Max diversity problem.

Diseño y análisis de algoritmos.

Alumno: Sabato Ceruso.

Índice de contenido

Introducción

Max dispersion problem.

Se pretende realizar una implementación de diversos algoritmos para la resolución del problema del problema de la máxima dispersión y del posterior estudio del comportamiento de estos algoritmos.

Concretamente se hizo una implementación de dos algoritmos GRASP constructivos distintos, uno que partiendo de una solución vacía añade nodos uno a uno, y otro que partiendo de todos los nodos elimina nodos uno a uno. También se implementó dos algoritmos Greedy, siguiendo la misma filosofía que antes; un GRASP con post-procesamiento, y un algoritmo de ramificación y poda.

A continuación se explicara cada uno de estos algoritmos, seguido de una explicación de la estructura del programa y estructuras de datos empleadas y finalmente se detallará la experiencia computacional realizada.

Algoritmos

Algoritmos Greedy.

Se implementó, como se ha mencionado antes, dos algoritmos greedy, uno que como solución inicial parte de una solución vacía, y, va agregando nodos hasta que no encuentra ningún nodo que consiga dar una mejor solución que la que tiene. Para decidir qué nodo añadir, primero se calcula el centroide de los nodos que no forman parte de la solución actual y posteriormente se elije el nodo más alejado del centroide.

El otro, parte desde una solución con todos los nodos y elimina hasta que se encuentra con el mismo criterio de parada.

En la práctica, estos algoritmos se implementaron como casos especiales de la fase constructiva del algoritmo GRASP pero con la particularidad de poseer una lista restringida de candidatos de tamaño 1, de este modo, se asegura que en cada iteración se escoge siempre el mejor candidato.

Algoritmos GRASP.

Los algoritmos GRASP se diseñaron permitiendo cambiar el tamaño de la lista restringida de candidatos a conveniencia. El tamaño de esta lista deberá ser fijado antes de cada ejecución y será un número fijo, el tamaño real de la lista podrá ser menor pero nunca mayor.

A la hora de seleccionar una solución de la lista de candidatos se selecciona realizando una tirada completamente aleatoria entre las soluciones mejores.

En la fase de post-procesamiento, se realiza una búsqueda local utilizando como entorno aquellas soluciones que pueden ser producidas como resultado de insertar y eliminar un único elemento. El muestreo del entorno se realizará, en este algoritmo y en los demás, de forma ansiosa, con el fin de evitar recorrer todo el entorno de la solución y además, no se podrá pasar a una solución de peor calidad de la que ya se tiene. El criterio de parada será el encontrar un óptimo local.

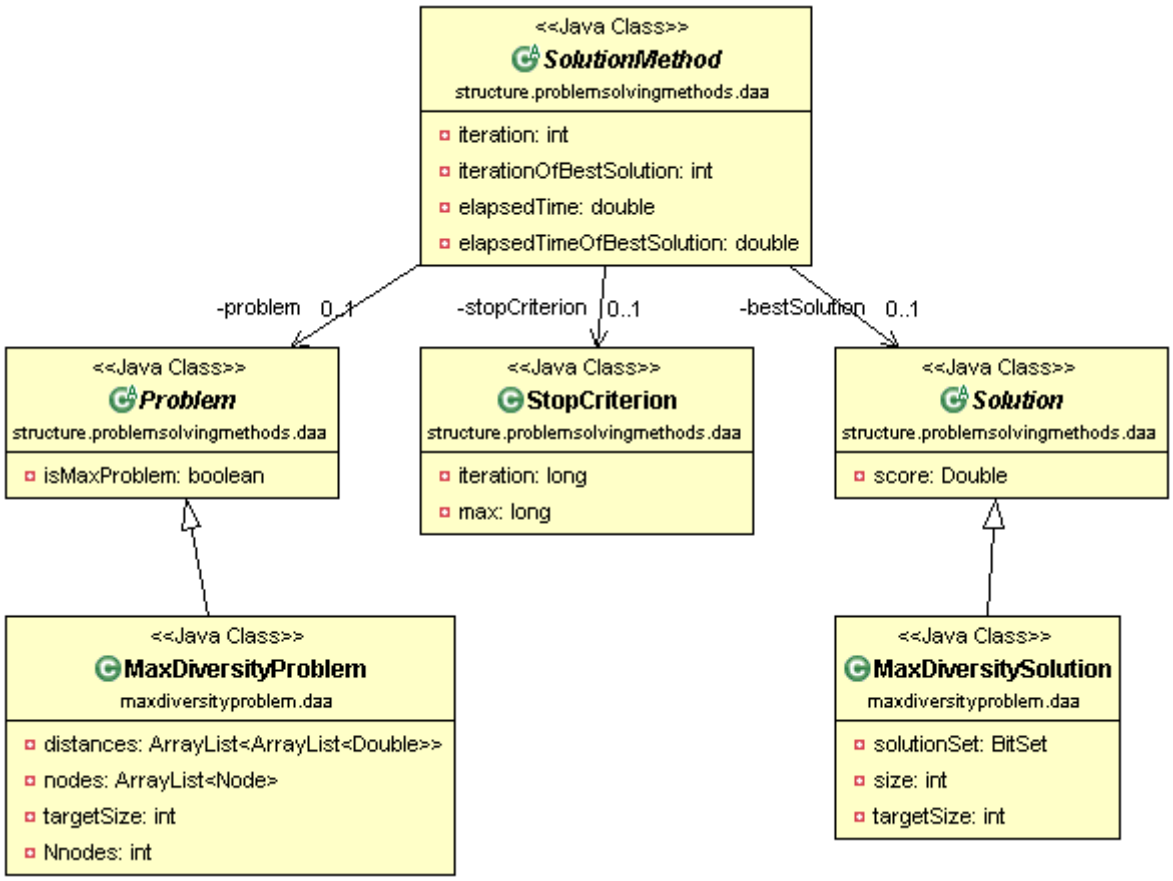
Método ramificación y acotación.

Se implementó un algoritmo de ramificación y acotación, con la flexibilidad de poder decidir, en el momento de la definición del algoritmo, como se escogerá la cota inferior, superior y qué nodo se ramificará primero.

Estructuras utilizadas.

Para la implementación se realizó una estructura de clases donde hay una clase abstracta problema que representa todos los problemas, una clase abstracta solución que representa las soluciones a un problema abstracto y una clase abstracta método de solución que representará los algoritmos a utilizar para resolver un determinado problema.

A continuación se extiende las clases de problema y solución para dar forma a nuestro problema en particular. A continuación se muestra el diagrama de clases que se acaba de explicar.



Para la implementación de los algoritmos, cada uno de ellos extenderá la clase SolutionMethod, pues, cada uno de ellos será un método para proporcionar una solución.

En primer lugar, la clase ConstructiveGRASP se ocupará de generar una solución siguiendo un método GRASP, a ella la extienden DirectConstructiveGRASP y ReverseConstructiveGRASP que se ocupan de generar una solución añadiendo o quitando nodos siguiendo un método GRASP.

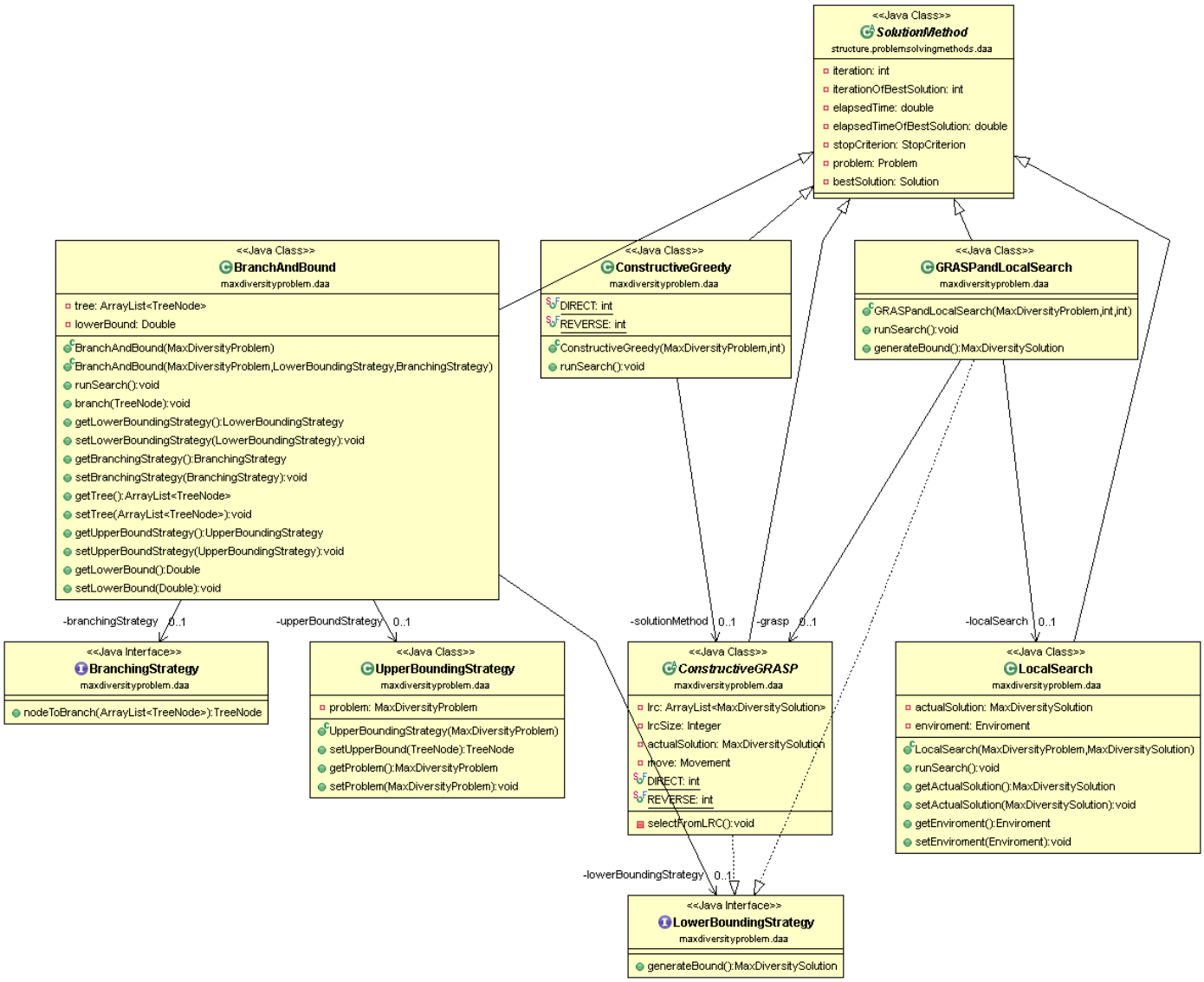
El algoritmo Greedy estará compuesto por un ConstructiveGRASP con un tamaño de lista restringida de candidatos igual a 1 y simplemente se ocupará de crear dicho GRASP y ejecutarlo.

Luego se implementó la búsqueda local, donde se realizará una búsqueda a partir de una solución utilizando una estructura de entorno dada. Para realizar esta búsqueda se definió primero la clase Enviroment, que define a un entorno, según esta estructura, un entorno está definido por un movimiento de agregar y otro de quitar un elemento de la solución.

Una vez que se tiene la búsqueda local, se pudo implementar la clase GRASPandLocalSearch que se ocupa de realizar una construcción mediante un método GRASP de una solución para luego realizar una búsqueda local sobre dicha solución. Esta clase es muy simple, está compuesta por un ConstructiveGRASP y un LocalSearch y solamente se ocupa de ejecutar ambos algoritmos uno tras otro.

Finalmente, se implementó el algoritmo de ramificación y acotación, el cual está compuesto por una estrategia de ramificación, una estrategia de calculo de cota superior y otra para la inferior. El algoritmo se encarga de primero, iniciar con una solución vacía, utilizar la estrategia de cota inferior para obtener la primera cota inferior, y luego, mientras queden nodos para expandir, pide a la estrategia de ramificación el siguiente nodo a expandir, lo expande y a través de la estrategia de calculo de cota superior calcula la cota superior de cada posible nodo a añadir a la lista de nodos a expandir, y, en caso de cumplir con los criterios de poda, no lo añade.

A continuación se muestra el diagrama UML de clases de los algoritmos:



Experiencia computacional

Sistema sobre el que se realiza:

SO: Ubuntu.

Procesador: Intel core i5.

RAM: 6 GB.

Algoritmo Greedy Directo:

Tamaño (n)	Valor objetivo	m	S	Tiempo (Milis)
15	1,8	2	{1,8}	1
15	4,85	3	{1, 3, 8}	1
15	13,93	4	{1, 3, 4, 8}	2
15	38,8	5	{0, 1, 3, 4, 8}	2
20	1,5	2	{2, 17}	2
20	5,6	3	{2, 13, 17}	2
20	11,3	4	{2, 13, 16, 17}	3
20	18,67	5	{2, 9, 13, 16, 17}	2
30	11,65	2	{8, 27}	2
30	28,94	3	{1, 8, 27}	2
30	52,77	4	{1, 8, 10, 27}	2
30	78,99	5	{1, 8, 10, 27, 29}	2
15	2,24	2	{9, 11}	1
15	5,34	3	{9, 11, 12}	2
15	19,27	4	{9, 10, 11, 12}	2
15	46,32	5	{9, 10, 11, 12, 13}	2
20	7,17	2	{12, 16}	1
20	16,29	3	{11, 12, 16}	2
20	47,11	4	{11, 12, 13, 16}	3
20	71,97	5	{11, 12, 13, 14, 16}	3
30	11,45	2	{16, 23}	2
30	33,5	3	{13, 16, 23}	2
30	56,47	4	{7, 13, 16, 23}	2
30	85,56	5	{7, 13, 16, 23, 26}	3

Algoritmo Greedy Inverso:

Tamaño (n)	Valor objetivo	m	S	Tiempo (Milis)
15	11,8	2	{6,8}	4
15	27,37	3	{0, 6, 8}	4
15	49,86	4	{0, 5, 6, 8}	4
15	78,88	5	{0, 1, 5, 6, 8}	4
20	8,5	2	{17, 18}	7
20	21,99	3	{8, 17, 18}	6
20	39,82	4	{1, 8, 17, 18}	7
20	63,65	5	{1, 8, 13, 17, 18}	6
30	11,65	2	{8, 27}	13
30	28,94	3	{1, 8, 27}	4
30	52,77	4	{1, 8, 10, 27}	3
30	80,91	5	{1, 8, 10, 12, 27}	4
15	11,76	2	{4, 11}	3
15	30,5	3	{3, 4, 11}	2
15	59,27	4	{3, 4, 11, 13}	1
15	96,08	5	{3, 4, 8, 11, 13}	1
20	11,59	2	{2, 16}	6
20	29,15	3	{2, 12,16}	5
20	56,69	4	{2, 12, 13, 16}	2
20	92,82	5	{2, 7, 12, 13, 16}	<1
30	13,07	2	{6, 16}	14
30	33,84	3	{6, 16, 23}	5
30	63,51	4	{6, 13, 16, 23}	4
30	98,58	5	{3, 6, 13, 16, 23}	4

Algoritmo GRASP directo con búsqueda local:

Tamaño (n)	Valor objetivo	m	LRC	S	Tiempo (Milis)

15	11,85	2	2	{6, 8}	6
15	11,85	2	3	{6, 8}	1
15	27,37	3	2	{0, 6, 8}	3
15	27,37	3	3	{0, 6, 8}	4
15	49,82	4	2	{0, 5, 6, 8}	6
15	49,82	4	3	{0, 5, 6, 8}	3
15	79,12	5	2	{0, 3, 5, 6, 8}	9
15	79,12	5	3	{0, 3, 5, 6, 8}	8
20	8,5	2	2	{17, 18}	8
20	8,5	2	3	{17, 18}	2
20	21,99	3	2	{8, 17, 18}	4
20	21,99	3	3	{8, 17, 18}	4
20	40	4	2	{1, 2, 8, 18}	5
20	40	4	3	{1, 2, 8, 18}	3
20	63,65	5	2	{1, 8, 13, 17, 18}	2
20	63,65	5	3	{1, 8, 13, 17, 18}	1
30	11,65	2	2	{8, 27}	8
30	11,65	2	3	{8, 27}	2
30	26,01	3	2	{8, 10, 12}	8
30	28,94	3	3	{1, 8, 27}	9
30	52,77	4	2	{1, 8, 10, 27}	11
30	52,77	4	3	{1, 8, 10, 27}	10
30	80,91	5	2	{1, 8, 10, 12, 27}	7
30	80,91	5	3	{1, 8, 10, 12, 27}	3

Conclusiones:

El algoritmo constructivo a partir de la solución vacía calculando el centroide suele dar resultados poco buenos, dado que dos nodos pueden estar muy separados del centroide pero cerca entre sí.

Los algoritmos GRASP también dan resutltados no tan buenos, aunque bastante mejor que los

anteriores, dado que, en este caso, se utiliza un GRASP pero construyendo la solución sin utilizar el centroide y partiendo de la solución con todos los nodos. La razón del resultado regular se debe a que una selección aleatoria en este problema en particular puede provocar que se llegue a un óptimo local de poca calidad.

El mejor algoritmo hasta hora ha sido el Greedy inverso, que tiene las ventajas del último GRASP implementado, con la diferencia que siempre se escoge el mejor nodo a quitar.

Tablas ramificación y poda.

Algoritmo greedy directo con selección de nodo con cota mas baja.

Tamaño (n)	Valor objetivo	m	Nodos generados	Tiempo (Milis)
15	11,85	2	106	7
15	27,37	3	355	5
15	49,82	4	1232	7
15	79,12	5	5058	19
20	8,51	2	191	10
20	21,99	3	888	9
20	40	4	3496	18
20	63,65	5	17323	58
30	11,65	2	1	5
30	28,94	3	412	22
30	52,77	4	8749	101
30	80,91	5	79482	263
15	13,27	2	106	7
15	31,86	3	343	6
15	59,76	4	1530	6
15	96,08	5	5343	29
20	11,8	2	172	11
20	30,87	3	487	6
20	56,69	4	3105	22
20	92,82	5	14481	110
30	13,07	2	56	11
30	34,29	3	433	10
30	63,7	4	9251	82
30	99,59	5	90263	306

Algoritmo greedy inverso con selección de nodo con cota mas baja.

Tamaño (n)	Valor objetivo	m	Nodos generados	Tiempo (Milis)
15	11,85	2	1	7
15	27,37	3	92	5
15	49,82	4	851	8
15	79,12	5	4000	25
20	8,51	2	1	9
20	21,99	3	172	9
20	40	4	2253	19
20	63,65	5	11294	99
30	11,65	2	1	17
30	28,94	3	412	21
30	52,77	4	8749	114
30	80,91	5	67605	270
15	13,27	2	16	8
15	31,86	3	128	5
15	59,76	4	1005	7
15	96,08	5	4039	24
20	11,8	2	4	9
20	30,87	3	229	8
20	56,69	4	2286	16
20	92,82	5	8432	53
30	13,07	2	1	1
30	34,29	3	418	21
30	63,7	4	8647	148
30	99,59	5	68125	253

Algoritmo GRASP inverso con post-procesamiento con selección de nodo con cota mas baja.

Tamaño (n)	Valor objetivo	m	Nodos generados	Tiempo (Milis)
15	11,85	2	1	9
15	27,37	3	92	6
15	49,82	4	851	10
15	79,12	5	5034	31
20	8,51	2	1	13

20	21,99	3	172	12
20	40	4	2190	18
20	63,65	5	17323	144
30	11,65	2	1	21
30	28,94	3	412	19
30	52,77	4	8749	147
30	80,91	5	107754	277
15	13,27	2	1	9
15	31,86	3	105	5
15	59,76	4	970	8
15	96,08	5	5343	30
20	11,8	2	1	12
20	30,87	3	229	9
20	56,69	4	2286	20
20	92,82	5	14716	70
30	13,07	2	1	21
30	34,29	3	407	18
30	63,7	4	8640	147
30	99,59	5	93444	260

Algoritmo Greedy directo con selección de nodo mas profundo.

Tamaño (n)	Valor objetivo	m	Nodos generados	Tiempo (Milis)
15	11,85	2	106	7
15	27,37	3	332	5
15	49,82	4	1144	7
15	79,12	5	3473	26
20	8,51	2	191	9
20	21,99	3	231	4
20	40	4	2038	16
20	63,65	5	9958	54
30	11,65	2	1	6
30	28,94	3	409	15
30	52,77	4	7177	122
30	80,91	5	57715	264
15	13,27	2	106	8

15	31,86	3	259	4
15	59,76	4	981	14
15	96,08	5	3186	27
20	11,8	2	172	12
20	30,87	3	504	5
20	56,69	4	2128	18
20	92,82	5	8134	97
30	13,07	2	56	12
30	34,29	3	16	10
30	63,7	4	481	79
30	99,59	5	59710	328

Algoritmo Greedy inverso con selección de nodo mas profundo.

Tamaño (n)	Valor objetivo	m	Nodos generados	Tiempo (Milis)
15	11,85	2	1	6
15	27,37	3	92	5
15	49,82	4	729	7
15	79,12	5	2928	25
20	8,51	2	1	10
20	21,99	3	172	12
20	40	4	1904	17
20	63,65	5	8603	55
30	11,65	2	1	17
30	28,94	3	409	17
30	52,77	4	7177	103
30	80,91	5	51192	291
15	13,27	2	16	8
15	31,86	3	141	6
15	59,76	4	766	7
15	96,08	5	2811	25
20	11,8	2	4	10
20	30,87	3	280	9
20	56,69	4	1850	15
20	92,82	5	7038	62

30	13,07	2	1	19
30	34,29	3	437	18
30	63,7	4	7262	101
30	99,59	5	51409	283

Algoritmo GRASP inverso con post-procesamiento con selección de nodo mas profundo.

Tamaño (n)	Valor objetivo	m	Nodos generados	Tiempo (Milis)
15	11,85	2	1	6
15	27,37	3	92	5
15	49,82	4	729	7
15	79,12	5	2919	25
20	8,51	2	4	14
20	21,99	3	172	10
20	40	4	1823	16
20	63,65	5	8603	66
30	11,65	2	1	20
30	28,94	3	409	17
30	52,77	4	7177	161
30	80,91	5	51192	229
15	13,27	2	1	9
15	31,86	3	92	6
15	59,76	4	737	8
15	96,08	5	2811	30
20	11,8	2	1	12
20	30,87	3	172	8
20	56,69	4	1850	15
20	92,82	5	7038	126
30	13,07	2	1	21
30	34,29	3	407	18
30	63,7	4	7232	94
30	99,59	5	59817	261

Conclusiones sobre la Ramificación y Poda:

Tal y como muestran las tablas de resultados, se obtiene una mayor eficiencia del algoritmo al

utilizar los algoritmos constructivos al inverso en la generación de la cota inferior. También, se puede observar que cuanto más sofisticado el algoritmos (por ejemplo, GRASP con post-procesamiento) menos nodos expanderá el árbol, aunque en muchas ocasiones el tiempo empleado para resolver el problema es mayor al que se necesitaría si no se utiliza un método de acotación inferior tan complejo. En definitiva, para este tipo de problemas, es conveniente utilizar algún método constructivo al inverso para acotar inferiormente, mientras que para expandir parece ser casi que casi no hay diferencia entre una estrategia de cota más pequeña frente a nodo más profundo.