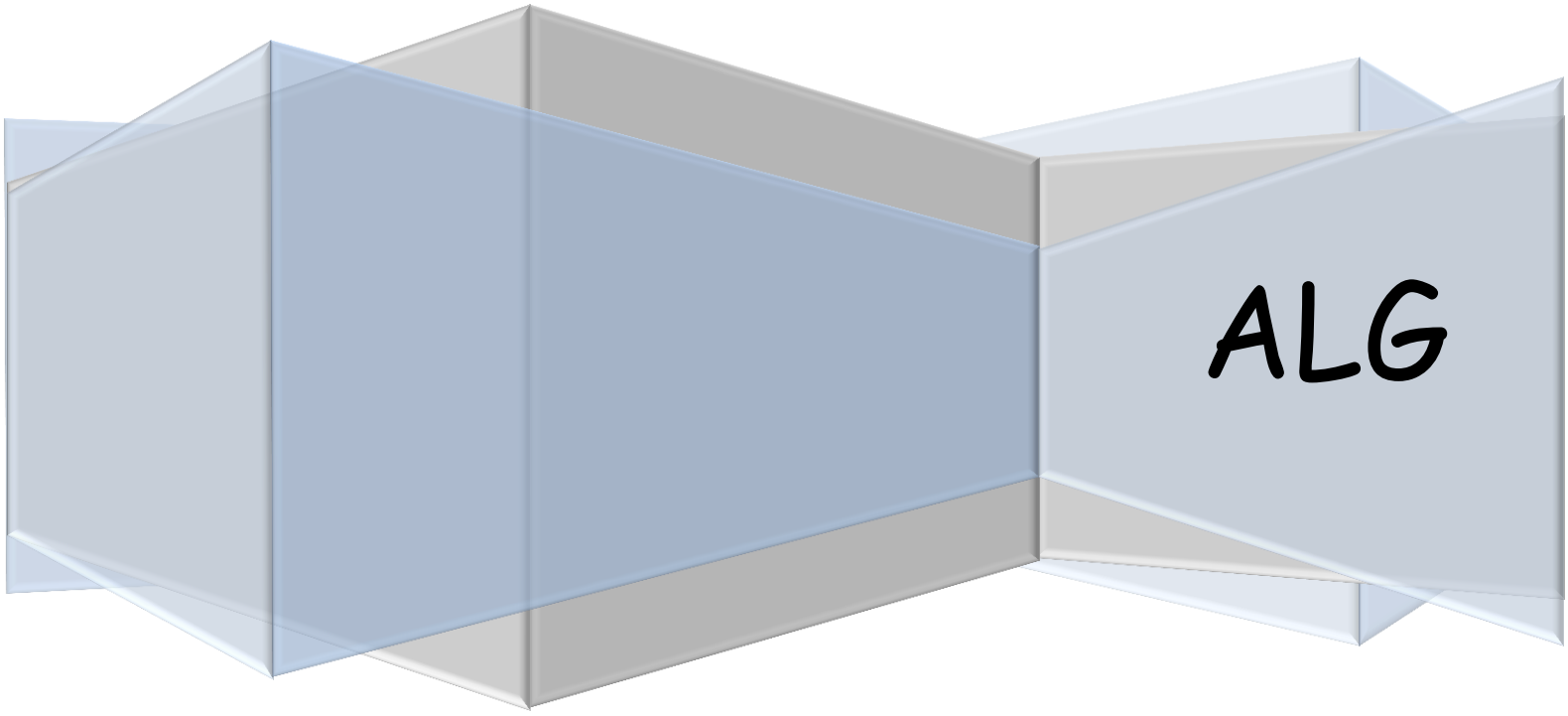
Práctica 1

MDP con búsqueda aleatoria, búsqueda local, VND, enfriamiento simulado y búsqueda tabú

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 en pseudocódigo del operador de vecino considerado.

```
Vecino (){
        NumAleatorio1=GeneraNumAleatorio(0,Lista1.size-1)
        NumAleatorio2=GeneraNumAleatorio(0,Lista0.size-1)
        Lista1.remplazar(Lista1[NumAleatorio1],Lista0[NumAleatorio2])
        Lista0.remplazar(Lista0[NumAleatorio2],Lista1[NumAleatorio1])
}
```

e) Descripción en pseudocódigo Generar Solución aleatoria.

```
GeneraSolAleatria (ListaU,ListaO){
    Desde i=0 hasta i=M{
        NumAleatorio=GeneraNumAleatorio(0,N)
        Mientras(NumAleatorio in Lista1)
            NumAleatorio=GeneraNumAleatorio(0,N)
        ListaU.añadir (NumAleatrorio)
        ListaO.borrar(NumAleatrorio)
    }
}
```

f) Operador Buscar más diverso (Greedy))

```
BuscarMasDiver(self){
        max=0;
        posicionmaxima=0;
        cont=0;
        desde i=0 hasta i=N{
            desde s=0 hasta s=N{
                cont=MatrizDiversidad[s,i]+cont
                si(max<cont){
                    posicionmaxima =i
                    max=cont
                }
            }
        }
    devolver sal
    }
```

g) Operador Buscar mas diverso con respecto a un conjunto (Greedy)

```
Buscar(Lista1,Lista0){
max=0
dese i =0 hasta i=Lista0.size{
        aux=0
        dese s =0 hasta s=Lista1.size {
            aux=self.ma[ListaU[s],ListaC[i]]+aux
        }
        si(max<aux):
            pos=i
            max=aux
}
return [pos,max]
}
```

3. Pseudocódigo

A. *Greedy*

```
Greedy(){
        elemento= BuscarMasDiver
        Lista1.insertar(elemento)
        Lista0.borrar(elemento)
        desde i=0 hasta i= M{
                elemento=Buscar(Lista1,Lista0)
                Lista1.insertar(elemento)
                Lista0.borrar(elemento)
        }
}
```

B. *Búsqueda aleatoria*

```
BA(){
        GenerarSolAleatoria(Lista1,Lista0)
        Max= FunObjetivo()
        ListaU=[]
        ListaO=[0..N]
        desde i=0 hasta i= 100000{
                GenerarSolAleatoria(ListaU,ListaO)
                x= FunObjetivo()
                si(max<x){
                Lista1=ListaU
                Lista0=ListaO
                }
        }
}
```

C. *Búsqueda local primer mejor vecino*

```
def BL(self,semilla):
        GenerarSolAleatoria(Lista1,Lista0)
        max=FunObjetivo(Lista1,Lista0)
        print "sol: "+str(self.max)
        mientras(mejora==True){
                #print "entra"
                mejora=False
                barajar(self.Lista1)
                ListaU=Lista1.clone
                ListaO=Lista0.clone
                desde i hasta i=Lista1.size and mejora==False{
```

```
                    desde i hasta i=Lista1.size and mejora==False{
                        ListaU.insertar(ListaO[j])
                        ListaO.insertar(ListaU[i])
                        ListaO.borrar(ListaO[j])
                        ListaU.borrar(ListaU[i])
                        num=max=FunObjetivo(ListU1,ListaO)
                        if(self.max<max2):
                            mejora=True
                            Lista1=ListaU.clone
                            Lista0=ListaO.clone
                            max=max2
                        else:
                            ListaU=Lista1.clone
                            ListaO=Lista0.clone
                    }
                }
            }
```

D. Búsqueda descendente de entorno variable

```
def BVND(self,semilla):
        GenerarSolAleatoria(Lista1,Lista0)
        k=1
        max=FunOjetivo(Lista1,Lista0)
        N=0
        mientras(k<=3 and N<=100000):
            mejora=False
            ListaU=Lista1.clone
            ListaO=Lista0.clone
            si(k==1):
                Vecindad=int(0.2*self.m*(self.n-self.m))
                s=GeneroNumAletorio(0, (ListaU.size)-1)
                desde j hasta(j=Vecindad):
                    ss=GeneroNumAletorio(0, (ListaO.size)-1)
                    ListaU.insertar(ListaO[s])
                    ListaO.insertar(ListaU[ss])
                    ListaO.borrar(ListaO[s])
                    ListaU.borrar(ListaU[ss])
                    num=FunObjetivo()
                    si(max<num):
                        CambiamosSolActual(ListaU,ListaO)
                    sino
                        ListaU=Lista1.clone
                        ListaO=Lista0.clone
                N=N+Vecindad
            si(k==2):
                Vecindad=int(0.5*self.m*(self.n-self.m))
                Genero2NumAleatorio()
                desde j hasta(j=Vecindad):
```

Generar Vecino

```
                    Genero2NumAleatorio()
                    Para los 2 numeros:                          ┐
                        ListaU.insertar(ListaO[s])               │
                        ListaO.insertar(ListaU[ss])              │  Generar vecino 2
                        ListaO.borrar(ListaO[s])                 │
                        ListaU.borrar(ListaU[ss])                ┘
                    num=FunOjetivo(ListaU)
                    si(max<num):
                        mejora=True
                        Lista0=ListaO.clone
                        Lista1=ListaU.clone
                        max=num
                    sino:
                        ListaU=Lista1.clone
                        ListaO=Lista0.clone
            N=N+Vecindad
    si(k==3):
            Vecindad=int(self.m*(self.n-self.m))
            Genero3NumAleatorio()
            desde j hasta(j=Vecindad):
                    Genero3NumAleatorio()
                    Para los 3 numeros:                          ┐
                        ListaU.insertar(ListaO[s])               │
                        ListaO.insertar(ListaU[ss])              │  Generar vecino 3
                        ListaO.borrar(ListaO[s])                 │
                        ListaU.borrar(ListaU[ss])                ┘
                    num=FunOjetivo(ListaU)
                    si(max<num):
                        mejora=True
                        Lista0=ListaO.clone
                        Lista1=ListaU.clone
                        max=num
                    sino:
                        ListaU=Lista1.clone
                        ListaO=Lista0.clone
            N=N+Vecindad
    si(mejora==False):
            k=k+1
            mejora=True
```

E. Enfriamiento simulado

```
Bmax=∑[desde i=0 hasta n-2] ∑[desde j=i+1 hasta n-1] d[i][j]
tfinal=1/Bmax
Lista1=GeneraSolAleatoria()
T=0,3/-log(0,3)*(C(Lista1))
β=(T-tfinal)/(333*T*tfinal)
desde 0 a 333{
  desde 0 a 300{
        ListaU=GeneraVecino(Lista1)
        ∂=C(ListaU)-C(Lista1)
        Si(GeneroNumAleatorio (0,1)< e^(-∂/T) o ∂<0){
        Lista1 = ListaU
        }
  }
  T=T/(1+ β*T)
}
```

```
CalBmax():
        Bmax=0
        Desde i=0 hasta n-2:
            Desde s=i+1 hasta n-1:
                Bmax=MatrizDiversidad[i,s]+Bmax

        devolver Bmax
```

F. Búsqueda Tabú

```
BT(){
  ListaTabu=vacia
  TamTabu=50,
  Frecuencias [0..n]=0;
  GeneraSolAletoria(Lista1)

  desde i=0, hasta i=19{
        aleatorio= GeneraNumAleatorio(0,1);
        Si(aleatorio<0.25){
            GeneraSolAleatoria(Lista1);
        }Sino Si(aleatorio<0.5){
            Lista1=global;
        }Sino{
            Lista1=[];
```

```
            Mientras(Lista1.size() != M){
                aleatorio= GeneraNumAleatorio(0,1);
                punto= PuntoAlAzar(ceros);
            Si(num_aleatorio < (1-Frecuencias[punto])/(SolAceptadas)){
                Lista1=Lista1 U {punto};
                }
            }
        }
        Repetir(desde j=0, hasta j<5000){
            ListaU=GeneraVecino(Lista1);
            Si(ListaU no in ListaTabu or Obj(ListaU)>Obj(global)){
                SolAceptadas++;
                Si(Obj(ListaU) > Obj(MejorVec)){
                    MejorVec=Scand;
                    Si(Obj(ListaU) < Obj(global)){
                                global=ListaU;
                    }
                }
                Si(ListaTAbu.size ¡= TAmListaTAbu)
                    ListaTabu=ListaTabuU {ListaU};
                Sino
                   ListaTabu.Reemplaza(mas_antiguo, ListaU);
                   Frecuencias[Punto(ListaU)]=Frecuencias[Punto(ListaU)]+1;
            }
            Si(j%300 == 0){
                Sact=MejorVec;
            }
        }

        lista_tabu=[]
        Si(GeneraNumAleatorio(0,1) <0.5){
            TamTabu=TamTabu-0.25*TamTabu;
        }Sino{
            TamTabu=TamTabu+0.25*TamTabu;
        }
}
```

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.

b) Resultados para cada problema.

1. Greedy

	SOM-b-10	MDG-b-1	SOM-b-20
Greedy	9523	754034,96	96499
Optimo	9689	778030,625	97344

Análisis de los resultados:

Los algoritmos Greedy generalmente no tienen muy buenos resultados, pero para el MDP, la heurística de escoger el punto más diverso al subconjunto ya escogido es una buena heurística. Por ello los resultados están muy cercanos al óptimo.

2. Búsqueda Aleatoria

	SOM-b-20	MDG-b-1	SOM-b-10
	BA	BA	BA
Semillas	Resultados	Resultados	Resultados
12345678	89705	607934,15	7826
23456781	90482	611755,22	7919
34567812	89530	619709,24	7942
45678123	89737	622449,79	8032
56781234	88983	606892,68	7996
67812345	89394	615695,45	7835
78123456	89493	629632,3	7931
81234567	89719	611945,11	7930
87654321	89075	609491,19	7881
18765432	90087	619180,97	8155
Optimo	97344	778030,625	9689

Análisis de resultados

Esta búsqueda da malos resultados, entre ellos algunos más cerca del óptimo que otros es porque aleatoriamente ha tenido más suerte.

3. Búsqueda Local primer mejor vecino

	SOM-b-20	MDG-b-1	SOM-b-10
	BL	BL	BL
Semillas	Resultados	Resultados	Resultados
12345678	97273	768142,13	9592
23456781	97153	766386,11	9601
34567812	97258	763946,73	9598
45678123	97268	776378,48	9599
56781234	97094	769215,64	9608
67812345	97295	765377,44	9616
78123456	97286	767888,65	9482
81234567	97215	768457,88	9581
87654321	97130	762175,77	9572
18765432	97218	761295,8	9532
Optimo	97344	778030,625	9689

Análisis de resultados

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

4. Búsqueda Descendente de Entorno Variable

	SOM-b-20	MDG-b-1	SOM-b-10
	BVND	BVND	BVND
Semillas	Resultados	Resultados	Resultados
12345678	93619	717772,59	9090
23456781	93922	709012,4	9178
34567812	93762	701401,67	9166
45678123	93275	725263,59	9240
56781234	93395	717581,63	8884
67812345	93708	723053,58	9172
78123456	93714	717480,87	9216
81234567	92975	702501,09	9088
87654321	93363	723920,03	9191
18765432	94017	689333,26	9053
Optimo	97344	778030,625	9689

Análisis de resultados

Esta búsqueda no da buenos resultados, se queda pronto en máximos locales que no son muy buenos.

5. Enfriamiento simulado.

	SOM-b-20	MDG-b-1	SOM-b-10
	TS	TS	TS
Semillas	Resultados	Resultados	Resultados
12345678	97220	764570,7	9503
23456781	96961	765330,1	9551
34567812	97192	764240	9558
45678123	97213	758780,3	9603
56781234	97101	758378,9	9522
67812345	97130	763582,7	9586
78123456	97187	763814,2	9525
81234567	97077	763244,9	9502
87654321	97142	764054,6	9516
18765432	97064	772215,3	9463
Optimo	97344	778030,625	9689

Análisis de resultados

Esta búsqueda da muy buenos resultados la temperatura desciende poco a poco y así podemos encontrar muy buenos máximos locales.

6. Búsqueda Tabú

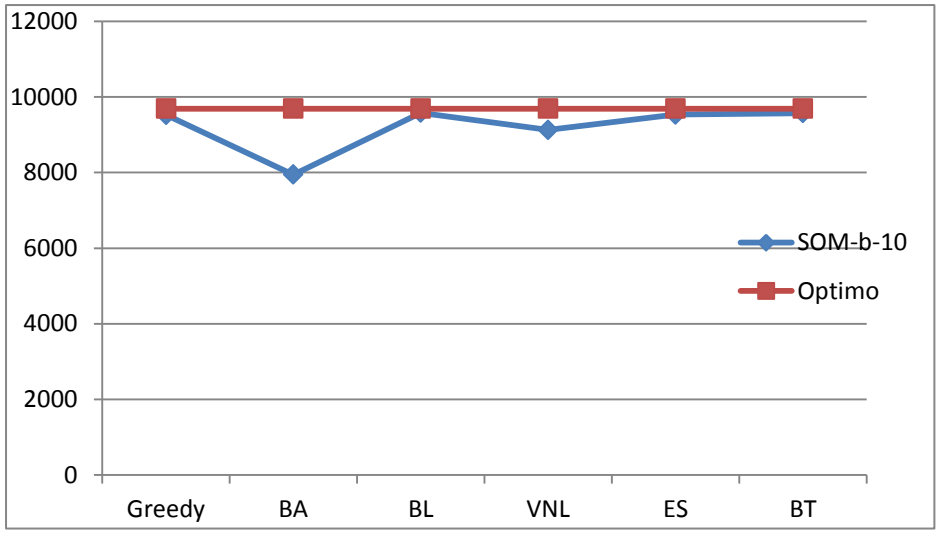
	SOM-b-20	MDG-b-1	SOM-b-10
	BT	BT	BT
Semillas	Resultados	Resultados	Resultados
12345678	97054	773927,1	9589
23456781	97255	755256,7	9598
34567812	97012	761283,4	9537
45678123	97001	762694,9	9592
56781234	97046	756876,8	9606
67812345	97200	763107,8	9579
78123456	97190	765566,4	9604
81234567	97131	763012,1	9541
87654321	97146	757189,1	9595
18765432	96997	759264,6	9489
Optimo	97344	778030,625	9689

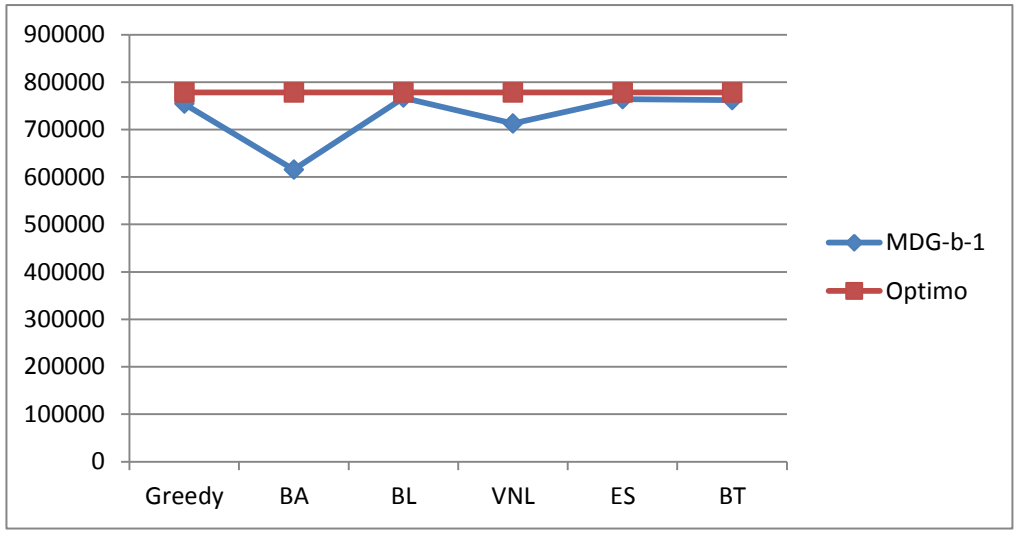
Análisis de resultados

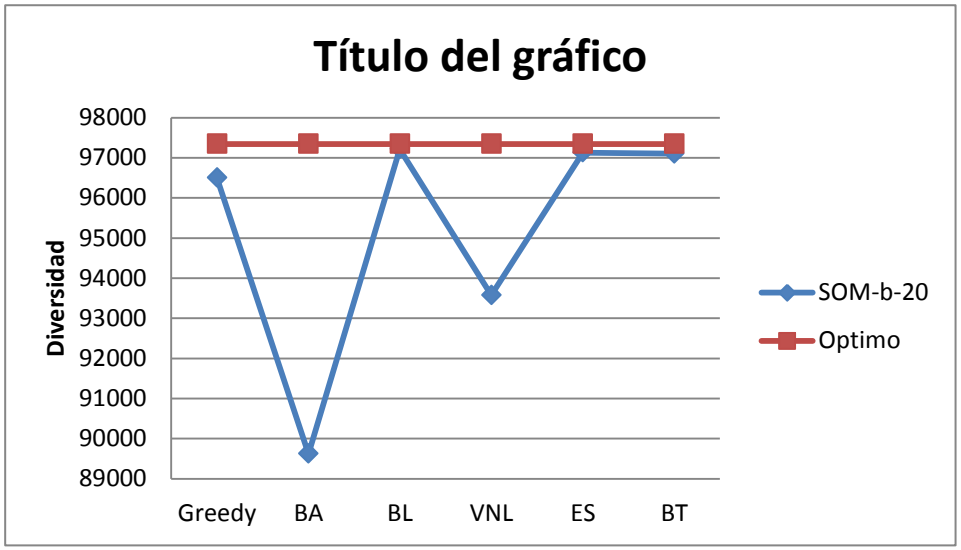
Esta búsqueda da buenos resultados aunque no intensifica lo suficiente la búsqueda si pusiéramos mas de 5000 ejecuciones podría mejorarnos.

c) Resultados Globales obtenidos

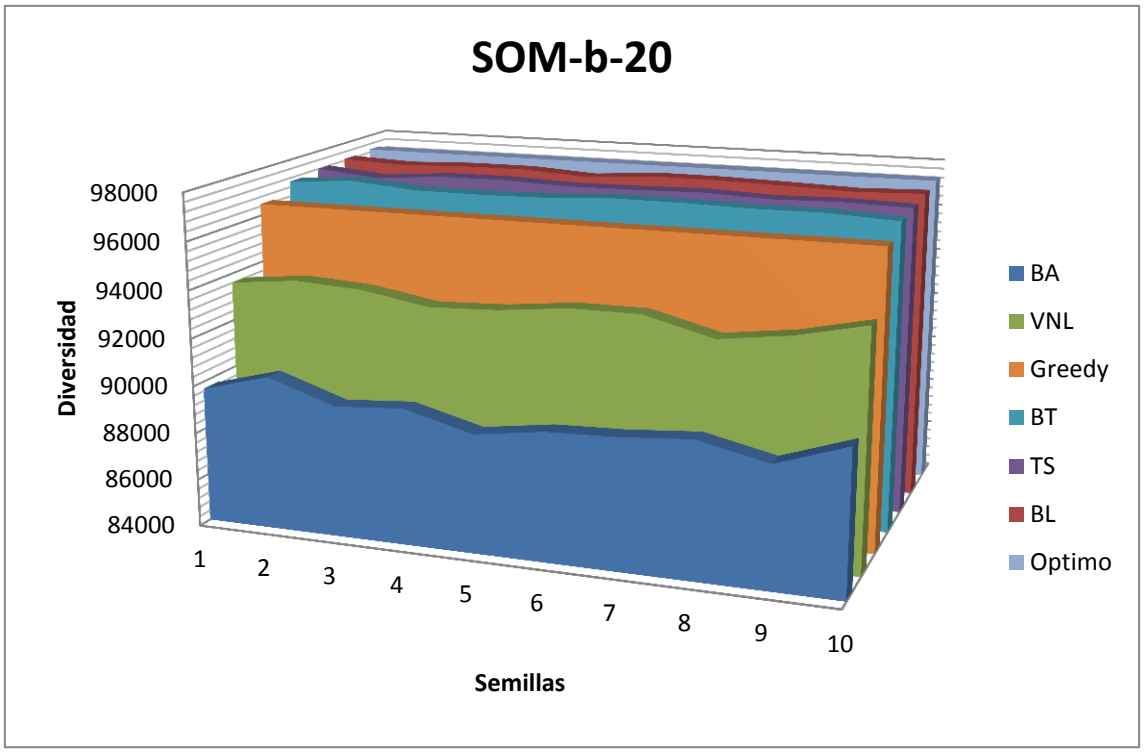
	SOM-b-20				MDG-b-1				SOM-b-10			
Método	mejor	media	peor	σ	m	x	p	σ	m	x	p	σ
Optimo	-	97344	-	-	-	778030,625	-	-	-	9689	-	-
Greedy	-	96499	-	-	-	754034,96	-	-	-	9523	-	-
BA	90482	89620,5	88983	420,707083	629632,3	615468,61	606892,68	6880,49079	8155	7944,7	7826	92,59163
BL	97295	97219	97094	67,0984352	776378,48	766926,463	761295,8	4064,15498	9616	9578,1	9482	39,098465
VNL	94017	93575	92975	302,825362	725263,59	712732,071	689333,26	11200,9391	9240	9127,8	8884	99,167333
ES	97220	97128,7	96961	76,6655725	772215,25	763821,157	758378,93	3590,9151	9603	9532,9	9463	39,8659
BT	97255	97103,2	96997	88,4791501	773927,07	761817,884	755256,73	5101,34744	9606	9573	9489	36,315286

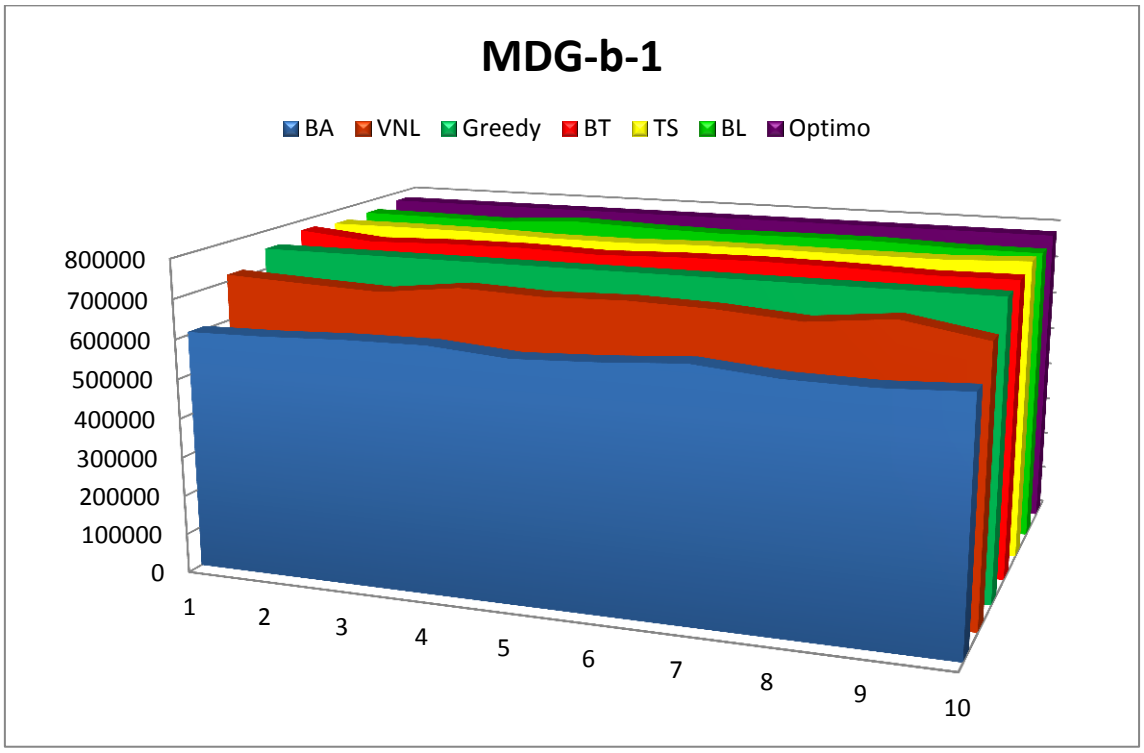

12000
10000
8000
6000
4000
2000
0
Greedy
BA
BL
VNL
ES
BT
SOM-b-10
Optimo

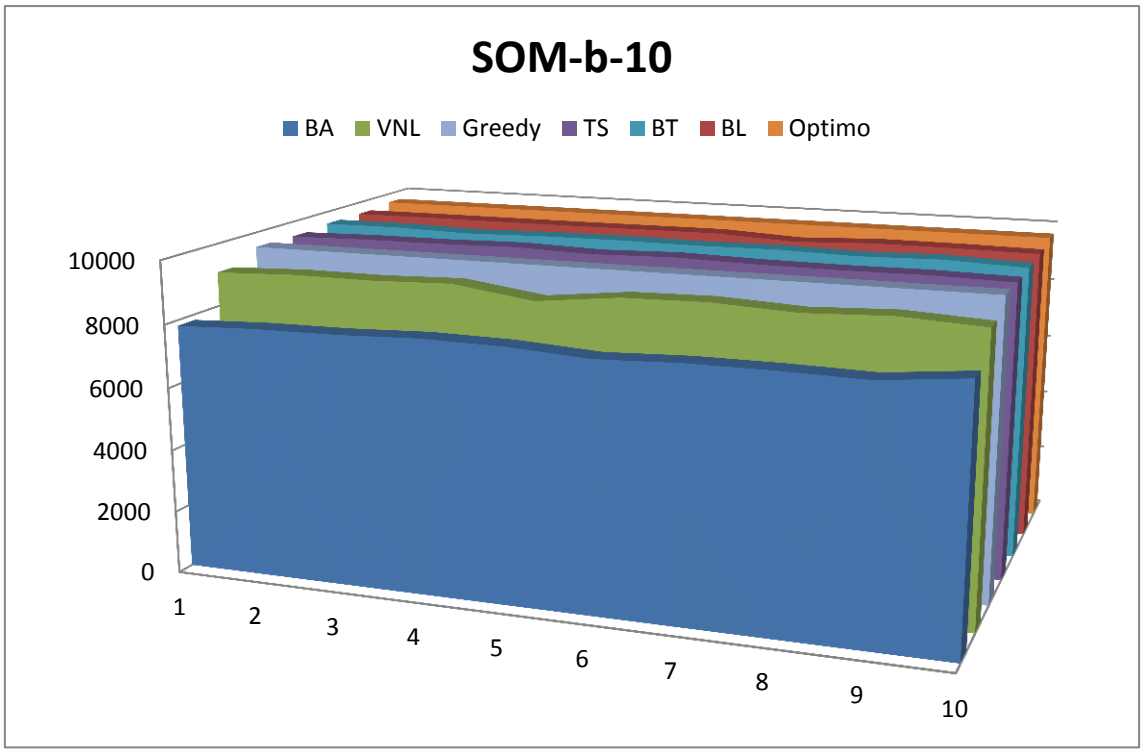

900000
800000
700000
600000
500000
400000
300000
200000
100000
0
Greedy
BA
BL
VNL
ES
BT
MDG-b-1
Optimo


Título del gráfico
Diversidad
98000
97000
96000
95000
94000
93000
92000
91000
90000
89000
Greedy
BA
BL
VNL
ES
BT
SOM-b-20
Optimo

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)
    - Tema 1 Metaheurísticas: Introducción y Clasificación
    - Tema 2 Algoritmos de Búsqueda Local Básicos
    - Tema 3 Algoritmos de Enfriamiento Simulado

6. Manual de usuario

El programa se puede ejecutar de dos maneras si esta instalado python solo abriendo el archivo, y sino el ejecutable esta dentro de bin\dist.

El programa funciona leyendo de un fichero llamado datos.txt, en el cual detemos poner Numero del algoritmo:0 greedy,1 BA,2 BL,3 BVND,4ES,5 BT, y después podremos espacio, numero de semillas a utilizar , y espacio y número de ficheros , y después las semillascon un \n al final y los nombres de archivo igual pero el ultimo sin el \n.