DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico Nro. 3

26/6/2009 Algoritmos y Estructuras de Datos III

Integrante	LU	Correo electrónico
Dinota, Matías	076/07	matiasgd@gmail.com
Huel, Federico Ariel	329/07	federico.huel@gmail.com
Leveroni, Luciano	360/07	lucianolev@gmail.com
Mosteiro, Agustín	125/07	agustinmosteiro@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

Aclaraciones generales

Antes de comenzar el análisis de los algoritmos, cabe mencionar lo siguiente:

- La implementación de los todos algoritmos se realizó en **lenguaje Java**, haciendo uso de las librerías estándar del mismo.
- Para el cálculo de tiempo de los algoritmos se utilizó la función **nanoTime()** de la clase System de Java. Con el fin de aumentar la precisión de las mediciones, se utilizó el comando **nice** para darle máxima prioridad a la tarea.
- El código fuente de los algoritmos aquí analizados se encuentran en el archivo *ResolvedorCIPM.java*.
- El código fuente de los programas encargados de hacer uso de los algoritmos y necesarios para compilar la aplicaciones son: MainCIPM.java, ResolvedorCIPM.java, GrafoNPonderados.java, PesoNodoComparator.java, LectorDeGrafos.java y EscritorDeSoluciones.java.
- Para la lectura y escritura de los datos se utilizaron clases provistas por el lenguaje Java. No se hará referencia a estos algoritmos ya que no resultan de interés para el trabajo aquí presentado.
- Los gráficos se realizaron con **GNUPlot** y las tablas con OpenOffice Calc. En los casos considerados pertinentes, se utilizó una escala logarítmica con el fin de poder visualizar mejor los resultados.

Con respecto al análisis de resultados y a las pruebas realizadas vale hacer la siguiente aclaración. Los grafos generados aletoriamente poseerán nodos con pesos comprendidos entre 10 y cantidad de nodos del grafo por 10. Las adyacencias de los nodos es generada también aleatoriamente de modo que el grado de cada nodo sea un número entre 0 y (n-1)*k, con k un parámetro que llamaremos *densidad* de aquí en adelante, donde $0 < k \leq 1$. De este modo, los experimentos mostrarán el comportamiento de los algoritmos para grafos "promedio" (sin particularidades) con una cierta cantidad de nodos y de cierta densidad.

1. Introducción

El objetivo del siguiente trabajo es presentar diversos métodos para encontrar soluciones exactas y aproximadas para el problema del Conjunto Independiente de Peso Máximo (CIPM). En particular, se desarrollará un algoritmo que utiliza la técnica de *backtracking* para resolver el problema de manera exacta para instancias pequeñas del mismo. Además, se implementarán heurísitcas constructivas, de búsqueda local y una metaheurística GRASP, la cual será el foco principal de nuestro estudio.

Para cada tipo de heurística (constructivas y de búsqueda local), se realizarán pruebas para determinar cual es la más eficaz y, en caso de ser posible, cuál es la más precisa en relación a la solución exacta del problema. En estas decisiones también se tendrá en cuenta el tiempo de ejecución de cada una las heurísticas, procurando obtener un balance entre eficacia de la solución y tiempo de ejecución.

Finalmente, se desarrollará una metaheurística GRASP modificando ligeramente las heurísticas constructivas y de búsqueda local previamente mencionadas. Se realizarán pruebas para determinar con cuales de estas heurísticas la solución de GRASP es más precisa. También se harán estudios sobre la eficacia de la solución en relación a los parámetros involucrados en esta metaheurística. A partir de estos resultados se podrán concluir los parámetros adecuados y las heurísticas a utilizar para que el resultado de GRASP sea eficaz y eficiente en relación al tiempo de ejecución.

Por último, para la metaheurística GRASP obtenida y para las mejores heurísticas (constructiva y de búsqueda local) se mostrarán casos en donde el resultado de las mismas difiera del exacto en gran medida. Además, se estudiará la complejidad teórica de estos algoritmos y se realizarán comparaciones con el tiempo de ejecución obtenido en las pruebas.

1.1. Situaciones modeladas

A continuación presentamos tres situaciones diferentes en las cuales problemas reales pueden ser modelados y resueltos como un problema de conjunto independiente de peso máximo.

Situación 1:

Hundidos en una profunda crisis económica por las malas gestiones y desconcertados por los pésimos resultados obtenidos en la última temporada, un equipo de fútbol decide cambiar su política de compra/venta de jugadores. Como no se puede comprar sin tener el dinero, el primer paso que llevan a cabo los directivos del club es evaluar cuáles son los jugadores del plantel que van a estar a la venta. Para no desequilibrar el equipo no debería haber dos jugadores en vidriera que:

- Tengan similar jerarquía.
- Jugasen en la misma posición.
- Compartan alguna habilidad especial.
- Tengan la misma experiencia y lleven el tiempo en el club.

Teniendo en cuenta estas restricciones, el manager del club decide plantear esta situación como un problema de conjunto independiente de peso máximo. En su modelo los nodos son los jugadores, el peso de éstos se define por su cotización y las adyacencias por las restricciones a tener en cuenta al querer incluir dos jugadores.

Felices por el funcionamiento de su política de ventas, el club decide también modelar el problema de la compra de los jugadores como el de un conjunto independiente de peso máximo, comprando sólo jugadores del fútbol local, debido a la alta cotización de los jugadores residentes en Europa.

En este caso, los nodos también son los jugadores, pero el peso está dado por la calidad del jugador. Las adyacencias se dan entre los nodos que representan a dos jugadores que:

- Jueguen en la misma posición.
- Sean de equipos opuestos.
- Compartan alguna habilidad especial.
- La suma de sus cotizaciones superen los dos millones de dólares.

Situación 2:

Ante la inminencia de las próximas elecciones, De Narvaez decide cerrar su campaña con una "gira" por las localidades de la provincia de Buenos Aires. Su objetivo es que sus actos sean presenciados por la mayor cantidad de personas posibles, pero evitando dar un discurso en dos localidades limítrofes. Esto es debido a que las personas pueden moverse a su localidad vecina para presenciar el acto y será una pérdida de tiempo innecesaria recorrer cada una de las zonas si no aumenta tanto la cantidad de asistentes. Además que al hacer un acto en localidades vecinas puede suceder que varias personas escuchen más de una vez el acto y empiecen a sospechar de la sinceridad en los proyectos de Francisco. Para decidir cuáles serán las provincias a visitar, el PRO cuenta con una gran base de datos que contiene la cantidad de personas aproximada que podrían asistir al evento al realizarse en cada localidad. Utilizando estos datos, el grupo de inteligencia de Francisco modela el problema de decidir las provincias a visitar como un problema de conjunto independiente de peso máximo donde los nodos son las localidades, los pesos son la cantidad de personas que podrían asistir al acto y la condición de adyacencia es que las localidades sean vecinas.

Situación 3:

Cuando se acercan las vacaciones de invierno las autoridades de la Facultad de Ciencias Exactas de la UBA deciden cuáles serán las materias optativas a dictarse en el segundo cuatrimestre del año. Para esto suben una encuesta a la página web, intentando saber cuántos alumnos se anotarían en cada materia si se dictase. Una vez obtenida esta información se eligen las materias que permitan cursar a la mayor cantidad de alumnos tales que no pase que dos materias:

- Compartan los mismos días y horario.
- Sean de la misma área de estudio.
- La dé el mismo profesor.
- Requieran más de 12 horas de cursada cada una.

Para esto organizan los datos y se los derivan a los alumnos de algoritmos 3 para que éstos lo resulevan como un problema de conjunto independiente máximo utilizando lo programado en el TP3. En este modelo los nodos son las materias y los pesos la cantidad de personas que cursarían la materia, mientras que dos nodos son adyacentes si representan a dos materias no pueden darse conjuntamente en el cuatrimestre por las condiciones mencinadas anteriormente.

2. Algoritmo Exacto

En primer lugar, nos ocuparemos del análisis de un algoritmo exacto propuesto para resolver el problema en cuestión. A continuación, se presenta el pseudocódigo que esboza el comportamiento del mismo:

```
para cada nodo (nodoActual) del grafo
    apilar nodoActual
```

```
    mientras la pila no sea vacia
        para cada nodo desde nodoActual+1 en adelante
            si ese nodo no es adyacente a ninguno de la pila
                apilar nodo
        finPara
        si la suma de pesos de los nodos de la pila > peso de la solucion
            solucion = pila
        nodoActual = desapilar nodo
    finMientras
finPara
```

Como se puede observar, el algoritmo utiliza la técnica de *backtracking* para resolver el problema. La idea general del mismo consiste en observar todos conjuntos independientes (CI) de cada subgrafo S de G inducido por los nodos desde un nodo i hasta n que contengan al nodo i, para $1 \leq i \leq n$[1]. Cada iteración del ciclo externo del algoritmo se encargará de buscar todos los CI para cada uno de los subgrafos.

Antes de continuar, es importante notar que esto garantiza encontrar el conjunto independiente de peso máximo de G. La demostración es simple: Sea j el nodo mínimo del CI de peso máximo del grafo G. Como en cada iteración i, el algoritmo busca todos los conjuntos independientes que contengan al nodo i en S, con i desde 1 hasta n, entonces existe una iteración del algoritmo donde $i = j$. En dicha iteración, por hipótesis, se observan todos los conjuntos independientes que están compuestos por j y nodos mayores a j, y como j es el nodo mínimo de la solución, uno de esos conjuntos deberá ser dicha solución.

A continuación, se mostrará como en cada iteración del ciclo externo, el algoritmo observa los CI de los subgrafos S mencionados.

El ciclo utiliza una pila que comienza conteniendo únicamente al nodo i. Luego, se van agregando todos los nodos que vayan conformando un conjunto independiente a partir de ese nodo i hasta el último del grafo. Inmediatamente después, en caso de que dicho conjunto independiente sea el de peso máximo[2] hasta el momento, se almacena la pila como solución. El siguente paso consiste en desapilar el tope de la pila (que contiene al último nodo insertado), observando si cada nodo posterior a éste puede ser apilado de modo que se siga manteniendo el invariante de conjunto independiente, es decir, se insertan los nodos que no sean adyacentes a ningún nodo de la pila. De esta manera, al salir del ciclo, se habrán observado todos conjuntos independientes que contengan al nodo i, ya que es condición del ciclo que dicho nodo esté en la pila por ser el primer nodo apilado.

Por todo esto, podemos afirmar que el algoritmo encontrará el conjunto independiente de peso máximo para cualquier grafo G.

Complejidad

En la presente sección se calculará la complejidad en el peor caso del algoritmo exacto presentado para resolver el problema. Como se mostró en anteriores secciones, para un grafo G el algoritmo, en cada iteración i, busca todos los conjuntos independientes formados por el nodo i y todos los nodos mayores que i en el subgrafo inducido por dichos nodos. El peor caso

[1] Recordemos que cada nodo del grafo está numerado desde 1 hasta n.

[2] Dicho peso se encuentra ya almacenado en simple acumulador.

de este algoritmo ocurre cuando el grafo está formado solamente por nodos aislados (grado 0). En este caso, el algoritmo deberá buscar todos los conjuntos posibles que se pueden formar a partir de los nodos del grafo ya que cualquier combinación de nodos forma un conjunto independiente. Como la cantidad de conjuntos posibles que se pueden formar con n nodos es $2^n - 1$, se deduce que la complejidad en el peor caso será $O(2^n)$. Sin embargo, en casos donde la cantidad de aristas sea mayor, la cantidad de conjuntos independientes que deberá comprobar el algoritmo será mucho más reducida. Evitando el chequeo de conjuntos innecesarios se espera que el tiempo de ejecución sea más reducido para grafos con densidades altas, es decir, con gran cantidad de aristas.

Análisis de tiempo de ejecución

El siguiente analisis está orientado a estudiar el tiempo de ejecución del algoritmo exacto en relación a los datos de entrada (cantidad de nodos de los grafos), comparando el costo real del algoritmo junto con su complejidad teórica calculada. Como vimos anteriormente el método exacto tiene una alta complejidad temporal por lo que, al momento de realizar las mediciones, se utilizaron grafos en los que su cantidad de nodos varía entre 1 y 36.

En cuanto al tipo de grafos utilizados, se optó por analizar el comportamiento del algoritmo sobre grafos aleatorios con distintas densidades debido a que la cantidad de conjuntos independientes se modifica notablemente según este parámetro. De esta manera, para cada grafo de un mismo tamaño, se realizaron tres pruebas, variando entre altas, medias y bajas densidades.

Los resultados arrojados fueron los siguientes:

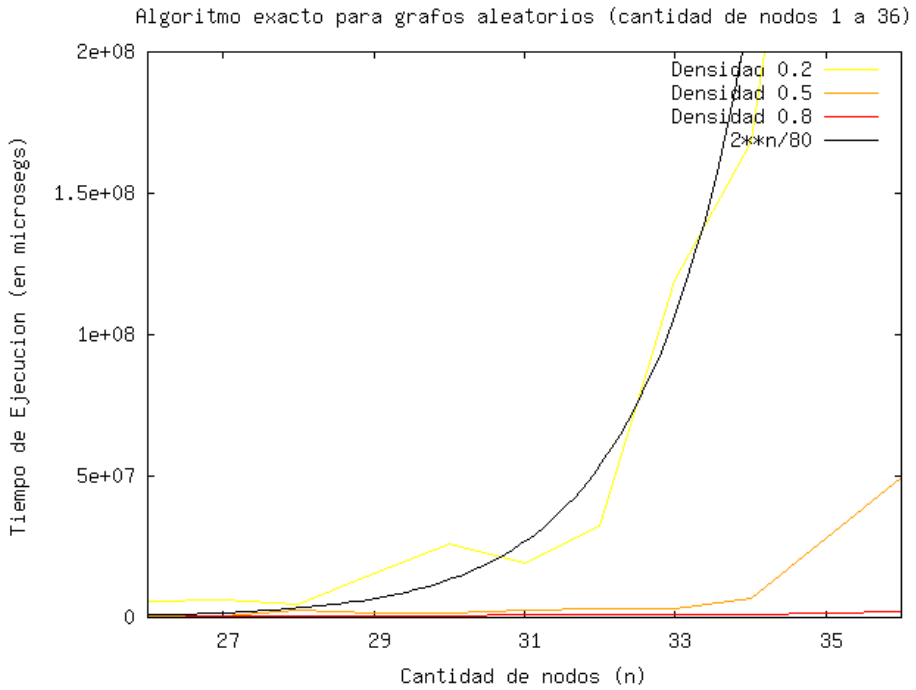

Figura 2.1

Como se puede apreciar en el gráfico, el tiempo de ejecución del algoritmo aplicado sobre los distintos grafos seleccionados se comporta de acuerdo a la complejidad teórica estimada. Además, podemos ver que se mantiene muy alejado de la cota para el peor caso (2^n).

Otro aspecto a tener en cuenta es la notable diferencia de los tiempos en relación a las diversas densidades. Se puede ver que si la densidad del grafo es baja, el tiempo de busqueda del conjunto independiente de mayor peso es mucho mayor que si se lo busca sobre un grafo de densidad media. Lo mismo sucede entre media y alta densidad. Esto se debe a que al disminuir la cantidad de aristas (menor densidad), la cantidad de conjuntos independientes en el grafo aumenta, ya que mientras menos aristas haya, mayor cantidad de nodos no adyacentes entre sí habrá. De esta manera, mientras menor sea la densidad, mayor será la cantidad de conjuntos independientes a tener en cuenta por el algoritmo con el fin de buscar el de mayor peso, por lo que demandará mayor tiempo de ejecución.

Finalmente estudiaremos la complejidad en función del tamaño de la entrada. Sea t el tamaño de la entrada, n la cantidad de nodos, m la cantidad de aristas (m_i la cantidad de aristas del nodo i), p el arreglo de pesos y $a[i]$ las adyacencias del nodo i. Tenemos entonces que:

$$t = \log(n) + \sum_{i=1}^{n} \log(p) + \sum_{i=1}^{n} \sum_{j=1}^{m_i} \log(a[i]) > \log(n) + \sum_{i=1}^{n} 1 + \sum_{i=1}^{n} \sum_{j=1}^{m_i} 1$$

$$> \log(n) + n + \sum_{i=1}^{n} m > \log(n) + n + nm$$

$$\implies \text{como } T(n) \in O(2^n) \text{ y } 2^n < 2^{\log(n)+n+nm} < 2^t \implies T(t) \in O(2^t)$$

3. Heurísticas constructivas

A continuación se desarrollará el trabajo realizado sobre heurísticas constructivas con el objetivo de reducir significativamente la complejidad temporal a merced de una pérdida de eficacia en las soluciones. Para esto, se utilizaron distintos algoritmos golosos que tienen en cuenta diversas condiciones sobre los nodos para ir armando una solución de manera constructiva. Se pensaron varias ideas para buscar aproximarse a la solución óptima, todas siguiendo el siguiente esquema de algoritmo goloso:

```
mientras haya nodos en el grafo
  maxF = 0 | minF = n
  para cada nodo del grafo
    si grado(nodo) = 0
      agregar nodo a solucion
      borrar nodo del grafo
    sino
      si f(nodo) >= maxF | f(nodo) <= minF
        maxF | minF = f(nodo)
        nodoMax | nodoMin = nodo
  finPara
```

```
agregar nodoMax a solucion
borrar vecindad de nodoMax
```

Podemos apreciar que en cada iteración (del ciclo interno) el algoritmo recorre todos los nodos verificando que se cumpla una condición de acuerdo a una función aplicada a los mismos (función f). Una vez observados todos los nodos, se agrega el nodo máximo o mínimo de acuerdo a la condición mencionada al conjunto solución. De inmediato, se procede a borrar la vecindad de dicho nodo del grafo. De esta manera, en cada iteración, el grafo contendrá una cantidad cada vez menor de nodos de modo de garantizar que el algoritmo termina (sale del ciclo principal).

Cabe aclarar que si el grado de los nodos que el algoritmo va recorriendo es igual a 0, inmediatamente son agregados a la solución. Esto se debe a que, al ser todos los pesos mayores que 0, si un nodo no es adyacente a ningún otro (grado 0), siempre deberá pertenecer a la solución.

Las distintas funciones f aplicadas sobre los nodos para armar el conjunto solución fueron:

- **Peso:** en cada iteración del algoritmo se agrega a la solución el nodo de mayor peso.
- **Grado:** en cada iteración del algoritmo se agrega a la solución el nodo de menor grado. La idea de aplicar dicha función es que, al agregar a la solución el nodo de menor grado, solo se borrarán del grafo los nodos de su vecindad (que es la de menor tamaño) descartando asi la menor cantidad de nodos posible.
- **Peso/Grado:** en cada iteración del algoritmo se agrega a la solución el nodo de mayor cociente peso/grado. Simplemente una combinación de las dos técnicas mencionadas anteriormente.
- **Peso/Vecindad:** en cada iteración del algoritmo se agrega a la solución el nodo de mayor cociente peso*grado/pesoVecindad. La intención en este caso es ponderar también el peso y la cantidad de nodos de la vecindad del nodo a agregar a la solución (si la vecindad del nodo es muy pesada y posee pocos nodos no quiero descartarla).

Análisis de eficacia de las heurísticas constructivas

Con el objetivo de decidir que heurística es más eficaz en relación a la precisión de la solución obtenida, se realizaron pruebas para 27 grafos distintos generados de manera aleatoria. En los casos que fue posible, se comparó el resultado obtenido con la solución exacta del problema, obtenida por medio del algoritmo exacto presentado anteriormente.

Los grafos generados de manera aleatoria se dividen en 3 tipos según la cantidad de nodos. Se crearon 9 grafos de cada tipo, con cantidad de nodos 30, 300 y 600. A su vez, cada tipo de grafo se divide según la densidad del grafo, es decir, de los 9 grafos de cada tipo se crearon 3 con baja, media y alta densidad. El peso de cada uno de los nodos también es aleatorio, en un rango de 1 a $n * 10$, siendo n la cantidad de nodos. Estos grafos serán utilizados en todas las pruebas que se realizarán en el trabajo, tanto para las heurísticas constructivas como para las búsquedas locales y la metaheurística GRASP.

Para cada uno de los grafos mencionados se ejecutaron todas las heurísticas constructivas mencionadas en la sección anterior, obteniendo los siguientes resultados.

Tipo de Grafo	Exacto	Peso/grado	Peso	Grado	Peso/vecindad
1) 30 nodos BD	2500	2060	2180	1700	1930
2) 30 nodos BD	2540	2310	2170	1830	2000
3) 30 nodos BD	2680	1910	1930	1650	1570
1) 30 nodos MD	2490	1670	1900	1350	1750
2) 30 nodos MD	2250	1710	1570	1200	1210
3) 30 nodos MD	2450	1490	1810	1420	1470
1) 30 nodos AD	1930	810	1750	580	1030
2) 30 nodos AD	1830	1280	1410	1080	790
3) 30 nodos AD	1930	1090	1200	830	910
1) 300 nodos BD		61590	56280	45810	52810
2) 300 nodos BD		68210	59940	51540	53870
3) 300 nodos BD		59970	51680	44630	50230
1) 300 nodos MD		44690	34930	28880	36310
2) 300 nodos MD		43390	35740	32630	33060
3) 300 nodos MD		40090	32510	26850	31840
1) 300 nodos AD		34480	25830	17610	26240
2) 300 nodos AD		34830	23920	13490	27770
3) 300 nodos AD		33240	23540	19810	29950
1) 600 nodos BD		185800	139740	129770	136250
2) 600 nodos BD		186190	145770	113440	134870
3) 600 nodos BD		181980	153510	150400	132540
1) 600 nodos MD		125810	111680	88130	101540
2) 600 nodos MD		114880	94050	70530	88270
3) 600 nodos MD		125940	99820	83710	88000
1) 600 nodos AD		110060	76640	75230	89770
2) 600 nodos AD		104810	76800	60330	64790
3) 600 nodos AD		102590	77150	71930	82780

Tabla 3.1

Las pruebas realizadas a grafos de 30 nodos pueden ser comparadas con el resultado exacto obtenido por medio del algoritmo de *backtracking*. Como se puede observar, todos los resultados de las heurísticas difieren del resultado exacto en distinta medida. Esto es un resultado esperable ya que en la mayoría de los casos todas las heurísticas golosas no son eficaces y se pueden encontrar casos en los que el resultado difiera en gran medida del exacto.

En la tabla presentada fueron remarcados los resultados más altos para cada tipo de grafo. Como se puede apreciar, para grafos con 30 nodos, los mejores resultados fueron obtenidos con la heurística de Peso. Sin embargo, para grafos con mayor cantidad de nodos, como los de 300 y 600 nodos, los resultados de la heurística de Peso/grado son ampliamente mejores.

A partir de estos resultados, se puede concluir que la heurística de Peso/Grado es la más eficaz de las heurísticas implementadas. El fundamento más importante para la elección de la misma fue que los resultados para grafos con gran cantidad de nodos, son mucho más eficaces a las demás heurísticas constructivas. Como las heurísiticas fueron pensadas para resolver instancias de tamaño mucho mayor a lo que el algoritmo exacto puede resolver, es razonable que se elija la heurística que mejor se comporte para grafos con gran cantidad de nodos. Además, para grafos pequeños los resultados no difieren demasiado de los obtenidos con la heurística de Peso.

En posteriores secciones estudiaremos el comportamiento de la heurística Peso/grado en relación al tiempo de ejecución y compararemos dichos resultados con su complejidad teórica.

Complejidad

El modelo elegido para calcular la complejidad de este algoritmo es el uniforme debido a que lo que la define es la cantidad de nodos y de aristas y como inciden las aristas sobre los nodos. Esto se da porque de ellos depende la cantidad de operaciones que deba realizar el algoritmo de acuerdo a las condiciones de los ciclos.

Antes de analizar las operaciones, cabe aclarar que verificar si dos nodos son adyacentes puede hacerse en tiempo constante, dado que se cuenta con la matriz de adyacencia del grafo. Además se puede obtener el grado asociado a un nodo con la misma complejidad ya que esto se puede realizar viendo el tamaño de su lista de adyacencias.

El cálculo de la complejidad de la heurística es bastante sencilla. Primero observamos el ciclo interno.

```
para cada nodo del grafo //n
  si grado(nodo) = 0
    agregar nodo a solucion
    borrar nodo del grafo
  sino
    si F(nodo) >= maxF
      maxF = F(nodo)
      nodoMax = nodo
finPara
```

En el caso en que el nodo sea de grado 0, es insertado en la solucion ($O(1)$ al insertar en una lista) y eliminado del grafo ($O(1)$ ya que como el nodo no tiene adyacentes sólo borramos su lista de adyacencias que está vacía). Es importante comentar que la matriz de adyacencia no es modificada para no aumentar la complejidad, pero que la información que no es modificada no será utilizada ya que está asociada a nodos que fueron borrados.

En el caso en que la relación de peso/grado del nodo obtenido sea mayor o igual al maxF, se actualiza maxF con el número de dicha relación y se lo guarda al nodo en $nodoMax$. En este caso sólo se definen dos variables por lo que es de orden constante. Como dentro del ciclo se realizan sólo operaciones constantes y este recorre todos los nodos del grafo una vez, el orden del ciclo es lineal.

Luego hay dos operaciones:

```
agregar nodoMax a solucion
borrar vecindad de nodoMax
```

Agregar $nodoMax$ a la solución es constante debido a que la solución usa como estructura una lista. Para borrar la vecindad de $nodoMax$ del grafo, recorremos las listas de adyacencias y borramos de ellas los nodos que son adyacentes a $nodoMax$. Además, si la lista pertenece a un nodo adyacente a $nodomax$, se borra entera. Como la complejidad de esta operacion se basa en recorrer las listas de adyacencias de todos los nodos, tiene orden $O(n+m)$ ($O(m)$ para los casos en los que no haya nodos aislados).

Por último, el ciclo de afuera se ejecuta mientras haya nodos en el grafo. Teniendo en cuenta que en cada iteracion se elimina por lo menos un nodo del grafo (ya que seguro se borra el nodo que se agrega al grafo) el ciclo se ejecutará como máximo n veces. Esto sumado a que el orden de las operaciones realizadas dentro de él es $O(n+m)$, podemos concluir que

la complejidad de la heurística constructiva es $O(n * (n + m))$. Pero como los nodos de grado cero fueron eliminados, sabemos que $n \leq m$, lo que deriva a un orden de $O(n * m)$.

Sin embargo, si se realiza un analisis más minusioso del comportamiento del algoritmo se puede observar lo siguiente. Si el grafo a evaluar resulta muy denso (es decir m cercano n^2) la cantidad de nodos de grado alto aumenta proporcionalmente, por lo que la función encargada de borrar la vecindad eliminará del grafo una alta cantidad de nodos en comparación al caso que se trate de un grafo poco denso. Esto último, dará lugar a que la cantidad de veces que itere el ciclo externo disminuya notablemente en grafos densos ya que en cada paso, el grafo resultante contiene una cantidad menor de nodos que en el caso donde la densidad es baja. Este fenómeno de "equilibro" podría dar lugar a que el algoritmo en la práctica se comporte mejor de lo que se espera, e incluso que su tiempo de ejecución no dependa de la cantidad de aristas. En la siguiente sección, observaremos empíricamente esta cuestión.

Analizaremos ahora la complejidad en función del tamaño de la entrada. Sea t el tamaño de la entrada, n la cantidad de nodos, m la cantidad de aristas (m_i la cantidad de aristas del nodo i), p el arreglo de pesos y $a[i]$ las adyacencias del nodo i. Tenemos entonces que:

$$t = \log(n) + \sum_{i=1}^{n} \log(p) + \sum_{i=1}^{n} \sum_{j=1}^{m_i} \log(a[i]) > \log(n) + \sum_{i=1}^{n} 1 + \sum_{i=1}^{n} \sum_{j=1}^{m_i} 1$$

$$> \log(n) + n + \sum_{i=1}^{n} m > \log(n) + n + nm$$

$$\implies \text{como } T(n, m) \in O(nm) \text{ y } nm < \log(n) + n + nm < t \implies T(t) \in O(t)$$

Análisis de tiempo de ejecución

Al igual que para el análisis del algoritmo exacto, se generaron grafos aleatorios con 3 tipos de densidades con el fin de evaluar el comportamiento de la heurística constructiva elegida sobre diversos grafos según la cantidad de nodos. Sin embargo, al tratarse de un algoritmo polinomial, la cantidad de nodos analizados será mucho mayor (desde 10 hasta 1400 nodos) con el fin de poder observar con claridad su comportamiento.

Como hemos visto anteriormente, la complejidad de la heurística constructiva resulta del orden $O(n * m)$. Por este motivo, se han incluido en el gráfico las curvas $n^2/25$ y $n^3/15000$ con el fin de observar el comportamiento para grafos de distinta densidad. En principio, los tiempos de ejecución de grafos de alta densidad (donde m tiende a n^2) deberían ser similar a la curva $n^3/15000$ mientras que los de baja (donde m tiene a n) deberían asimilarse a la curva $n^2/25$.

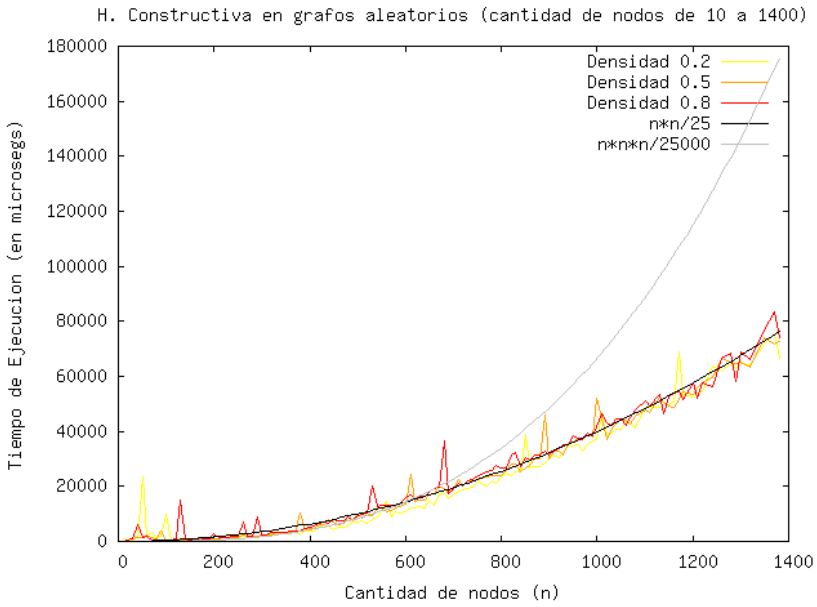

Figura 3.1

Como se observa, el comportamiento del algoritmo no tiene correspondencia directa con la complejidad teórica calculada, ya que que para grafos de densidad variada, los tiempos de ejecución siempre se comportan de la forma de n^2, lo que haría suponer que la complejidad real del algoritmo podría ser de orden cuadrático con respecto a la cantidad de nodos. La razón de esto probablemente radique en el fenómeno de "equilibro" mencionado anteriormente en el apartado de complejidad, el cuál logra que, para un grafo promedio, el algoritmo se comporte en un orden de complejidad menor al calculado y dependiendo únicamente de la cantidad de nodos.

Casos Patológicos

En la siguiente sección se estudiará el comportamiento de la heurística constructiva seleccionada anteriormente cuando recibe como parámetro un tipo especifico de grafos. Podremos apreciar que en estos casos patológicos el peso del conjunto independiente máximo retornado difiere notablemente del óptimo.

Tomemos el siguiente grafo sobre el cual se aplica la heurística:

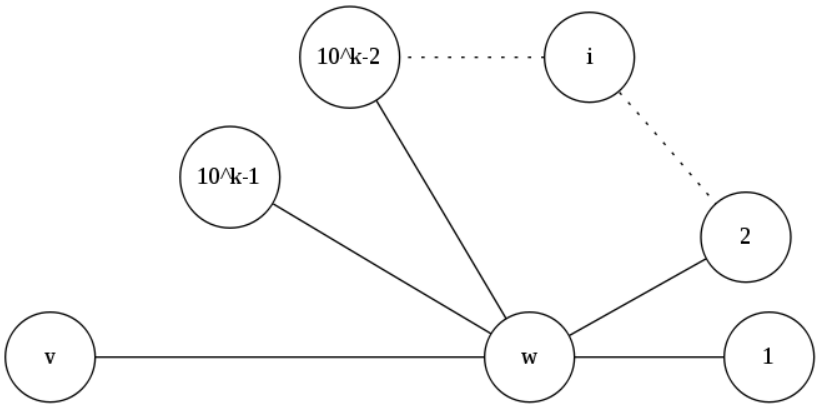

Figura 3.2

Supongamos ahora que los nodos numerados desde 1 a $c^k - 1$ tienen peso 1, el nodo v peso $c^k + 1$ y el w c^{2k}. De esta manera, como w es adyacente a c^k nodos (los nodos numerados y v), el cociente $peso(w)/grado(w) = c^{2k}/c^k = c^k$. Como v es únicamente adyacente a w, su relacion peso-grado será $c^k + 1$. Concluimos entonces que el algoritmo seleccionará a v como primer nodo de la solución. Al hacer esto, descartamos w como posible nodo perteneciente a la solución por ser adyacente a v. El resto de los nodos $(1..c^k - 1)$ pertenecerá a la solución debido a que no son adyacentes a v ni adyacentes entre ellos. Como el peso de estos últimos nodos mencionados es 1, el peso final de la solución brindada por la heurística constructiva es $c^k + 1 + \sum_{i=1}^{c^k-1} 1 = c^k + 1 + c^k - 1 = 2c^k$. A simple vista podemos notar que en realidad, el conjunto independiente de mayor peso es el formado solamente por el nodo w cuyo peso es c^{2k} (tomando $c, k > 2$). Finalmente vemos que el peso del conjunto independiente máximo hallado por nuestro algoritmo y el óptimo difieren en $c^{2k} - 2c^k = c^k(c^k - 2)$.

Vemos entonces que podemos generar un grafo donde el algoritmo sea tan impreciso como uno desee(ajustando los parámetros c y k como se considere necesario).

4. Heurísticas de Búsqueda Local

A continuación se presentarán las heurísticas de búsqueda local implementadas. El objetivo de dichos algoritmos es mejorar una solucón dada (obtenida por medio de alguna de las heurísticas constructivas mencionadas), hasta obtener una solución localmente óptima. Cada heurística de búsqueda local define su concepto de localidad estableciendo una vecindad de soluciones que se revisarán para obtener el máximo local. Las búsquedas locales implementadas corresponden al siguiente esquema.

```
obtengo una solucion (solucionLocal) con una heuristica constructiva

para i desde 1 hasta cantIteraciones hacer
  pesoMaximoAnterior = peso(solucionLocal)
    solucionLocal = intercambioDeNodo(solucionLocal) o
                    intercambioDeUnoAMuchos(solucionLocal)
    si peso(solucionLocal) = pesoMaximoAnterior
      devolver solucionLocal
```

```
devolver solucionLocal

intercambioDeNodo(solucion)
  diferenciaMaxima = 0
  sacar los nodos de grado 0 de solucion e insertarlos en nodosGradoCero

  para cada nodo (nodoActual) de solucion
    para cada nodo (adyacenteANodoActual) adyacente a nodoActual
      diferenciaActual = peso(adyacenteANodoActual) - peso(nodoActual)
      si (adyacenteANodoActual no es adyacente a algun nodo de solucion
         excepto nodoActual y diferenciaActual > diferenciaMaxima)
           diferenciaMaxima = diferenciaActual
           nodoASacar = nodoActual
           nodoAInsertar = adyacenteANodoActual
    finPara
  finPara

  si diferenciaMaxima > 0
    sacar nodoASacar de solucion
    agregar nodoAInsertar en solucion

  insertar todos los nodos de nodosGradoCero en solucion


intercambioDeUnoAMuchos(solucion)
  diferenciaMaxima = 0
  sacar los nodos de grado 0 de solucion e insertarlos en nodosGradoCero

  para cada nodo (nodoActual) de solucion
    nodosAInsertarActual = lista vacia
    para cada nodo (adyacenteANodoActual) adyacente a nodoActual
      si adyacenteANodoActual no es adyacente a algun nodo de solucion excepto nodoActual
        si adyacenteANodoActual no es adyacente a algun nodo de nodosAInsertarActual
          agregar adyacenteANodoActual en nodosAInsertarActual
    finPara
    diferenciaActual = peso(nodosAInsertarActual) - peso(nodoActual)
    si diferenciaActual > diferenciaMaxima
      diferenciaMaxima = diferenciaActual
      nodosAInsertar = nodosAInsertarActual
      nodoASacar = nodoActual
  finPara

  si diferenciaMaxima > 0
    sacar nodoASacar de solucion
    insertar todos los nodos de nodosAInsertar en solucion
```

```
insertar todos los nodos de nodosGradoCero en solucion
```

Como fue mencionado anteriormente, las búsquedas locales implementadas difieren en la forma en que definen su vecindad, pero tienen el mismo esquema básico. Las dos heurísticas implementadas obtienen una solución inicial por medio de alguna de las heurísticas constructivas (Peso, Grado, Peso/Grado, Peso/Vecindad) estudiadas en secciones anteriores. Luego, intentan mejorar dicha solución obtenida una cierta cantidad de veces, establecida mediante el parámetro cantIteraciones. Si en alguno de los pasos la solución no mejorase el algoritmo termina ya que, como los procedimientos utilizados para mejorar la solución son completamente determinísticos, no habría cambios para las posteriores iteraciones del mismo. Cuando esto sucede, se puede afirmar que la solución obtenida por la heurística corresponde a un máximo local según la vecindad definida por dicha heurística.

La primer heurística de búsqueda local (a la que llamaremos BL1) mejora la solución por medio de la función *intercambioDeNodo*. Esta función saca un nodo de la solución que recibe como parámetro y agrega otro distinto que forme una solución válida y que maximice el peso de la solución. Es decir, saca un nodo de la solución, al que llamaremos nodoActual y para cada nodo adyacente a nodoActual, verifica si lo puede insertar en la solución viendo si no es adyacente a ningún nodo de la misma. De los nodos adyacentes a nodoActual que se pueden insertar, se elijen sólo los de peso mayor al peso de nodoActual ya que son los únicos que aumentan el peso total de la solución. A su vez, se elije el máximo entre dichos nodos para que el peso que se agrega a la solución sea mayor. Este proceso se repite tomando como nodo nodoActual a todos los nodos de la solución, por lo que, al terminar de ejecutar la función se obtendrá el nodo a sacar y el nodo a insertar que hacen que el peso de la solución aumente en máxima medida. Es importante aclarar que al sacar un nodo de la solución sólo se pueden agregar los adyacentes a dicho nodo debido a que las soluciones que recibe la búsqueda local están generadas por las heurísticas constructivas y estas siempre retornan conjuntos independientes maximales. Además, a partir de lo visto anteriormente, se puede definir la vecindad para esta búsqueda local como todas las soluciones que se pueden formar sacando un nodo de la solución original e insertando otro que forme un conjunto independiente.

La segunda búsqueda local implementada (a la que llamaremos BL2) mejora la solución a través de la función *intercambioDeUnoAMuchos*. A diferencia de la búsqueda local anterior, esta función saca un nodo de la solución y agrega varios nodos nuevos a la misma. Más precisamente, saca un nodo de la solución (nodoActual) y se recorren los nodos adyacentes al mismo (en el orden en que estén dispuestos en la lista de adyacencia), agregando en una lista (nodosAInsertarActual) los que no sean adyacentes a alguno de la solución ni a alguno de dicha lista para asegurar que los nodos de la lista nodosAInsertarActual formen un conjunto independiente. Si la suma de los pesos de la lista mencionada es mayor al peso de nodoActual esta puede ser agregada a la solución aumentando así su peso. Este paso se repite para cada nodo de la solución, quedándose con la lista de mayor peso que cumpla con las condiciones mencionadas (nodosAInsertar) y con el nodo a sacar (nodoASacar) a partir del cual se la obtuvo. Finalmente, se saca de la solución el nodo nodoASacar y se agregan a la solución todos los nodos de la lista nodosAInsertar.

A partir del algoritmo descripto, se puede concluir que la vecindad de la segunda búsqueda local se puede definir como todas las soluciones que se pueden armar sacando un nodo y agregando una cierta cantidad de nodos de la manera que fue mencionada anteriormente.

Cabe aclarar que los nodos agregados a cada lista nodosAInsertarActual depende fuertemente del orden en que se encuentren en la lista de adyacencia del nodoActual. Es decir,

nodosAInsertarActual puede contener distintos conjuntos independientes, dependiendo del orden en que estén los nodos en la lista de adyacencia (en posteriores secciones veremos que ordenar las listas de adyacencias según el peso de los nodos de mayor a menor ayuda a reducir casos "patológicos"). Además, se puede ver claramente que el conjunto independiente obtenido en dicha lista puede no ser máximo, ya que obtener un conjunto de este tipo sería una tarea exponencial. Por último, es importante mencionar que al comienzo de las funciones *intercambioDeNodo* y *intercambioDeUnoAMuchos* se extraen los nodos de grado 0 (con respecto al grafo original) de la solución antes de comenzar a mejorar la misma, ya que estos nodos siempre van a formar parte de la solución óptima y no tiene sentido sacarlos para intentar insertar sus nodos adyacentes. Al finalizar la mejora de la solución estos nodos son insertados nuevamente. Como se verá más adelante, realizar esta operación reduce la complejidad del algoritmo.

Análisis de eficacia

En la presente sección se intentará observar cuán eficaces resultan las heurísticas de búsqueda local propuestas a la hora de hallar un conjunto independiente de mayor peso. Además, veremos como mejora la calidad de la solución con respecto a cada heurística constructiva presentada anteriormente. El motivo de analizar nuevamente todas las heurísticas es que no necesariamente las soluciones provistas por la mejor heurística hallada harán que las busquedas locales provean los mejores resultados. Es decir, puede ocurrir que, por la forma en la que comporta la busqueda local (en otras palabras, la vecindad a analizar), las soluciones provistas por la heurística golosa de peso/grado no sean las soluciónes iniciales que dén mejores resultados.

Con el fin de realizar las comparaciones mencionadas se optó por realizar una versión extendida de la tabla referida a las heurísticas constructivas. Al igual que antes, cada celda de la tabla contiene el peso de la solución obtenida, aunque para el caso de las búsquedas locales, se muestra también (entre paréntesis) la cantidad de iteraciones realizadas por dicha búsqueda.

Tipo de Grafo	Exacto	Heuristica peso/grado			Heuristica con peso			Heuristica con grado			Heuristica peso/vecindad		
		Sin BL	Con BL1	Con BL2	Sin BL	Con BL1	Con BL2	Sin BL	Con BL1	Con BL2	Sin BL	Con BL1	Con BL2
1) 30 nodos BD	2500	2060	2060 (1)	2060 (1)	2180	2180 (1)	2180 (1)	1700	1960 (3)	2410 (6)	1930	1930 (1)	2400 (4)
2) 30 nodos BD	2540	2310	2330 (2)	2440 (2)	2170	2170 (1)	2170 (1)	1830	2190 (4)	2360 (2)	2000	2200 (4)	2430 (3)
3) 30 nodos BD	2680	1910	1910 (1)	2610 (8)	1930	1930 (1)	2250 (2)	1650	1880 (3)	2680 (5)	1570	1570 (1)	1890 (2)
1) 30 nodos MD	2490	1670	1800 (3)	1800 (1)	1900	1900 (1)	2380 (3)	1350	1740 (6)	2490 (7)	1750	1750 (1)	1750 (1)
2) 30 nodos MD	2250	1710	1810 (3)	1810 (1)	1570	1570 (1)	2100 (4)	1200	1580 (5)	2180 (5)	1210	1280 (3)	2170 (6)
3) 30 nodos MD	2450	1490	1550 (2)	2030 (3)	1810	1810 (1)	2110 (2)	1420	1420 (1)	1420 (1)	1470	1470 (1)	1700 (2)
1) 30 nodos AD	1930	810	1010 (2)	1800 (5)	1750	1750 (1)	1810 (2)	580	930 (4)	1520 (3)	1030	1120 (2)	1450 (3)
2) 30 nodos AD	1830	1280	1370 (3)	1710 (4)	1410	1410 (1)	1410 (1)	1080	1350 (4)	1710 (4)	790	870 (3)	1830 (6)
3) 30 nodos AD	1930	1090	1170 (3)	1720 (5)	1200	1200 (1)	1690 (4)	830	1290 (4)	1550 (4)	910	970 (2)	1740 (5)
1) 300 nodos BD		61590	70680 (24)	122910 (18)	56280	56280 (1)	117770 (22)	45810	74190 (26)	109680 (14)	52810	53910 (6)	114020 (23)
2) 300 nodos BD		68210	70480 (11)	137780 (23)	59940	59940 (1)	135650 (34)	51540	75030 (27)	134250 (20)	53870	57060 (9)	130680 (24)
3) 300 nodos BD		59970	67170 (17)	109270 (18)	51680	51680 (1)	113190 (26)	44630	71150 (24)	121340 (22)	50230	56620 (15)	115680 (22)
1) 300 nodos MD		44690	52670 (14)	94900 (16)	34930	34930 (1)	91240 (17)	28880	59070 (32)	94760 (20)	36310	39760 (15)	99200 (18)
2) 300 nodos MD		43390	45900 (10)	90010 (14)	35740	35740 (1)	92110 (22)	32630	54900 (24)	96320 (13)	33060	36160 (9)	86390 (16)
3) 300 nodos MD		40090	45510 (14)	99450 (22)	32510	32510 (1)	96070 (25)	26850	52620 (23)	85580 (12)	31840	34470 (12)	94420 (16)
1) 300 nodos AD		34480	37920 (14)	77030 (15)	25830	25830 (1)	81140 (15)	17610	36510 (17)	74790 (15)	26240	28480 (7)	76090 (19)
2) 300 nodos AD		34830	39880 (14)	76810 (12)	23920	23920 (1)	70680 (16)	13490	44860 (20)	71420 (9)	27770	28450 (7)	77030 (13)
3) 300 nodos AD		33240	37700 (15)	77250 (12)	23540	23540 (1)	73640 (10)	19810	42020 (23)	80310 (16)	29950	31190 (7)	78380 (14)
1) 600 nodos BD		185800	210010 (32)	398540 (36)	139740	139740 (1)	411530 (40)	129770	222800 (56)	412590 (44)	136250	150580 (22)	399890 (33)
2) 600 nodos BD		186190	212850 (36)	426550 (45)	145770	145770 (1)	382090 (41)	113440	218650 (64)	416750 (46)	134870	143550 (12)	411800 (52)
3) 600 nodos BD		181980	201970 (18)	434590 (31)	153510	153510 (1)	438430 (46)	150400	219100 (40)	421380 (37)	132540	144050 (24)	424520 (47)
1) 600 nodos MD		125810	137350 (24)	310930 (25)	111680	111680 (1)	281010 (24)	88130	146790 (42)	309170 (23)	101540	108260 (11)	273750 (23)
2) 600 nodos MD		114880	131390 (25)	296050 (33)	94050	94050 (1)	280830 (22)	70530	154610 (41)	282220 (21)	88270	97490 (17)	293230 (30)
3) 600 nodos MD		125940	143960 (26)	298620 (33)	99820	99820 (1)	293900 (29)	83710	151470 (36)	310800 (27)	88000	99650 (18)	307530 (27)
1) 600 nodos AD		110060	120580 (12)	281340 (19)	76640	76640 (1)	244060 (17)	75230	125860 (30)	279540 (19)	89770	92630 (12)	285920 (29)
2) 600 nodos AD		104810	115850 (23)	247970 (19)	76800	76800 (1)	258090 (29)	60330	121380 (39)	259830 (21)	64790	69580 (15)	243880 (28)
3) 600 nodos AD		102590	115480 (22)	270330 (25)	77150	77150 (1)	239410 (19)	71930	126310 (36)	278050 (22)	82780	85450 (8)	269310 (25)

Tabla 4.1

En primer lugar, podemos observar que la busqueda local 2 resulta ampliamente superior a la busqueda local 1 para todos los casos evaluados. Lo interesante de esta tabla es ver como se destacan particularmente 2 heurísticas constructivas como generadores de soluciones iniciales "buenas" para esta búsqueda. Por un lado la heurística peso/grado muestra ser eficaz tal como lo era por sí sola. Por otra parte, la heurística de grado que resultaba particularmente ineficiente en análisis previos, resulta aún más eficaz que esta en la gran mayoría de los casos.

Otro aspecto interesante a observar es lo que sucede con la cantidad de iteraciones de cada búsqueda en general. La tabla muestra que aún para grafos grandes la cantidad de iteraciones se encuentra acotada por un valor inferior a 100. Teniendo en cuenta el crecimiento de dicho valor con respecto al tamaño de los grafos y que, debido a cuestiones de complejidad (como veremos más adelante), las grafos sobre los cuales se hará uso este algoritmo no serán mucho más grandes a los grafos analizados (a los sumo del orden de los 10^3 nodos), podemos concluir que las iteraciones locales nunca superarán un valor del orden de 10^2. Además, por este motivo y por lo que veremos luego, se puede observar que la cantidad de iteraciones no es un factor determinante sobre el tiempo de ejecución. Por lo tanto, no resulta de importancia al momento de optar por alguna de las 8 busquedas locales (2 por cada heurística constructiva).

Por lo aquí analizado, se puede concluir que la busqueda local 2 junto con la heurística contructiva de grado resultan la combinación más eficaz para la resolución del problema. De aquí en adelante, nos ocuparemos únicamente de dicha heurística. Además, por lo recién mencionado, fijaremos como cota de iteraciones locales el valor 200.

Complejidad

A continuación procederemos a analizar la complejidad de la busqueda local.

```
obtengo una solucion (solucionLocal) con la heuristica de grado

para i desde 1 hasta cantIteraciones hacer
  pesoMaximoAnterior = peso(solucionLocal)
    solucionLocal = intercambioDeUnoAMuchos(solucionLocal)
    si peso(solucionLocal) = pesoMaximoAnterior
      devolver solucionLocal

devolver solucionLocal
```

En primer lugar, se genera una solución inicial con la heuristica de grado elegida. Como vimos anteriormente, dicho algoritmo tiene un costo de orden $O(n * m)$. A continuación, se itera una cantidad constante de veces ya que cantIteraciones es un parámetro acotado, por lo que la complejidad real estará dada por las operaciones definidas dentro del mismo. Es fácil ver que la única operación de complejidad no trivial es la llamada a *intercambioDeUnoAMuchos(solucionLocal)*, ya que el peso del conjunto solución se encuentra almacenado en una variable por lo que su acceso es constante. Por lo tanto, la complejidad estará determinada por esta función cuya primer parte es la siguiente:

```
diferenciaMaxima = 0
sacar los nodos de grado 0 de solucion e insertarlos en nodosGradoCero
```

Aqui resulta también facil ver que el orden es lineal con respecto a n ya que es necesario recorrer todos los nodos de la solución. La parte interesante del algoritmo es su ciclo principal:

```
para cada nodo (nodoActual) de solucion
  nodosAInsertarActual = lista vacia
  para cada nodo (adyacenteANodoActual) adyacente a nodoActual
    ...
  finPara
  diferenciaActual = peso(nodosAInsertarActual) - peso(nodoActual)
  si diferenciaActual > diferenciaMaxima
    diferenciaMaxima = diferenciaActual
    nodosAInsertar = nodosAInsertarActual
    nodoASacar = nodoActual
finPara
```

En primer lugar, el ciclo exterior encargado de recorrer todos los nodos de la solución sería de orden $O(n)$. El ciclo interno recorre los adyacentes a cada uno de esos nodos también en un orden $O(n)$, lo que a primera vista daría a pensar que ambos ciclos en conjunto serían de orden cuadrático. Sin embargo, en total se recorrerán todos los nodos adyecentes a cada nodo de la solución por lo cual como máximo, en caso de que la solución contenga a todos los nodos, se harán $O(m)$ iteraciones (ya que $m \geq n$ porque los nodos aislados ya fueron extraidos previamente de la solución). Por lo tanto, la complejidad de los ciclos anidados es $O(m)$. (Cabe aclarar que el resto de las operaciones no mencionadas resultan trivialmente constantes). Veremos ahora que ocurre con las operaciones dentro del ciclo interno.

```
si adyacenteANodoActual no es adyacente a algun nodo de solucion excepto nodoActual
    si adyacenteANodoActual no es adyacente a algun nodo de nodosAInsertarActual
        agregar adyacenteANodoActual en nodosAInsertarActual
```

La parte importante de este fragmento se encuentra en la verificar que cierto nodo en cuestión (adyacente a un nodo solución) no es adyacente a ningun nodo ya insertado del conjunto de nodos a insertar parcial ni a ningun nodo de la solución. Como verificar una adyacencia requiere tiempo constante, la complejidad estará dada por la cantidad de nodos a verificar. Es fácil ver que esta cantidad está acotada por n ya que los nodos ya insertados son nodos de los adyacentes a los nodos solución por lo cual no pertecen al mismo conjunto que los nodos de la solución. Por este motivo, al tratarse de conjuntos disjuntos de nodos, la suma de dichos elementos debe ser menor a n. Por último, como agregar a un conjunto es $O(1)$ entonces es claro que la complejidad de esta parte en cuestión es $O(n)$.

Por lo analizado hasta aquí, podemos concluir entonces que el ciclo general del algorimto es de $O(m * n)$. Por último, resta ver que sucede con el final del pseudocódigo.

```
si diferenciaMaxima > 0
    sacar nodoASacar de solucion
    insertar todos los nodos de nodosAInsertar en solucion

insertar todos los nodos de nodosGradoCero en solucion
```

Como sacar un nodo resulta de orden lineal respecto a n al igual que la última operación, la complejidad total de esta parte es de O(n). Del mismo modo, insertar nuevamente todos los nodos de grado cero también se encuentra acotado por n.

Con todo lo analizado, podemos ver que la complejidad total del algoritmo completo es $O(n * m + n + m * n + 2 * n) = O(n * m)$.

Finalmente analizaremos la complejidad en función del tamaño de la entrada. Sea t el tamaño de la entrada, n la cantidad de nodos, m la cantidad de aristas (m_i la cantidad de aristas del nodo i), p el arreglo de pesos y $a[i]$ las adyacencias del nodo i. Tenemos entonces que:

$$t = \log(n) + \sum_{i=1}^{n} \log(p) + \sum_{i=1}^{n} \sum_{j=1}^{m_i} \log(a[i]) > \log(n) + \sum_{i=1}^{n} 1 + \sum_{i=1}^{n} \sum_{j=1}^{m_i} 1$$

$$> \log(n) + n + \sum_{i=1}^{n} m > \log(n) + n + nm$$

$$\implies \text{como } T(n,m) \in O(nm) \text{ y } nm < \log(n) + n + nm < t \implies T(t) \in O(t)$$

Casos Patológicos

En esta sección se estudiará un caso en el que, dado un tipo de solución inicial, el algoritmo de busqueda local implementado no realiza una mejora sustancial.

Supongamos que el algoritmo de busqueda local recibe una solución compuesta por dos nodos que llamaremos v y w. Supongamos tambien que el peso de v es 1 y que es adyacente a un único nodo a de peso 2. Además, a es adyacente a todos los nodos adyacentes a w pero no a este último. De esta manera, el algoritmo de busqueda local comenzará por colocar en la lista de nodos a insetrar a a, ya que sacando a v de la solución y agregando a a, el peso parcial aumenta en 1. En el siguiente paso, el algoritmo intentara reemplazar w por uno o más nodos adyacentes a el que mejoren la solución. Como el nodo a es adyacente a todos los adyacentes a w, y ya es considerado como uno de los nodos a insertar, no podre agregar ninguno de los nodos adyacentes a w a la lista de nodos a insertar en la solución. En el caso en que los nodos adyacentes a w tuvieran mucho peso y no fueran adyacentes con el nodo a intercambiado ni entre ellos, no estaria teniendolos en cuenta por lo mencionado anteriormente.

Vemos asi que mientras mayor sea el peso de la vecindad de w, mas alejado del mejor intercambio se encontrará la solución. Este caso patológico podria ser evitado si en lugar de arrancar los intercambios por el nodo v el algoritmo comenzara por w.

Análisis de tiempo de ejecución

Manteniendo la política utilizada en los ejercicios anteriores, el análisis empiríco se realizó sobre grafos aleatorios. En este caso el algoritmo se corrió para grafos entre 20 y 4000 nodos, nuevamente con 3 tipos de densidades para poder apreciar si existe una relación entre ellas y el tiempo de ejecución demandado.

Para poder apreciar el comportamiento del algoritmo con respecto a la complejidad teórica se agregaron al gráfico las curvas definidas por $n^3/8000$ y $n^2/9$, dado que como el orden del algoritmo es $n * m$ debe alternar entre esos valores(ya que $n \leq m \leq n^2$).

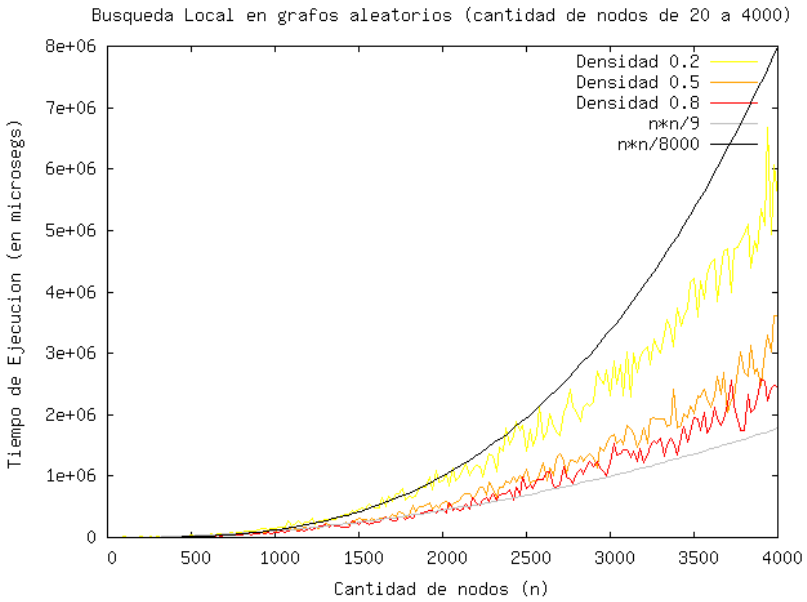

Figura 4.1

En el gráfico se puede apreciar como la curva de orden $(O(n^3))$ acota superiormente a las curvas que representan los resultados de los tiempos de ejecución para los tres tipos de densidades, mientras la curva de orden $(O(n^2))$ la acota inferiormente lo que coincide con los esperado debido a la relación entre n y m. De todas formas se observa que los ejemplos analizados tienden más a una complejidad cuadràtica que cùbica.

Otra característica observable es que la densidad del grafo influye directamente en el tiempo de ejecución del algoritmo, ya que a pesar de no mejorar en complejidad, se puede notar que a medida que se van aumentando las densidades disminuye el tiempo requerido.

5. Metaheurística GRASP

La última parte del trabajo es, probablemente, la más interesante. En esta sección se estudiará el comportamiento de la heurística de Grasp, lo que incluye la utilización de las heurísticas anteriores y permite la comparación (para grafos con un número limitado de nodos) de su rendimiento con respecto al valor óptimo obtenido utilizando el algoritmo exacto. Aquí se terminan de tomar las decisiones con respectos a los parámetros a utilizar en las heurísitcas debiendo privilegiar ciertos aspectos por encima de otros. Una vez analizado Grasp, se podrá tener una conclusión en conjunto de lo realizado y detallar los beneficios y las contras de la utilzación de esta heurística para este problema. A continuación se presenta el pseudocódigo del algoritmo.

```
pesoMaximo = 0
para i desde 1 hasta cantIteracionesGrasp
  unaSolucion = heuristicaConstructiva(alfa)
  unaSolucion = busquedaLocal(unaSolucion, cantIteracionesBL)
  si peso(unaSolucion) > pesoMaximo
    laMejorSolucion = unaSolucion
    pesoMaximo = peso(laMejorSolucion)
```

El algoritmo utilizado consta de un ciclo general que está determinado por la cantidad de iteraciones de Grasp. El ciclo se ejecutará tantas veces como este parámetro lo indique. Para determinar su valor apropiado se efectuarán las pruebas necesarias más adelante. Dentro del ciclo se construye una solución mediante la heurística constructiva, se le aplica búsqueda local y, si consigue mejorar a la óptima obtenida hasta el momento, ocupa su lugar. De esta manera, al terminar el ciclo se obtendrá la mejor solución encontrada.

Un punto importante a señalar es el hecho de que las heurísticas constructivas fueron modificadas con el fin de construir distintas soluciones iniciales, tal como indica el esquema de GRASP. Por este motivo, se añadió un parámetro que llamaremos $alfa$ cuyo valor tiene relacion con el factor de aleatoriedad del algoritmo. Más precisamente, en lugar de que la heurística golosa siempre vaya añadiendo a la solución el mejor candidato, agregue alguno de los mejores cantidatos.

Dicho esto, los análisis posteriores se encargarán analizar que valores de los 2 parámetros mencionados son los más adecuados para la metaheurística.

Por un lado, el alfa determinará qué tan aleatoria será la heurística constructiva. Es importante su buena elección ya que una mala decisión podría limitar notablemente la cantidad de soluciones a verificar, mientras que por otro lado podría construir malas soluciones iniciales lo que influiría negativamente en la eficiencia y eficacia de la búsquela local que se le aplique.

Por otro lado, la otra decisión consiste en determinar una cota para la cantidad de iteraciones de GRASP a realizar. Nuevamente, la relevancia de encontrar un buen valor para éste parámetro se debe a que es necesario encontrar un equilibrio entre la eficiencia temporal y la optimalidad de la solución buscada. Para su determinación resulta fundamental ver qué tanto sigue mejorando la solución a medida que van aumentando las iteraciones para seleccionar el número de iteraciones para el cual la solución deja de conseguir mejoras notables.

Análisis de eficacia

En la presente sección estudiaremos la eficacia de la metaheurística GRASP en relación a la precisión de la solución obtenida. Con este objetivo, compararemos los resultados de GRASP con las mejores heurísticas constructivas y de búsqueda local según lo visto en anteriores secciones. Se probará con distintas heurísiticas ya que anteriormente se concluyó que la mejor heurística constructiva es la de Peso/Grado pero los mejores resultados para la búsqueda local se lograron con una heurística constructiva de Grado y la búsqueda local 2 (BL2). Es decir, en principio, se desconoce que combinación de heurísticas tendrá mejores resultados.

Las siguientes pruebas realizadas tienen como objetivo estudiar los resultados de la metaheurística variando los parámetros de la misma (cantIteracionesGRASP y alfa). Es decir, se intentarán decidir los parámetros que logren un balance entre la calidad de las soluciones y el tiempo de ejecución de la metaheurística.

Siguiendo la misma línea utilizada en las pruebas anteriores, para todas las pruebas se utilizaran los grafos aleatorios con densidades baja, media y alta.

Análisis de heurísticas

Como se mencionó anteriormente, la primer prueba realizada tiene como objetivo decidir que heurísiticas constructivas y de búsuqeda local hacen los resultados de GRASP sean más eficaces. Para esto se realizarán pruebas de GRASP con la mejor heurística constructiva (Peso/Grado) y la heurística constructiva que hace que la búsqueda local sea más eficaz (Grado).

La búsqueda local utilizada fue BL2 ya que se mostró que los resultados son ampliamente mejores que los de BL1.

Las tablas que se presentarán a continuación, muestran la diferencia de los resultados obtenidos de GRASP utilizando la heurística de Grado y los resultados utilizando Peso/Grado (ambas usando BL2 y variando el parámetro alfa de 0.1 a 1). Es decir, cada celda de la tabla contiene la diferencia $resultadoGrado - resultadoPeso/Grado$. Además, se presentan estos resultados variando la cantidad de iteraciones de GRASP 30, 100 y 300.

Tipo de Grafo	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1) 30 nds BD	420	320	90	200	120	100	0	0	0	0
2) 30 nds BD	60	50	50	10	80	0	70	0	0	0
3) 30 nds BD	70	70	0	0	0	0	0	0	0	0
1) 30 nds MD	690	0	0	0	0	0	0	0	0	0
2) 30 nds MD	10	0	0	0	0	10	10	0	0	0
3) 30 nds MD	500	-280	-280	0	0	0	0	0	0	0
1) 30 nds AD	100	0	0	0	0	0	0	0	0	0
2) 30 nds AD	130	120	0	0	0	0	0	0	0	0
3) 30 nds AD	190	190	130	190	0	0	0	0	0	0
1) 300 nds BD	-1360	2210	3040	-750	480	3100	1840	3390	2760	5810
2) 300 nds BD	4320	40	1390	2500	2150	4260	810	5620	5190	2790
3) 300 nds BD	-2030	-270	-960	-640	-2960	-4640	410	-3120	-1200	10810
1) 300 nds MD	2950	2770	5300	-820	10	5750	5290	2650	4150	4800
2) 300 nds MD	2980	2450	2380	-1460	2810	1450	2090	3800	1080	3200
3) 300 nds MD	-1010	1590	-1160	4630	-580	2580	-2190	1550	4900	7090
1) 300 nds AD	-400	1530	2750	3320	790	-920	-1720	5150	1360	3350
2) 300 nds AD	1090	-1090	2170	2540	5710	1090	4770	6010	2550	9020
3) 300 nds AD	770	4640	-200	-3480	2010	3200	5000	2910	200	2000
1) 600 nds BD	-1000	-3240	6320	-6430	13270	6840	90	10460	1870	-3210
2) 600 nds BD	2330	-11090	980	1710	-3090	-15800	8330	6070	4380	23130
3) 600 nds BD	20400	22020	3660	11170	-7330	-2060	12970	2590	12190	14030
1) 600 nds MD	2440	16540	-3370	3220	520	10130	10800	-8770	5380	-3540
2) 600 nds MD	-5950	10570	-150	1710	780	22190	8260	11880	8670	1930
3) 600 nds MD	80	2860	11320	-680	9530	-5290	16090	4110	7570	5140
1) 600 nds AD	-1660	6650	9300	6650	9240	-1650	11020	-1660	2850	23770
2) 600 nds AD	-3620	5470	2310	7910	6810	-6930	11200	2970	2680	8660
3) 600 nds AD	780	-7720	-8190	-3350	680	-2830	7190	13260	6160	5250

Tabla 5.1 (30 iteraciones de GRASP)

Tipo de Grafo	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1) 30 nds BD	420	320	90	90	0	10	0	0	0	0
2) 30 nds BD	-10	40	-20	10	10	0	0	0	0	0
3) 30 nds BD	70	70	0	0	0	0	0	0	0	0
1) 30 nds MD	690	0	0	0	0	0	0	0	0	0
2) 30 nds MD	10	0	0	0	0	0	0	0	0	0
3) 30 nds MD	500	-280	-280	0	0	0	0	0	0	0
1) 30 nds AD	100	0	0	0	0	0	0	0	0	0
2) 30 nds AD	130	120	0	0	0	0	0	0	0	0
3) 30 nds AD	190	190	0	0	130	0	0	0	0	0
1) 300 nds BD	-4820	-3010	-2820	-1710	-2620	980	-2750	2580	-2410	4510
2) 300 nds BD	-2710	-1180	-4260	-920	3410	2850	-900	250	3070	1590
3) 300 nds BD	-4170	-110	-3810	-4950	-7990	-730	-2790	-6940	-2140	1460
1) 300 nds MD	1700	60	-1930	-1040	130	-2670	-2810	3120	-1900	3100
2) 300 nds MD	2980	420	1410	-1080	-1840	-630	-3430	-1240	-160	670
3) 300 nds MD	-610	-990	1800	1310	-5870	-3560	100	-2180	-240	-1710
1) 300 nds AD	-740	-2060	-2590	-2780	-1660	1280	1020	-130	2940	-3790
2) 300 nds AD	-1990	-4080	370	-1420	-1290	-1560	-1690	2380	2110	-1570
3) 300 nds AD	4520	-360	-4730	80	-2960	-2310	-2230	-810	-1280	-1360
1) 600 nds BD	-11410	-7960	-5360	-5740	-13030	-15560	6130	-10530	7560	2900
2) 600 nds BD	-3520	-6160	-9900	-15650	-7310	-8280	3040	4190	-710	-5670
3) 600 nds BD	8150	8030	420	-2730	6200	-6370	-5250	-10990	1480	-1000
1) 600 nds MD	-12670	-3540	-6290	-6640	-100	-6720	-8420	-6650	-5380	80
2) 600 nds MD	-15200	-6910	-22210	-16910	-24030	-5920	-8840	-7090	-18450	-8510
3) 600 nds MD	-6010	-14890	-11610	-7440	-7440	3090	10100	-900	-14520	6820
1) 600 nds AD	-2420	-1170	1920	-5470	2670	-8200	-770	-7210	18960	5470
2) 600 nds AD	-11000	3410	-6420	-6050	-2830	-5390	-1710	1920	5330	-6320
3) 600 nds AD	-3420	-12220	-13210	-9230	910	-7690	-8910	2400	5760	-170

Tabla 5.2 (100 iteraciones de GRASP)

Tipo de Grafo	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1) 30 nds BD	420	90	90	90	0	10	0	0	0	0
2) 30 nds BD	60	40	-20	0	0	0	0	0	0	0
3) 30 nds BD	70	70	0	0	0	0	0	0	0	0
1) 30 nds MD	690	0	0	0	0	0	0	0	0	0
2) 30 nds MD	10	0	0	0	0	0	0	0	0	0
3) 30 nds MD	500	-280	-280	0	0	0	0	0	0	0
1) 30 nds AD	100	0	0	0	0	0	0	0	0	0
2) 30 nds AD	130	120	0	0	0	0	0	0	0	0
3) 30 nds AD	190	190	0	0	0	0	0	0	0	0
1) 300 nds BD	-4140	-1010	-40	-1500	-2640	-1680	-650	-780	-1380	1090
2) 300 nds BD	3860	-1280	-1650	-1570	770	-480	-1020	2190	-510	3460
3) 300 nds BD	-3890	-4350	-6090	-5400	-5120	-5760	-5450	-4680	-2120	3740
1) 300 nds MD	1700	-1000	3780	-2060	-2040	4760	-1730	-650	-1030	1520
2) 300 nds MD	1660	1190	170	-500	960	-4470	400	1350	-2080	1080
3) 300 nds MD	-440	-240	680	1040	-3190	-1400	-3050	-80	-720	1140
1) 300 nds AD	-400	-950	-1510	-730	-2270	-2020	-930	940	-1630	-970
2) 300 nds AD	1090	-1760	290	-210	1960	-700	0	1360	-810	170
3) 300 nds AD	-2690	2680	-5320	-2970	1000	0	-2040	-5040	2630	-1670
1) 600 nds BD	-9430	-4110	370	-14820	1470	-3480	-13060	-22190	-7200	-580
2) 600 nds BD	-15730	-12310	-6310	-10420	-7140	-9030	-3950	-2970	-2230	11360
3) 600 nds BD	-1080	11560	-1900	5390	-5630	1770	4090	-11230	8830	-7000
1) 600 nds MD	1710	1420	-4260	570	-4900	-2380	-4880	600	-3720	1910
2) 600 nds MD	-1900	-720	-16880	-5970	-7930	1940	-9530	-4550	80	1500
3) 600 nds MD	-3610	-12470	720	-7870	-6650	-11670	-3980	-9090	-6670	1780
1) 600 nds AD	-4220	-5550	-4010	-8020	-1430	-2300	-7170	-10750	-4970	9860
2) 600 nds AD	-8700	-1140	5470	-6660	2950	-2580	7140	-6630	-5810	-1790
3) 600 nds AD	-1640	-11430	-2580	-15410	-1160	-12000	-930	2190	2230	-5100

Tabla 5.3 (300 iteraciones de GRASP)

Las celdas remarcadas representan los casos en que la diferencia $resultadoGrado - resultadoPeso/Grado$ es mayor que 0, es decir, los casos en los los resultados de GRASP utilizando la heurística de Grado son mayores que utilizando la heurística de Peso/Grado. Como se puede apreciar, cuando la cantidad de iteraciones de GRASP es baja (30) la heurística de Grado es la que otorga los mejores resultados. Sin embargo al aumentar la cantidad de iteraciones los mejores resultados pasan a ser los que se obtienen por medio de la heurística de Peso/Grado. Además, se puede apreciar que la diferencia en las tablas de 100 y 300 iteraciones es muy remarcada, es decir, los números negativos son bastante grandes en módulo, lo que implica que la diferencia entre los resultados es bastante amplia.

A partir de esto se puede concluir que utilizar la heurística de Grado tiene buenos resultados para poca cantidad de iteraciones, pero al aumentar la cantidad de iteraciones, los resultados se estancan. Por el contrario, si se utiliza la heurística de Peso/Grado, la calidad de los resultados para mayor cantidad de iteraciones aumenta. El siguiente gráfico muestra esta tendencia.

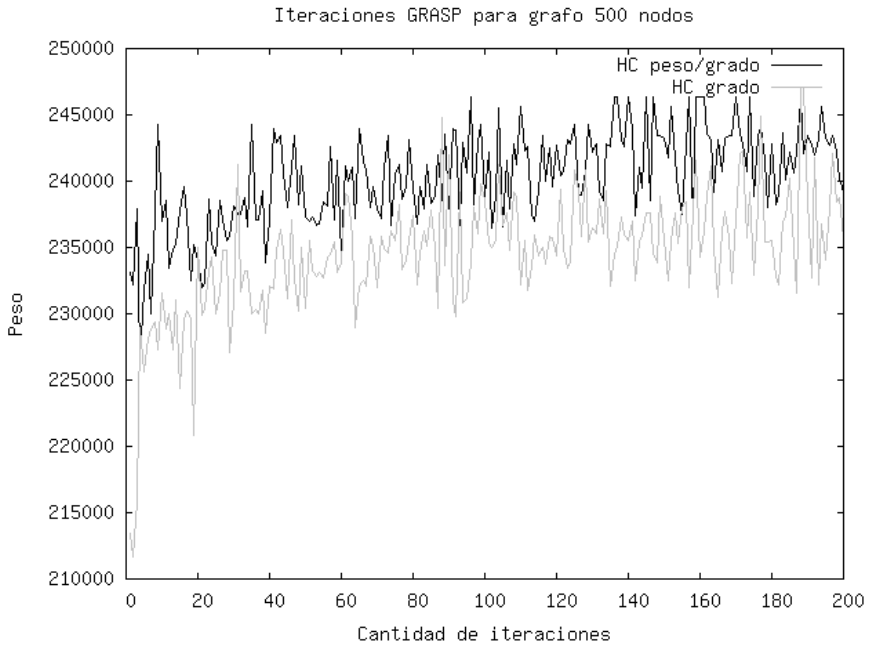

Figura 5.1

Como se puede observar, el gráfico muestra que la heurística de Peso/Grado hace que los resultados de GRASP sean mejores que cuando se utiliza la heurística de grado a medida que aumenta la cantidad de iteraciones, tal y como se había notado en las tablas anteriores. Entonces, utilizar la heurística de Peso/Grado brinda mejores resultados a costa de realizar más cantidad de iteraciones y por tanto, más tiempo de ejecución. Por esta razón, podemos concluir que utilizar dicha heurística mejor en cuanto a la calidad de las soluciones, cuestión que resulta de mayor interés que los tiempos de ejecución del algoritmo.

Análisis de los parámetros

Como se mencionó anteriormente, se estudiarán los parámetros que hacen que la metaheurística GRASP tenga un balance entre la calidad de los resultados obtenidos y el tiempo de ejecución. Las experiencias realizadas para decidir estos parámetros, se hicieron utilizando la heurística de Peso/Grado ya que en la sección anterior concluímos que es la mejor, independientemente del alfa y la cantidad de iteraciones.

En primer lugar, presentaremos las pruebas para decidir que alfa hace que los resultados sean mejores. Para esto se ejecutó GRASP con 30, 100 y 300 iteraciones, variando en cada caso el alfa de 0.1 a 1 a intervalos de 0.1. Los resultados obtenidos se presentan en las siguientes tablas.

Tipo de Grafo	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1) 30 nds BD	2080	2180	2410	2300	2380	2400	2500	2500	2500	2500
2) 30 nds BD	2450	2460	2460	2530	2460	2540	2470	2540	2540	2540
3) 30 nds BD	2610	2610	2680	2680	2680	2680	2680	2680	2680	2680
1) 30 nds MD	1800	2490	2490	2490	2490	2490	2490	2490	2490	2490
2) 30 nds MD	2240	2250	2250	2250	2250	2240	2240	2250	2250	2250
3) 30 nds MD	1670	2450	2450	2450	2450	2450	2450	2450	2450	2450
1) 30 nds AD	1830	1930	1930	1930	1930	1930	1930	1930	1930	1930
2) 30 nds AD	1700	1710	1830	1830	1830	1830	1830	1830	1830	1830
3) 30 nds AD	1740	1740	1800	1740	1930	1930	1930	1930	1930	1930
1) 300 nds BD	126980	128300	124600	128010	127030	124520	126970	125230	122870	121440
2) 300 nds BD	139640	143510	140680	139630	141170	138760	141030	136720	136380	139890
3) 300 nds BD	124630	124580	122510	123080	127010	126320	123570	126590	122830	118560
1) 300 nds MD	104180	105170	105710	107950	105920	104890	102700	104440	102070	102800
2) 300 nds MD	96680	97210	96080	99060	96820	95390	96340	94860	94920	94760
3) 300 nds MD	105130	104220	107250	102820	104150	103450	106090	103340	98320	98080
1) 300 nds AD	86670	84800	85000	83450	84400	86450	87460	82130	83120	82930
2) 300 nds AD	84870	86260	84590	84380	82640	84820	80800	80400	83510	78560
3) 300 nds AD	83290	87050	87290	89600	87090	85320	85370	84040	89120	85230
1) 600 nds BD	431330	435600	431130	432420	424090	424860	424490	417660	419710	425130
2) 600 nds BD	435340	443700	441800	440750	443500	453620	430760	428280	421560	420030
3) 600 nds BD	437810	448410	457490	452170	462240	461540	448970	447300	447150	435370
1) 600 nds MD	334190	325450	339040	331990	333050	326820	322680	342600	324730	329890
2) 600 nds MD	329570	315260	315380	317070	311590	301320	306610	300170	307700	310590
3) 600 nds MD	341240	338500	338580	342210	328050	341450	327920	328960	331420	331200
1) 600 nds AD	305830	296360	298430	297820	298480	302840	291010	300570	293770	288050
2) 600 nds AD	278040	275070	280760	275580	277110	284740	271730	272170	272920	262530
3) 600 nds AD	293930	301670	303320	293530	295730	290250	285360	286450	286630	284160

Tabla 5.4 (30 iteraciones de GRASP)

Tipo de Grafo	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1) 30 nds BD	2080	2180	2410	2410	2500	2490	2500	2500	2500	2500
2) 30 nds BD	2450	2470	2530	2530	2530	2540	2540	2540	2540	2540
3) 30 nds BD	2610	2610	2680	2680	2680	2680	2680	2680	2680	2680
1) 30 nds MD	1800	2490	2490	2490	2490	2490	2490	2490	2490	2490
2) 30 nds MD	2240	2250	2250	2250	2250	2250	2250	2250	2250	2250
3) 30 nds MD	1670	2450	2450	2450	2450	2450	2450	2450	2450	2450
1) 30 nds AD	1830	1930	1930	1930	1930	1930	1930	1930	1930	1930
2) 30 nds AD	1700	1710	1830	1830	1830	1830	1830	1830	1830	1830
3) 30 nds AD	1740	1740	1930	1930	1800	1930	1930	1930	1930	1930
1) 300 nds BD	129760	128410	127830	129120	129310	128120	127640	126790	127490	123120
2) 300 nds BD	143510	143640	143510	142630	140810	140030	139310	137640	137060	137810
3) 300 nds BD	124630	124830	126360	125680	128810	124860	125970	129120	126540	121190
1) 300 nds MD	105430	107320	108100	107950	107740	107000	107380	103330	108630	106000
2) 300 nds MD	96680	99240	96720	98910	98570	98130	98690	98370	97650	96550
3) 300 nds MD	104730	105870	103890	104260	107490	104410	105610	106580	103580	105440
1) 300 nds AD	86670	87070	86800	86290	86550	85750	85480	85630	83340	86870
2) 300 nds AD	84870	86390	86470	85760	85770	85210	85740	84080	83820	84620
3) 300 nds AD	86750	86030	88750	86880	88770	88170	90130	86710	87550	86430
1) 600 nds BD	432140	435630	428610	429960	437950	437530	422520	430080	417230	417200
2) 600 nds BD	441770	440480	440800	445680	445200	441460	429070	428590	426760	439800
3) 600 nds BD	453570	451280	456280	459180	451160	460690	462740	460350	451320	459030
1) 600 nds MD	340090	340100	335730	337790	334080	331980	332580	338300	335090	323450
2) 600 nds MD	327040	320970	322950	324810	327360	320360	319160	317210	323760	309210
3) 600 nds MD	341530	353830	342480	345110	344170	339250	331040	333100	338600	330940
1) 600 nds AD	305830	310070	302690	305420	301660	303160	301680	299890	291280	289990
2) 600 nds AD	279630	278980	279220	283410	277760	285450	274030	275190	272310	276020
3) 600 nds AD	288980	298640	303340	298560	293050	294360	297670	281380	286400	293120

Tabla 5.5 (100 iteraciones de GRASP)

Tipo de Grafo	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1) 30 nds BD	2080	2410	2410	2410	2500	2490	2500	2500	2500	2500
2) 30 nds BD	2450	2470	2530	2540	2540	2540	2540	2540	2540	2540
3) 30 nds BD	2610	2610	2680	2680	2680	2680	2680	2680	2680	2680
1) 30 nds MD	1800	2490	2490	2490	2490	2490	2490	2490	2490	2490
2) 30 nds MD	2240	2250	2250	2250	2250	2250	2250	2250	2250	2250
3) 30 nds MD	1670	2450	2450	2450	2450	2450	2450	2450	2450	2450
1) 30 nds AD	1830	1930	1930	1930	1930	1930	1930	1930	1930	1930
2) 30 nds AD	1700	1710	1830	1830	1830	1830	1830	1830	1830	1830
3) 30 nds AD	1740	1740	1930	1930	1930	1930	1930	1930	1930	1930
1) 300 nds BD	129760	131520	127680	128760	130150	129300	129460	129400	127010	126160
2) 300 nds BD	140100	144830	143720	143700	142550	143500	142860	140150	142080	139220
3) 300 nds BD	126490	128660	127640	127840	129170	127440	129430	128150	123750	125630
1) 300 nds MD	105430	108940	107230	109190	107970	105880	109720	107740	107250	106080
2) 300 nds MD	98000	98470	98290	98100	98670	101310	98030	97310	98080	96880
3) 300 nds MD	104560	106050	105410	106410	106760	107430	106950	104970	103940	104030
1) 300 nds AD	86670	87280	89260	87500	87460	87550	86670	86340	86110	87250
2) 300 nds AD	84870	86930	86470	87130	86390	86610	85570	85050	86870	87410
3) 300 nds AD	86750	89010	92410	89090	88100	88520	92410	91990	86690	88900
1) 600 nds BD	439760	436470	437080	440810	435890	435180	437640	450310	428780	422500
2) 600 nds BD	453400	444920	449090	452880	447550	446850	443040	437320	428170	431800
3) 600 nds BD	459290	458870	463050	457950	460540	457710	457850	461120	450510	456400
1) 600 nds MD	334920	340570	339930	334640	338470	339330	338360	333230	333830	324440
2) 600 nds MD	325520	326550	332110	324750	320300	321570	324400	316600	316290	311020
3) 600 nds MD	344930	353830	349180	349400	344230	347830	347990	342160	345660	334560
1) 600 nds AD	308390	308560	311740	312490	309150	303490	309200	309660	301590	301960
2) 600 nds AD	283120	281680	277600	290150	280970	280390	275790	281770	281410	272980
3) 600 nds AD	296350	305380	297710	305590	297570	299420	293480	297520	290560	294510

Tabla 5.6 (300 iteraciones de GRASP)

En las tablas se encuentran remarcados los mayores resultados para cada nodo estudiado. Se puede notar que para grafos pequeños (30 nodos) el parámetro alfa no incide demasiado en los resultados obtenidos ya que para alfas desde 0.2 a 1 se obtienen prácticamente los mismos resultados, independientemente de la cantidad de iteraciones. A medida que el tamaño de los grafos crece los mayores resultados obtenidos aparecen para distintos alfa. En particular, se puede notar una tendencia que muestra que los mayores resultados se obtienen para alfas entre 0.2 y 0.3, siendo 0.2 el que más resultados máximos registra. Esta tendencia ocurre en las tres tablas presentadas, pero es mucho más marcada en las que muestran gran cantidad de iteraciones de GRASP. A partir de estos resultados, podemos concluir que el parámetro alfa indicado para GRASP es 0.2, ya que es efectivo tanto para grafos pequeños como para grafos con gran cantidad de nodos. Al parecer resulta que dicho valor equilibra el aspecto aleatorio con la calidad del algoritmo goloso generando así buenas soluciones para distintos tipos de grafos.

Otra de las pruebas realizadas tiene como objetivo determinar la cantidad de iteraciones de GRASP que hacen que la calidad de las soluciones sea buena y que, a su vez, el tiempo de ejecución no sea excesivo. Con este objetivo se creo una prueba que ejecuta GRASP para grafos aleatorios de 500 nodos y densidad media, variando la cantidad de iteraciones de 1 a 200. Además fijamos el parámetro alfa en 0.2 ya que las pruebas anteriores mostraron que es el valor adecuado para cualquier tipo de grafos. Los resultados de esta prueba fueron los siguientes.

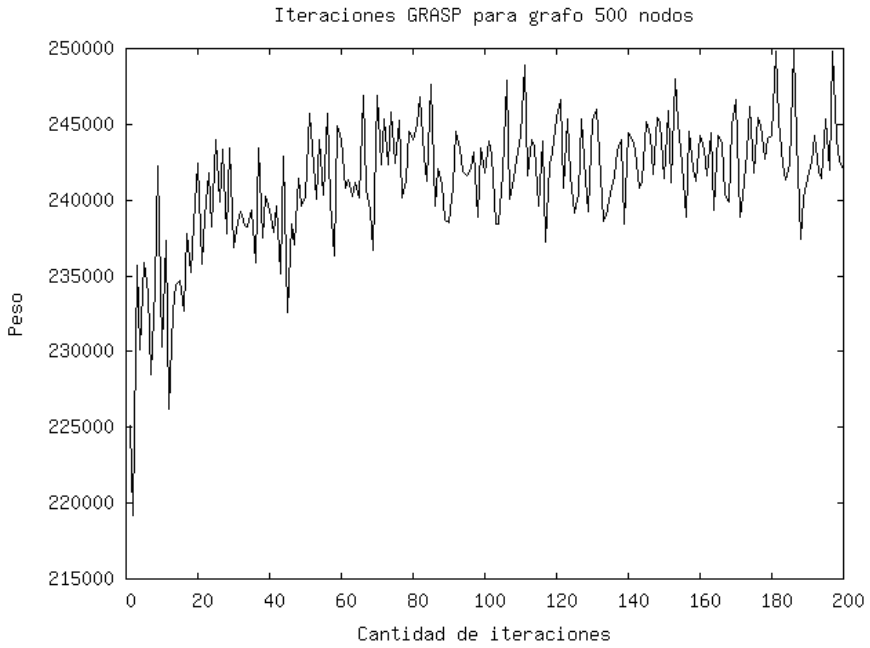

Figura 5.2

En el gráfico presentado se puede notar que los resultados de GRASP crecen de manera sostenida hasta que la cantidad de iteraciones es 70. A partir de este punto, el crecimiento se vuelve muy leve, es decir, a pesar de que la cantidad de iteraciones aumente la calidad de las soluciones crecerá muy poco. Por esta razón, establecer la cantidad de iteraciones en algún valor mayor que 70 no mejoraría demasiado la solución que se tiene hasta ese paso y el tiempo de ejecución sería mayor. En consecuencia, fijaremos la cantidad de iteraciones de GRASP en 70, ya que establece un balance entre la calidad de la solución y el tiempo de ejecución.

Para finalizar, se presenta una tabla que resume el comportamiento de todos los algoritmos heurísticos con el fin de poder comparar su eficacia y eficiencia para grafos de 600 nodos con distintas densidades.

Tipo de Grafo	H. Constructiva	Búsqueda Local	GRASP
1) 600 nds BD	185800 (99 ms.)	366250 (204 ms.)	440560 (9479 ms.)
2) 600 nds BD	186190 (68 ms.)	433830 (177 ms.)	442610 (9923 ms.)
3) 600 nds BD	181980 (65 ms.)	418860 (144 ms.)	449960 (10852 ms.)
1) 600 nds MD	125810 (75 ms.)	320270 (133 ms.)	339880 (8405 ms.)
2) 600 nds MD	114880 (63 ms.)	279480 (187 ms.)	323150 (7851 ms.)
3) 600 nds MD	125940 (72 ms.)	318310 (123 ms.)	340820 (8261 ms.)
1) 600 nds AD	110060 (101 ms.)	289470 (240 ms.)	304990 (8745 ms.)
2) 600 nds AD	104810 (74 ms.)	240830 (169 ms.)	280910 (7701 ms.)
3) 600 nds AD	102590 (70 ms.)	266770 (129 ms.)	297060 (7810 ms.)

Figura 5.3

Como se observa, GRASP encuentra soluciones aún mejores que la búsqueda local sola en todos los casos para grafos de gran tamaño a costa de un mayor tiempo de ejecución. Por todo lo visto, estos resultados confirman todo lo conjeturado anteriormente.

Complejidad

```
pesoMaximo = 0
para i desde 1 hasta cantIteracionesGrasp
  unaSolucion = heuristicaConstructiva(alfaRCL)
  unaSolucion = busquedaLocal(unaSolucion, cantIteracionesBL)
  si peso(unaSolucion) > pesoMaximo
    laMejorSolucion = unaSolucion
    pesoMaximo = peso(laMejorSolucion)
```

En primer lugar, cabe notar que el ciclo principal itera una cantidad constante de veces ya que, tal como se mencionó anteriormente, el parámetro *cantIteracionesGrasp* es un número prefijado que no depende del tamaño de la entrada. Por lo tanto, la complejidad estará dada por el resto de las operaciones dentro del mismo. En particular, la llamada a la heurística contructiva junto con la llamada a la búsqueda local son las operaciones relevantes. Como se mencionó, se optó por utilizar la heurística constructiva peso/grado de complejidad de orden $O(n * m)$. Asímismo, la búsqueda local elegida tiene la misma complejidad, lo que resulta en que la complejidad total de GRASP es de orden $O(n * m)$. A continuación, analizaremos empíricamente este resultado.

A contunuación analizaremos la complejidad en función del tamaño de la entrada. Sea t el tamaño de la entrada, n la cantidad de nodos, m la cantidad de aristas (m_i la cantidad de aristas del nodo i), p el arreglo de pesos y $a[i]$ las adyacencias del nodo i. Tenemos entonces que:

$$t = \log(n) + \sum_{i=1}^{n} \log(p) + \sum_{i=1}^{n} \sum_{j=1}^{m_i} \log(a[i]) > \log(n) + \sum_{i=1}^{n} 1 + \sum_{i=1}^{n} \sum_{j=1}^{m_i} 1$$

$$> \log(n) + n + \sum_{i=1}^{n} m > \log(n) + n + nm$$

$$\implies \text{como } T(n,m) \in O(nm) \text{ y } nm < \log(n) + n + nm < t \implies T(t) \in O(t)$$

Análisis de tiempo de ejecución

El siguiente gráfico muestra los tiempos de ejecución del algoritmo de GRASP propuesto. Al igual que antes, se analizan resultados con grafos aleatorios para 3 densidades, en este caso con nodos desde 10 hasta 800.

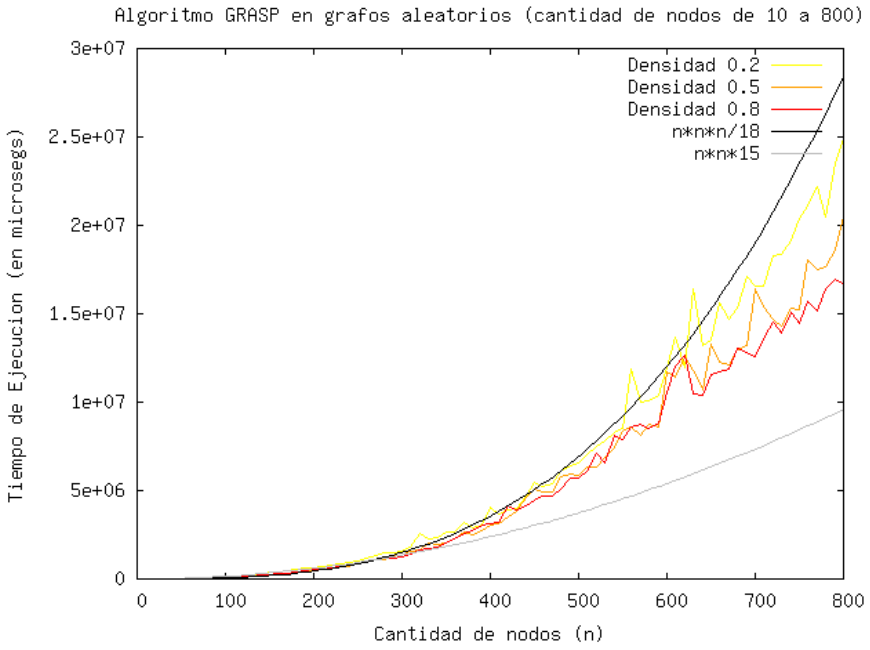

Figura 5.4

Como era de esperarse, el comportamiento del algoritmo es similar al comportamiento de la búsqueda local analizada previamente, ya básicamente el algorimto consiste en realizar sucesivas búsquedas locales. Sin embargo, se puede observar que en este caso la complejidad de GRASP se asemeja más a n^3 que en el caso de la búsqueda local. La explicación de este hecho podría recidir en que la búsqueda local de GRASP utiliza la heurística de grado mientras que en GRASP se ha optado por utilizar una búsqueda local con la heurística de peso/grado. De todos modos, se puede observar como claramente el algoritmo respeta la complejidad teórica $O(n * m)$ calculada.

En cuanto a casos patológicos para la metaheurística resulta complejo detectarlos ya que el algoritmo tiene aspectos aleatorios por lo cuál un peor caso para la búsqueda local no necesariamente implica que el algoritmo no pueda hallar una buena solución en otra de sus iteraciones ya que se parten de soluciones iniciales distintas (no determinísticas).

6. Conclusiones Generales

El trabajo aqui realizado se encargó de mostrar diversos algoritmos para resolver el problema del conjunto independiente de peso máximo para un grafo de nodos ponderados cualquiera. En primer lugar se realizó un simple algoritmo exacto de orden $O(2^n)$ para posteriores comparaciones con las heuristicas propuestas. Este orden exponencial, mostró luego que el algoritmo sólo termina en un tiempo razonable para grafos relativamente pequeños (hasta 36 nodos, con baja densidad).

Luego, se comenzó por mostrar distintas heurísticas constructivas para generar soluciones al problema para grafos más grandes. Como vimos, si bien se mostraron heurísticas mejores que otras, todas ellas resultaron estar relativamente lejos de la solución exacta. Por pruebas realizadas, se concluyó que la mejor heurística resultó ser una heurística golosa que añade los nodos con mayor relación peso/grado del grafo a la solucion parcial. De este modo, el

algorimto encontraría soluciones en un tiempo polinomial $O(n*m)$ permitiendo ser ejecutado sobre grafos de un orden de 10^3 de baja densidad en tiempos razonables.

En la siguiente etapa se estudiaron 2 heurísticas de búsqueda local con el fin de mejorar las soluciones contructivas obtenidas. Resultó interesante ver como una busqueda local bastante más inteligente dió resultados notablemente mejores, aún siendo de una complejidad temporal aceptable como ser $O(n*m)$. Más aún, realizar la heurística sola o utilizar una busqueda local a posteriori conserva la complejidad de $O(n*m)$ por lo cual se puede concluir que la heurística por sí sola carece de utilidad. Otro punto interesante, es que la búsqueda realiza una cantidad de iteraciónes que aumenta muy levemente con respecto al tamaño del grafo, por lo cual fue posible acotar dicho parámetro sin perder en calidad de soluciones encontradas.

Por último se estudió una versión del esquema metaheurístico GRASP que muestra como la mejora en la calidad de las soluciones es aún mayor que realizar una búsqueda local. Tal como explica la bibliografía asociada, este resultado tiene relación con el hecho de que se realizan distintas busquedas locales en diversas soluciones iniciales lo que resulta en el aumentar notablemente la posibilidad de hallar soluciones mejores. Además, al haber fijado la cantidad de iteraciones del procedimiento, la complejidad teórica sigue siendo del mismo órden que realizar una única busqueda local, es decir $O(n*m)$. Esto último muestra como este esquema metaheurístico resulta de gran utilidad al momento de querer encontrar soluciones de calidad para problemas de gran tamaño imposibles de resolver de forma exacta.

Referencias

- "Greedy Randomized Adaptive Search Procedures" - Thomas A. Feo, Mauricio G.C. Resende
- Artículo de Wikipedia sobre Conjunto independiente