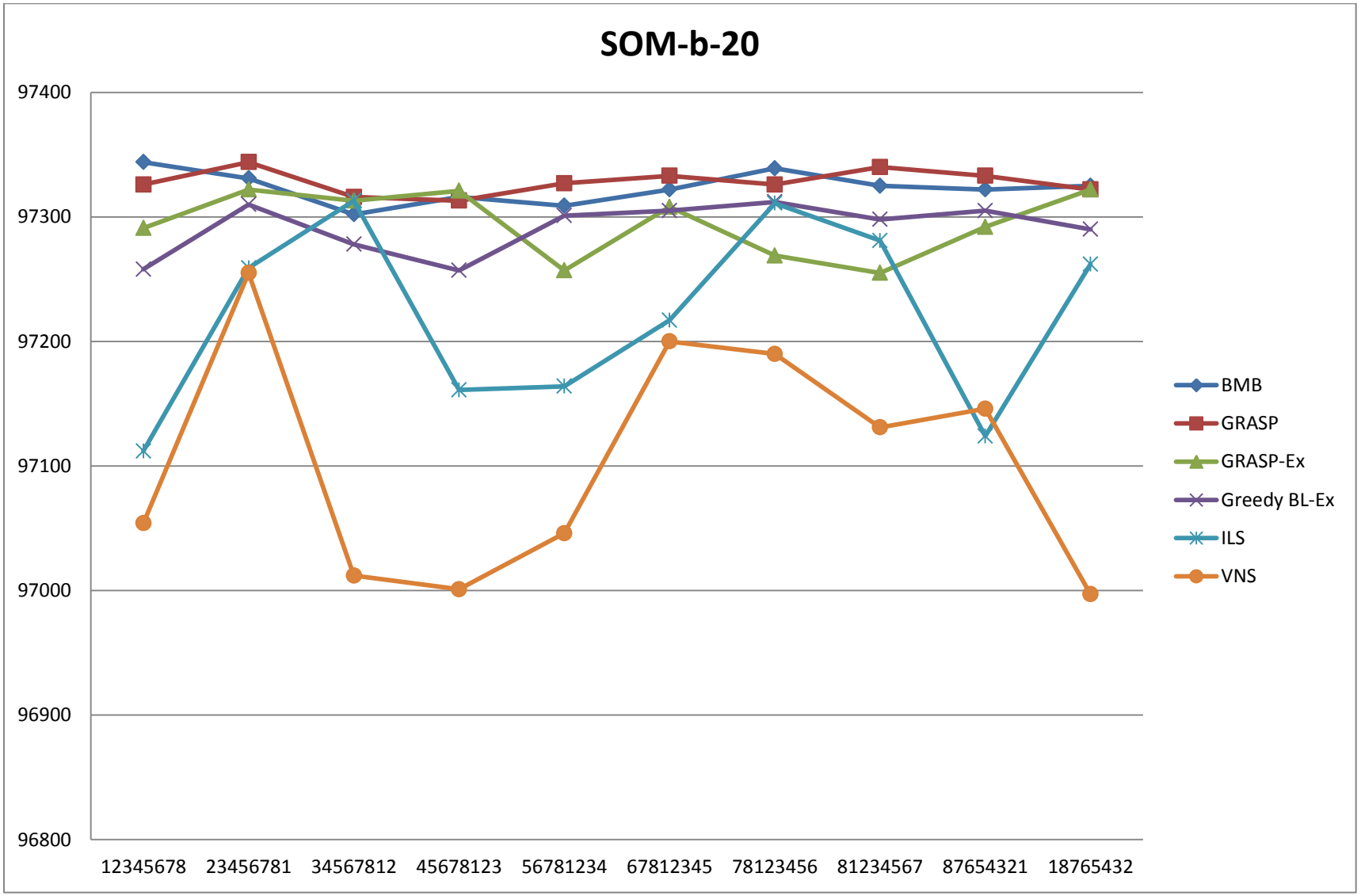
Análisis de resultados

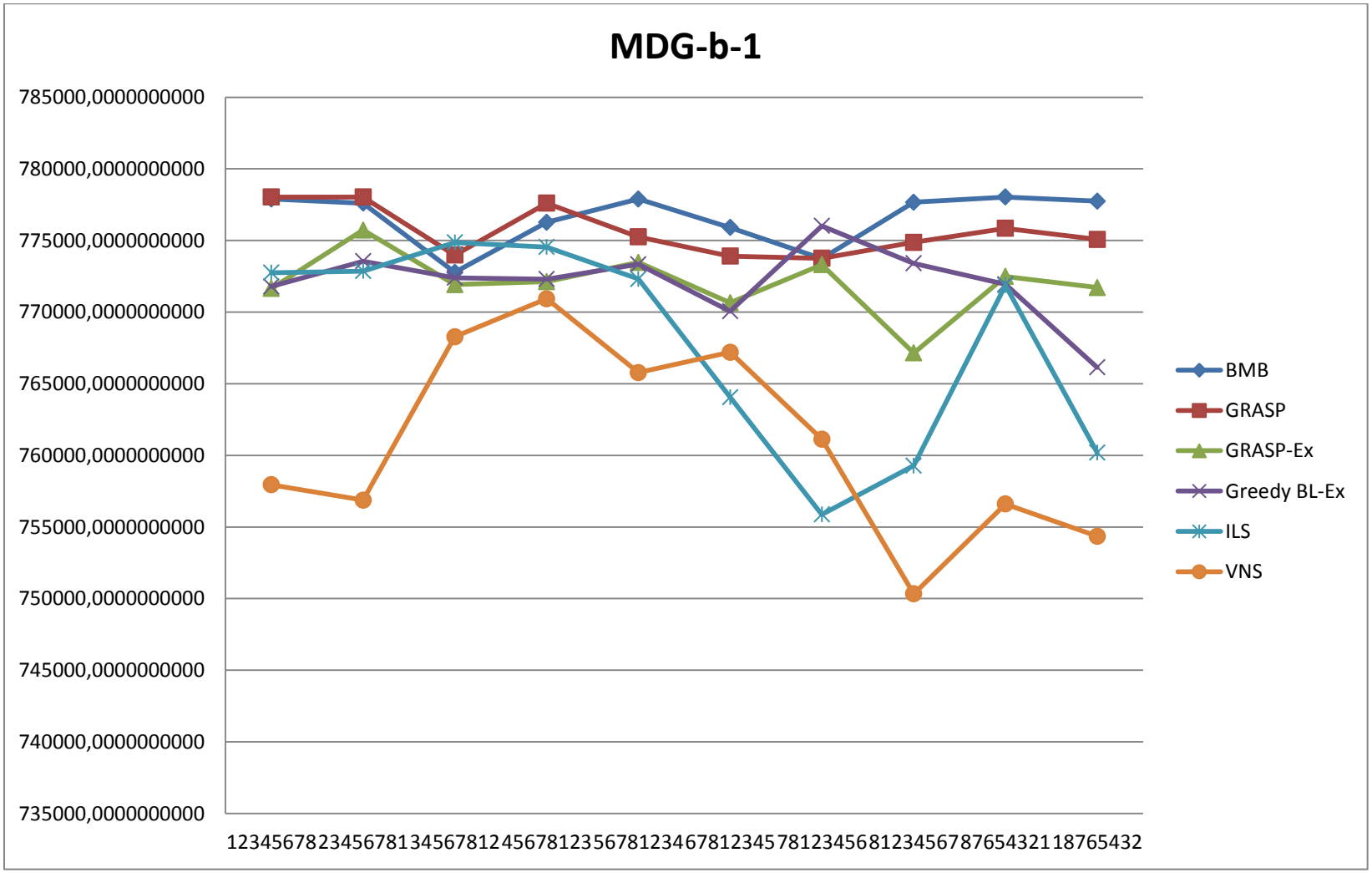
Esta búsqueda da muy malas soluciones porque como las mutaciones va poco a poco se estanca al final en mínimos locales muy malos.

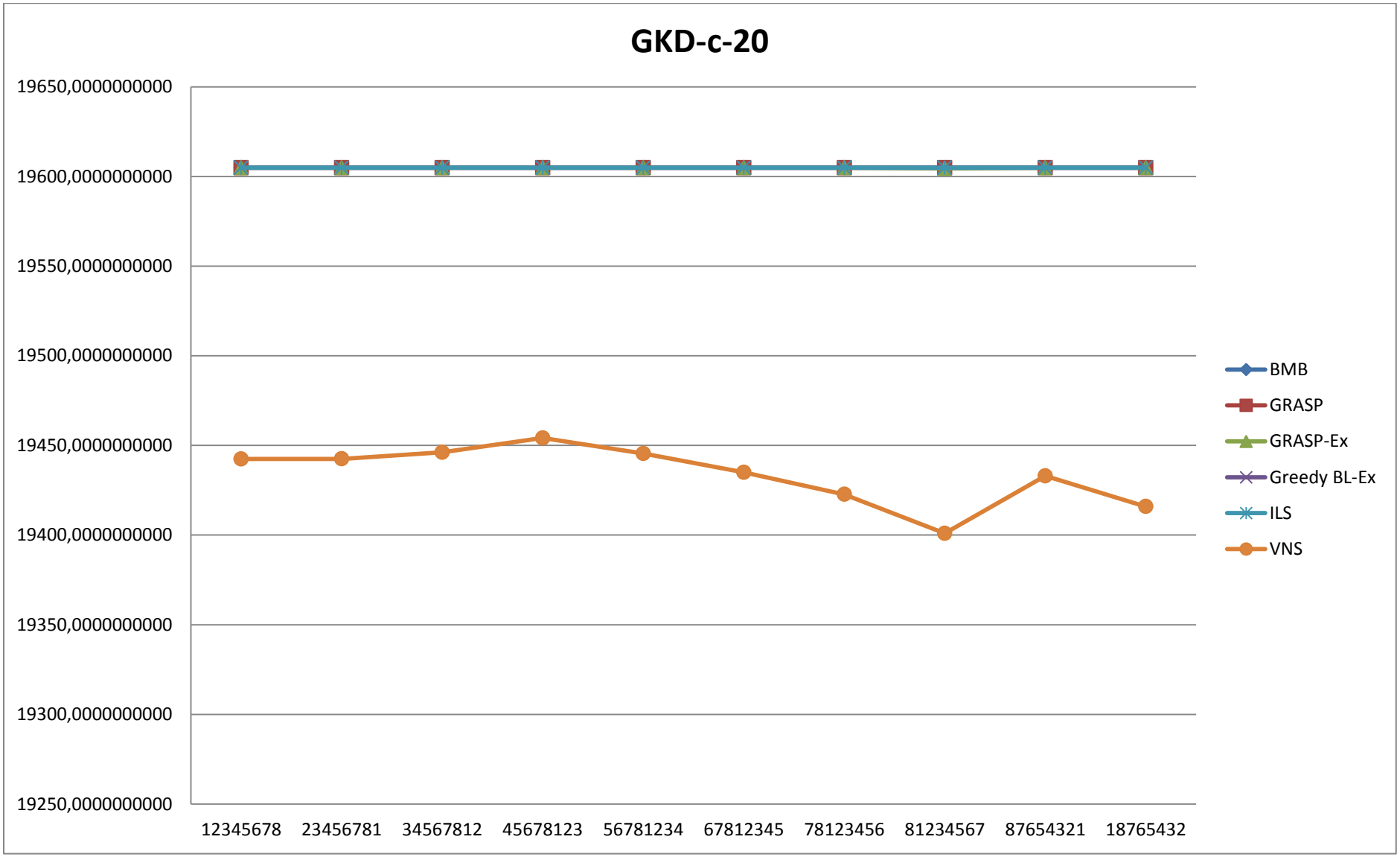
c) Resultados Globales obtenidos

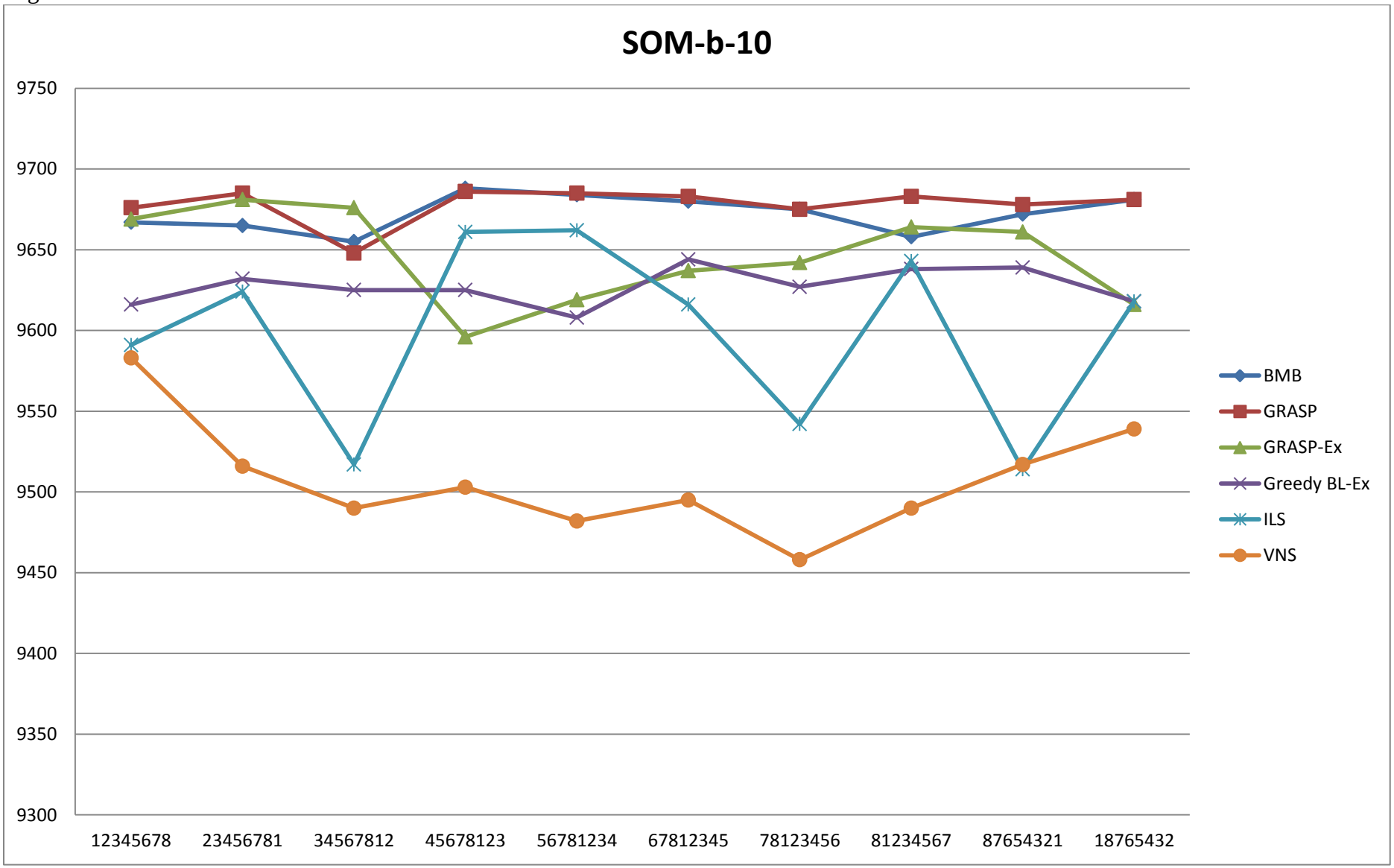
	SOM-b-20				MDG-b-1			
Metodo	mejor	media	peor	σ	m	x	p	σ
Optimo	-	97344	-	-	-	778030,625	-	-
Greedy	-	96499	-	-	-	754034,96	-	-
Greedy+BL	-	97263	-	-	-	760737,5900	-	-
BMB	97344	97323,50	97302	12,0602653	778030,5700000020	776551,66200000	772781,8300000010	1795,554651
GRASP	97344	97328,00	97313	9,29516003	778030,5700000030	775636,8940000000	773756,4100000010	1602,341635
GRASP-EX	97322	97295,00	97255	25,2428208	775728,260000000	772019,567999999	767148,499999999	2087,723026
Greedy+BL-EX	97312	97291,40	97257	19,3814344	776017,929999999	772093,942000000	766141,760000001	2461,9523
ILS	97313	97220,40	97112	71,7219632	774863,300000000	767859,475000000	755872,029999999	6849,040742
VNS	97255	97103,20	96997	88,4791501	770934,650000000	760941,239000000	750328,660000000	6456,218966

SOM-b-10				GKD-c-20			
m	x	p	σ	m	x	p	σ
-	9689	-	-	-	19604,84375	-	-
-	9523	-	-	-	19604,84356	-	-
-	9620	-	-	-	19604,843560	-	-
9688	9672,5	9655	10,538026	19604,8435600000	19604,8435600000	19604,8435600000	3,63798E-12
9686	9678	9648	10,648944	19604,8435600000	19604,8435600000	19604,8435600000	3,63798E-12
9681	9646,1	9596	27,256009	19604,8435600000	19604,8027640000	19604,4356000001	0,122388
9644	9627,2	9608	10,72194	19604,8435600000	19604,8435600000	19604,8435600000	3,63798E-12
9662	9598,8	9514	53,184208	19604,843560000	19604,843560000	19604,843560000	5,09317E-11
9583	9507,3	9458	32,778194	19454,145350000	19433,831830000	19400,962730000	15,41851577

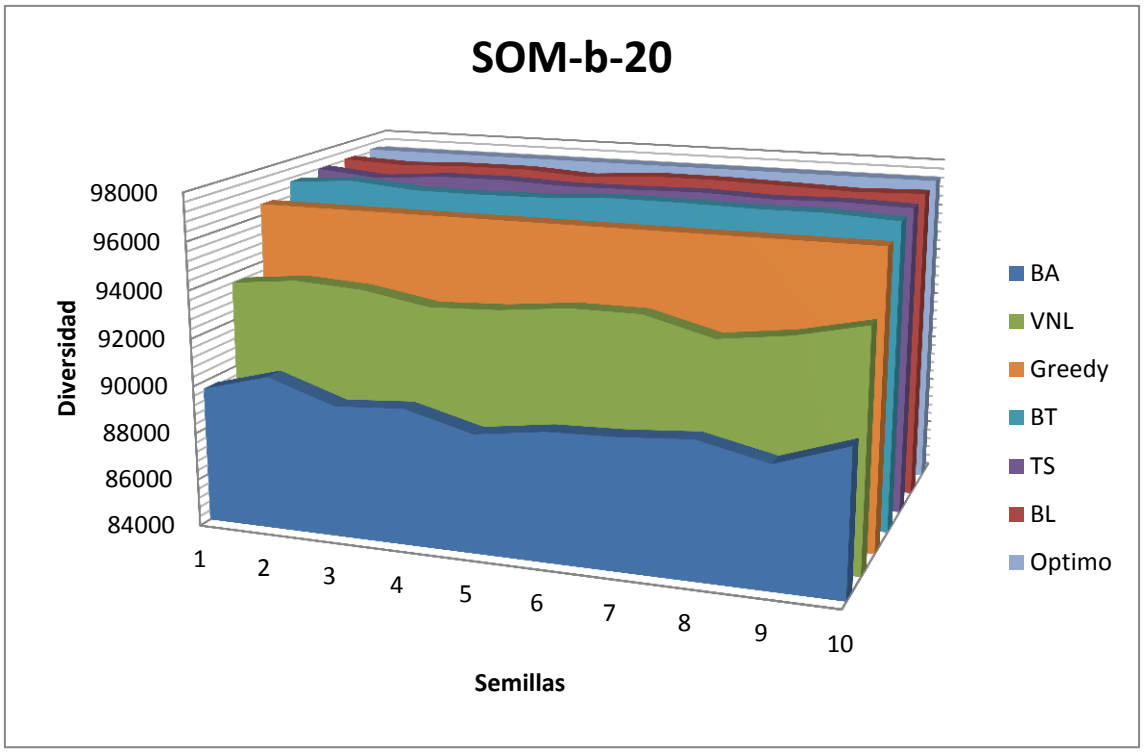

SOM-b-20
97400
97300
97200
97100
97000
96900
96800
12345678
23456781
34567812
45678123
56781234
67812345
78123456
81234567
87654321
18765432
BMB
GRASP
GRASP-Ex
Greedy BL-Ex
ILS
VNS

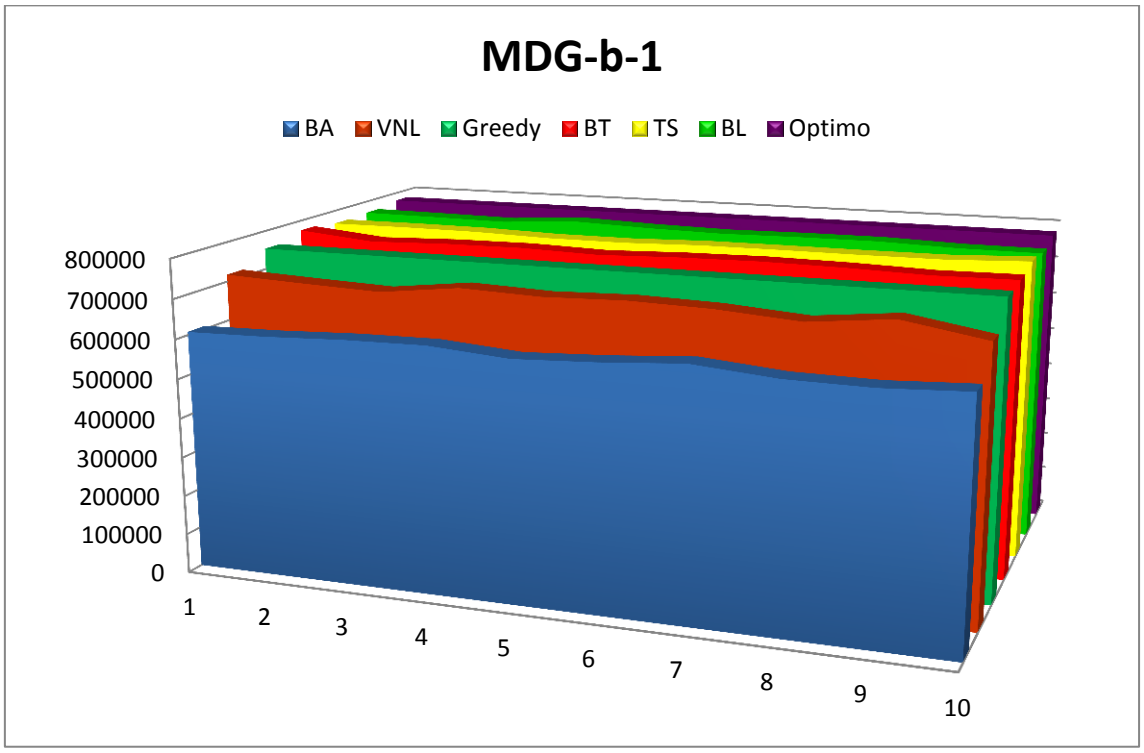

MDG-b-1
785000,0000000000
780000,0000000000
775000,0000000000
770000,0000000000
765000,0000000000
760000,0000000000
755000,0000000000
750000,0000000000
745000,0000000000
740000,0000000000
735000,0000000000
12345678234567813456781245678123567812346781234578123456812345678765432118765432
BMB
GRASP
GRASP-Ex
Greedy BL-Ex
ILS
VNS

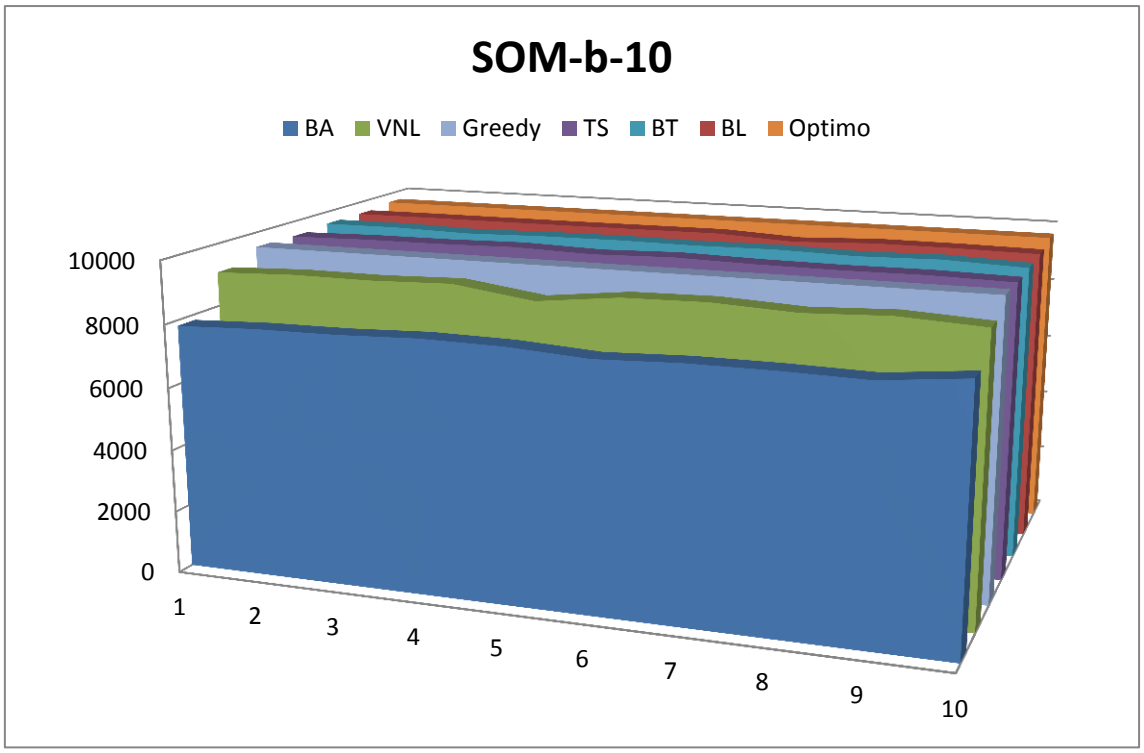

GKD-c-20
19650,0000000000
19600,0000000000
19550,0000000000
19500,0000000000
19450,0000000000
19400,0000000000
19350,0000000000
19300,0000000000
19250,0000000000
12345678
23456781
34567812
45678123
56781234
67812345
78123456
81234567
87654321
18765432
BMB
GRASP
GRASP-Ex
Greedy BL-Ex
ILS
VNS


SOM-b-10
9750
9700
9650
9600
9550
9500
9450
9400
9350
9300
12345678
23456781
34567812
45678123
56781234
67812345
78123456
81234567
87654321
18765432
BMB
GRASP
GRASP-Ex
Greedy BL-Ex
ILS
VNS

d) Graficas de comparación.







En estas graficas podemos ver que los algoritmos toman diferencias cuanto mas grande es el conjunto de datos y mas grande es la solución a elegir es decir:

Diferencia entre las soluciones de los Algoritmos => ↑M ↑N

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 BMB,1 GRASP,2 Greedy más Búsqueda Local,3 GRASP-Ex,4 Greedy +BL-Ex,5 ILS,6 VNS, 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.