Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Informe de la Práctica 8 de DAA
Curso 2023-2024

Maximum diversity problem

STEPHAN BROMMER GUTIÉRREZ

La Laguna, 7 de mayo de 2024

Resumen

Este informe aborda el desafío de optimización combinatoria en el problema de la diversidad máxima. El objetivo es encontrar el subconjunto de elementos de diversidad máxima de un conjunto dado de elementos.

Se han propuesto una serie de algoritmos para abordar este problema. Estos incluyen métodos constructivos (voraz), el algoritmo GRASP (Greedy Randomized Adaptive Search Procedure), búsqueda tabú con memoria a corto plazo, y un algoritmo de ramificación y poda.

Estos algoritmos se implementarán en C++, y se evaluarán utilizando instancias del problema proporcionadas en archivos de texto. Estos archivos contendrán información detallada sobre el conjunto de elementos y las coordenadas donde se situan.

Los resultados de estas implementaciones, presentados en tablas y gráficos, demostrarán la efectividad de los algoritmos en diferentes escenarios.

En resumen, el informe ofrecerá una descripción detallada de los algoritmos desarrollados, las estructuras de datos utilizadas y los resultados obtenidos, evidenciando su eficacia para resolver el problema de la diversidad máxima.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Contexto

En el ámbito de la informática y la ingeniería de sistemas, el diseño y análisis de algoritmos son áreas de estudio fundamentales. En este contexto, se han desarrollado diversas técnicas y métodos, como el algoritmo de ramificación y poda, los métodos constructivos (voraz), con su respectiva búsqueda local, y el Greedy Randomized Adaptive Search Procedure (GRASP).

1.1.1. Objetivos

Durante el desarrollo de esta práctica, se han cumplido los objetivos listados a continuación:

- Comprender y aplicar el concepto del problema de diversidad máxima
- Diseñar e implementar el algoritmo constructivo voraz, y la búsqueda local del mismo.
- Diseñar e implementar un algoritmo GRASP para el problema. Implementar sólo la fase constructiva.
- Diseñar e implementar el algoritmo de ramificación y poda con la estrategia de expandir el nodo más profundo y la estrategia de expandir el nodo con cota superior más pequeña.

  Nota: Destacar que en el GRASP se ha creado mediante multiarranque para intentar alcanzar una mejor solución.

1.2. Motivación

La motivación para realizar este trabajo proviene de la creciente necesidad de optimizar los procesos computacionales. Con el avance de la tecnología, los problemas de diversidad máxima se han vuelto cada vez más comunes, y con ellos, la necesidad de desarrollar algoritmos eficientes para su resolución. Además, el estudio de diferentes métodos de optimización puede abrir nuevas vías para mejorar el rendimiento de estos sistemas.

Capítulo 2

Maximum diversity problem

2.1. Descripción

En el problema de diversidad máxima se desea encontrar el subconjunto de elementos de diversidad máxima de un conjunto dado de elementos.

Sea dado un conjunto $S = \{s_1, s_2, ..., s_n\}$ de n elementos, en el que cada elemento s_i es un vector $s_i = (s_{i1}, s_{i2}, ..., s_{ik})$. Sea, asimismo, la distancia entre los elementos i y j. Si $m < n$ es el tamaño del subconjunto que se busca el problema puede formularse como:

$$\text{Maximizar } z = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d_{ij} x_i x_j$$

sujeto a:

$$\sum_{i=1}^{n} x_i = m$$
$$x_i \in \{0, 1\} \quad i = 1, 2, ...n$$

donde:

$$x_i = \begin{cases} 1 & \text{si } s_i \text{ pertenece a la solución} \\ 0 & \text{en caso contrario} \end{cases}$$

La distancia d_{ij} depende de la aplicación real considerada. En muchas aplicaciones se usa la distancia euclídea. Así:

$$d_{ij} = d(s_i, s_j) = \sqrt{\sum_{r=1}^{k} (s_{ir} - s_{jr})^2}$$

Capítulo 3

Algoritmos

3.1. Algoritmo constructivo voraz

Un constructivo voraz

Un algoritmo constructivo voraz muy sencillo para este problema parte del subconjunto, S, formado por las m tareas con menores valores de t_{0j} asignadas a los m arrays que representan la secuenciación de tareas en las máquinas. A continuación, añade a este subconjunto, iterativamente, la tarea-máquina-posición que menor incremento produce en la función objetivo. El pseudocódigo de este algoritmo se muestra a continuación.

Algoritmo constructivo voraz

1: Elem = S;
2: S = $\emptyset$;
3: Obtener s_c = centro(Elem);
4: **repeat**
5: Obtener el elemento $s_* \in$ Elem más alejado de s_c;
6: S = S $\cup$ s_*;
7: Elem = Elem - {s_*};
8: Obtener s_c = centro(S);
9: **until** ($|S|$==m)
10: Devolver S;

Figura 3.1: Algoritmo constructivo voraz

GRASP

El Algoritmo GRASP es un procedimiento de búsqueda adaptable, aleatorio y codicioso que se utiliza para resolver problemas de optimización combinatoria. Este algoritmo consta de dos fases principales: la Fase Constructiva y la Búsqueda Local.

Fase Constructiva: En esta fase, se genera una solución inicial de manera aleatoria pero guiada por un criterio codicioso. El proceso comienza con una solución vacía y en

cada iteración se añade un elemento a la solución. La elección del elemento se realiza de manera aleatoria pero dentro de una lista restringida de candidatos (RCL).

Búsqueda Local: Una vez generada la solución inicial, se aplica un algoritmo de búsqueda local para mejorarla. La búsqueda local explora las soluciones vecinas de la solución actual (es decir, todas las soluciones que se pueden obtener mediante una pequeña modificación de la solución actual) y se compara esta nueva solución óptima con la óptima anterior. Esto se realiza de manera iterativa poniendo una condición de parada como un contador. (Omitida por falta de tiempo y por existir en este problema una poca cantidad de soluciones posibles, es decir, que la solución óptima no estará tan lejos de la solución inicial.)

El Algoritmo GRASP es muy flexible y puede adaptarse a diferentes problemas.

Algoritmo GRASP

1: Dado un problema, un número de elementos en la solución y un número de iteraciones
2: Obtener las coordenadas y los índices del problema
3: Generar una solución inicial con la fase constructiva
4: Inicializar la mejor solución y la mejor función objetivo
5: Inicializar un contador a 0
6: **repeat**
7: Generar una nueva solución con la fase constructiva
8: Obtener la función objetivo de la nueva solución
9: If la nueva función objetivo mejora, actualizar la mejor solución y la mejor función objetivo
10: **until** (El contador llegue al número de iteraciones especificado)
11: Actualizar la solución y la función objetivo con los mejores valores encontrados
12: Devolver la mejor solución

Figura 3.2: Algoritmo GRASP

Búsqueda Tabú

La Búsqueda Tabú es una técnica heurística de optimización utilizada para encontrar soluciones aproximadas a problemas de optimización combinatoria. Se basa en realizar exploraciones en el espacio de búsqueda mientras se evitan soluciones previamente visitadas o ciertos movimientos que podrían conducir a soluciones subóptimas.

Principios Clave:

1. Memoria a Corto Plazo:
   La Búsqueda Tabú utiliza una lista tabú para almacenar temporalmente movimientos o soluciones exploradas recientemente. Esta lista tabú tiene un tamaño limitado (tenencia) y se actualiza dinámicamente durante la búsqueda.

2. Evitación de Ciclos:
   La memoria a corto plazo permite evitar ciclos en la búsqueda, impidiendo que el algoritmo se estanque en regiones del espacio de soluciones que ya han sido exploradas.

3. Diversificación y Exploración:
   A través de la lista tabú, la Búsqueda Tabú fomenta la diversificación de la búsqueda al forzar movimientos hacia nuevas regiones del espacio de soluciones, incluso si temporalmente parecen subóptimos.

4. Criterio de Aspiración:
   La Búsqueda Tabú puede incluir un criterio de aspiración que permite aceptar ciertos movimientos tabú si conducen a soluciones significativamente mejores.

BusquedaTabú

1: Dado un problema, un número de elementos en la solución y un número de iteraciones
2: Inicializar el algoritmo GRASP
3: Generar una solución inicial con el algoritmo GRASP
4: Obtener los elementos que no están en la solución inicial
5: Inicializar la mejor solución y la mejor función objetivo con la solución inicial
6: Inicializar un contador a 0
7: **repeat**
8: Aplicar la búsqueda tabú a la solución actual
9: Obtener la función objetivo de la solución actual
10: If la nueva función objetivo mejora, actualizar la mejor solución y la mejor función objetivo
11: Generar una nueva solución con el algoritmo GRASP
12: Obtener los elementos que no están en la nueva solución
13: **until** (El contador llegue al número de iteraciones especificado)
14: Actualizar la solución y la función objetivo con los mejores valores encontrados
15: Devolver la mejor solución

Figura 3.3: BusquedaTabú

Ramificación y Poda

El algoritmo de ramificación y poda es un método de búsqueda en árbol que se utiliza para resolver problemas de optimización combinatoria. Este algoritmo divide el problema en subproblemas (ramificación) y descarta aquellos que no pueden producir una mejor solución que la mejor conocida hasta el momento (poda).

Estrategia de Profundidad: En esta estrategia, el algoritmo explora el árbol de búsqueda a profundidad, es decir, explora primero los nodos hijos antes que los nodos

hermanos.

Estrategia de Expansión del Nodo con Cota Superior Más Pequeña: En esta estrategia, el algoritmo siempre expande el nodo con la cota superior más pequeña.

Cuando se llega a una solución completa, si la función objetivo de dicha solución es mejor que la de la cota inferior actual, dicha cota se actualiza.

En cuanto a las pruebas con soluciones iniciales mediante el algoritmo GRASP (Greedy Randomized Adaptive Search Procedure) y Voraz, estos son métodos heurísticos que se utilizan para obtener una solución inicial (cota inferior) que luego se mejora con el algoritmo de ramificación y poda. Cabe destacar, que ambas estrategias llegan siempre a la solución óptima.

Ramificación y Poda (Cota superior más pequeña)

1: Dado un problema, un número de elementos en la solución y un número de iteraciones
2: Generar una solucion inicial S mediante GRASP o Voraz (Representa cota inferior)
3: Calcular dMin y dMax para todos los puntos.
4: Implementar función de ramificación y poda (extendiendo nodo con cota superior más pequeña).
5: Devolver la mejor solución

Figura 3.4: Ramificación y Poda (Cota superior más pequeña)

Ramificación y Poda (Profundidad)

1: Dado un problema, un número de elementos en la solución y un número de iteraciones
2: Generar una solucion inicial S mediante GRASP o Voraz (Representa cota inferior)
3: Calcular dMin y dMax para todos los puntos.
4: Implementar función de ramificación y poda (extendiendo nodo más profundo).
5: Devolver la mejor solución

Figura 3.5: Ramificación y Poda (Profundidad)

Destacar también que se ha conseguido optimizar el cálculo de la función objetivo, al no calcular continuamente las distancias entre los diferentes puntos del espacio, sino una vez se ha calculado la matriz de distancias euclidianas, tras el intercambio de un punto de la solución con otro punto del exterior simplemente se realizan las restas sucesivas de las distancias de todos los puntos de la solución respecto al que punto que se ha extraido, y a continuación, las sucesivas sumas con el punto que se ha insertado.

Sucede lo mismo cuando no se realiza un intercambio, sino una única inserción de un punto en la solución, pero en dicho caso solo se realizan sumas sucesivas con ese nuevo punto.

Capítulo 4

Experimentos y resultados computacionales

Nota: Destacar que estos experimentos fueron realizados con 1000 iteraciones, excepto greedy que no es multiarranque.

4.1. Constructivo voraz

	Algoritmo voraz					
Problema	n	m	k	Ejecución	TCT	$CPU(microsegundos)$
max_div_15_2.txt	15	8	2	1	200.6026	422
max_div_15_3.txt	15	4	3	2	59.7638	243
max_div_20_2.txt	20	11	2	3	299.2226	646
max_div_20_3.txt	20	15	3	4	754.2476	699
max_div_30_2.txt	30	9	2	5	249.9406	1464
max_div_30_3.txt	30	6	3	6	141.9952	598

Cuadro 4.1: Algoritmo voraz. Tabla de resultados

4.2. GRASP

	GRASP					
Problema	n	m	k	Ejecución	TCT	$CPU(microsegundos)$
max_div_15_2.txt	15	8	2	1	201.6509	560624
max_div_15_3.txt	15	4	3	2	59.7638	292764
max_div_20_2.txt	20	11	2	3	295.8432	730886
max_div_20_3.txt	20	15	3	4	748.1408	1057920
max_div_30_2.txt	30	9	2	5	238.2712	733332
max_div_30_3.txt	30	6	3	6	137.7758	512459

Cuadro 4.2: GRASP. Tabla de resultados

4.3. Búsqueda Tabú

Nota: Destacar que la tenencia tabú utilizada ha sido de 5, 10 iteraciones del búcle del método de la búsqueda tabu, y si en 3 iteraciones seguidas no se encuentra una mejor solución, también finaliza el algoritmo.

	Búsuqeda Tabú					
Problema	n	m	k	Ejecución	TCT	$CPU(microsegundos)$
max_div_15_2.txt	15	8	2	1	202.655	1299669
max_div_15_3.txt	15	4	3	2	59.7638	974074
max_div_20_2.txt	20	11	2	3	299.2226	2148888
max_div_20_3.txt	20	15	3	4	754.2476	2217616
max_div_30_2.txt	30	9	2	5	252.5945	4454398
max_div_30_3.txt	30	6	3	6	142.5346	3529397

Cuadro 4.3: Búsuqeda Tabú. Tabla de resultados

4.4. Ramificación y Poda (Estrategia expandir nodo con cota superior más pequeña

Con solución inicial Voraz:

	Ramificación y Poda Cota superior expandida					
Problema	n	m	k	Ejecución	TCT	$CPU(microsegundos)$
max_div_15_2.txt	15	8	2	1	202.655	232647
max_div_15_3.txt	15	4	3	2	59.7638	12172
max_div_20_2.txt	20	11	2	3	299.2226	8546463
max_div_20_3.txt	20	15	3	4	754.2476	17758176
max_div_30_2.txt	30	9	2	5	252.5945	163468857
max_div_30_3.txt	30	6	3	6	142.5346	2874620

Cuadro 4.4: Ramificación y Poda Cota superior expandida. Tabla de resultados

Con solución inicial GRASP con 1000 iteraciones:

	Ramificación y Poda Cota superior expandida					
Problema	n	m	k	Ejecución	TCT	$CPU(microsegundos)$
max_div_15_2.txt	15	8	2	1	202.655	468703
max_div_15_3.txt	15	4	3	2	59.7638	130150
max_div_20_2.txt	20	11	2	3	299.2226	9420882
max_div_20_3.txt	20	15	3	4	754.2476	18456866
max_div_30_2.txt	30	9	2	5	252.5945	227809289
max_div_30_3.txt	30	6	3	6	142.5346	5210191

Cuadro 4.5: Ramificación y Poda Cota superior expandida. Tabla de resultados

4.5. Ramificación y Poda (Estrategia expandir nodo más profundo

Con solución inicial Voraz:

	Ramificación y Poda Profundidad					
Problema	n	m	k	Ejecución	TCT	$CPU(microsegundos)$
max_div_15_2.txt	15	8	2	1	202.655	22273
max_div_15_3.txt	15	4	3	2	59.7638	991
max_div_20_2.txt	20	11	2	3	299.2226	893538
max_div_20_3.txt	20	15	3	4	754.2476	2512640
max_div_30_2.txt	30	9	2	5	252.5945	11703052
max_div_30_3.txt	30	6	3	6	142.5346	198320

Cuadro 4.6: Ramificación y Poda Profundidad. Tabla de resultados

Con solución inicial GRASP con 1000 iteraciones:

	Ramificación y Poda Profundidad					
Problema	n	m	k	Ejecución	TCT	$CPU(microsegundos)$
max_div_15_2.txt	15	8	2	1	202.655	258287
max_div_15_3.txt	15	4	3	2	59.7638	124996
max_div_20_2.txt	20	11	2	3	299.2226	1275312
max_div_20_3.txt	20	15	3	4	754.2476	3046746
max_div_30_2.txt	30	9	2	5	252.5945	12149063
max_div_30_3.txt	30	6	3	6	142.5346	432782

Cuadro 4.7: Ramificación y Poda Profundidad. Tabla de resultados

4.6. Análisis compartativo entre algoritmos

1. **Algoritmo voraz**: Este algoritmo es eficiente en términos de tiempo de CPU, lo que indica que es rápido. Sin embargo, su valor de TCT no es el más alto en todos los casos, lo que sugiere que puede no encontrar la solución óptima. A pesar de esto, a menudo encuentra la solución óptima, lo que indica que es una opción viable para problemas de maximización.

2. **GRASP (Greedy Randomized Adaptive Search Procedure)**: Este algoritmo tiene un tiempo de CPU significativamente más alto que el algoritmo voraz, lo que indica que es más lento, debido al multiarranque. Sin embargo, podría alcanzar una mejor solución que el algoritmo voraz, pero por falta de tiempo no ha podido ser implementado. El rendimiento de GRASP podría mejorar con más tiempo o con la implementación de una búsqueda local efectiva.

3. **Búsqueda Tabú**: Este algoritmo tiene uno de los tiempos de CPU más altos. Sin embargo, siempre logra el valor de TCT más alto, lo que indica que siempre encuentra la solución óptima. Esto lo convierte en la mejor opción para problemas de maximización, a pesar de su mayor costo computacional.

4. **Ramificación y Poda**: En ambas estrategias se consigue la solución óptima. Aunque cabe destacar que la estrategia en profundidad suele ser mucho más directa y rápida que la de extender el nodo con cota superior más pequeña, esto se debe a que esta última estretgia explora una mayor cantidad de ramas en los diferentes niveles. Además, se ha conseguido no repetir ninguna rama del árbol gracias a la fórmula general $n-(m-k)-i+1$ que indica la cantidad de hijos que tiene que generar cada nodo padre, siendo esta fórmula específica para el nodo raíz $m-m+1$, extraida del artículo *A branch and bound algorithm for the maximum diversity problem*

En resumen, se produce una compensación entre el tiempo de CPU y el valor de TCT. El algoritmo voraz es el más rápido pero puede no encontrar la solución más óptima, mientras que la Búsqueda Tabú y la Ramificación y Poda siempre encuentran la solución óptima pero pueden ser más lentos. GRASP parece ser un compromiso entre los dos, con la posibilidad de mejorar con más tiempo o una búsqueda local efectiva.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

En este informe, se abordó el desafío de optimización combinatoria mediante el estudio del problema de la diversidad máxima. Se propusieron y desarrollaron varios algoritmos para resolver este problema, incluyendo métodos constructivos (voraces), el algoritmo GRASP, búsqueda tabú con memoria a corto plazo y un enfoque de ramificación y poda.

Algoritmos Implementados y Resultados

- Algoritmo Constructivo Voraz: Se diseñó e implementó un algoritmo voraz que parte de un subconjunto inicial vacío y al cuál se le agregan iterativamente elementos que maximizan la función objetivo.
- GRASP (Greedy Randomized Adaptive Search Procedure): Se desarrolló la fase constructiva de GRASP, generando soluciones iniciales aleatorias pero guiadas por la construcción de la LRC.
- Búsqueda Tabú: Se aplicó una técnica de búsqueda tabú para explorar soluciones aproximadas. Esta estrategia incluyó un enfoque de memoria a corto plazo para evitar ciclos y fomentar la exploración diversificada.
- Ramificación y Poda: Se implementaron dos estrategias de ramificación y poda para abordar el problema, expandiendo nodos basados en diferentes criterios (cotas superiores y profundidad).

Resultados y Análisis Comparativo

Se realizaron experimentos computacionales utilizando instancias del problema proporcionadas en archivos de texto. Los resultados fueron presentados en tablas y gráficos para demostrar la eficacia relativa de cada algoritmo en diferentes escenarios.

5.2. Trabajo Futuro

- Optimización de Algoritmos: Se podría explorar la optimización de los algoritmos implementados mediante técnicas avanzadas, como la paralelización o el uso de estructuras de datos más eficientes.

- Exploración de Nuevas Heurísticas: Investigar el desarrollo de nuevas heurísticas adaptativas o metaheurísticas que puedan mejorar la calidad de las soluciones encontradas para el problema de la diversidad máxima.
- Aplicaciones Prácticas: Estudiar aplicaciones prácticas del problema de la diversidad máxima en contextos específicos, como la optimización de recursos en redes de comunicación, logística o diseño de experimentos.
- Evaluación en Grandes Conjuntos de Datos: Extender la evaluación de los algoritmos a conjuntos de datos más grandes y complejos para comprender mejor su escalabilidad y comportamiento en situaciones realistas.

En conclusión, este informe proporciona una visión detallada de varios enfoques algorítmicos para resolver el problema de la diversidad máxima. Los resultados experimentales muestran que cada método tiene sus propias fortalezas y debilidades en términos de eficiencia y calidad de las soluciones encontradas. Se identificaron diversas áreas para futuras investigaciones y mejoras, lo que demuestra el potencial continuo de este campo en la optimización combinatoria.

Bibliografía

[1] Rafael Martí, Micael Gallego, Abraham Duarte. A branch and bound algorithm for the maximum diversity problem .

[2] Ramificación y poda. Ejemplo Problema de Asignación.

[3] RamificaciónPoda. Problema de la mochila.