DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Heurísticas y Metaheurísticas

CIDM

Algoritmos y Estructuras de Datos III
Primer Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Barañao, Facundo	480/11	facundo_732@hotmail.com
Confalonieri, Gisela Belén	511/11	gise_5291@yahoo.com.ar
Mignanelli, Alejandro Rubén	609/11	minga_titere@hotmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción

En el presente trabajo, se pretende analizar y comparar diferentes maneras de encarar el problema de *Conjunto Independiente Dominante Mínimo (CIDM)*, el cual consiste en hallar un conjunto independiente dominante de un grafo, con mínima cardinalidad. En particular se utilizará un algoritmo exacto, una heurística golosa, una heurística de busqueda local, y la metahuristica de GRASP.

A continuación se presentan algunas definiciones que nos serán útiles para comprender y abordar el problema:

- Sea $G = (V, E)$ un grafo simple. Un conjunto $D \subseteq V$ es un conjunto dominante de G si todo vértice de G está en D o bien tiene al menos un vecino que está en D. Por otro lado, un conjunto $I \subseteq V$ es un conjunto independiente de G si no existe ningún eje de E entre dos vértices de I. Definimos entonces un conjunto independiente dominante de G como un conjunto independiente que a su vez es un conjunto dominante del grafo G.
- Un conjunto independiente de $I \subseteq V$ se dice maximal si no existe otro conjunto independiente $J \subseteq V$ tal que $I \subset J$, es decir tal que I está incluido estrictamente en J.

2. Relación con trabajos previos y aplicaciones

En el TP1 de la materia, hemos encontrado y analizado un algoritmo que resolvía un problema que fue llamado **El Señor de los Caballos**. El problema era el siguiente:

Se tiene un juego de mesa cuyo tablero, dividido en casillas, posee igual cantidad de filas y columnas y hace uso de una conocida pieza del popular ajedrez: el caballo. El juego es solamente para un jugador y consiste en, teniendo caballos ubicados en distintos casilleros, insertar en casilleros vacíos la mínima cantidad de caballos extras, de manera tal que, siguiendo las reglas del movimiento de los caballos en el ajedrez, todas las casillas se encuentren ocupadas o amenazadas por un caballo. Aspectos a tener en cuenta:

- Se conoce la cantidad de filas y columnas del tablero.
- Se conoce la cantidad de caballos que ocupan el tablero inicialmente.
- Para cada uno de estos caballos, se sabe su ubicación en el tablero.
- Una casilla se considera amenazada si existe un caballo tal que en una movida pueda ocupar dicha casilla.

Observemos que si consideramos a cada casilla como un nodo, y que dos nodos son adyacentes cuando un caballo puede llegar de uno a otro con un movimiento, entonces **El Señor de los Caballos** puede ser visto como la búsqueda de un conjunto dominante minimal que tenga a los nodos/casillas que tienen un caballo preubicado. Sin embargo, no podemos traducir este problema a un CIDM, puesto que no necesariamente debe cumplir el requisito de independencia. O sea, si la solución tiene un caballo en una casilla determinada, no necesariamente ocurre que no haya ningún caballo ubicado en alguna de las casillas que este caballo amenaza. Veamos un ejemplo:

Si nuestro tablero fuera el de la Figura 1, una solución posible (de hecho, la encontrada con el algoritmo propuesto en el TP1) es la Figura 2. Esta solución podemos observar que es dominante, pero no independiente puesto que el caballo de la fila 3 - columna 2 amenaza al de la fila 4 - columna 4. Sin embargo en este caso, existe una solución CIDM como se muestra en la Figura 3.

(Dejo la parte de trabajos previos hasta aca, puesto que hay un ejemplo en el txt que quiero probar si funciona y agregaría un par de líneas)

Como por lo visto hasta ahora CIDM no se adapta del todo al problema de los caballos(asumo eso por ahora, desp de última se cambia(ALE)) cabe preguntarse, en qué situaciones de la vida real podría aplicarse este problema. Veamos algunos ejemplos realistas y alguno no tanto.

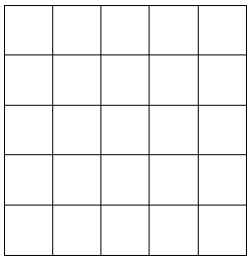

Figura 1: Tablero vacío de 5×5

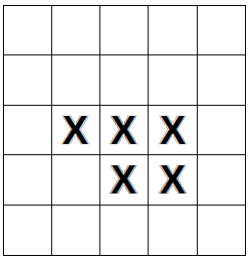

Figura 2: Sol. no independiente

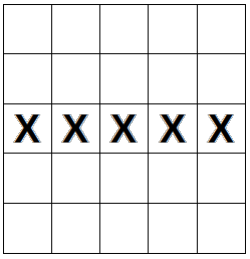

Figura 3: Solución independiente

Gaseosas: Una empresa de bebidas gaseosas tiene sus actividades divididas en áreas, las cuales se interrelacionan de cierta manera (no necesariamente todas con todas). Esta empresa está por sacar una nueva bebida sabor cola y quiere hacerlo antes que su rival, quien está más adelantado en la producción. Para lograr sacar el producto antes que su contrincante, nuestra empresa debe lograr una mejora en el trabajo de sus diferentes áreas. Por cómo está estructurada la empresa, si un área tiene una mejora considerable en tiempo, todas las áreas que se relacionan con ella se verán beneficiadas, pero este beneficio ya no impacta en las áreas relacionadas con estas últimas. Queremos entonces lograr impulsar mejoras considerables en las áreas que sean necesarias para que toda la empresa funcione más rápido y logre sacar su producto a tiempo. Para ello se contratarán especialistas con sueldos sumamente importantes, por lo cual se desea que la cantidad de áreas a mejorar considerablemente sea mínima (para minimizar la inversión en especialistas). Además, sabemos que estos especialistas son excelentes, pero muy soberbios/tercos, y ponerlos a cargo de áreas relacionadas puede devenir en discusiones que retrasarían a la empresa, por lo cual procuraremos colocarlos en áreas ”no adyacentes”.

Detectives: Se tiene un caso de asesinato, y debemos resolverlo. Nuestras investigaciones previas nos han dado información de todas las personas que tuvieron algo que ver con el incidente, y sabemos cuáles se conocen entre sí y cuáles no, además de que sabemos que toda persona que conoce a otra, sabe lo que hizo dicha persona el día del incidente. Debido a que somos detectives principiantes, el alquiler por día de nuestra oficina nos sale caro, y entrevistar a un testigo, independientemente del volumen de información obtenida, nos toma un día. Si para resolver el caso debiéramos escuchar información sobre todas las personas involucradas, ¿a quienes deberíamos llamar de testigos, de manera tal que reduzcamos el alquiler de oficina al mínimo? (Estos testigos no pueden conocerse entre sí, para evitar complot y falsos datos).

Caballeros del Zodíaco: Milo de Escorpio[1], guardián de la casa de Escorpio, ha recibido muchas quejas de parte del gran patriarca, debido a que por su casa pasa todo el mundo. Cansado de ser tantas veces derrotado, nos pide ayuda, y para esto nos explica la verdad sobre su gran técnica, la aguja escarlata. Lejos de lo que se cuenta en el animé *Los caballeros del zodíaco*[2], el verdadero poder de la aguja escarlata es el siguiente:

Milo nos explica que el cuerpo de cada ser humano puede ser dividido en sectores energéticos. Estos sectores energéticos pueden tener una correspondencia entre sí, las cuales no son necesariamente a nivel local (por ejemplo, una porcion del dedo indice de un humano, puede estar conectada a una porción de la cabeza). Cuando la aguja escarlata toca un sector energético de un cuerpo, éste y todos aquéllos sectores que tengan una correspondencia, se inmovilizan. Pero esto sucede sólo si la aguja escarlata impacta contra un sector energético “sano”. Si el sector en cuestión ya estuviese inmovilizado, entonces la aplicación de la aguja escarlata no hace ningun efecto.

Dado que el uso de una aguja escarlata, resulta en un fuerte uso del cosmos de Milo, se nos pide que, dado un oponente y la información sobre todos sus sectores energéticos y correspondencias, nosotros le fabriquemos un algoritmo tal que le diga en qué sectores del cuerpo debe usar la aguja escarlata, de manera tal que deba utilizar la mínima cantidad de agujas escarlata posibles. Milo nos asegura que el

[1] http://es.wikipedia.org/wiki/Milo_de_Escorpio
[2] http://es.wikipedia.org/wiki/Saint_Seiya

tiempo de dicho algoritmo no tiene importancia, dado que le pidió prestada la habitación del tiempo[3] a Kamisama[4], que está equipada con una notebook y un enchufe para dejar la bateria cargada (eso sí, no tiene wifi dado a una muy mala señal, por lo que debemos darle un pendrive con el código).

3. Propiedades

En esta sección demostraremos ciertas propiedades que serán usadas para elaborar una conclusión que nos ayudará a abordar el problema propuesto.

3.1. Todo conjunto independiente maximal es dominante.

Lo probaremos por absurdo. Supongamos que existe algún grafo $G = (V, E)$ tal que tiene un conjunto independiente maximal C que no es un conjunto dominante. Como C no es dominante, entonces existe un nodo $v \in V$ tal que $v \notin C$, y que además dentro del grafo original, v no es vecino de ningún elemento de C. Pero entonces, si añadimos a v a C se obtiene un conjunto $C' = C + v$ que es independiente, o sea que $C \subset C'$, lo cual es absurdo, puesto que habíamos dicho que C era maximal. El absurdo proviene de suponer que el conjunto no es dominante, por lo tanto, el conjunto debe ser dominante.

3.2. Todo conjunto independiente dominante es independiente maximal.

Lo probaremos por absurdo. Supongamos que existe un grafo $G = (V, E)$ tal que tiene un conjunto independiente dominante I que no es independiente maximal. Como I no es independiente maximal, podemos suponer que existe algún conjunto $J \subseteq V$ independiente, tal que $I \subset J$. Entonces, existe v un nodo que pertenece a J pero no a I, tal que v no es vecino de ningún elemento de I, lo cual es absurdo, ya que suponer eso es decir que I no era dominante.

3.3. Conclusión

Por las propiedades anteriormente demostradas, se puede concluir que el problema de buscar un CIDM es idéntico al problema de buscar un conjunto independiente maximal mínimo(CIMM), o sea, de todos los conjuntos independientes maximales que son posibles formar dado un grafo cualquiera, aquel cuya cardinalidad es la menor. Por esta razón, nuestros algoritmos buscarán encontrar o aproximar un CIMM para un grafo dado.

4. Plataforma de pruebas

Para toda la experimentación se utilizará un procesador Intel Core i3, de 4 nucleos a 2.20 GHZ.
El software utilizado será Ubuntu 14.04, y G++ 4.8.2.

5. Algoritmo Exacto

5.1. Desarrollo de la idea y correctitud.

5.2. Análisis de complejidad.

[3] http://es.dragonball.wikia.com/wiki/Habitacion_del_Tiempo
[4] http://es.wikipedia.org/wiki/Kamisama_(personaje)

CIDM_EXACTO($lista\_nodos\ cidm, lista\_nodos\ cidm\_sol, nodo, int\ n, int\ cota, int\ res$)

```
 1  if se superó la cota
 2      return
 3  if se encontró una solución
 4      cidm_sol ← cidm
 5      cota ← res
 6      return
 7  if se llegó al final y no se encontró una solución
 8      return
 9  if nodo no está "tomado"
10      cidm ← agregar nodo
11      incrementar res
12      marcar a nodo y a sus vecinos como "tomados"
13      CIDM_EXACTO(cidm, cidm_sol, nodo_siguiente, n, cota, res)
14  if nodo tiene vecinos ó todavía no se llegó a una solución
15      if se modificó en la rama anterior
16          cidm ← sacar nodo
17          decrementar res
18          marcar a nodo y a sus vecinos como "no tomados"
19      CIDM_EXACTO(cidm, cidm_sol, nodo_siguiente, n, cota, res)
```

Figura 4: Algoritmo exacto para CIDM

5.3. Experimentación y gráficos.

5.3.1. Test 1

5.3.2. Test 2

```
CIDM_GOLOSO(lista_nodos cidm_sol)
1  res ← 0
2  while no se hayan "tomado" todos los nodos
3        elegido ← nodo "óptimo"
4        cidm_sol ← agregar elegido
5        incrementar res
6        marcar a elegido y a sus vecinos como "tomados"
7  return res
```

Figura 5: Heurística golosa constructiva para CIDM

6. Heurística Golosa Constructiva

6.1. Desarrollo de la idea.

Sea G un grafo cualquiera, I un conjunto independiente de ese nodo, y n_1,n_2 nodos de G que no pertenecen a I y no tienen aristas en común con ningún elemento de I, diremos que n_1 es óptimo si no existe ningún n_2 tal que $(\#(Vecinos(n_2)) - \#(Vecinos(n_2) \cap Vecinos(I))) > (\#(Vecinos(n_1)) - \#(Vecinos(n_1) \cap Vecinos(I)))$. Definimos $Vecinos(\alpha)$ como el conjunto de nodos a los cuales α lleva una arista si α es un nodo, y si α es un conjunto, entonces es el conjunto de nodos a los cuales les llega un arista desde por lo menos un elemento de α.

Nuestra heurística se basa en formar un conjunto independiente maximal de la siguiente manera: Primero, considerando al conjunto independiente vacío, tomamos a un nodo óptimo, y lo agregamos como nuevo elemento de nuestro conjunto independiente. Luego con nuestro nuevo conjunto independiente, tomamos un nodo óptimo, y lo agregamos a nuestro conjunto independiente. Repetimos esto hasta que no exista un nodo óptimo, puesto que nuestro conjunto independiente se transformó en maximal, y por lo tanto en un conjunto dominante.

6.2. Análisis de complejidad.

6.3. Instancias no óptimas.

6.4. Experimentación y gráficos.

6.4.1. Test 1

6.4.2. Test 2

```
CIDM_BUSQUEDA(int mej)
1  cidm_sol ← lista vacía de nodos
2  res ← CIDM_GOLOSO(cidm_sol)
3  while haya mejoras y la última solución tenga más de un nodo
4       if mej == 1
5            MEJORADOR1(cidm_sol, res)
6       else MEJORADOR2(cidm_sol, res)
```

Figura 6: Heurística de búsqueda local para CIDM

```
MEJORADOR1(lista_nodos cidm_sol, int res)
 1  for cada par de nodos en cidm_sol
 2       "sacar" ambos nodos
 3       for cada vecino n de estos nodos
 4            if n quedó "desconectado"
 5                 "agregar" n
 6                 if se forma una solución válida
 7                      salir del ciclo
 8            if se encontró una solución mejor
 9                 actualizar cidm_sol
10                 actualizar res
11                 return
12  return
```

Figura 7: Pseudocódigo de la mejora 1

7. Heurística de Busqueda Local

7.1. Desarrollo de la idea y correctitud.

7.2. Análisis de complejidad de una iteración.

7.3. Experimentación y gráficos.

7.3.1. Test 1

7.3.2. Test 2

```
MEJORADOR2(lista_nodos cidm_sol, int res)
 1  for cada nodo n
 2      if n se conecta con al menos dos nodos de cidm_sol
 3          for cada nodo de cidm_sol que se conectan n
 4              "sacar" el nodo
 5              if se forma una solución válida
 6                  salir del ciclo
 7          if se encontró una solución mejor
 8              actualizar cidm_sol
 9              actualizar res
10              return
11  return
```

Figura 8: Pseudocódigo de la mejora 2

8. Metaheurística de GRASP

8.1. Desarrollo de la idea y correctitud.

8.2. Experimentación y gráficos.

8.2.1. Test 1

8.2.2. Test 2

9. Comparación de los distintos métodos

9.1. Experimentación y gráficos.

9.1.1. Test 1

9.1.2. Test 2

10. Apéndice 1: acerca de los tests

10.1. Código del Problema 1

10.2. Código del Problema 2

10.3. Código del Problema 3