1 Quadratic Assignment Problem:

Tenemos *n* instalaciones a emplazar en *n* ubicaciones. Para cada par de ubicaciones conocemos la distancia que las separa y para cada par de instalaciones conocemos el tráfico que habrá entre ellas (transporte de mercancías, tráfico de red, etc). El objetivo es encontrar el emplazamiento óptimo que minimice la suma de las distancias de los caminos multiplicadas por el tráfico que ha de pasar.

El Quadratic Assignment Problem pertenece a la clase de problemas NP-completos y es considerado uno de los más difíciles de resolver dentro de ésta. En la página web dedicada QPALib[1] se hace mención a que problemas de tamaño moderado (n=25) ya son considerados computacionalmente difíciles incluso para los algoritmos más potentes.

También en la página web QPALib[1] se pueden encontrar instancias resueltas junto a sus costes óptimos.

1.1 Enfoque *Branch* & *Bound*

La manera tradicional de resolver este tipo de problemas ha sido mediante algoritmos de *Branch* & *Bound*. En este tipo de algoritmos el espacio de soluciones factibles se estructura en árbol que se construye dinámicamente. En todo momento nos guardamos el valor de la mejor solución encontrada hasta el momento y alternativamente se ejecutan dos pasos:

- **Branching:** En un momento dado nos encontramos en un nodo del árbol con una solución parcial factible. El proceso de *Branching* nos dirá qué soluciones factibles podemos obtener añadiendo un solo elemento a nuestra solución parcial.
- **Bounding:** Para cada una de las soluciones candidatas que han resultado del proceso de *Branching* obtenemos una cota de la mejor solución obtenible a partir de ésa, mediante el proceso de *Bounding*. Si la cota es peor que el valor de la mejor solución encontrada hasta el momento, entonces descartamos la solución parcial que estamos considerando. De entre todas las soluciones restantes escogemos una y seguimos explorando.

El algoritmo acaba cuando se ha explorado todo el árbol de soluciones (teniendo en cuenta que se espera que varias ramas hayan sido podadas durante el proceso) y se retorna la mejor solución encontrada.

Un primer truco para acelerar el proceso es empezar con una cota relativamente buena. Por ejemplo, se podría emplear un algoritmo *greedy* para encontrar una primera solución y así intentar podar gran parte del árbol desde el principio.

Sin embargo, la clave para un algoritmo de *Branch & Bound* es tener una buena función de *bounding* para descartar ramas del árbol lo antes posible. Aún así, para el Quadratic Assignment Problem las funciones de cotas que se conocen no son eficientes de computar ni podan tan bien como cabría esperar.

Otro aspecto clave a tener en cuenta es la estrategia a seguir en el momento de hacer el *branching*. Por lo general hay dos estrategias clásicas:

- **Eager:** Entendemos por *eager* aquella estrategia que pretende calcular los *boundings* para las soluciones parciales lo antes posible.

  Al seleccionar un nodo se hace el *branching* de éste y se calcula la cota para todos los nodos resultantes. Todos los nodos no explorados se almacenan en una especie de "cola de prioridad" y siempre se explora el nodo con la menor cota, en busca de soluciones buenas. La primera solución completa que se desencole será la óptima, puesto que cualquiera otra solución que nos llegue despues tendrá una cota peor, y por lo tanto se ha terminado la búsqueda.

  La forma de visitar las soluciones parciales recuerda a un recorrido en anchura.

- **Lazy:** Entendemos por *lazy* aquella estrategia que pretende calcular los *boundings* para las soluciones parciales lo más tarde posible.

  Al seleccionar un nodo se hace el *branching* de éste y se selecciona uno de los nodos sucesores. Se calcula la cota para éste último; si es mayor que la mejor solución encontrada se descarta, sino se repite el proceso hasta llegar a una solución completa. El algoritmo termina cuando no quedan partes del espacio de soluciones para ser exploradas.

  La forma de visitar las soluciones parciales recuerda a un recorrido en profundidad.

1.2 La cota de Gilmore-Lawler

Para el problema que nos concierne se explicará la Gilmore-Lawler *bound*, ya que es una de las clásicas y es bastante sencilla de entender. Sin embargo, este *bounding* es muy caro computacionalmente y se estima que cada vez que se aumenta en uno el tamaño del problema se multiplica por 4 el tiempo de resolución.

Se basa en el hecho de que se puede obtener una cota inferior para el producto escalar de dos vectores *a* y *b* multiplicando el elemento más grande de *a* por el elemento más pequeño de b, el segundo elemento más grande de *a* por el segundo elemento más pequeño de *b*, etc.

Dada una solución parcial donde *m* instalaciones ya han sido emplazadas, la cota Gilmore-Lawler se puede calcular mediante la adición de tres términos:

- El primer término se puede calcular de manera exacta y hace referencia al coste del tráfico entra las instalaciones ya colocadas ($O(m^2) \subseteq O(N^2)$).

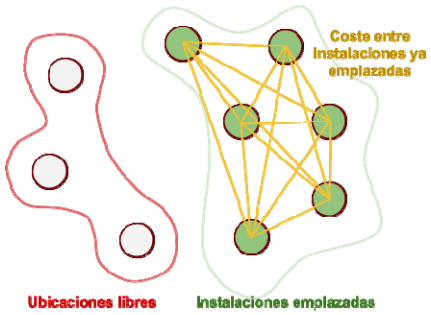


Figure 1: El primer término resulta de sumar los costes de las aristas señaladas.

- El segundo término aproxima el coste del tráfico entre las instalaciones colocadas y las que queden por colocar. A priori no se puede calcular de manera exacta en tiempo razonable.

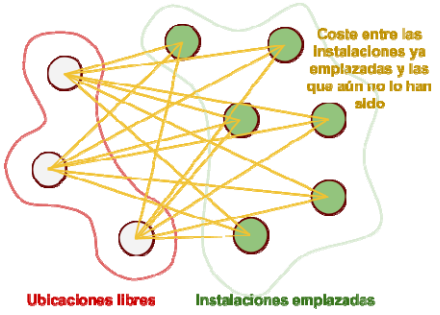


Figure 2: El segundo término intenta aproximar la suma de los costes de las aristas señaladas.

- El tercer término contempla el coste del tráfico entre las instalaciones aún no emplazadas. A priori no se puede calcular de manera exacta en tiempo razonable.

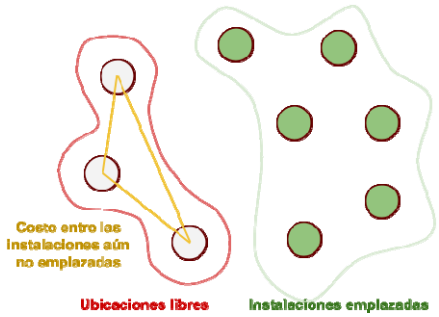


Figure 3: El tercer término intenta aproximar la suma de los costes de las aristas señaladas.

Para estimar los valores del segundo y del tercer término lo que hacemos es construir una matriz de dimensión $(N - m) \times (N - m)$ donde el elemento (i, k) intenta acotar el coste adicional que supondría colocar la iéssima instalación no emplazada en la késsima ubicación. Una vez la tenemos, intentamos encontrar la asignación óptima para las instalaciones no emplazadas y el coste de ésta se usa como estimador de la suma del segundo y tercer término.

La matriz mencionada en el apartado anterior se puede calcular como la suma de dos matrices C1 y C2.:

- En C1 el elemento (i, k) representa el coste de colocar la iéssima instalación no emplazada en la késsima ubicación, con respecto a las instalaciones ya emplazadas. Los valores de esta matriz se pueden calcular de forma exacta $(O(m(N - m)^2) \subseteq O(N^3))$.
  Los costes de esta matriz están relacionados con el segundo término.

- En C2 el elemento (i, k) representa el coste de colocar la iéssima instalación no emplazada en la késsima ubicación, con respecto a las instalaciones aún no emplazadas. Sin embargo, los elementos de esta matriz no se pueden calcular de manera exacta y se han de aproximar. Sea T el vector de tráficos desde la iéssima instalación al resto (de las no emplazadas) y sea D el vector de distancias desde la késsima ubicación al resto (de las no ocupadas); la cota inferior del coste de emplazar i en k resulta del productor escalar del T ordenado crecientemente con D ordenado decrecientemente $(O((N - m)^3) \subseteq O(N^3))$.
  Los costes de esta matriz están relacionados con el tercer término.

Lo único que queda ahora es encontrar el coste de la asignación óptima de las instalaciones no emplazadas, teniendo en cuenta la matriz de costes C1+C2. Un primer enfoque sencillo seria el de asignar, para cada instalación no emplazada, el emplazamiento de coste mínimo

(aunque de esta manera se estuviesen repitiendo ubicaciones). Al fin y al cabo, esto nos daría una cota inferior que ya nos serviría para el *Branch & Bound*.

Sin embargo, si pudiésemos hacer una asignación correcta de peso mínimo estaríamos encontrando una cota inferior pero mayor a la mencionada anteriormente. Cuanto mayor se nuestra cota inferior más nos estaremos aproximando al valor real y, por tanto, nuestro algoritmo podará más ramas y funcionará más rápido. El problema que nos concierne se conoce formalmente como *Linear Assignment* y existen varios algoritmos para resolverlo, entre ellos, el llamado *Hungarian Algorithm*[2][4][5].

En resumen, el valor del *bounding* Gilmore-Lawler para una solución parcial de un problema QAP es la suma del primer término, descrito anteriormente, y del coste de la asignación lineal óptima de C1+C2.

2 El Hungarian Algorithm

El *Hungarian Algorithm* resuelve instancias de *Linear Assignment Problems*, el cual es equivalente al problema de encontrar *Minimum Weighted Matchings* en grafos bipartidos.

El problema puede entenderse como ¿Cómo asignar n trabajadores a n tareas, sabiendo el coste que implica que el iéssimo trabajador haga la jéssima tarea, de manera que el coste total sea mínimo? o, en nuestro caso, ¿Cómo asignar n instalaciones a n ubicaciones, sabiendo el coste que implica emplazar la iéssima instalación en la jéssima ubicación, de manera que el coste total sea mínimo?

Todo el proceso se sustenta sobre el siguiente teorema: Si un número se añade o substrae de todos los valores de un columna o fila en una matriz de costes, entonces la asignación óptima para la matriz resultante es también óptima para la matriz original.

Este algoritmo se compone de una fase de preproceso y de una fase iterativa. La primera consiste en:

- Se organizan los costes en una matriz, donde las filas corresponden a las instalaciones y las columnas a las ubicaciones (Figura 4a).
- Si la matriz no es cuadrada, se añaden filas y columnas auxiliares con costes igual al mayor coste en la matriz.
- Se substrae de cada fila el mínimo valor de esa fila (Figura 4b).
- Se substrae de cada columna el mínimo valor de esa columna (Figura 4b).

En este punto, si cada fila y columna de la matriz contiene un cero entonces hemos acabado. Si no, se ejecuta la siguiente fase iterativa hasta que se cumpla esta condición:

- Recubre cada cero en la matriz usando el mínimo número de líneas posible (Figura 4c). Si se tienen que usar n líneas en una matriz de $n \times n$ entonces nos encontramos en el caso en que hay un cero para cada columna y fila y por tanto hemos acabado (Figura 4e).
- Añade a cada número cubierto el mínimo número no cubierto. Si un número esta cubierto por dos líneas, añade el mínimo dos veces (Figura 4d).
- Substrae el mínimo elemento de la matriz de cada elemento de la matriz. Vuelve al primer paso (Figura 4d).

Al acabar sabemos seguro que hay un 0 para cada fila y columna de la matriz. El conjunto de n zeros que nos de una asignación factible nos da la asignación óptima que estábamos buscando (Figura 4f). Se pueden consultar guías visuales para este algoritmo en los links [4] y [2].

21	34	31	43
20	35	32	44
20	34	33	45
21	34	31	43

(a) Matriz de costes original.

0	0	0	0
0	2	2	2
0	1	3	3
0	0	0	0

(b) Matriz de costes después de substraer el mínimo por fila y columna.

0	**0**	**0**	**0**	x
0	2	2	2	
0	1	3	3	
0	**0**	**0**	**0**	x
x				

(c) Se necessitan mínimo 3 líneas para recubrir todos los ceros.

1	0	0	0
0	1	1	1
0	0	2	2
1	0	0	0

(d) Matriz de costes después de añadir a cada número cubierto el mínimo número no cubierto y de substraer el mínimo elemento de la matriz de cada elemento de la matriz.

1	**0**	**0**	**0**	x
0	**1**	**1**	**1**	x
0	**0**	**2**	**2**	x
1	**0**	**0**	**0**	x

(e) Se necessitan mínimo 4 líneas para recubrir todos los ceros. Como N = 4 hemos acabado.

1	0	0	**0**
0	1	1	1
0	**0**	2	2
1	0	**0**	0

(f) Hemos encontrado una asignación óptima. El coste de ésta es 20 + 31 + 34 + 43 = 128.

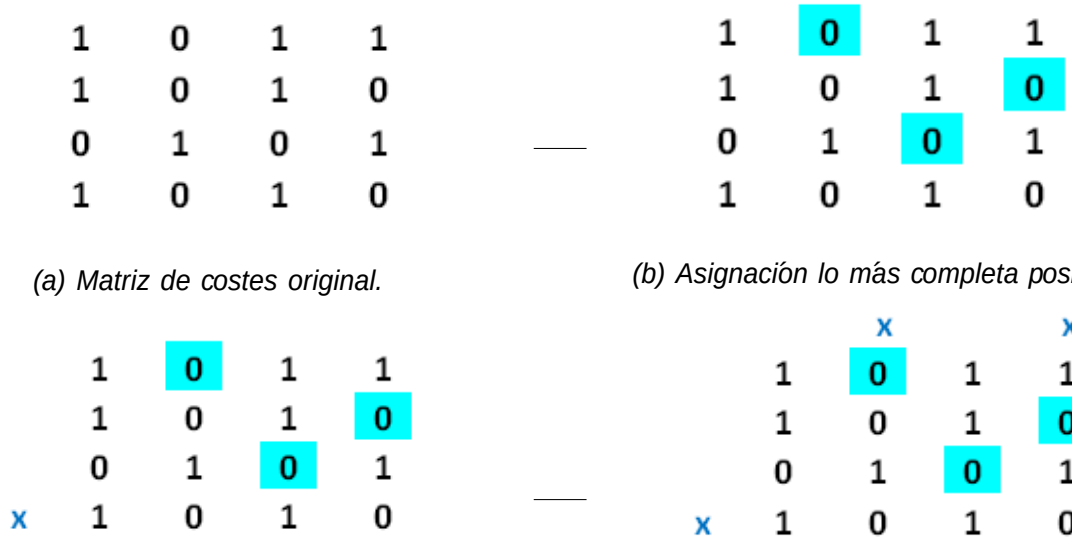
Figure 4: Ejemplo de la aplicación del Hungarian Algorithm.

2.1 Calcular líneas mínimas

Un método para calcular las líneas mínimas necesarias para recubrir todos los zeros es el siguiente [3].

- Primero se debe obtener una asignación lo más completa posible, es decir con el mayor número posible de filas con un zero asignado. Este paso se puede realizar fácilmente con un algorismo backtracking.
- Una vez tenemos hecha esta asignación, se marcan todas las filas que no tengan asignación.
- Despues se marcan todas las columnas que tengan algun zero en una fila marcada. Despues se marcan todas las filas que tengan su asignación en una columna marcada. Se repiten estos dos pasos hasta que se cierre el bucle.

- Finalmente las líneas mínimas son las que resultan de recubrir las columnas marcadas y las filas no marcadas.



(a) Matriz de costes original.

(b) Asignación lo más completa posible.

(c) Marcamos las filas sin asignación (la quarta fila).

(d) Marcamos las columnas que tienen algun zero en alguna fila marcada (la segunda y la quarta columna).

(e) Marcamos las filas que tienen su zero asignado en alguna columna marcada (la primera y la segunda fila).

(f) Recubrimos las columnas marcadas y las filas no marcadas (segunda y quarta columna, y tercera fila).

Figure 5: Ejemplo de cómo encontrar las líneas mínimas.

3 References

[1] Rainer E. Burkard, Stefan E. Karisch, and Franz Rendl. Qaplib; a quadratic assignment problem library, 1997. URL https://coral.ise.lehigh.edu/data-sets/qaplib/.

[2] Hungarian algorithm.com - solve the assignment problem, 2013-2023. URL http://www.hungarianalgorithm.com/.

[3] Hungarian algorithm, 2023. URL http://en.wikipedia.org/wiki/Hungarian_algorithm

[4] Wikihow - how to do anything. URL http://www.wikihow.com/Use-the-Hungarian-Algorithm.

[5] Bipartite Matching & the Hungarian Method, 2006. URL https://cse.hkust.edu.hk/~golin/COMP572/Notes/Matching.pdf.