Búsquedas Heurísticas

Iñigo Tamayo Uria

7 de abril de 2016

Resumen

En este artículo se analiza el comportamiento de varios algoritmos de búsqueda huerística sobre los paradigmas que representan el problema de la asignación cuadrática y la bipartición de un grafo ambos de índole combinatoria. Con este objetivo, se ha implementado el algoritmo GRASP basado en búsquedas locales y un algoritmo genético enmarcado dentro de los algoritmos poblacionales. Los experimentos realizados, comparan ambos algoritmos a nivel de computación y eficiencia.

1. Análisis de la optimización

1.1. Problema de asignación cuadrática

El problema de asignación cuadrática, es un problema de Optimización Combinatoria (OC) de alta complejidad computacional, y consiste en encontrar una permutación de asignación óptima de n instalaciones a n localidades con el propósito de minimizar el costo de transporte, dadas dos matrices simétricas una de distancias y otra de flujos.

El QAP (*Quadratic assignment problem*) fue propuesto por Koopmans y Beckmann en 1957. En 1976, Shani y González probaron que QAP es un problema NP-completo. Hasta hoy sólo se han encontrado soluciones óptimas usando métodos exactos para instancias de tamaño menores que 30. El QAP aparece en muchas aplicaciones, tales como el diseño de teclados de computadora, la programación de manufactura, el diseño de terminales en aeropuertos y procesos de comunicaciones, entre otras. En los últimos años se han propuesto métodos de búsqueda de soluciones conocidas como metaheurísticas (MH), que son procedimientos de aproximación de propósito general, tales como los algoritmos genéticos, búsqueda tabú y GRASP [4] entre otras.

Para definir estos datos, se definen dos matrices (la matriz de distancias y la matriz de flujos de material) que modelan el comportamiento del sistema de instalaciones. La matriz de distancias se define como $D = [d_{ij}]$, donde d_{ij} representa el coste de transporte de la localización i a la localización j. La

matriz de flujos se define como $F = [f_{kl}]$, que representa el flujo de material entre la instalación k a la instalación l.

Cada configuración (una posible ubicación para todas las instalaciones) se representa en forma de permutación (ver ecuación 1).

$$\sigma = (\sigma_1\sigma_2\sigma_3...\sigma_n) \tag{1}$$

La función de coste a minimizar por lo tanto, tendrá en cuenta el coste de transporte dependiendo del flujo entre las instalaciones y la distancia entre las localizaciones. Esta función se puede modelar como muestra la ecuación 2.

$$f(\sigma) = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{\sigma(i)\sigma(j)}f_{ij} \tag{2}$$

1.2. Problema de Bipartición del grafo

El problema de la partición de un grafo consiste en un grafo no dirigido $G = (X, U)$ con $X \neq \emptyset$ y $U = (u, v)|u, v \in X$ en el que cada arista (u, v) tiene asociado un peso $p(u, v)$ y el número de vértices es par. Se trata de dividir el conjunto de vértices en dos subconjuntos iguales, de forma que se maximice la suma de los pesos asociados a aristas que unen vértices de diferentes conjuntos.

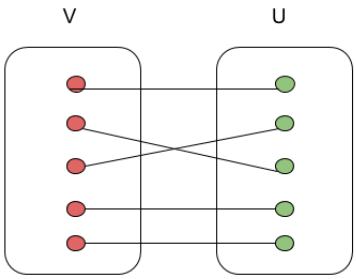

Figura 1: Ejemplo de una biparticion del grafo.

Un caso práctico, sería la de una empresa con V puestos vacantes y U trabajadores con cualidades significativas, donde maximizar la biparticion del grafo permitiría asignar los puestos aprovechando al máximo las cualidades de una forma global.

2. Propuestas de solución basadas en búsqueda local

Como se ha mencionado anteriormente, el algoritmo implementado basado en búsqueda local es el "Greedy randomized adaptive search procedure",

más conocido como GRASP. Es un método metaheurístico que fue creado a finales de los 80 (Feo, Resende,1989) para resolver problemas de optimización combinatoria. Se puede ver como un algoritmo multi-arranque, que genera soluciones iniciales de cierta calidad que se apoya en búsquedas locales con el objetivo de encontrar mejores resultados y así evitar caer en óptimos locales.

El algoritmo GRASP se divide principalmente en dos fases; una fase denominada construcción y otra fase de búsqueda. En la fase de construcción se genera una solución partiendo de un nodo seleccionado aleatoriamente. Sobre este nodo se buscan los siguientes nodos que tengan el menor o mayor coste, minimizando o maximizando respectivamente, y obteniendo una lista de nodos ordenados por coste como resultado. Sobre esta lista, elegimos un candidato dada la lista de probabilidades de cada uno de los candidatos. Se procede así sucesivamente hasta generar la solución.

La fase de búsqueda, se basa en un bucle con cierto criterio de parada. Cada iteración genera una nueva solución a través del constructor. Sobre esta primera solución se realiza una búsqueda local en búsqueda de nodos vecinos que mejoren el resultado actual. En cada iteración, se analizan los resultados y se actualiza la lista de los mejores resultados obtenidos.

Existen infinidad de variaciones del planteamiento inicial de GRASP. En este trabajo se ha utilizado GRASP basado en cardinalidad por su sencillez y robustez.

El pseudo-código general para este algoritmo sería el siguiente:

```
procedure GRASP
Require: i_máx
  f* ← ∞;
  for i ≤ i_máx do
    x ← GreedyRandomized();
    x ← LocalSearch(x);
    if f(x) < f* then
      f* ← f(x);
      x* ← x;
    end if
  end for
  return x*;
```

Figura 2: Pseudocódigo de *GRASP* fase de búsqueda.

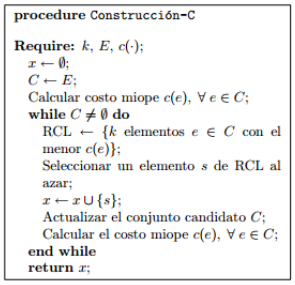

Figura 3: Pseudocódigo de *Constructor* basado en cardinalidad.

La diferencia principal en lo que respecta a la algoritmia de los dos problemas que se están analizando, reside en la forma de generar las mejores soluciones iniciales. Hay que tener en cuenta, que en la problemática QAP el objetivo es minimizar los resultados, mientras que en la bipartición del grafo el objetivo es maximizar el resultado. Además ambos son de naturaleza diferentes; la primera es una permutación, mientras que la segunda es un vector de posiciones con sus respectivas características.

En el caso de QAP, se calcula la probabilidad de cada uno de los nodos de una lista reducida, según el coste del mismo con probabilidad inversa, ya que es un problema de minimización, invirtiendo la lista que se genera con la ecuación 3:

$$P = \frac{Coste(n,i)}{\sum_{i}^{n} Coste(n,i)} \tag{3}$$

En el caso de la optimización de la bipartición del grafo, se ha optado por ir generando el vector comparando el nodo con el vector en construcción. Es decir, la comparación se realiza, con el coste total que generaría añadir un 1 o 0 al vector en construcción.

A lo que al criterio de parada se refiere, se ha optado por además de tener un máximo de iteraciones posibles del bucle principal, también se ha limitado la búsqueda local al máximo de iteraciones. Otro criterio de parada se da si la búsqueda local no ha mejorado el mejor resultado en 4 iteraciones, y el numero de iteraciones ha sobrepasado el 66 % de las iteraciones máximas permitidas.

2.1. Configuración final

Con el objetivo de realizar una evaluación eficiente, se han establecido algunos parámetros definidos en la tabla 1, con los cuales se obtienen resultados en tiempo razonable. A su vez, se ha observado que el GRASP y el respectivo algoritmo evolutivo necesitan un tiempo relativamente parecido, lo cual permite realizar mediciones aproximadas entre ambos algoritmos.

Parámetro	Valor
Número máximo de evaluaciones	200
Número de iteraciones antes de guardar	20
Número de iteraciones sin mejorar tras mínimo local	3
% de máxima evaluaciones para activar contador de mejoras	66

Cuadro 1: Lista de parámetros de configuración para el algoritmo GRASP

3. Propuestas de solución basadas en algoritmos poblacionales

El algoritmo poblacional implementado es un algoritmo genético. Este algoritmo se basa en las teorías clásicas de la evolución darwiniana como base para generar una serie de iteraciones que generan una población mejor que la generación anterior.

Un algoritmo genético consta de cuatro fases principales: evaluación, selección, cruce y mutación. Cada iteración genera una nueva generación después de pasar por todos estos estados. Se puede poner un criterio de terminación, con el que podemos decidir si la generación actual es ya lo suficientemente buena para el resultado, o si se han utilizado los recursos asignados independientemente del resultado. La figura 4 muestra gráficamente la secuencia de estados de una iteración.

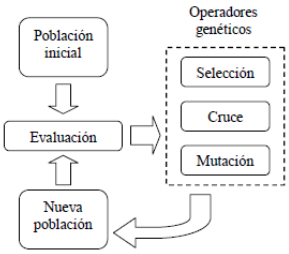

Figura 4: Secuencia de una iteración de un algoritmo genético.

En este trabajo se ha realizado una extensión al algoritmo genético simple. El algoritmo genético simple genera la población de forma aleatoria, lo cual hace que se necesiten más generaciones para llegar a una población con cualidades que son interesantes heredar. Es por ello, que se ha propuesto utilizar el algoritmo Greedy para generar la población inicial [2]. Este algoritmo posibilita que la población inicial esté compuesta por individuos que tenga mejores cualidades que las de una aleatoria.

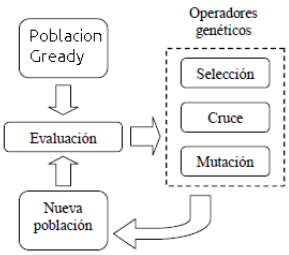

Figura 5: Secuencia de una iteración de un algoritmo genético con generación de población a través de greedy.

3.1. Fase de Evaluación

En la fase de evaluación se evalúan todos los individuos de la población actual. Esto se hace utilizando las funciones de coste que se han expuesto en el apartado 1.1.

Por lo tanto, cada individuo tendrá asociado el valor de su resultado, dando un indicador de la çalidad"de dicho individuo.

3.2. Fase de Selección

En la fase de selección se escogen los individuos de la población con mejores resultados. Para ello se pueden utilizar diferentes opciones. De todas las opciones existentes, se ha escogido la *selección por torneo*.

Esta selección se realiza mediante una comparación entre un subconjunto de individuos de tamaño concreto (definido como parámetro), escogidos al azar dentro de la población.

Este tipo de selección se efectúa de manera muy rápida dado que no es necesario evaluar la totalidad de la población. No obstante, hay que definir un nuevo parámetro por lo que añade complejidad a la configuración.

3.3. Fase de Cruce

En esta fase, se unen y combinan los diferentes individuos en el subconjunto de *mejores* individuos. Es necesario encontrar una función de cruce que genere nuevos individuos aprovechando las cualidades de los originales, y tratando de transmitir esas cualidades a las nuevas generaciones.

La primera de ellas, combina dos individuos por mitades. Es decir, coge la primera mitad de un individuo A, y la segunda mitad de un individuo B. Los nuevos individuos (denominados A' y B') tendrán la primera parte de A y la segunda de B respectivamente. Para rellenar la mitad restante, lo que se hace es colocar los valores que faltan por incluir con el mismo orden que aparece en el individuo original. Por lo tanto, A' tendrá la primera mitad de A, y los elementos restante en el mismo orden que aparecen en B, y B'

tendrá la segunda mitad de B y los demás valore en el mismo orden que aparecen en A.

Esta opción se descarto, ya que por un lado no garantizaba la herencia de las mejores cualidades de los individuos padres y por otro lado surgían problemas a la hora de mantener las restricciones.

Considerando que la generación de la población inicial ya asegura ciertas características que mejoran los resultados, se optó por un algoritmo capaz de transmitir estas cualidades en un gran porcentaje, y por ello se eligió el algoritmo "swap path crossover" [1].

El algoritmo swap path crossover funciona de forma que se elige un nodo del vector/permutación de forma aleatoria del individuo, y se intercambia con el nodo precedente. Después se guarda el hijo generado y se calcula su calidad/fitness. Ahora en el individuo dos, se cambian los mismo elementos de sitio, y se genera un nuevo hijo. Seguidamente, se comparan los hijos de forma que se guarda el mejor de ellos. Este proceso se repite hasta recorrer todo el índice del vector/permutación. Una vez acabado el proceso, se seleccionan de la lista de hijos generados los mejores dos. De esta manera no sólo se heredan las mejores características sino que es capaz de encontrar nuevos espacios de búsqueda.

Parent 1:	5-2-3-4-1-7-6	**Swap in Parent 1:**	2-5-3-4-1-7-6
Parent 2:	2-1-3-4-6-5-7	**Swap in Parent 2:**	5-1-3-4-6-2-7

Figura 6: Secuencia parical del algoritmo path crossover.

Figura 7: La figura muestra de forma gráfica el proceso de cruce escogido para el algoritmo genético. En el primer paso (izquierda), se seleccionan los elementos con un índice escogido al azar. En el segundo (centro), se intercambia el elemento entre los dos individuos, y se detecta la posición del número que va a ser sustituido. El tercer paso (derecha), se pueden ver los nuevos individuos, y como no tienen repeticiones.

3.4. Fase de Mutación

Para aumentar la variedad de posibilidades, se añade un factor de mutación. Este factor hace que una parte de la población mute (tenga un cambio adicional) en cada iteración. Con esto conseguimos configuraciones que no estuviesen en la generación anterior, y que la fase de cruce no puede generar.

La función de mutación escogida es la de intercambiar dos elementos del individuo. Para ello, se escogen dos índices al azar, y se intercambian los elementos.

Esta fase añade un elemento a configurar, que es la probabilidad de mutación de un individuo.

3.5. Configuración final

Con el objetivo de poder evaluar de forma eficiente se han establecido algunos parámetros según los 2 cuales se obtiene resultado más o menos en tiempo razonables. A su vez, se ha observado que el GRASP y el algoritmo evolutivo necesitan un tiempo relativamente parecido, lo cual permite realizar las comparaciones entre ambos.

Parámetro	Valor
Tamaño población	200
Número de generaciones	20
Factor Mutación	0.2
Tamaño subconjunto torneo	3
Factor de cruce	0.7

Cuadro 2: Lista de parámetros de configuración para el algoritmo genético.

4. Experimentación

La experimentación se ha divido en dos sub-secciones una por cada problemática analizada. Se ha optado por esta distribución dado que los análisis realizados y los sets de datos empleados son de diferente naturaleza y de diferentes dimensiones.

Con el objetivo de analizar el comportamiento de los algoritmos desarrollados, se ha realizado varios test empíricos. Las pruebas se han basado en realizar varias ejecuciones de los algoritmos con diferentes parámetros para aproximar el tiempo de computo de los dos algoritmos analizados y adecuar los parámetros. El objetivo marcado era obtener el mejor resultado posible en un máximo de 3 minutos 8.

Una vez ajustado los parámetros, se ha procedido a la recogida de datos, a través de un script desarrollado. Dado que los algoritmos no son deterministas se ha procedido a ejecutar cada uno de los algoritmos 10 veces con los N sets de datos para recoger datos tales como; el mejor resultado,la media de los mejores resultados, la varianza de los resultados así como la media del tiempo de cómputo.

4.1. QAP

Para la experimentación de la problemática QAP se han utilizando 10 sets de datos diferentes. Las dimensiones de los sets de datos oscilan entre 10

y 50 nodos, por cada dimensión se ha utilizado dos sets de datos diferentes.

Por otro lado, se ha procedido a ejecutar los algoritmos con parámetros exigentes en tiempo de computo para poder calcular los mejores resultados a través del los algoritmos desarrollados. Los mejores resultados se han utilizado posteriormente como referencia de calculo de la desviación.

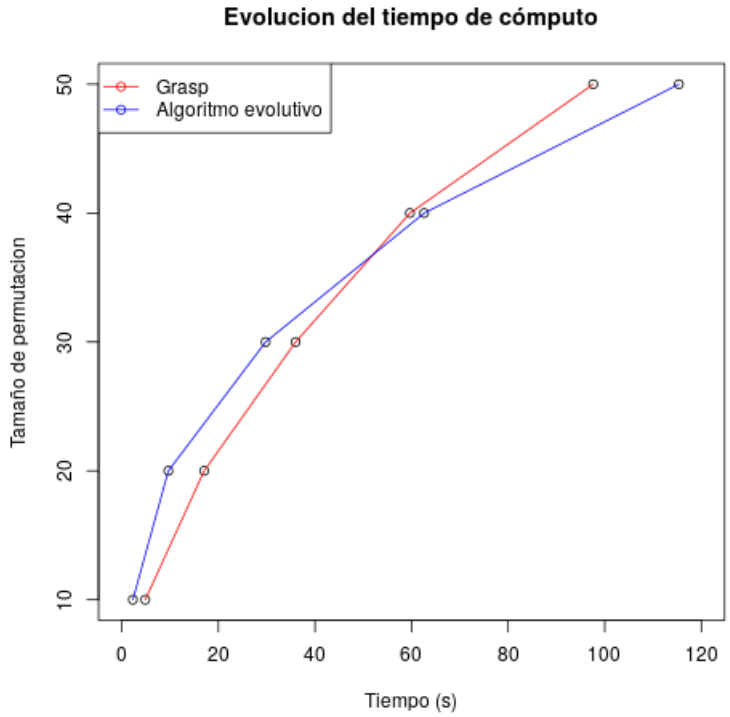

Figura 8: Tiempo de cómputo de los algoritmos desarrollados.

En la siguiente tabla se muestra los datos relacionados con la búsqueda de diferentes sets de datos utilizando para ello el algoritmo de búsqueda local GRASP 3.

Num Edificios	Media	Varianza	Mejor Valor
10.1	3.971134+06	6.974391222+09	3.971134+06
10.2	1.92157+05	8.8350+04	1.91405+05
20.1	2.2668727+07	2.365639690102+12	2.1122412+07
20.2	8.30385+05	8.32517+05	8.23442+05
30.1	1.35997151+08	3.2409157443866+013	1.30125487+08
30.2	1.845181+06	1.04514178+08	1.834046+06
40.1	2.20375820+08	1.01157799328448+014	2.03647638+08
40.2	7.69172+05	1.64398078839+10	7.54934+05
50.1	3.83819392+08	9.26172128058705+013	3.83819392+08
50.2	5.5893422+06	1.76830320+08	5.556157+08

Cuadro 3: Resultados de la problemática QAP con el algoritmo Grasp.

En la siguiente tabla se muestra los datos relacionados con la búsqueda de diferentes sets de datos utilizando para ello el algoritmo de evolución genética. 4.

Num Fabricas	Media	Varianza	Mejor Valor
10.1	4.473354+06	7.6676915183+010	3.986340+06
10.2	1.94520+05	5.43667+05	1.93258+05
20.1	2.9312464+07	4.792201405630+12	2.5591115+07
20.2	8.51987+05	1.2557748+07	8.44886+05
30.1	1.54982818+08	4.7044080478746+013	1.40335056+08
30.2	1.881750+06	1.23993034+08	1.846891+06
40.1	2.54484262+08	3.3421645702534+013	2.45261563+08
40.2	7.66769+05	7.327327+06	7.62695+05
50.1	4.23455419+08	6.71382282628101+013	4.08923255+08
50.2	5.6294542+06	1.61706692+08	5.5970780+08

Cuadro 4: Resultados obtenidos por el algoritmo Genético.

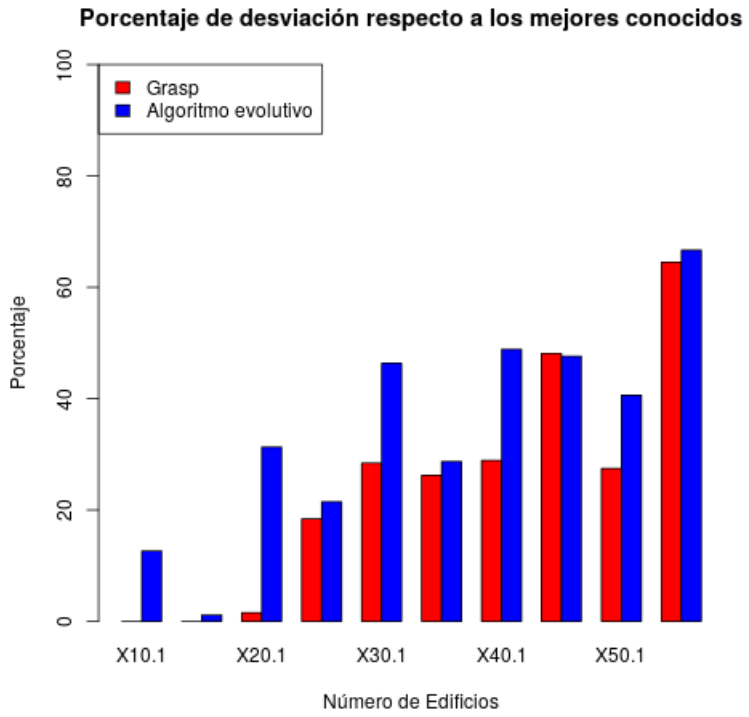

Figura 9: Desviación de los algoritmos desarrollados sobre los mejores resultados conocidos.

Num Fabricas	Mejor Valor conocido
10.1	3.971134+06
10.2	1.91405e+05
20.1	2.1023498e+07
20.2	7.01322+05
30.1	1.05899563+08
30.2	1.461874+06
40.1	1.70962023+08
40.2	5.19440+05
50.1	3.01133828e+08
50.2	3.377636e+06

Cuadro 5: Los mejores valores conocidos para cada dataset.

En las tablas se puede ver la media de los resultados, la varianza y el mejor valor entre las 10 ejecuciones. Estos datos se pueden comparar con los mejores valores conocidos de cada uno sets de datos, listados en la tabla 5.

Con estos datos, podemos comparar el desviación aproximada de cada uno de los algoritmos. En la figura 9 podemos ver, que los algoritmos no siguen un patrón respecto van aumentado la dimensión del problema, lo que puede indicar que aleatoriedad tiene más peso del que debería tener.

GRASP	AG	diff	Rank
3.971134+06	4.473354+06	502220	6
1.92157+05	1.94520+05	2363	1
2.2668727+07	2.9312464+07	6643737	7
8.30385+05	8.51987+05	21602	3
1.35997151+08	1.54982818+08	18985667	8
1.845181+06	1.881750+06	35569	4
2.20375820+08	2.54484262+08	34108442	9
7.69172+05	7.66769+05	-2403	2
3.83819392+08	4.23455419+08	39636027	10
5.5893422+06	5.6294542+06	401120	5

Cuadro 6: Lista de valores utilizados para el test de Wilcoxon. Las primeras dos columnas listan los datos medios de los resultados. La tercera columna muestra las diferencias entre ellos, y la tercera el ranking de menor a mayor valor absoluto de las diferencias.

A continuación con los datos de la tabla 6 realizamos el test de Wilcoxon, para ver si hay diferencias significativas entre ambos algoritmos. Estableciendo H_0 de que los dos algoritmos son iguales, para ello establecemos un error 5 %. Dado que las instancias son menores que 25:

$$\sum R_+ = 53 \; ; \sum R_- = 2$$

$$T = min(\sum R_+, \sum R_-) = 2$$

$$T_c = 8$$

$$T_c > T$$

Dado que el T crítico es mayor que el T calculado, se puede descartar la hipótesis nula por lo que se confirma que hay diferencias significativas entre los dos algoritmos.

4.2. Bipartición del grafo

Para la experimentación de la problemática de la bipartición del grafo se han utilizando 10 sets de datos diferentes. Las dimensiones de los sets de datos oscilan entre 10 y 50 nodos. Se han utilizado 4 instancias de 10 nodos, otras 4 instancias de 20 nodos y 2 instancias compuestas por 50 nodos.

Por otro lado, se ha procedido a ejecutar los algoritmos con parámetros exigentes en tiempo de computo para poder calcular los mejores resultados a través del los algoritmos desarrollados. Los mejores resultados se han utilizado posteriormente como referencia de calculo de la desviación.

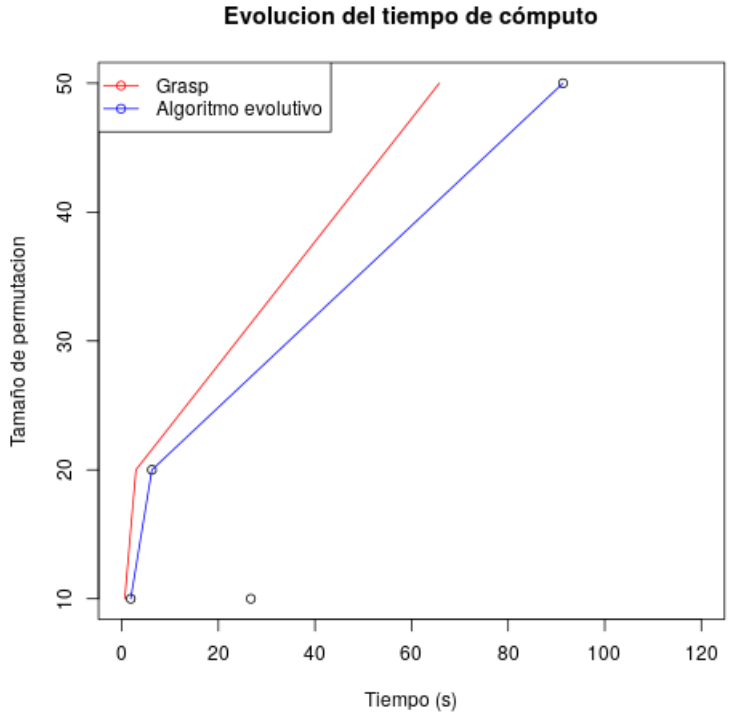

Figura 10: Tiempo de cómputo de los algoritmos desarrollados sobre la problemática de la bipartición del grafo.

En la siguiente tabla se muestra los datos relacionados con la búsqueda de diferentes sets de datos utilizando para ello el algoritmo de búsqueda local GRASP 3.

Num Edificios	Media	Varianza	Mejor Valor
10.1	821.5	26.7	822
10.2	147983	518242	1480840
10.3	152205	3357334	158853
10.4	180488	2610319	1807710
20.1	3605	1554	3732
20.2	520721	48701803	542505
20.3	611427	22883301	625461
20.4	604105	98182848	633943
50.1	22099	38520	22782
50.2	3169869	675860045	3181822

Cuadro 7: Resultados de la problemática BIP con el algoritmo Grasp.

En la siguiente tabla se muestra los datos relacionados con la búsqueda de diferentes sets de datos utilizando para ello el algoritmo genético 4.

Num Edificios	Media	Varianza	Mejor Valor
10.1	822	0	822
10.2	148084	0	148084
10.3	158853	0	158853
10.4	180771	0	180771
20.1	3736	284	3748
20.2	540700	5825245	542505
20.3	625386	13125	625461
20.4	634874	17830392	640059
50.1	22587	6392	22724
50.2	3180589	71765892	3189004

Cuadro 8: Resultados de la problemática BIP con el algoritmo genético.

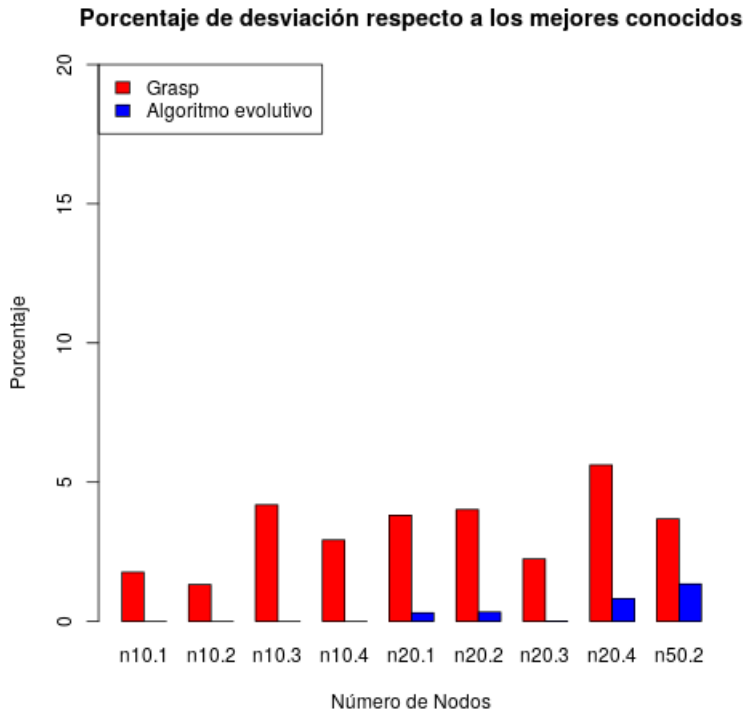

Figura 11: Desviación de los algoritmos desarrollados respecto a los mejores resultados encontrados 9 por los mismo algoritmos con parámetros más permisivos.

Num Fabricas	Mejor Valor conocido
10.1	822
10.2	148084
10.3	158853
10.4	180771
20.1	3748
20.2	542505
20.3	625461
50.1	640059
50.1	23009
50.2	3223847

Cuadro 9: Los mejores valores conocidos para cada dataset.

En las tablas se puede ver la media de los resultados, la varianza y el mejor valor entre las 10 ejecuciones. Estos datos se pueden comparar

con los mejores valores conocidos de cada uno sets de datos, listados en la tabla 9. Con estos datos, podemos comparar el desviación aproximada de cada uno de los algoritmos. En la figura 11 podemos ver, que en general la desviación es mínima, lo que sugiere que los mejores valores obtenidos por la ejecución de los algoritmos en forma permisiva, no ha sido muy eficaz, lo que podría reflejar que los algoritmos están bloqueados en óptimos locales. De la misma manera, se puede observar que la naturaleza del problema hace que el algoritmo genético sea más óptimo.

También se ha realizado un test de hipótesis, utilizando los datos de los experimentos. Los datos utilizados se listan en la tabla 10.

GRASP	AG	diff	Rank
821.5	822	0.5	2
147983	148084	101	4
152205	158853	6648	6
180488	180771	283	3
3736	3736	0	1
520721	540700	19979	9
611427	625386	13959	8
604105	634874	20769	10
22099	22587	488	5
3169869	3180589	10720	7

Cuadro 10: Lista de valores utilizados para el test de Wilcoxon. Las primeras dos columnas listan los datos medios de los resultados. La tercera columna muestra las diferencias entre ellos, y la tercera el ranking de menor a mayor valor absoluto de las diferencias.

A continuación con los datos de la tabla 10 realizamos el test de Wilcoxon, para ver si hay diferencias significativas entre ambos algoritmos. Estableciendo H_0 de que los dos algoritmos son iguales, para ello establecemos un error 5 %. Dado que las instancias son menores que 25:

$$\sum R_+ = 54 \; ; \sum R_- = 0$$

$$T = min(\sum R_+, \sum R_-) = 0$$

$$T_c = 8$$

$$T_c > T$$

Dado que el T crítico es mayor que el T calculado, se puede descartar la hipótesis nula por lo que se confirma que hay diferencias significativas entre los dos algoritmos.

5. Conclusiones

Se ha podido observar que los algoritmos se comportan de forma muy diferente según el tipo de problemática. Ejemplo claro es la relación que parece existir entre la problemática de la bipartición del grafo y el algoritmo genético; obtiene muchos mejores resultados que el algoritmo GRASP y además utiliza menos tiempo de computo. En cambio, en la problemática QAP; el algortimo GRASP es más óptimo, pero no es igual de estable. Se debería analizar los algoritmos para verificar el porque de la aleatoriedad que se produce en los resultados.

Para limitar el tiempo de procesado, se han tenido que acotar tanto la cantidad de vecinos a analizar como la población y cantidad de generaciones, empeorando los resultados de la búsqueda. Para mejorar los resultados, habría que optimizar los parámetros de los algoritmos por separado y mejorar el punto de parada del algoritmo ya que parece ser fuente de los problemas en el caso de GRASP.

Referencias

[1] A Greedy Genetic Algorithm for the Quadratic Assignment Problem `http://cdn.intechopen.com/pdfs-wm/5841.pdf`

[2] Parallel Search Strategies for TSPs using a Greedy Genetic Algorithm `http://cdn.intechopen.com/pdfs-wm/5841.pdf`

[3] Heurísticos GRASP híbridos para el problema de rutas de vehículos con restricciones de capacidad `https://juangvillegas.files.wordpress.com/2011/08/tesisvrp-jgvr-matematicasaplicadas.pdf`

[4] GRASP: Greedy Randomized Adaptive Search Procedures `https://egela1516.ehu.eus/pluginfile.php/780430/mod_resource/content/1/GRASP.pdf`