DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico III

Algoritmos y Estructuras de Datos III
Primer Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Aldasoro Agustina	86/13	agusaldasoro@gmail.com
Noriega Francisco	660/12	frannoriega.92@gmail.com
Zimenspitz Ezequiel	155/13	ezeqzim@gmail.com
Zuker Brian	441/13	brianzuker@gmail.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

Resumen

Resumen va aquí

Índice

1. Introducción al problema

1.1. Conjunto Independiente Dominante Mínimo (CIDM)

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.

El problema de Conjunto Independiente Dominante Mínimo (CIDM) consiste en hallar un conjunto independiente dominante de G con mínima cardinalidad.

1.2. Paralelismo con "El señor de los caballos"

El problema "El señor de los caballos" es similar al problema de encontrar un Conjunto Independiente Dominante Mínimo en la familia de grafos en la que cada nodo modela un casillero en un tablero de ajedrez, y cada eje modela las posiciones a las que un caballo puede moverse desde determinado casillero.

Insertar imagen del grafo tablero

La relación con nuestro problema es la siguiente:

- "El señor de los caballos" busca un conjunto dominante, dado que intenta ocupar el tablero, y para ello requiere que o bien cada casillero tenga un caballo, o bien que cada casillero sea amenazado por un caballo.
- "El señor de los caballos" busca un conjunto mínimo, es decir, que utilice la menor cantidad de caballos posibles.
- "El señor de los caballos" **NO** busca un conjunto independiente, dado que si fuese necesario, un caballo puede ubicarse en un casillero que estuviese siendo amenazado por otro caballo.

1.3. Todo conjunto independiente maximal es un conjunto dominante

Sea $G = (V, E)$ un grafo simple, 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 que I está incluido estrictamente en J.

Todo conjunto independiente maximal es un *conjunto dominante*.

Demostración

Sean $G = (V, E)$ grafo simple, $I \subseteq V$ *conjunto independiente maximal*.

Quiero ver que I es un *conjunto dominante*:
Lo que es equivalente a probar que $(\forall\ nodo\ v \in V)\ ((v \in I) \vee (\exists\ nodo\ w \in adyacentes(v), w \in I))$

Supongo por el absurdo que: $(\exists\ nodo\ v \in V)\ tq\ ((v \notin I) \wedge (\forall\ nodo\ w \in adyacentes(v), w \notin I))$

Considero a $adyacentes(I)$ como el conjunto que se obtiene de concatenar todos los vecinos de cada elemento de I. Por lo tanto, es equivalente decir $(\forall\ nodo\ w\ \in adyacentes(v), w\ \notin\ I)$ y decir $(v \notin adyacentes(I))$.

$\Rightarrow v \notin I \wedge v \notin adyacentes(I)$
$\Rightarrow \exists$ conjunto independiente J: $J \subseteq V$ tq $J = I + \{v\}$

J es independiente porque I lo era y al agregarle el nodo v se mantiene esta propiedad ya que v no pertenecía a I y además no estaba conectado a ningún nodo del conjunto I.

$\Rightarrow \exists J$ conjunto independiente tq $I \subset J$. **Absurdo!**(I era un conjunto Independiente Maximal)

El absurdo provino de suponer que I era un conjunto independiente maximal, pero no dominante. Por lo tanto, I debe ser un conjunto dominante.

se fijan que onda?

1.4. Situaciones de la vida real

Situaciones de la vida real que puedan modelarse utilizando CIDM:

- **Ubicación de estudiantes al momento de rendir un examen:** Encontrar la mejor manera de ubicar a todos los estudiantes en el aula, tal que ninguno este suficientemente cerca de otro como para copiarse, pero tal que entre la mayor cantidad de estudiantes posibles, es lo mismo a modelar el aula como un grafo donde cada asiento es un nodo, y donde cada asiento es adyacente a los asientos que estan a sus costados (en todas las direcciones), y buscar el CIDM de dicho grafo.
- **Ubicación de servicios en ciudades:** Para minimizar costos, es probable que si se quiere situar centros de servicios (cualesquiera sean estos: hospitales, estaciones de servicio, distribuidoras, etc), se los situe de manera tal que tengan amplia cobertura, pero sin situar demasiados centros, es decir, situando la minima cantidad. Tampoco se querría que un centro cubra la misma zona que otro centro. Si se modela a la ciudad, tomando cada zona (arbitraria, como barrios, o manzanas, o conjunto de manzana) como un nodo, en los que cada nodo es adyacente al nodo que representa la zona vecina, entonces el problema de situar estos centros minimizando costos, es igual a encontrar un CIDM en el grafo mencionado.

Otra vez Noriega salvando las papas del fuego.. lo del aula en el examen y eso

2. Algoritmo Exacto

De acuerdo a lo ya explicado en el inciso 1.2, podemos establecer una analogía con este problema y "El señor de los caballos". Por lo tanto, la metodología empleada para la implementación del algoritmo exacto también fue la de *Backtracking*.

De este modo, nos vemos obligados a recorrer inteligentemente todos los conjuntos dentro del conjunto de partes del total de nodos V. Mediante el backtracking podemos realizar podas y estrategias para saltear algunas ramas de decisión donde se predice que no se va a poder encontrar la solución óptima allí.

2.1. Explicación y mejoras

Nuestro algoritmo recorre ordenadamente el conjunto de partes de V y por cada uno de ellos verifica que cumpla la función `esIndependienteMaximal()`. La misma devuelve 0 si es falso, la cantidad de nodos en el conjunto en caso contrario.

Es decir, se itera sobre los nodos y se pide el minimo conjunto independiente maximal considerando el nodo actual presente y ausente.

En una variable se acumula la solución óptima hasta el momento, la cual se actualiza cuando se encuentra un nuevo conjunto independiente maximal que tiene un cardinal menor al óptimo actual. En ella va a quedar la solución buscada luego de correr el algoritmo.

Las podas que implementamos fueron ... completar
Y las estrategias fueron ... completar

Poner el pseudocodigo...

```
unsigned int calcularCIDM(Matriz& adyacencia, unsigned int i, vector<unsigned int>& conjNodos,
vector<unsigned int>& optimo){

if (conjNodos.esIndependienteMaximal()) then
    optimo ⟵ conjNodos;
    return optimo.size();
end
if (i.estaEnRango()) then
    if (conjNodos es más grande que el optimo actual) then
        return 0;
    end
    siNoAgrego ⟵ calcularCIDM(adyacencia, i+1, conjNodos, optimo);
    agrego el nodo i a conjNodos;
    siAgrego ⟵ calcularCIDM(adyacencia, i+1, conjNodos, optimo);
    elimino el nodo i de conjNodos;
    return el mejor entre siNoAgrego y siAgrego;
end
return 0;
```

Algorithm 1: algoritmo exacto

2.2. Complejidad Temporal

2.3. Experimentación

3. Heurística Constructiva Golosa

3.1. Explicación

3.2. Complejidad Temporal

3.3. Comparación de resultados con solución óptima

3.4. Experimentación

4. Heurística de Búsqueda Local

4.1. Explicación

4.2. Complejidad Temporal

4.3. Experimentación

5. Metaheurística GRASP

5.1. Explicación

La metaheurística *Greedy Randomized Adaptive Search Procedure* (**GRASP**), es una mezcla de las dos heurísticas previas (vistas en 3 y 4). Dicho de manera simple: genera un punto de partida de forma golosa para el algoritmo de búsqueda local.

La distinción de este algoritmo radica en cómo se construye "*golosamente*" la solución inicial.

Como la sigla lo indica, consiste en un algoritmo *Goloso Randomnizado*. Es decir que se escogen candidatos a solución inicial de forma golosa, pero en vez de elegir al que mejor resultados nos pueda arrojar, elegimos alguno de los que mejor lo hará. Particularmente optamos por elegir un nodo entre los α % mejores.

Entonces, si nuestro grafo tiene un nodo de grado máximo mayor estricto a los demás, no necesariamente lo elegiremos.

Luego se aplica el algoritmo de búsqueda local explicado en el inciso anterior sin modificaciones.

Un aspecto también diferencial de esta heurística, es que no generamos una única instancia inicial, sino que se toman un número arbitrario de ellas (según el criterio de parada) y guardamos la solución óptima que encontramos, si en algún momento se mejora, se actualiza.

La RCL (Restricted Candidate List) varía según el $alpha$ elegido (si se elige 0, será un greedy), a mayor $alpha$, mayor diversidad de candidatos iniciales.

Las vecindades utilizadas son las mismas que se usan en la heurísitca de búsqueda local.

El criterio de parada que adoptamos fue el de llegar a una determina cantidad de repeticiones del algoritmo, búsqueda de soluci]'on inicial y búsqueda local a partir de ella, tal que la respuesta óptima encontrada, no se viera modificada durante todas ellas.

5.2. Experimentación

6. Comparación entre todos los métodos

Una vez elegidos los mejores valores de configuracion para cada heurıstica implementada (si fue posible), realizar una experimentacion sobre un conjunto nuevo de instancias para observar la performance de los metodos comparando nuevamente la calidad de las soluciones obtenidas y los tiempos de ejecucion en funcion de los parametros de la entrada. Para los casos que sea posible, comparar tambien los resultados del algoritmo exacto implementado. Presentar todos los resultados obtenidos mediante graficos adecuados y discutir al respecto de los mismos.