MAX MEAN DISPERSION PROBLEM

DISEÑO Y ANÁLISIS DE ALGORITMOS

Resumen del documento

Informe en el que se redacta brevemente los cinco algoritmos utilizados para la resolución del Max-Mean Dispersion Problem. Se presentan también los pseudocódigos y las partes del código más importantes, además de los resultados para varias muestras de cada uno de ellos.

Personas Implicadas

- **Profesora de la asignatura:** Belén Melián Batista.
- Eduardo Escobar Alberto (alu0100825985@ull.edu.es).

Control de versiones del documento

VERSIÓN	**1.0**	02 de mayo de 2017
REDACTOR	Eduardo Escobar Alberto	
REVISOR	Belén Melián Batista	

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES Y TABLAS

ILUSTRACIONES

TABLAS

ALGORITMO CONSTRUCTIVO VORAZ

Para este algoritmo se ha llevado a cabo la implementación del pseudocódigo propuesto en el enunciado de la práctica. En él se crea un subconjunto S y un subconjunto S auxiliar, ambos inicialmente vacíos. Luego se les añade la arista con mayor afinidad. Seguidamente, en cada iteración se obtiene un vértice candidato a maximizar y se añade al subconjunto S auxiliar. Si la dispersión media comparada con el subconjunto S aumenta, se añade definitivamente a la solución. Este proceso se realiza hasta que no mejore dicha solución.

```
1   Seleccionar la arista (i, j) con mayor afinidad;
2   S = {i,j};
3   repeat
4       S* = S;
5       Obtener el vértice k que maximiza md(S ∪ {k});
6       if md(S ∪ {k}) ≥ md(S) then
7           S = S ∪ {k};
8   until (S* = S)
9   Devolver S*;
```

Ilustración 1: Pseudocódigo Algoritmo Constructivo Voraz

```
1   @Override
2   public ArrayList<Integer> resolverProblema() {
3       Integer verticeCandidatoMaximizar;
4       ArrayList<Integer> subconjuntoS = new ArrayList<Integer>();
5       ArrayList<Integer> subconjuntoSAuxiliar = new ArrayList<Integer>();
6       ArrayList<Integer> subconjuntoSCandidato = new ArrayList<Integer>();
7       insertarAristaMayorAfinidad(subconjuntoS);
8       do {
9           subconjuntoSAuxiliar = new ArrayList<Integer>(subconjuntoS);
10          subconjuntoSCandidato = new ArrayList<Integer>(subconjuntoS);
11          verticeCandidatoMaximizar = obtenerVerticeCandidatoMaximizar(subconjuntoS);
12          subconjuntoSCandidato.add(verticeCandidatoMaximizar);
13          if (calcularDispersionMedia(subconjuntoSCandidato) > calcularDispersionMedia(subconjuntoS)) {
14              subconjuntoS.add(verticeCandidatoMaximizar);
15          }
16      } while(!subconjuntoSAuxiliar.equals(subconjuntoS));
17      return subconjuntoS;
18  }
```

Ilustración 2: Código Algoritmo Constructivo Voraz

Como observación, destacar el uso de un subconjunto más (subconjuntoSCandidato), ya que para realizar el cálculo de la dispersión media no podemos usar S U {k} como vemos en el pseudocódigo sin alterar el contenido de S.

MUESTRA DE 10 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	7.90	71.0	10.142857	0.97460
2	7.90	71.0	10.142857	0.96580
3	7.90	71.0	10.142857	0.97147
4	7.90	71.0	10.142857	0.97288
5	7.90	71.0	10.142857	0.98218

MUESTRA DE 15 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-8.26	38.0	9.50	0.65784
2	-8.26	38.0	9.50	0.62008
3	-8.26	38.0	9.50	0.56128
4	-8.26	38.0	9.50	0.56293
5	-8.26	38.0	9.50	0.62940

MUESTRA DE 20 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-4.90	90.0	12.877143	1.09085
2	-4.90	90.0	12.877143	1.03959
3	-4.90	90.0	12.877143	1.06208
4	-4.90	90.0	12.877143	1.03010
5	-4.90	90.0	12.877143	1.00414

MUESTRA DE 25 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-2.24	82.0	11.714286	1.35053
2	-2.24	82.0	11.714286	1.54602
3	-2.24	82.0	11.714286	1.23557
4	-2.24	82.0	11.714286	1.78749
5	-2.24	82.0	11.714286	1.76218

ALGORITMO DESTRUCTIVO VORAZ

Como algoritmo voraz alternativo al propuesto en el guión en de la práctica, se ha llevado a cabo un destructivo voraz. La implementación del mismo es muy similar al constructivo, teniendo como principal diferencia que al principio en el subconjunto S tenemos todos los vértices del grafo inicial. Posteriormente, en cada iteración se irá quitando el vértice que menos afinidad aporta y se comprueba la dispersión media, actualizando en caso de que mejore dicho valor.

```
1  S = {Todos los vértices del grafo}
2  repeat
3      S* = S;
4      Obtener el vértice k de menor afinidad md(S - {k});
5      if md(S - {k}) ≥ md(S) then
6          S = S - {k};
7  until (S* = S)
8  Devolver S*;
```

Ilustración 3: Pseudocódigo Algoritmo Destructivo Voraz

```
1  @Override
2  public ArrayList<Integer> resolverProblema() {
3      Integer verticeCandidatoMaximizar = null;
4      ArrayList<Integer> subconjuntoS = new ArrayList<Integer>();
5      ArrayList<Integer> subconjuntoSAuxiliar = new ArrayList<Integer>();
6      ArrayList<Integer> subconjuntoSCandidato = new ArrayList<Integer>();
7      insertarTodosVertices(subconjuntoS);
8      do {
9          subconjuntoSAuxiliar = new ArrayList<Integer>(subconjuntoS);
10         subconjuntoSCandidato = new ArrayList<Integer>(subconjuntoS);
11         verticeCandidatoMaximizar = obtenerVerticeCandidatoMaximizar(subconjuntoS);
12         subconjuntoSCandidato.remove(verticeCandidatoMaximizar);
13         if (calcularDispersionMedia(subconjuntoSCandidato) > calcularDispersionMedia(subconjuntoS)) {
14             subconjuntoS.remove(verticeCandidatoMaximizar);
15         }
16     } while(!subconjuntoSAuxiliar.equals(subconjuntoS));
17     return subconjuntoS;
18 }
```

Ilustración 3: Código Algoritmo Destructivo Voraz

Observamos que la función para obtener el vértice de menor afinidad tiene como nombre obtenerVerticeCandidatoMaximizar, pudiendo llevar a confusión, pero es que es esa su auténtica utilidad, obtener el vértice que al ser extraído del subconjunto S maximice la dispersión media.

MUESTRA DE 10 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	7.90	84.0	14.0	1.27661
2	7.90	84.0	14.0	1.62494
3	7.90	84.0	14.0	1.71977
4	7.90	84.0	14.0	1.27617
5	7.90	84.0	14.0	1.35453

MUESTRA DE 15 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-8.26	59.0	9.833333	1.96844
2	-8.26	59.0	9.833333	1.58252
3	-8.26	59.0	9.833333	2.31561
4	-8.26	59.0	9.833333	1.68176
5	-8.26	59.0	9.833333	2.52276

MUESTRA DE 20 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-4.90	85.0	12.142858	2.92043
2	-4.90	85.0	12.142858	2.68844
3	-4.90	85.0	12.142858	2.93522
4	-4.90	85.0	12.142858	3.76292
5	-4.90	85.0	12.142858	2.64338

MUESTRA DE 25 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-2.24	129.0	11.727272	4.10423
2	-2.24	129.0	11.727272	5.43276
3	-2.24	129.0	11.727272	5.82683
4	-2.24	129.0	11.727272	5.32210
5	-2.24	129.0	11.727272	4.94451

ALGORITMO GRASP

El algoritmo GRASP implementado es constructivo, es decir, está basado en el algoritmo voraz constructivo. Desarrolla el mismo proceso con dos diferencias principales y distintivas. La primera, es que se hace uso de una **lista de candidatos** en el proceso de obtener el vértice que maximiza. Para ello se obtienen tres posibles candidatos y se elige uno aleatoriamente. En segundo lugar, cabe destacar que a la solución obtenida se le realiza una **búsqueda local.** Finalmente obtendremos un subconjunto S con la solución.

```
1   Seleccionar la arista (i, j) con mayor afinidad;
2   S = {i, j}
3   repeat
4       S* = S;
5       Obtener el vértice k aleatorio de la LRC que maximiza md(S - {k});
6       if md(S - {k}) ≥ md(S) then
7           S = S - {k};
8   until (S* = S)
9   Realizar búsqueda local sobre S*;
10  Devolver S*;
```

Ilustración 5: Pseudocódigo Algoritmo GRASP

```
1   @Override
2   public ArrayList<Integer> resolverProblema() {
3       ArrayList<Integer> subconjuntoS = new ArrayList<Integer>(obtenerSolucionGRASP());
4       realizarBusquedaLocal(subconjuntoS);
5       return subconjuntoS;
6   }
7
8   public ArrayList<Integer> obtenerSolucionGRASP() {
9       Integer verticeCandidatoMaximizar;
10      ArrayList<Integer> subconjuntoS = new ArrayList<Integer>();
11      ArrayList<Integer> subconjuntoSAuxiliar = new ArrayList<Integer>();
12      ArrayList<Integer> subconjuntoSCandidato = new ArrayList<Integer>();
13      insertarAristaMayorAfinidad(subconjuntoS);
14      do {
15          subconjuntoSAuxiliar = new ArrayList<Integer>(subconjuntoS);
16          subconjuntoSCandidato = new ArrayList<Integer>(subconjuntoS);
17          verticeCandidatoMaximizar = obtenerVerticeCandidatoMaximizar(subconjuntoS);
18          subconjuntoSCandidato.add(verticeCandidatoMaximizar);
19          if (calcularDispersionMedia(subconjuntoSCandidato) > calcularDispersionMedia(subconjuntoS)) {
20              subconjuntoS.add(verticeCandidatoMaximizar);
21          }
22      } while(!subconjuntoSAuxiliar.equals(subconjuntoS));
23      return subconjuntoS;
24  }
```

Ilustración 6: Código Algoritmo GRASP

```
1   Random random = new Random();
2   int indiceAleatorio = (int)(random.nextDouble() * verticesCandidatos.size());
3   return new Integer(verticesCandidatos.get(indiceAleatorio));
```

Ilustración 7: Obtención aleatoria de la LRC

MUESTRA DE 10 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	7.90	22.0	7.333333	1.04582
2	7.90	22.0	7.333333	1.04013
3	7.90	10.0	5.0	0.82202
4	7.90	10.0	5.0	0.89856
5	7.90	10.0	5.0	1.35150

MUESTRA DE 15 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-8.26	46.0	9.20	1.30059
2	-8.26	45.0	9.0	1.27880
3	-8.26	23.0	7.666666	1.11736
4	-8.26	23.0	7.666666	1.08066
5	-8.26	28.0	7.0	1.56504

MUESTRA DE 20 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-4.90	90.0	12.857142	1.90933
2	-4.90	71.0	10.142857	1.86922
3	-4.90	77.0	12.833333	1.69215
4	-4.90	62.0	12.4	1.50639
5	-4.90	77.0	12.833333	1.59688

MUESTRA DE 25 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-2.24	26.0	8.666666	1.50730
2	-2.24	93.0	13.285714	2.49236
3	-2.24	99.0	12.375	3.52738
4	-2.24	22.0	7.333333	1.32050
5	-2.24	98.0	12.25	3.86482

ALGORITMO MULTIARRANQUE

El multiarranque, tal y como su nombre indica, consiste en empezar el algoritmo una determinada cantidad de veces determinada por una constante, la cual está establecida como 5. En este caso el algoritmo usado hace uso de una solución inicialmente obtenida por un GRASP como punto de partida. Seguidamente, realizamos una **búsqueda local** sobre dicha solución y se compara con las otras 5. Finalmente nos quedamos con la que mejor dispersión media nos ofrezca.

```
1   Procedure Búsqueda con Arranque Múltiple Begin
2       Genera (Solución Actual);
3       Mejor Solución := Solución Actual;
4       Repeat
5           Búsqueda Local(Solución Actual);
6           If Objetivo(Solución Actual) < Objetivo(Mejor Solución) then
7               Mejor Solución := Solución Actual; Genera (Solución Actual);
8       Until (Criterio de parada)
9   End;
```

Ilustración 8: Pseudocódigo Algoritmo Multiarranque

```
1   public ArrayList<Integer> resolverProblema() {
2       AlgoritmoGRASP algoritmoGRASP = new AlgoritmoGRASP(getGrafo());
3       ArrayList<Integer> solucionActual = new ArrayList<Integer>(algoritmoGRASP.resolverProblema());
4       ArrayList<Integer> mejorSolucion = new ArrayList<Integer>(solucionActual);
5       for (int i = 0; i < SOLUCIONES_GRASP; i++) {
6           realizarBusquedaLocal(solucionActual);
7           if (calcularDispersionMedia(solucionActual) > calcularDispersionMedia(mejorSolucion)) {
8               mejorSolucion = new ArrayList<Integer>(solucionActual);
9           }
10          solucionActual = new ArrayList<Integer>(algoritmoGRASP.resolverProblema());
11      }
12      return mejorSolucion;
13  }
```

Ilustración 9: Código Algoritmo GRASP

MUESTRA DE 10 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	7.90	22.0	7.333333	3.58445
2	7.90	10.0	5.0	2.66980
3	7.90	41.0	8.2	3.23319
4	7.90	22.0	7.333333	3.36714
5	7.90	22.0	7.333333	3.00566

MUESTRA DE 15 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-8.26	59.0	9.833333	4.14316
2	-8.26	36.0	9.0	4.19333
3	-8.26	36.0	9.0	3.29509
4	-8.26	46.0	9.2	5.21137
5	-8.26	38.0	9.5	4.32444

MUESTRA DE 20 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-4.90	62.0	12.4	5.26690
2	-4.90	62.0	12.4	4.70597
3	-4.90	62.0	12.4	4.58346
4	-4.90	90.0	12.857142	6.97547
5	-4.90	62.0	12.4	4.97841

MUESTRA DE 25 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-2.24	133.0	13.3	10.46197
2	-2.24	103.0	12.875	7.39374
3	-2.24	113.0	14.125	10.38844
4	-2.24	93.0	13.28571	6.00068
5	-2.24	121	13.44444	5.84122

ALGORITMO VNS

El VNS implementado en este caso es un VNS básico el cuál realiza una **Búsqueda por Recorrido al Azar (BRA)**. Para ello se obtiene inicialmente una solución aleatoria. Luego, se realiza una agitación, intercambiando uno de los vértices, elegido aleatoriamente también, por alguno de los posibles vecinos, y se comprueba la comparación entres dispersiones medias inicial y final. Seguidamente, si este valor no mejora, se realiza de nuevo la búsqueda por entorno variable, intercambiando en este instante un vértice más que en la iteración anterior. Este proceso terminará cuando no mejore la solución obtenida o cuando ya no sea posible intercambiar más elementos.

Dicho algoritmo cuenta con varias estructuras de entorno, las cuáles son:

- **INTERCAMBIO:** La cual aleatoriamente elije uno de los nodos dentro de la solución y lo intercambia aleatoriamente por los posibles vecinos.
- **ADD**. Añade un nodo de forma aleatoria.
- **REMOVE**. Elimina un nodo de forma aleatoria.

```
1   procedure Algoritmo VNS Begin
2       Genera(Solución Inicial Aleatoria);
3       Solución Actual:= Solución Inicial;
4       Mejor Solución := Solución Actual;
5       Repeat
6           BusquedaEntornoVariable(Solucion Actual, Numero Vecinos);
7           if Objetivo(Solución Actual) > Objetivo(MejorSolución) then
8               Mejor Solución := Solución Actual;
9           else
10              vecinosIntercambiar++;
11      Until (criterio de parada);
12  End;
```

Ilustración 10: Pseudocódigo Algoritmo VNS

```
1   @Override
2   public ArrayList<Integer> resolverProblema() {
3       int vecinosIntercambiar = VECINOS_INTERCAMBIAR_INICIAL;
4       ArrayList<Integer> subconjuntoS = new ArrayList<Integer>(obtenerSolucionAleatoria());
5       ArrayList<Integer> subconjuntoSAuxiliar = new ArrayList<Integer>(subconjuntoS);
6       do {
7           if (calcularDispersionMedia(subconjuntoSAuxiliar) > calcularDispersionMedia(subconjuntoS)) {
8               subconjuntoS = new ArrayList<Integer>(subconjuntoSAuxiliar);
9           }
10          else {
11              subconjuntoSAuxiliar = new ArrayList<Integer>(subconjuntoS);
12              realizarBusquedaEntornoVariable(subconjuntoSAuxiliar, vecinosIntercambiar);
13              vecinosIntercambiar++;
14          }
15      } while ((!subconjuntoS.equals(subconjuntoSAuxiliar)) && (vecinosIntercambiar < subconjuntoS.size()));
16      return subconjuntoS;
17  }
```

Ilustración 11: Código Algoritmo VNS

MUESTRA DE 10 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	7.90	61.0	7.625	1.61843
2	7.90	-10.0	-5.0	1.02142
3	7.90	52.0	10.4	0.92999
4	7.90	36.0	7.2	1.16194
5	7.90	55.0	7.857142	1.69060

MUESTRA DE 15 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-8.26	10.0	5.0	0.70152
2	-8.26	-1.0	-0.5	0.79388
3	-8.26	-31.0	-6.2	1.36648
4	-8.26	-26.0	-4.33333	0.84112
5	-8.26	16.0	3.2	0.88675

MUESTRA DE 20 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-4.90	21.0	2.333333	1.00239
2	-4.90	2.0	0.2	1.81317
3	-4.90	-19.0	-1.9	1.20729
4	-4.90	-20.0	-1.666666	1.41426
5	-4.90	4.0	0.8	0.88288

MUESTRA DE 25 VÉRTICES				
EJECUCIÓN	DISPERSIÓN MEDIA INICIAL	AFINIDAD SUBCONJUNTO	DISPERSION MEDIA SUBCONJUNTO	CPU (ms)
1	-2.24	47.0	3.35714	1.33905
2	-2.24	10.0	0.52631	2.45395
3	-2.24	3.0	0.333333	1.00492
4	-2.24	-36.0	-4.5	1.55076
5	-2.24	29.0	2.9	1.05435

REFERENCIAS

[5] CAMPUS VIRTUAL DE LA ASIGNATURA

Del campus virtual he utilizado los apuntes y diapositivas que desarrollan una descripción y explicación de todos los algoritmos utilizados en este trabajo para resolver el Max-Mean Dispersion Problem.

URL: https://campusvirtual.ull.es/1617/course/view.php?id=1138