DISEÑO Y ANÁLISIS DE ALGORITMOS
Algoritmos constructivos y búsquedas por entornos
Max-mean dispersión problem

Victoria Manrique Rolo

1. Introducción

En esta práctica se planteará una solución para el problema max-mean dispersión usando algoritmos constructivos y búsquedas por entornos.

1.1 El problema Max-mean dispersión

Dado un grafo completo G = (V, E), donde V es el conjunto de vértices y E el conjunto de aristas. Cada arista $(i, j) \in E$ tiene asociada una distancia d(i, j). En el max-mean dispersión se desea encontrar el subconjunto de vértices $S \subseteq V$ que maximiza la dispersión media dada por

$$md(S) = P\, i,j \in S\, d(i, j)/ |S|$$

1.2 Resolución

Para resolver el problema se utilizarán cinco algoritmos diferentes: voraz constructivo, variante de voraz, GRASP, multiarranque y VNS (búsqueda por entorno).

1.2.1 Voraz Constructivo

Este algoritmo función de la siguiente manera, partimos del subconjunto de vértices S conectados por la arista con mayor afinidad (en caso de haber más de una con le mismo valor máximo se eligirá una arista de manera aleatoria) e se irá añadiendo iterativamente al conjunto S, mientras sea posible el vértice que maximiza la función objetivo, es decir que consiga el mayor incremento. El pseudocódigo es el siguiente:

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 $md(S \cup \{k\})$;
6: if $md(S \cup \{k\}) \geq md(S)$ then
7: $S = S \cup \{k\}$;
8: until $(S* = S)$
9: Devolver $S*$;

1.2.2 Voraz (variante)

Este algoritmo tendrá una estructura similar al voraz constructivo pero en vez de partir de un subconjunto de vértices conectados por la arista con mayor afinidad, partiremos por el conjunto de todo los vértices e iremos eliminando iterativamente aquello vértices que su eliminación incremente la media de dispersión.

```
1: S = V;
2: repeat
3:    S∗ = S;
4:    Obtener el vértice k que maximiza md(S - {k});
5:    if md(S - {k}) ≥ md(S) then
6:       S = S - {k};
7: until (S∗ = S)
8: Devolver S∗ ;
```

1.2.3 GRASP

El algoritmo GRASP Greedy Randomized Adaptive Search Procedure es un algoritmo constructivo, es decir va añadiendo elementos a la solución que es inicialmente vacía, adaptativo, un elemento se evalúa teniendo en cuenta los elementos previamente incluidos en el solución, se mide la conveniencia de incluir el elemento como parte de la solución y tiene una estrategia greedy o voraz el cual escoge el elemento que optimiza la función objetivo.

El algoritmo tiene la siguiente estructura

```
1: Begin
2:  Preprocesamiento
3:    Repeat
4:       Fase Constructiva(Solución);
5:       PostProcesamiento(Solución);
6:       Actualizar(Solución, MejorSolución);
7:    Until (Criterio de parada);
8: End.
```

En la fase de preprocesamiento se eliminan las soluciones que no serán necesarias, en nuestro caso eliminaremos aquellos vértices cuyas arista sean todas negativas ya que no proporcionan ningún incremento positivo.

Iniciamos el algoritmo con el conjunto de nodos restantes de la fase de preprocesamiento y generamos su media para tener una solución inicial y pasamos a la fase constructiva, en esta fase se creará una solución según una lista de candidatos restringida, en esta lista se incluirá el 30% nodos con mejor media de dispersión de todas sus aristas, se seleccionará un nodo aleatoriamente de esa lista y se añadirá a la solución hasta que hayamos alcanzado el tamaño de la solución, este tamaño se generará de manera aleatoria.

```
1: repeat
2:   crearRCL();
3:   S = S U {elemento aleatorio de RCL};
4: hasta completar solucion
```

Esta solución generada pasará a la fase de postprocesamiento, es decir se evaluará la solución y se buscará algún a eliminar de la solución que mejore la función objetivo.

```
1: S = faseConstructiva();
2:repeat
3:   k = Obtener vértice que maximice la solución
4:   S = S - {k}
5: hasta no haya ningún vértice que eliminar
```

Con esta solución generada pasaremos a la fase de actualización en la cual evaluamos si la solución generada por la fase de postprocesamiento es mejor a la guardada hasta el momento, en caso de ser mejor la actualizamos.

Como condición de parada de este algoritmo se pueden utilizar dos criterios: número de iteraciones y número de iteraciones sin mejora, es decir sin actualización, estos se define en una constante.

1.2.4 Multiarranque

Este algoritmo es bastante parecido a GRASP con la única diferencia que las soluciones generadas en la fase constructivas son generadas aleatoriamente y no con una lista de candidatos.

```
1: 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)
7:      then
8:        Mejor Solución := Solución Actual;
7:        Genera (Solución Actual);
9:   Until (Criterio de parada)
10: End.
```

Los métodos usados en este algoritmo son bastantes similares a los utilizados en GRASP con la diferencia comentada anteriormente en el método “Genera”.

1.1.5 VNS (Variable neighbourhood Search)

Este algoritmo se basa en la búsqueda por entornos variable. Se inicia desde un solución aleatoria y se genera su entorno, una vez generado el entorno pueden ocurrir dos cosas, la solución es un mínimo/máximo local dependiendo de la función objetivo, si esto es así el siguiente paso es generar un entorno más grande que contenga la solución actual.

En caso de que se encuentre una máximo local distinto a la solución actual se tomará esta solución como mejor solución y se generará su entorno, y se buscará un mínimo máximo local. El número de entornos generados es un valor dictado por el usuario, suelen ser tres entornos como máximo.

Este proceso se repite hasta que se alcanza un número de iteraciones máximo o un número de iteraciones máximas sin mejora.

El pseudocódigo de este algoritmo es el siguiente:

```
1:  S = GenerarInicial();
2:  repeat
3:    repeat
4:      X = agitación(k, S);
5:      busquedaLocal(X)
6:      if X mejor que S
7:          S = X;
8:          reset del tamaño de entorno
9:     else
10:         ampliamos entorno
11:   hasta tamaño entorno == maximo
12: hasta condición de parada
```

2. Implementación. Estructura de clases

Para poder resolver este problema se hará uso de programación orientada a objetos. A continuación se explicará la estructura de clases creada para esta práctica, empezaremos por las clases “graph” y “solution”, las cuales definen el problema y la solución generada. Para la aplicación de los distintos algoritmos se utilizado un patrón estrategia, donde cada algoritmo se construye en un clase distinta.

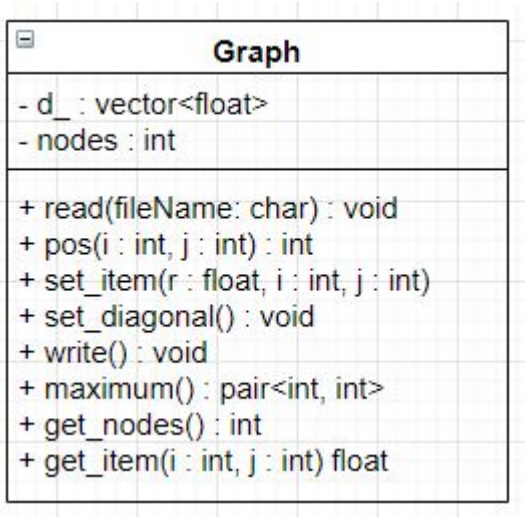
Para cada clase nombrará sus métodos y atributos acompañados de una explicación correspondiente a su importancia, además se incluirán diagramas UML que representarán sus atributos y métodos además de su jerarquía.

2.1. Clase “graph”

En esta clase se representará el grafo completo en forma de matriz de distancias, se recibirán los datos desde fichero y este tendrá el siguiente formato. En la primera línea estará el número de vértices total y a continuación se irán listando las distancias d(i, j) entre los vértices asumiendo que las afinidades son simétricas, es decir, d(i, j) = d(j, i). Un ejemplo de este fichero sería:

```
4 número de vértices -
3,90 d(1, 2) = d(2, 1)
-4,27 d(1, 3) = d(3, 1)
5,66 d(1, 4) = d(4, 1)
5,95 d(2, 3) = d(3, 2)
9,53 d(2, 4) = d(4, 2)
-6,91 d(3, 4) = d(4, 3)
```

La clase tendrá como atributos un vector que se implementará como matriz con las distancias (d_) y el número de nodos totales (nodes_).

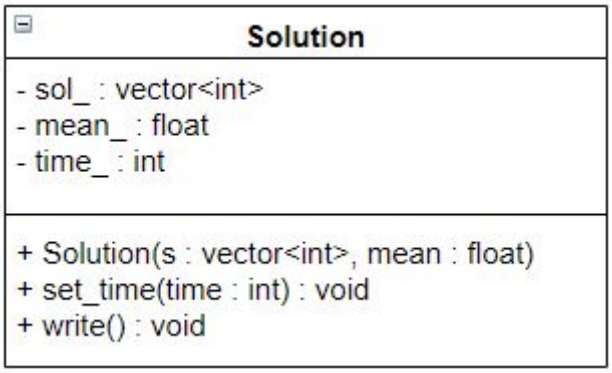


Métodos:

- **read**: se encarga de leer el fichero y almacenar las distancias en su posición correspondiente
- **pos**: devuelve la posición relativa al vector dado dos índices.
- **set_item**: dados dos índices almacena el dato en la posición correspondiente en el vector.
- **set_diagonal**: inicializa la diagonal a cero
- **write**: imprime por pantalla la matriz de distancias
- **maximum**: devuelve un par de enteros que forman la arista con mayor afinidad.
- **get_nodes**: devuelve el atributo nodes_
- **get_item**: dado dos índices devuelve el valor almacenado en la posición indicada.

2.2 Solution

Esta clase será la encargada de almacenar la solución del problema (sol_), contiene un vector con los vértices que pertenecen a la solución así como la media de dispersión (mean_) y el tiempo de ejecución (time_).



Métodos:

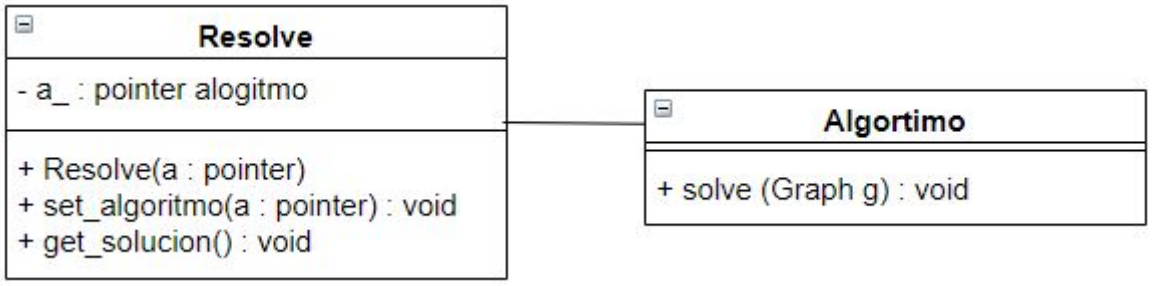
- **Constructor:** recibe el vector con el conjunto de nodos y la media
- set_time: asigna el valor de atributo time_.

2.3. Patrón estrategia

En el patrón estrategia participarán un total de siete clases, una clase que servirá de contexto para elegir el tipo de algoritmo a utilizar, una clase algoritmo virtual de la cual desde la cual heredarán todos las clases que implementan cada algoritmo.

2.3 Clase Resolve Y Algoritmo

La clase Resolve contará como atributo únicamente un puntero a la clase algoritmo para poder ejecutar el algoritmo y la clase Algoritmo tendrá un método virtual llamado solve el cual será el encargado de iniciar la ejecución del algoritmo.

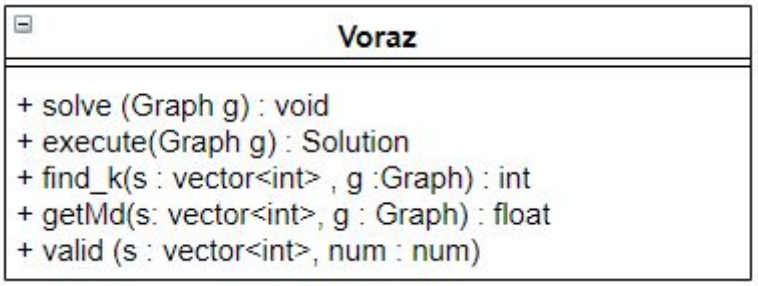


Métodos

- **set_algoritmo:** asigna valor al atributo a_:
- **get_solución**: llama al método *solve* de la clase algoritmo

2.4 Clase Voraz

Esta se encarga de implementar el algoritmo voraz constructivo

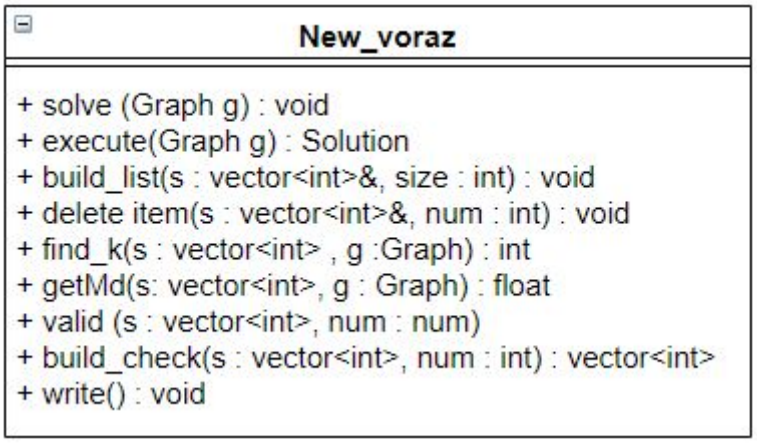


Métodos

- **solve:** inicia ejecución
- **execute:** lleva a cabo el algoritmo voraz
- **find_k**: busca vértice k para añadir a la solución
- **getMd**: calcula la media de dispersión de una solución
- **valid**: comprueba si un vértice se encuentra o no en la solución

2.5 Clase New_voraz

Esta clase se encarga de implementar el algoritmo voraz alternativo

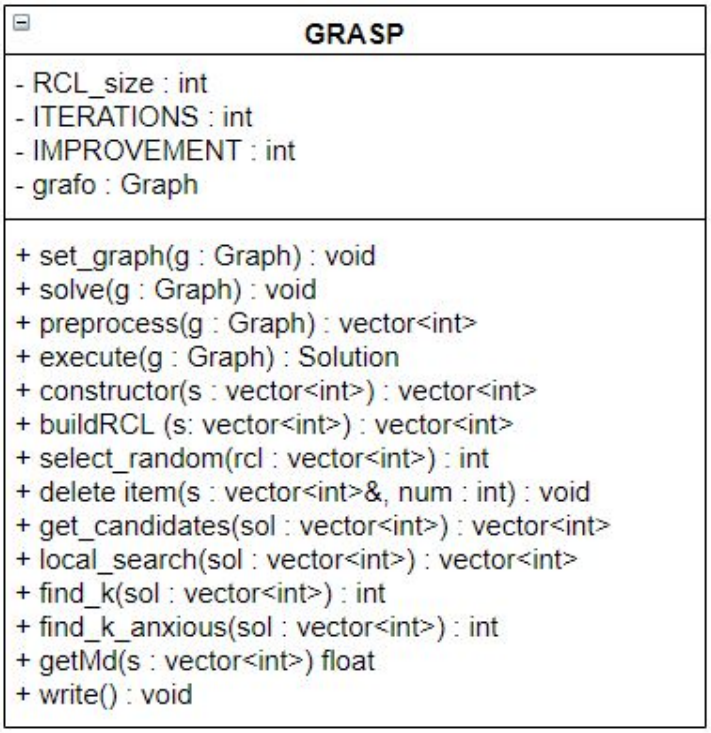


Métodos

- **solve:** inicia ejecución
- **execute:** lleva a cabo el algoritmo voraz alternativo
- **build_list**: genera conjunto con todos los vértices posibles
- **delete_item**: elimina un vértice de la solución
- **find_k**: busca vértice k para eliminar de la solución
- **getMd**: calcula la media de dispersión de una solución
- **valid**: comprueba si un vértice se encuentra o no en la solución
- **build_check**: crea solución sin elemento que se le manda

2.6 Clase GRASP

Implementa el algoritmo GRASP, tiene una serie de atributos: número máximo de iteraciones (ITERATIONS), número de máximo de iteraciones sin mejora (IMPROVEMENT), el tamaño de la RCL (RCL_size) y el objeto Graph con la matriz de distancias (grafo).

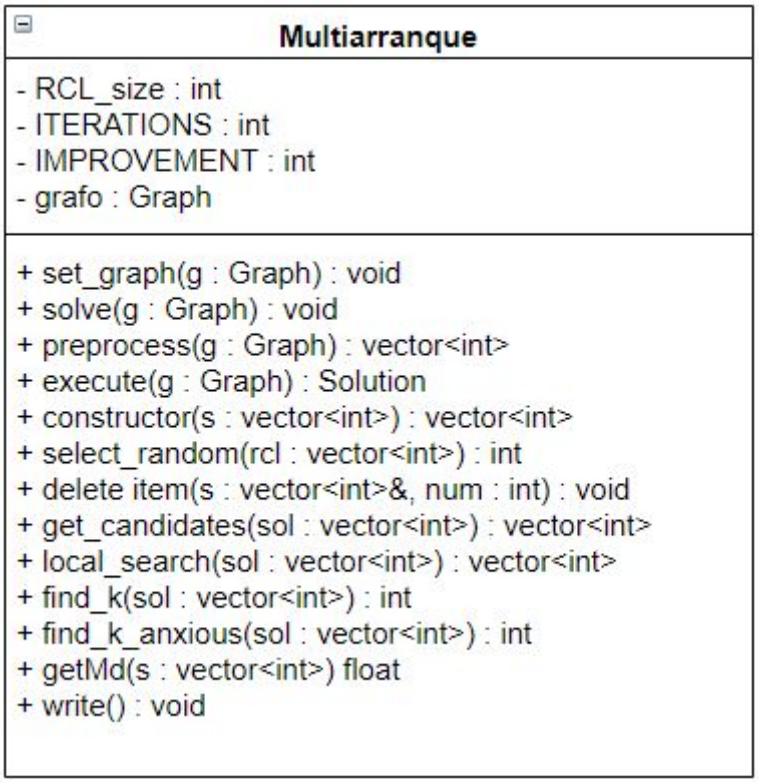


Métodos

- **set_graph:** asigna valor al atributo
- **solve:** inicia la ejecución
- **preprocess:** fase de pre procesamiento
- **constructor:** fase de construcción, genera soluciones a partir de las RCL
- **buildRCL:** se encarga de crear una lista de candidatos restringida
- **select_random:** elige elemento aleatorio de una RCL
- **get_candidates:** genera lista de candidatos no incluidos en la solución
- **delete_item:** elimina un vértice de una solución
- **local_search:** realiza la busqueda local de una solución
- **find_k:** busca vértice k que maximice la función objetivo (greedy)
- **find_k_anxious:** busca vértice k maximice la función objetivo (ansiosa)
- **getMd:** calcula media dispersión de una solución
- **write:** escribe una solución

2.7 Clase Multiarranque

Implementa el algoritmo multiarranque, tiene como atributos: número máximo de iteraciones (ITERATIONS), número de máximo de iteraciones sin mejora (IMPROVEMENT) y el objeto Graph con la matriz de distancias (grafo).

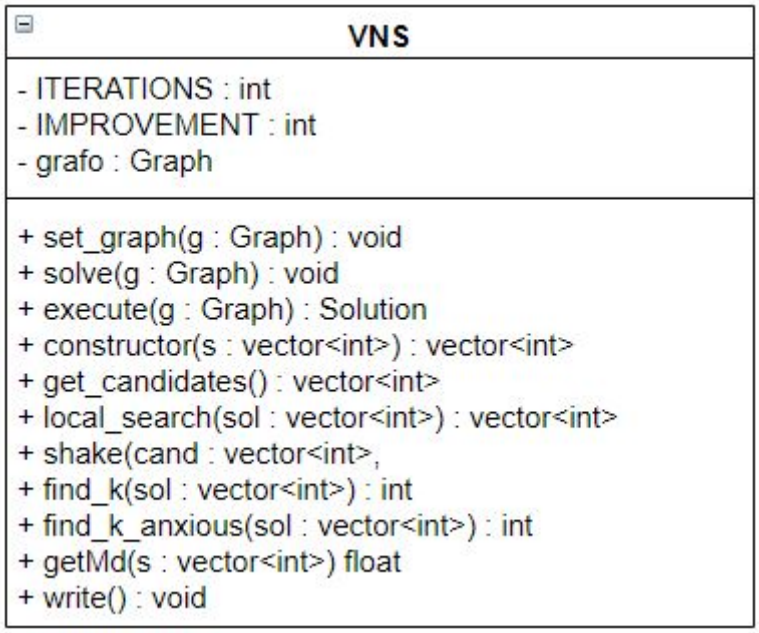


Métodos

- **set_graph:** asigna valor al atributo
- **solve:** inicia la ejecución
- **preprocess:** fase de pre procesamiento, elimina nodos
- **constructor:** fase de construcción genera soluciones aleatorias
- **select_random:** elige elemento aleatorio de una RCL
- **get_candidates:** genera lista de candidatos
- **delete_item:** elimina un vértice de una solución
- **local_search:** realiza la búsqueda local de una solución
- **find_k:** busca vértice k que maximice la función objetivo (greedy)
- **find_k_anxious:** busca vértice k maximice la función objetivo (ansiosa)
- **getMd:** calcula media dispersión de una solución
- **write:** escribe una solución

2.9 Clase VNS

Encargada de implementar el algoritmo VNS, tiene como atributos número máximo de iteraciones (ITERATIONS), número de máximo de iteraciones sin mejora (IMPROVEMENT) y el objeto Graph con la matriz de distancias (grafo).



Métodos

- **set_graph:** asigna valor al atributo
- **solve:** inicia la ejecución
- **constructor:** fase de construcción genera soluciones en base al entorno
- **select_random:** elige elemento aleatorio de una RCL
- **get_candidates:** genera lista de candidatos
- **local_search:** realiza la búsqueda local de una solución
- **shake:** agitación, genera un tamaño de entorno
- **find_k:** busca vértice k que maximice la función objetivo (greedy)
- **find_k_anxious:** busca vértice k maximice la función objetivo (ansiosa)
- **getMd:** calcula media dispersión de una solución
- **write:** escribe una solución

3. Resultados

A continuación mostraremos en forma de tabla los resultados de cada algoritmo en distinta ejecuciones.

Antes que nada especificaremos las caracteríticas de la máquina en la que se ha ejecutado el problema

Procesador: Intel(R) Core(™) i7-7500U CPU @ 2.70 GHz 2.90 GHz

Algoritmo voraz

Problema nº1: max-mean-div-10.txt **Vértices:** 10

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	14	7 6 8 5 9 4	247
2	10.1429	0 2 4 6 7 8 9	55
3	10.1429	2 0 4 6 7 5 8	61
4	14	7 6 8 5 9 4	55
5	13	7 6 8 5 3 4	60
6	13	5 8 6 7 3 4	53
7	14	8 5 6 7 9 4	120
8	10.1429	2 0 4 6 7 8 5	84
9	11.8571	9 1 8 7 6 5 4	70
10	11.8571	9 1 8 7 6 5 4	89

Promedio de las 10 ejecuciones: 12.21429 **/ Tiempo medio:** 90.4
Mejor valor: 14

Problema nº2: max-mean-div-15.txt **Vértice:** 15

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	7.5	13 14 9 3	58
2	9.75	8 4 11 10	76
3	9.75	8 4 11 10	82
4	9.75	8 11 4 10	86
5	9.5	8 1 10 3	70
6	9.5	1 8 10 3	60

7	9.75	11 8 4 10	62
8	9.5	3 10 1 8	76
9	7.5	13 14 9 3	57
10	7.5	13 14 9 3	90

Promedio de las 10 ejecuciones: 9 **/ Tiempo medio:** 71.7
Mejor valor: 9.75

Problema nº3: max-mean-20.txt **Vértices:** 20

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	12.8571	0 3 18 11 8 19 7	181
2	12	3 12 11 0 18	149
3	12.8571	0 18 3 11 8 19 7	147
4	12	4 9 16 8 14 17 7	278
5	12	12 3 11 0 18	105
6	13.1667	14 7 17 4 8 16	147
7	12	3 12 11 0 18	124
8	12	12 3 11 0 18	215
9	12	4 9 16 8 14 17 7	176
10	8.8	6 10 5 17 19	118

Promedio de las 10 ejecuciones: 11.96809 **/ Tiempo medio:** 164
Mejor valor: 13.1667

Algoritmo Voraz Alternativo
Problema nº1: max-mean-div-10.txt **Vértices:** 10

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	10.1429	0 2 4 6 7 5 8	155
2	14	6 7 8 5 9 4	136
3	14	6 7 8 5 9 4	112
4	14	6 7 8 5 9 4	134
5	13	6 7 8 5 3 4	157
6	13	5 8 6 7 3 4	163
7	14	4 5 6 7 8 9	154

8	14	4 5 6 7 8 9	147
9	14	6 7 8 5 9 4	164
10	14	4 5 6 7 8 9	132

Promedio de las 10 ejecuciones: 13.41429 **/ Tiempo medio:** 145.4
Mejor valor: 14

Problema nº2: max-mean-div-15.txt **Vértice:** 15

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	9.83333	1 3 6 7 8 10	429
2	9.75	8 4 11 10	606
3	9.83333	3 1 6 7 8 10	456
4	9.83333	1 3 6 7 8 10	411
5	9.83333	1 3 6 7 8 10	446
6	9.75	11 8 4 10	570
7	9.83333	1 3 6 7 8 10	418
8	9.83333	1 3 6 7 8 10	652
9	9.83333	1 3 6 7 8 10	433
10	9.83333	1 3 6 7 8 10	374

Promedio de las 10 ejecuciones: 9.81664 **/ Tiempo medio:** 479.5
Mejor valor: 9.83333

Problema nº3: max-mean-20.txt **Vértices:** 20

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	12.8571	0 7 8 11 17 18 19	779
2	12.1429	0 7 8 14 18 19	1043
3	12.1429	0 7 8 14 18 19	893
4	12.8571	0 7 8 11 17 18 19	974
5	12.8571	0 7 8 11 17 18 19	909
6	12.1429	0 7 8 14 18 19	954
7	12.1429	0 7 8 14 18 19	1266
8	12.8571	0 7 8 11 17 18 19	975

9	12.1429	0 7 8 14 18 19	901
10	12.8571	0 7 8 11 17 18 19	877

Promedio de las 10 ejecuciones: 12.5 **/ Tiempo medio:** 957.1
Mejor valor: 12.8571

Algoritmo GRASP

Problema nº1: max-mean-div-10.txt **Vértices:** 10 **Versión greedy**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	14	4 9 6 7 8 5	4289
2	14	4 7 6 8 5 9	4232
3	14	9 4 7 6 5 8	5954
4	14	6 7 8 5 9 4	4601
5	14	9 8 7 6 5 4	7748

Promedio de las 10 ejecuciones: 13.8 **/ Tiempo medio:** 3815.2
Mejor valor: 14

Problema nº1: max-mean-div-10.txt **Vértices:** 10 **Versión ansiosa**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	14	6 5 7 8 9 4	5227
2	14	7 6 5 4 8 9	3842
3	14	9 8 7 6 5 4	4438
4	14	4 9 7 8 6 5	5328
5	14	4 7 6 8 5 9	6083

Promedio de las 10 ejecuciones: 13.9 **/ Tiempo medio:** 5183.6
Mejor valor: 14

Problema nº2: max-mean-div-15.txt **Vértice:** 15 **Versión greedy**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	9.83333	8 10 1 6 3 7	14158
2	9.83333	3 7 1 8 10 6	18344
3	9.83333	3 10 1 7 6 8	15621

4	9.83333	10 8 1 6 3 7	14425
5	9.83333	8 3 6 7 1 10	12217

Promedio de las 10 ejecuciones: 9.8333 **/ Tiempo medio:** 12855.55
Mejor valor: 9.83333

Problema nº2: max-mean-div-15.txt **Vértice:** 15 **Versión ansiosa**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	9.83333	7 8 1 6 3 10	14881
2	9.83333	6 1 8 7 10 3	10870
3	9.83333	3 6 7 8 1 10	11733
4	9.83333	10 3 8 6 1 7	15915
5	9.83333	6 1 8 10 3 7	16245

Promedio de las 10 ejecuciones: 9.8333 **/ Tiempo medio:** 11624
Mejor valor: 9.83333

Problema nº3: max-mean-div-20.txt **Vértice:** 20 **Versión greedy**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	13.1667	4 16 8 17 14 7	44102
2	13.1667	4 16 8 14 17 7	41512
3	13.1667	4 16 8 14 17 7	40403
4	13.1667	4 16 8 14 17 7	39410
5	13.1667	4 16 8 17 14 7	40540

Promedio de las 10 ejecuciones: 13.1667 **/ Tiempo medio:** 49075.4
Mejor valor: 9.83333

Problema nº3: max-mean-div-20.txt **Vértice:** 20 **Versión ansiosa**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	13.1667	4 16 14 17 8 7	18359
2	13.1667	4 16 14 17 8 7	18205
3	13.1667	4 16 14 17 8 7	18404
4	13.1667	4 16 8 14 17 7	18533
5	13.1667	4 16 8 17 14 7	19527

Promedio de las 10 ejecuciones: 13.1667 **/ Tiempo medio:** 15009.2
Mejor valor: 13.1667

Algoritmo Multiarranque

Problema nº1: max-mean-div-10.txt **Vértices:** 10 **Versión greedy**

Ejecución nº	**md**	**Conjunto de nodos**	**Tiempo (micros)**
1	14	8 5 9 4 6 7	1269
2	14	8 5 9 4 6 7	884
3	14	8 5 9 4 6 7	897
4	14	6 7 8 5 9 4	883
5	14	6 8 9 7 4 5	1098

Promedio de las 10 ejecuciones: 14 **/ Tiempo medio:** 1006.2
Mejor valor: 14

Problema nº1: max-mean-div-10.txt **Vértices:** 10 **Versión ansiosa**

Ejecución nº	**md**	**Conjunto de nodos**	**Tiempo (micros)**
1	14	4 9 6 5 7 8	1166
2	14	4 7 6 5 8 9	1066
3	13	4 6 5 8 7 3	1067
4	14	4 5 9 7 6 8	1162
5	14	5 9 7 4 6 8	1065

Promedio de las 10 ejecuciones: 13.9 **/ Tiempo medio:** 907.2
Mejor valor: 14

Problema nº2: max-mean-div-15.txt **Vértice:** 15 **Versión greedy**

Ejecución nº	**md**	**Conjunto de nodos**	**Tiempo (micros)**
1	9.83333	1 6 8 10 3 7	3486
2	9.83333	10 1 8 7 6 3	2891
3	9.83333	3 10 1 7 6 8	3615
4	9.83333	10 8 6 3 1 7	2509
5	9.83333	3 1 8 7 6 10	2509

Promedio de las 10 ejecuciones: 9.8333 **/ Tiempo medio:** 3002
Mejor valor: 9.83333

Problema nº2: max-mean-div-15.txt **Vértice:** 15 **Versión ansiosa**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	9.83333	3 6 1 8 7 10	2379
2	9.83333	10 1 3 8 6 7	2119
3	9.83333	8 7 6 3 1 10	2099
4	9.83333	10 6 8 3 7 1	2136
5	9.83333	3 8 10 6 7 1	2101

Promedio de las 10 ejecuciones: 9.8333 **/ Tiempo medio: 2166.8**
Mejor valor: 9.83333

Problema nº3: max-mean-div-20.txt **Vértice:** 20 **Versión greedy**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	13.1667	14 4 8 7 16 17	7792
2	12.8571	7 19 18 11 0 8 17	8385
3	13.1667	4 16 7 14 8 17	8136
4	13.1667	17 4 14 16 8 7	6927
5	13.1667	16 7 17 14 8 4	14207

Promedio de las 10 ejecuciones: 13.10478 **/ Tiempo medio:** 9089.4
Mejor valor: 13.1667

Problema nº3: max-mean-div-20.txt **Vértice:** 20 **Versión ansiosa**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	13.1667	17 8 14 4 7 16	3617
2	13.1667	17 8 14 4 7 16	3590
3	13.1667	17 8 14 4 7 16	2747
4	13.1667	17 8 14 4 7 16	2716
5	13.1667	17 8 14 4 7 16	2952

Algoritmo VNS

Promedio de las 10 ejecuciones: 13.1667 **/ Tiempo medio:** 3124.4
Mejor valor: 13.1667

Problema nº1: max-mean-div-10.txt **Vértices:** 10 **Versión greedy**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	10.5	3 6 2 8 5 4	1135
2	14	0 2 3 8 9 7 1 4	2038
3	9.77778	4 8 0 9 3 1 6 7 5 2	3500
4	14	1 9 0 8 2 3 5	983
5	11.8571	5 3 1 4 9 7 2 8 6	1793

Promedio de las 10 ejecuciones: 12.0269 **/ Tiempo medio: 1889.8**
Mejor valor: 14

Problema nº1: max-mean-div-10.txt **Vértices:** 10 **Versión ansiosa**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	10.75	3 0 8 7 5 4	813
2	11.4	7 1 9 6 3 5 8	667
3	9.77778	6 8 5 7 9 0 4 2 1 3	1448
4	10.875	6 3 9 1 4 5 0 8 7	2090
5	12.75	5 2 7 3 0 6 8 4 9 1	1688

Promedio de las 10 ejecuciones: 11.11 **/ Tiempo medio: 1341.2**
Mejor valor: 12.75

Problema nº2: max-mean-div-15.txt **Vértice:** 15 **Versión greedy**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	3.5	6 1 8 13 4 11 0 10 3	1289
2	8.16667	3 13 4 2 12 1	1090
3	6.42857	11 7 1 12 14 6 2	1000
4	8.2	11 4 1 0 13 2 6	1399
5	9.5	1 7 13 9 14 11 10 5 0	2716

		3 4 8 2 6	

Promedio de las 10 ejecuciones: 7.15 / Tiempo medio: 1498.8
Mejor valor: 9.5

Problema nº2: max-mean-div-15.txt **Vértice:** 15 **Versión ansiosa**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	9	7 8 4 5 6 2 12 3 9 13 1	2007
2	9.75	6 9 12 3 13 14 5 0 8 7 10 1 2	2711
3	8	0 10 4 8 2 14 6	906
4	7.4	0 13 2 3 5 14	784
5	8.33333	13 12 10 2 3 11 8 1	2530

Promedio de las 10 ejecuciones: 8.496 / Tiempo medio: 1787.6
Mejor valor: 9.75

Problema nº3: max-mean-div-20.txt **Vértice:** 20 **Versión greedy**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	12.8333	8 2 5 12 13 4 3 19 1 7 0 16 10 14 18 9 15 6	20529
2	12.1429	0 15 5 12 19 2 1 14 11 3 8 17	19934
3	12.8	8 9 17 5 10 1 2 3 18 13 15 4 11 0	18292
4	11.5	2 16 13 10 3 9 5 4 11 12 17 19 18	13420
5	9.25	4 14 0 5 7 16 15 9 13 19 2 6 3 8 7	15606

Promedio de las 10 ejecuciones: 11.70524 / Tiempo medio: 17556.2
Mejor valor: 12.8333

Problema nº3: max-mean-div-20.txt **Vértice:** 20 **Versión ansiosa**

Ejecución nº	md	Conjunto de nodos	Tiempo (micros)
1	13.1667	17 8 14 4 7 16	3617

2	9	4 0 3 15 2 12 10 19 7 9 8 18 13	2046
3	7.33333	12 15 16 10 0 14 1 7 6 19	1545
4	12.8571	11 3 8 0 19 18 7	3954
5	12.5	10 2 4 9 14 17 5 18 6 3	2306

Promedio de las 10 ejecuciones: 10.97136 / Tiempo medio: 2693.6
Mejor valor: 13.1667