Proyecto de la Asignatura Análisis de Algoritmos

María Ximena Acuña Mosquera, Andrés Felipe Tocaruncho Vásquez

Abstract—Se presenta la descripción detallada de tres problemas planteados para el desarrollo del proyecto de la asignatura, de los siguientes temas: Programación lineal, árboles de expansión mínima y cliques en grafos. Adicional a ésto, se realizó una investigación de ejemplos para cada tipo de problema con su respectiva solución.

Index Terms—Programación lineal, logística, árbol de expansión mínima, distancia de Hamming, grafo de Hamming, clique máximo.

1 Programación Lineal (LP)

1.1 Enunciado

Suponga que se planea contruir una nueva cadena de tiendas en una ciudad dada, usted tiene identificadas una serie de ubicaciones potenciales en diferentes barrios. Además asuma que la demanda de productos en cada barrio de la ciudad es conocida. Si usted quiere construir exactamente k tiendas, ¿dónde debería localizarlas de forma que minimice la distancia promedio de los clientes?. Si en lugar usted dese construir una cantidad variable de tiendas, y el costo de construir una tienda en cada sitio es conocido, ¿dónde debería construir las tiendas de forma que minimice el costo total del construcción y la distancia promedio de los clientes?

1.2 Descripción detallada

Éste es un problema de localización en redes, entendiendo como red a un conjunto de nodos o vértices (barrios) unidos por un conjunto de aristas (caminos) que representan conexiones entre esos nodos. Consiste en determinar las ubicaciones más apropiadas para establecer las tiendas en un conjunto específico de barrios en una ciudad, de forma que se minimice la distancia promedio hacia los clientes desde las tiendas que se abran. Se debe asegurar que cada barrio se supla por al menos una tienda. Si una tienda se abre en un barrio, ésta podrá suplir el barrio en el que se encuentra y los barrios con los que se encuentra comunicada.

Algunos ejemplos de la vida cotidiana con los que se podría comparar éste problema son: la ubicación de farmacias para atender hospitales, centros de distribución para atender clientes, estaciones de gasolina en determinados barrios o pueblos, centros de conmutación en una red de comunicaciones, servicios informáticos en una red de ordenadores para minimizar almacenamiento, redes de tratamiento de aguas en una ciudad o pueblo, unidades de emergencia en áreas rurales para minimizar tiempo de intervención en otros poblados, entre otros.

1.2.1 Variante 1

El número de tiendas a construir debe ser exactamente k.

Éste es un problema comúnmente conocido en logística como p-medians, en el que se busca encontrar la ubicación de p instalaciones para servir a un conjunto de n nodos de demanda, minimizando el costo asociado a la distancia recorrida para satisfacer la demanda de un conjunto de clientes. No hay limitación de capacidad en ninguna de las instalaciones. El número de posibles soluciones es $\frac{n!}{p!(n-p)!}$.

1.2.2 Variante 2

Se puede construir una cantidad variable de tiendas y cada tienda tiene un costo fijo por abrirse. Adicional a la función objetivo inicial, debe minimizarse el costo total por construcción.

Én este caso no se tiene una restricción del número de tiendas a abrir, por lo que se debe iterar hasta satisfacer la demanda de todos los clientes. No sólo se tienen en cuenta las distancias (o los costos variables a los que están asociadas), sino también el costo que implica ubicar una instalación.

1.3 Modelo Matemático

Se planteó un modelo matemático para describir el problema:

CONJUNTOS

I: Barrios

PARÁMETROS

D_i: Demanda del barrio i $\quad \forall i \in I$
A_{ij}: Distancia del barrio i al barrio j $\quad \forall i \in I \ \ \forall j \in I$
$B_{ij} = \begin{cases} 1 \text{ Si existe un arco del barrio i al barrio j} & \forall i \in I \ \forall j \in I \\ 0 \text{ De lo contrario} \end{cases}$
k: Número de tiendas a abrir
F_i: Costo fijo de abrir una tienda en el barrio i $\quad \forall i \in I$

VARIABLES DE DECISIÓN

$$X_i = \begin{cases} 1 \text{ Si se abre una tienda en el barrio i} & \forall i \in I \\ 0 \text{ De lo contrario} \end{cases}$$

$$Y_{ij} = \begin{cases} 1 \text{ Si la tienda en el barrio i atiende al barrio j} \\ \quad \forall i \in I \ \forall j \in I \\ 0 \text{ De lo contrario} \end{cases}$$

Z_i: Distancia promedio hacia los clientes posibles de la tienda i $\quad \forall i \in I$

FUNCIÓN OBJETIVO

Variante 1: Minimizar distancia promedio

$$\min Z = \sum_{i \in I} Z_i$$

Variante 2: Minimizar distancia promedio y costos fijos

$$\min Z = \sum_{i \in I} Z_i + \sum_{i \in I} X_i F_i$$

sujeto a:

1. $$Z_i = \frac{\sum_{j \in I} X_i A_{ij} B_{ij}}{\sum_{j \in I} B_{ij}} \quad \forall i \in I$$

La distancia promedio corresponde a la distancia (A_{ij}) que hay entre las tiendas que se abren (X_i) y sus potenciales clientes (B_{ij}) dividido en el número de clientes potenciales de cada una.

2. $$\sum_{i \in I} Y_{ij} = 1 \quad \forall j \in I$$

Cada barrio j sólo puede ser atendido por una tienda i.

3. $$X_i \leq \sum_{j \in I} Y_{ij} \quad \forall i \in I$$

Un barrio sólo puede atender a otros si se abre una tienda en él.

4. $$\sum_{j \in I} Y_{ij} \leq X_i \mu \quad \forall i \in I$$

Donde μ corresponde a un número lo suficientemente grande.
Una tienda debe abrirse si está atendiendo a algún otro barrio.

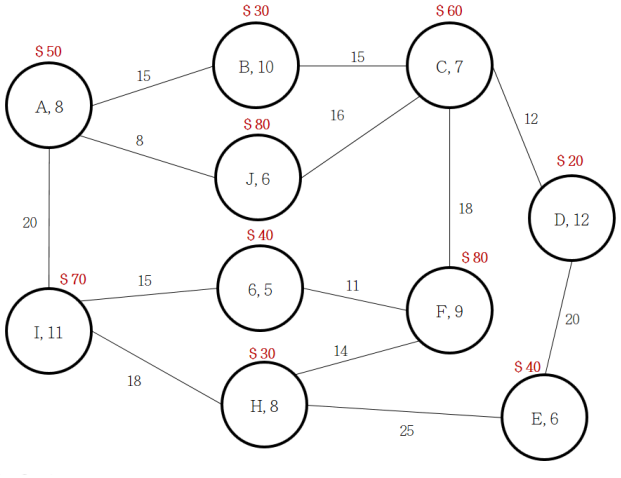
4. $$\sum_{i \in I} X_i = k$$

Variante 1: Se deben abrir exactamente k tiendas.

1.4 Ejemplo

Se tiene un conjunto de ciudades con una demanda establecida y se quieren ubicar centros de distribución que puedan suplir esas demandas al costo mínimo. Se tienen costos fijos por construir el centro de distribución y costos variables por kilómetro recorrido de cada centro de distribución a las ciudades que atienda. El problema puede tener variantes como costos de producción, o penalizaciones por no abrir un centro de distribución en cada ciudad.

Fig. 1. Representación del grafo de ciudades.



La Figura 1 representa la localización de las ciudades como un grafo, donde los nodos corresponden a las ciudades y dentro de ellos se observan las demandas, sobre cada nodo se encuentra el costo fijo de abrir un centro de distribución en esa ciudad y las aristas corresponden a la distancia de una ciudad a otra. El costo variable es de $0,15 \frac{\$}{km}$.

1.4.1 Algoritmo ADD

Para resolverlo se utilizará el algoritmo ADD, que es un método heurístico que busca las mejores ciudades para abrir los centros de distribución de la siguiente forma:

1. Se debe hallar la matriz de distancias mínimas entre ciudades, para ésto se pueden utilizar algoritmos como Dijkstra o Floyd Warshall.

2. En la primera iteración se calculan los costos tanto fijos como variables si se abre sólo un centro. En éste caso se calcularían los costos para abrir A, B, C, ... J. Se abre el centro de distribución en la que tenga menor costo total.

3. Para las siguientes iteraciones, se combinan los centros de distribución abiertos con las demás ciudades y se vuelven a calcular costos, en el caso de los costos variables se parte del hecho de que cada ciudad es atendida desde el centro de distribución al que represente menor costo de transporte (menor distancia). Por ejemplo, si en la primera iteración se decidió abrir A, en la segunda se calculan costos si se abrieran centros de distribución en A-B, A-C, A-D ... A-J. De nuevo, se escoge la combinación que tenga menor costo total.

4. El algoritmo se detiene en la iteración cuyo costo total aumente respecto al anterior y se toma la solución de la iteración anterior.

La solución para el grafo de la Figura 1 sería la siguiente:

1. Matriz de distancias mínimas

Fig. 2. Matriz de distancias mínimas.

Min Dij	A	B	C	D	E	F	G	H	I	J
A	0	15	24	36	56	42	35	38	20	8
B	15	0	15	27	47	33	44	47	35	23
C	24	15	0	12	32	18	29	32	44	16
D	36	27	12	0	20	30	41	44	56	28
E	56	47	32	20	0	39	50	25	43	48
F	42	33	18	30	39	0	11	14	26	34
G	35	44	29	41	50	11	0	25	15	43
H	38	47	32	44	25	14	25	0	18	46
I	20	35	44	56	43	26	15	18	0	28
J	8	23	16	28	48	34	43	46	28	0

2. Primera iteración

Fig. 3. Iteración 1.

Ubicacion	Costo Variable ($)	Costo Fijo ($)	Costo Total ($)
A	331,65	50	381,65
B	337,05	30	367,05
C	273,15	60	333,15
D	355,95	20	375,95
E	441,6	40	481,6
F	306,45	80	386,45
G	365,55	40	405,55
H	360,15	30	390,15
I	355,35	70	425,35
J	334,05	80	414,05

Se abre un centro de distribución en C.

3. Segunda iteración

Fig. 4. Iteración 2.

Ubicacion	Costo Variable ($)	Costo Fijo ($)	Costo Total ($)
C-A	197,55	110	307,55
C-B	225	90	315
C-D	240,67	80	320,67
C-E	234,3	100	334,3
C-F	184,05	140	324,05
C-G	185,7	100	285,7
C-H	177,15	90	267,15
C-I	168,45	130	298,45
C-J	213,15	140	353,15

Se abre un centro de distribución en H.

4. Tercera iteración

Fig. 5. Iteración 3.

Ubicacion	Costo Variable ($)	Costo Fijo ($)	Costo Total ($)
C-H-A	141,15	140	281,15
C-H-B	143,85	120	263,85
C-H-D	151,05	110	261,05
C-H-E	154,65	130	284,65
C-H-F	147,75	170	317,75
C-H-G	149,4	130	279,4
C-H-I	135,15	160	295,15
C-H-J	143,55	170	313,55

Se abre un centro de distribución en D.

5. Cuarta iteración

Fig. 6. Iteración 4.

Ubicacion	Costo Variable ($)	Costo Fijo ($)	Costo Total ($)
C-H-D-A	115,05	160	275,05
C-H-D-B	117,75	140	257,75
C-H-D-E	133,05	150	283,05
C-H-D-F	121,65	190	311,65
C-H-D-G	123,3	150	273,3
C-H-D-I	109,05	180	289,05
C-H-D-J	117,45	190	307,45

Se abre un centro de distribución en B.

6. Quinta iteración

Fig. 7. Iteración 5.

Ubicacion	Costo Variable ($)	Costo Fijo ($)	Costo Total ($)
C-H-D-B-A	92,55	190	282,55
C-H-D-B-E	99,75	180	279,75
C-H-D-B-F	88,35	220	308,35
C-H-D-B-G	90	180	270
C-H-D-B-I	80,55	210	290,55
C-H-D-B-J	94,95	220	314,95

Debido a que el costo total aumenta de 257.75 a 270.00, se deben abrir centros de distribución en C, H, D y B.

1.5 Algoritmos propuestos

1.5.1 Variante 1

Descripción

Se diseñó un algoritmo que lee la información del archivo $input\_n\_k$, donde el número que sigue a n es la cantidad de barrios y el número que sigue a k es la cantidad de barrios a abrir. De este archivo se lee el valor de k, el nombre de los barrios y la distancia desde cada uno de los barrios hacia todos los demás. Teniendo en cuenta el k, realiza todas las posibles combinaciones de barrios a abrir utilizando el método $combinations$ de la librería $itertools$. Posteriormente, para cada una de las combinaciones de barrios a abrir, calcula la distancia recorrida para atender a todos los demás barrios. Se escoge la combinación con menor distancia y se imprime tanto por pantalla como en un archivo llamado $output\_n\_k$ donde n y k corresponden a los mismos del archivo de entrada.

Análisis del tiempo de ejecución

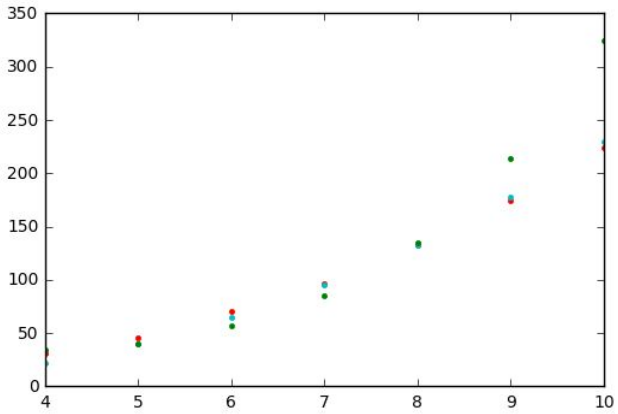
Se utilizó un contador de pasos para determinar el número de pasos que realizaba el algoritmo para solucionar el problema, el cual se probó con una entrada de 5 barrios. Posteriormente, se realizó un análisis experimental comparando el número de pasos, el tiempo de ejecución y el tiempo teórico del algoritmo (que se tomó como $n^2\ log_n$), ésto con entradas desde 4 hasta 10 barrios, para todos con k=3.

Es importante mencionar que el método

$itertools.combinations$ tiene un tiempo de ejecución $O(n)$.

Se obtuvieron los siguientes resultados:

Fig. 8. Análisis del tiempo de ejecución - Algoritmo 1



Donde los puntos rojos corresponden al número de pasos, los verdes al tiempo de ejecución y los azules al tiempo teórico.

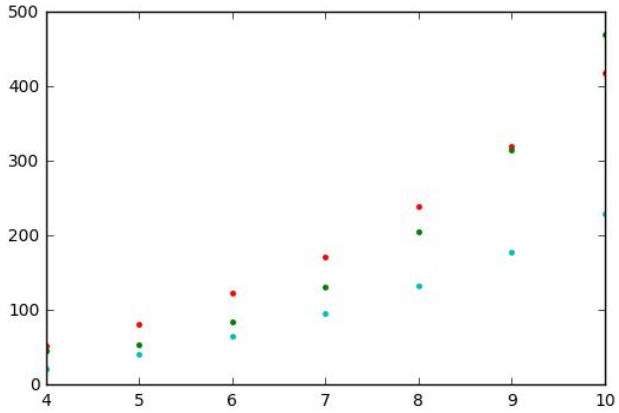
Podemos observar que el algoritmo es $O(n^2\ log_n)$, ya que con los casos de prueba, el número de pasos fue muy cercano a éste valor al variar el tamaño de entrada.

1.5.2 Variante 2

Descripción
Se diseñó un algoritmo que lee la información del archivo $input\_n\_k\_c$, donde el número que sigue a n es la cantidad de barrios y el número que sigue a k es la cantidad máxima de barrios a abrir. De este archivo se lee el valor de k, el nombre de los barrios, el costo fijo de abrir una tienda y la distancia desde cada uno de los barrios hacia todos los demás. Teniendo en cuenta el k, realiza todas las posibles combinaciones (de tamaño 1 hasta k) de barrios a abrir utilizando el método $combinations$ de la librería $itertools$. Posteriormente, para cada una de las combinaciones de barrios a abrir, calcula el costo de abrir una tienda en cada uno de esos barrios más la distancia recorrida para atender a todos los demás barrios. Se escoge la combinación con menor distancia y se imprime tanto por pantalla como en un archivo llamado $output\_n\_k$ donde n y k corresponden a los mismos del archivo de entrada.

Análisis del tiempo de ejecución
Se utilizó un contador de pasos para determinar el número de pasos que realizaba el algoritmo para solucionar el problema, el cual se probó con una entrada de 5 barrios. Posteriormente, se realizó un análisis experimental comparando el número de pasos, el tiempo de ejecución y el tiempo teórico del algoritmo (que se tomó como $n^2\ log_n$), ésto con entradas desde 4 hasta 10 barrios, para todos con k=3. Es importante mencionar que el método $itertools.combinations$ tiene un tiempo de ejecucion $O(n)$.

Fig. 9. Análisis del tiempo de ejecución - Algoritmo 2



Se obtuvieron los siguientes resultados:
Donde los puntos rojos corresponden al número de pasos, los verdes al tiempo de ejecución y los azules al tiempo teórico.

Podemos observar que el algoritmo es $o(n^2\ log_n)$, ya que con los casos de prueba, el número de pasos fue significativamente mayor a éste valor al variar el tamaño de entrada; sin embargo, al compararlo con n^3, el número de pasos era significativamente menor.

2 ÁRBOL DE EXPANSIÓN MÍNIMA (MST)

2.1 Enunciado

Se tiene un grafo G=(V,E) no dirigido, el cual tiene n vértices y m aristas. Cada una de las aristas puede ser roja o azul, sin importar el peso o algún otro parámetro. Se espera encontrar el árbol de expansión mínima (MST) para el grafo G dado.

2.2 Descripción detallada

La idea principal de este problema es encontrar el árbol de expansión mínima (MST) para un grafo G determinado, el cual puede tener el tamaño que desee. El árbol de expansión mínima (MST) es un subgrafo que contiene todos los nodos o vértices y algunas aristas del grafo G. Para este ejercicio se proporcionará una variable adicional k, que indica que el árbol de expansión mínima que se desea encontrar debe contener exactamente, k aristas azules y $n - k - 1$ aristas rojas.

2.3 Ejemplos

Para encontrar el árbol de expansión mínima (MST) se puede recurrir a la implementación de ciertos algoritmos. Como ejemplos se tienen el Algoritmo de Kruskal y el Algoritmo de Prim:

2.3.1 Algoritmo de Kruskal

El Algoritmo de Kruskal permite hallar el árbol de expansión mínima (MST) por medio de los siguientes pasos:

1. Se debe marcar la arista con menor peso. En caso que haya más de una, se escoge cualquiera de ellas.

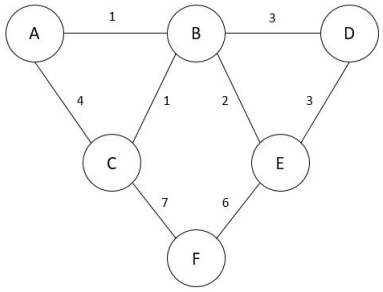
2. Se debe marcar la arista con menor peso que no se haya marcado aún. En caso que haya más de una, se escoge cualquiera de ellas.

3. Se deber macar la arista con menor peso que no se haya marcado aún y que no forme un ciclo con otras.

4. Repetir el paso 3 hasta que se hayan seleccionado $n-1$ aristas, siendo n el número de nodos que tiene el grafo.

Como ejemplo tomaremos el siguiente grafo:

Fig. 10. Grafo.



La solución para dicho grafo sería:

1. Buscamos la arista con menor valor, las cuales son las aristas A-B y B-C. En este caso escogeremos A-B.

2. Escogemos la siguiente arista con menor valor, en este caso B-C.

3. Escogemos la siguiente arista con menor valor, en este caso B-D.

4. La siguiente arista con menor valor sería B-C.

5. Por último, escogemos la arista E-F.

De esta forma el grafo obtenido es el siguiente:

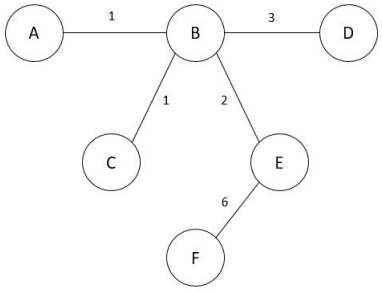
2.3.2 Algoritmo de Prim

El Algoritmo de Prim permite hallar el árbol de expansión mínima (MST) por medio de los siguientes pasos:

1. Elegir un nodo cualquiera como punto de partida.

2. Seleccionamos la arista de menor valor que este unida al nodo seleccionado anteriormente.

Fig. 11. Árbol de expansión mínima.

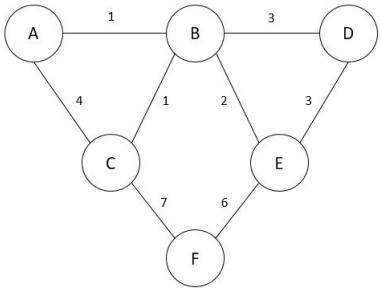


3. Repetir el paso 2 mientras la arista seleccionada se comunique con un nodo no visitado anteriormente.

4. Seleccionar aristas hasta haber marcado todos los nodos.

Como ejemplo tomaremos el grafo usado en el ejemplo anterior:

Fig. 12. Grafo.



La solución para dicho grafo sería:

1. Escogemos un nodo cualquiera de inicio, en este caso el nodo A.

2. Marcamos la arista A-B y dado que estamos en B, marcamos la arista de menor valor, en este caso B-C.

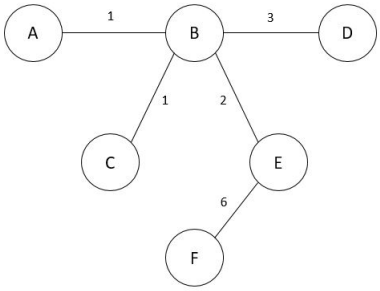
3. Dado que estamos en C, la arista que comunica C-E es mayor a B-D, por lo tanto seleccionamos B-D.

4. Estando en D, observamos que D-E es mayor que B-E, por lo tanto marcamos B-E.

5. Por último, escogemos la arista E-F, ya que F es el ultimo nodo que nos falta por visitar y E-F es menor que C-F.

De esta forma el grafo obtenido es el siguiente:

Fig. 13. Árbol de expansión mínima.



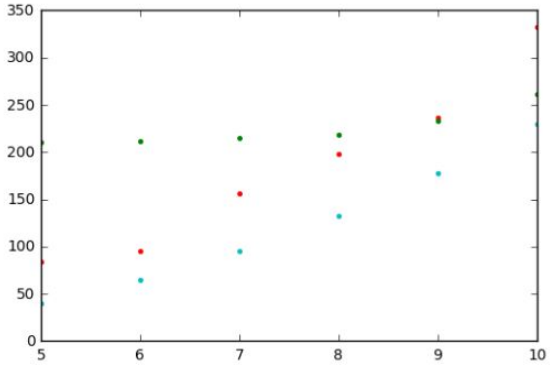
2.4 Algoritmo propuesto

Descripción
Se diseñó un algoritmo que lee la información del archivo $input\_n\_b\_r$, donde el número que sigue a n es la cantidad de nodos, el número que sigue a b es la cantidad de aristas azules y el número que sigue a r es la cantidad de aristas rojas. De este archivo se lee el valor de n, de b y las aristas que conectan cada nodo con su respectivo peso y color. Se calcula el valor de r como $n-b-1$. Se ordenan las aristas de menor a mayor peso y se itera sobre el grafo seleccionando las aristas de menor peso que cumplan con las siguientes condiciones: no se debe superar el parámentro de número de aristas rojas o azules y cada uno de los nodos debe ser visitado mínimo una vez. Se escoge el conjunto de aristas que forman el árbol de expansión mínima y se imprimen las aristas tanto por pantalla como en un archivo llamado $output\_n\_b\_r$ donde n, b y r corresponden a los mismos del archivo de entrada.

Análisis del tiempo de ejecución
Se utilizó un contador de pasos para determinar el número de pasos que realizaba el algoritmo para solucionar el problema, el cual se probó con una entrada de 8 nodos. Posteriormente, se realizó un análisis experimental comparando el número de pasos, el tiempo de ejecución y el tiempo teórico del algoritmo (que se tomó como $n^2\ log_n$), ésto con entradas desde 5 hasta 10 nodos, para todos con b=3.

Se obtuvieron los siguientes resultados:

Fig. 14. Análisis del tiempo de ejecución - Algoritmo 3



Donde los puntos rojos corresponden al número de pasos, los verdes al tiempo de ejecución y los azules al tiempo teórico.

Podemos observar que el algoritmo es $o(n^2\ log_n)$, ya que con los casos de prueba, el número de pasos fue un poco mayor a éste valor al variar el tamaño de entrada. Así mismo, podemos observar cómo el tiempo es constante para los tamaños de entrada pequeõs, y sólo aumenta un poco a medida que el tama no aumenta.

3 Clique máximo en el Grafo de Hamming

3.1 Enunciado

La distancia de Hamming $dist(u, v)$ entre dos vectores binarios $v = (v_1, ..., v_n)$ y $w = (w_1, ..., w_n)$ es el número de índices k tal que $v_k \neq w_k$. Una pregunta fundamental en la teoría de la codificación es determinar el número

$$A(n, d) = max|S \subset \{0, 1\}\ n \mid \text{dist(u, v)} \geq d \text{ for all distinct } u, v \in S|,$$

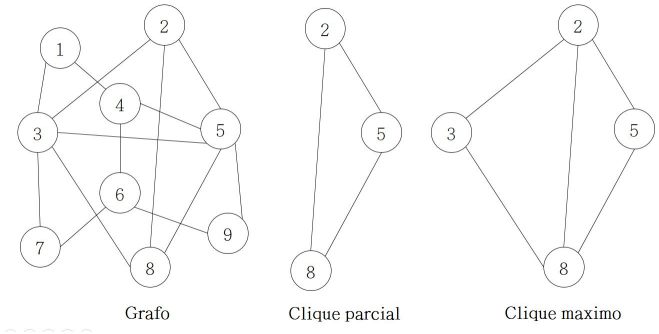
el máximo número de vectores binarios de longitud n que uno puede encontrar tal que dos vectores diferentes tienen una distancia de Hamming $\geq d$. Por ejemplo, $A(5, 4)$ = 2. El grafo de Hamming $H(n, d) = (V, E)$ es el gráfo con 2^n vertices V dados por cadenas binarias de longitud n. Nosotros tenemos $(u, v) \in$ E si y solo si $dist(u, v) \geq d$. El número $A(n, d)$ coincide con el tamaño de un clique máximo en $H(n, d)$. Encuentre un algoritmo "eficiente" para calcular el clique máximo en el grafo de Hamming (calcular el clique máximo es un problema NP-difícil).

3.2 Descripción

El grafo de Hamming $H(n, d) = (V, E)$ es un grafo con 2^n vértices que corresponden a vectores binarios de tamaño n. Las aristas están dadas por cada par de vértices en que la distancia de Hamming es mayor o igual a d. El problema consiste en hallar el clique máximo del grafo de Hamming. Un Clique es un conjunto C de vértices donde todo par de vértices de C está conectado por una arista, es decir, C es un subgrafo completo. El clique máximo es el

clique con mayor cardinalidad posible; el tamaño del clique máximo en $H(n,d)$ corresponde a $A(n,d)$, definido como el máximo número de vectores binarios de tamaño n que se pueden encontrar tal que dos vectores diferentes tienen una distancia de Hamming mayor o igual a d.

Fig. 15. Ejemplo de un clique máximo.



3.3 Ejemplo

Los algoritmos de Optimización de Colonia de Hormigas (OCH) han sido utilizados para resolver diferentes problemas de optimización combinatoria clasificados dentro de los NP-complejos. La idea principal del OCH es modelas los problemas para buscar el camino de costo mínimo en un grafo. Las hormigas recorren el grafo en busca de buenos caminos (soluciones). Cada hormiga es un agente que tiene un comportamiento simple de modo que no siempre encuentra caminos de calidad cuando está sola. Las hormigas encuentran mejores caminos como resultado de la cooperación global entre la colonia. Ésta cooperación se realiza de una manera indirecta al depositar la feromona, una sustancia que es depositada por una hormiga en su recorrido.

El algoritmo de colonia de hormigas general para el problema del clique máximo fue propuesto por Fenet y Solon:

Inicializar los rastros de feromonas
Repetir
Para *k en 1 ... nbAnts*
Actualizar los rastros de feromonas $\{C_1, ..., C_{nbAnts}\}$ (1)
Hasta *Número de ciclos o encontrar la solución óptima*

Inicialización de la feromona: Las hormigas se comunican a través de la feromona, la cual es depositada en las aristas del grafo. La concentración de feromona en la arista (v_i, v_j) es denotada por $\tau(v_i, v_j)$. El rastro de feromona inicial es denotada por c, aunque se pueden utilizar límites τ_{min} o τ_{max}, los cuales están basados en el algoritmo MAX-MIN, con el propósito de tener una mayor exploración del espacio de búsqueda.

Construcción de los cliques con las hormigas: Se selecciona aleatoriamente un vértice inicial, e iterativamente se escogen vértices para agregar al clique, dentro de un conjunto de candidatos que contiene todos los vértices que estén conectados con todos los vértices del clique. A continuación se presenta el pseudocódigo:

Escoger aleatoriamente el primer vértice $v_f \in V$
C $\leftarrow \{v_f\}$
Candidatos $\leftarrow \{v_i/(v_i, v_f) \in E\}$
Mientras *Candidato* $\neq 0$ **Hacer**
Escoger un vértice $v_i \in$ *Candidatos con una propabilidad* $p(v_i)$ (2)
$C \leftarrow C \cup \{v_i\}$
Candidatos $\leftarrow$ *Candidatos* $\cap \{v_i/(v_i, v_f) \in E\}$
Fin Mientras
Retorna C

La actualización de rastro de feromona se realiza de la siguiente forma:

$$\tau_{ij}(t+n) = \rho\tau_{ij}(t) + \delta\tau_{ij} \quad (1)$$

La probabilidad de los candidatos se halla como

$$p(v_i) = \frac{[\tau c(v_i]^\alpha}{\sum v_j \, [\tau c(v_j]^\alpha} \quad (2)$$

REFERENCES

[1] 2.3 Problema Árbol Expandido Mínimo, Apr. 2010.
[2] JAQUE, A. Análisis de Algoritmos - Proyecto.
[3] JIMENEZ, J. A. L. 1 C. ÁRBOL DE EXPANSIÓN MÍNIMA.
[4] MARCOS COLEBROOK, J. S. Localización de servicios en redes.
[5] PABÓN, J. D. M. Localización en Redes.
[6] PONCE, J. C., DE LEÓN, E. E. P., PADILLA, A., PADILLA, F., AND OCHOA, A. Algoritmo de Colonia de Hormigas para el Problema del Clique Máximo con un Optimizador Local K-opt.