**Subgrafo K-Conexo Minimo**

Andrés Felipe Movilla Obregón

Fundación Universidad del Norte

[movillaf@uninorte.edu.co](mailto:movillaf@uninorte.edu.co)

Octubre 17, 2017

**Terminologia:**

1. **Vertice:** Representacion de una informacion.
2. **Arista:** Representacion de conexion entre dos vertices.
3. **Grafo:** Conjunto de vertices unidos por aristas.
4. **Camino:** Conjunto de aristas con comienzo en un vertice y fin en otro vertice.
5. **Grafo conexo:** Grafo que para cualquier dos vertices existe un camino que los conecte.
6. **Conexidad:** Propiedad de un grafo que implica la cantidad de vértices o aristas que al ser removidos el grafo deja de ser un grafo conexo.
7. **Grafo k-conexo:** Grafo conexo que tras remover ‘k’ vértices, se desconecta.

**Resumen:**

Dado un entero positivo k y un grafo G k-conexo, y un subgrafo de expansión k-conexo de G con un número mínimo de bordes. Este problema es conocido por ser NP-complete. Khuller y Raghavachari presentaron el primer algoritmo con una relación de rendimiento menor a 2 para todos k. Resultaron un límite superior de 1.85 para el rendimiento relativo de su algoritmo. Luego se demostro que la relación de rendimiento de su algoritmo es menor que 1.7 para k lo suficientemente grande, y que es como máximo 1.75 para todo k. Entonces se mejoran las relaciones más conocidas para cualquier k >= 4, en particular, para k = 4 de 1.75 a 1.65, y para k = 5 de 1.733 ... a 1.68. Por último, mostramos que el mínimo subgrafo de expansión k-conexo es MAX SNP-hard, incluso para k = 2. Todo esto se ve afectado por que el solo computar la conexidad de vértices de un grafo G tiene un tiempo de procesamiento polinomial, con una complejidad de min(k3+n,kn)\*m.

**Planteamiento:**

**Antecedentes:**

* “On the Structure of Minimum-Weight k-Connected Spanning Networks” [Online] Available: http://epubs.siam.org/doi/abs/10.1137/0403027 (Daniel Bienstock, Ernest F. Brickell, and Clyde L. Monma, 1988)

**Complejidad:** O( (p (log n))1/2/k )

* “A Better Approximation Ratio for the Minimum Size k-Edge-Connected Spanning Subgraph Problem” [Online] Available: https://pdfs.semanticscholar.org/1ec3/62164016964b14589b288c02384505361343.pdf (Cristina G Fernandes, 1997)
* “On Approximability of the Minimum-Cost k-Connected Spanning Subgraph Problem” [Online] Available: https://www.hni.uni-paderborn.de/publikationen/publikationen/?tx\_hnippview\_pi1%5Bpublikation%5D=1533&tx\_hnippview\_pi1%5Bfelder%5D%5Blade%5D=394 (Artur Czumaj and Andrzej Lingasy, 1999)

**Complejidad:** O( c2k4 ), O(c log W)

* “Approximation algorithms for minimum-cost k-vertex connected subgraphs” [Online] Available: https://dl.acm.org/citation.cfm?id=509955&dl=ACM&coll=DL&CFID=995825419&CFTOKEN=36763894 (Joseph Cheriyan, Adrian Vetta and Santosh Vempala, 2002)
* “Approximating Minimum-Size k-Connected Spanning Subgraphs via Matching” [Online] Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.58.8937&rep=rep1&type=pdf (Joseph Cheriyan and Ramakrishna Thurimella, 2006)
* “The k-edge connected subgraph problem: Valid inequalities and Branch-and-Cut” [Online] Available: https://limos.isima.fr/IMG/pdf/rr-07-01.pdf (F. Bendali, I. Diarrassouba, M. Didi Biha, A. R. Mahjoub, and J. Mailfert, 2007)
* “Approximating minimum cost connectivity problems” [Online] Available: <http://www.crab.rutgers.edu/~guyk/pub/book/nabs.pdf> (Guy Kortsarz and Zeev Nutov, 2008)
* “Minimum Cost ≤k Edges Connected Subgraph Problems” [Online] Available: http://www.sciencedirect.com/science/article/pii/S1571065310000053 (Firdovsi Sharifov and Hakan Kutucu, 2010)
* “On the minimum-cost λ-edge-connected k-subgraph problem” [Online] Available: https://link.springer.com/article/10.1007/s10287-016-0260-7 (Elham Sadeghi Neng Fan, 2016)

**Algoritmo(s):**

Para hallar K,

for (int i = 0; i < this.allSetups.length; i++) {

if (this.allSetups[i].getK() == -1) {

this.setup(i);

currentSetup.setK(findPathsForVertex(0)? -1 : 0);

}

}

for (int i = 0; i < this.allSetups.length; i++) {

GraphSetup set = this.allSetups[i];

if (set.getK() == -1) {

int minDist = N;

for (int j = 0; j < this.allSetups.length; j++) {

GraphSetup other = this.allSetups[j];

if (i != j && !other.isConnected()) {

minDist = Integer.min(minDist,set.distanceTo(other));

}

}

set.setK(minDist);

}

}

Para hallar el minimo subgrafo,

int minimumC = N;

int index = -1;

for (int i = 0; i < this.allSetups.length; i++) {

GraphSetup setup = this.allSetups[i];

if (setup.getK() == K) {

if (setup.getC() < minimumC) {

minimumC = setup.getC();

index = i;

}

}

}

this.setup(index);

**Algoritmo Concreto:**

boolean checkConnection = false;

boolean[] donePaths = new boolean[N];

for (int j = 0; j < N; j++) {

donePaths[j] = false;

}

boolean done = false;

int goalID = 0;

int changes = 0;

Vertex goal = V[goalID];

Vertex start = V[a];

if (!start.isActive()) {

return true;

}

while (!done) {

if (start.getP()[goalID].getGoal() != goal && goal.isActive()) {

int currentPathIndex = -1;

int currentLength = N;

for (int j = 0; j < N; j++) {

if (start.getP()[j].getGoal() == V[j]) {

Vertex midVertex = start.getP()[j].getGoal();

int toMid = start.getP()[j].getLength();

int toGoal = midVertex.getP()[goalID].getLength();

int possibleLength = toMid + toGoal;

if (midVertex.getP()[goalID].getGoal() == goal && midVertex.isActive() && possibleLength < currentLength) {

currentPathIndex = j;

currentLength = possibleLength;

}

}

}

if (currentPathIndex != -1) {

start.getP()[goalID].add(start.getP()[currentPathIndex]);

start.getP()[goalID].add(start.getP()[currentPathIndex].getGoal().getP()[goalID]);

X[currentPathIndex]++;

donePaths[goalID] = true;

changes++;

}

} else {

donePaths[goalID] = true;

}

**Accion de mejora**

No revisar todo vértice con todo otro vértice para revisar conexidad, sino revisar un vértice con todo otro vértice.

**Algoritmo Concreto Mejorado:**

boolean checkConnection = false;

boolean[] donePaths = new boolean[N];

for (int j = 0; j < N; j++) {

donePaths[j] = false;

}

boolean done = false;

int goalID = 0;

int changes = 0;

Vertex goal = V[goalID];

Vertex start = V[a];

if (!start.isActive()) {

return findPathsForVertex(a+1);

}

while (!done) {

if (start.getP()[goalID].getGoal() != goal && goal.isActive()) {

int currentPathIndex = -1;

int currentLength = N+1;

for (int j = 0; j < N; j++) {

if (start.getP()[j].getGoal() == V[j]) {

Vertex midVertex = start.getP()[j].getGoal();

if (midVertex.isActive() && midVertex.getP()[goalID].getGoal() == goal) {

int possibleLength = start.getP()[j].getLength() + midVertex.getP()[goalID].getLength() - 1;

if (possibleLength < currentLength) {

currentPathIndex = j;

currentLength = possibleLength;

}

}

}

}

if (currentPathIndex != -1) {

start.getP()[goalID].add(start.getP()[currentPathIndex]);

start.getP()[goalID].add(start.getP()[currentPathIndex].getGoal().getP()[goalID]);

donePaths[goalID] = true;

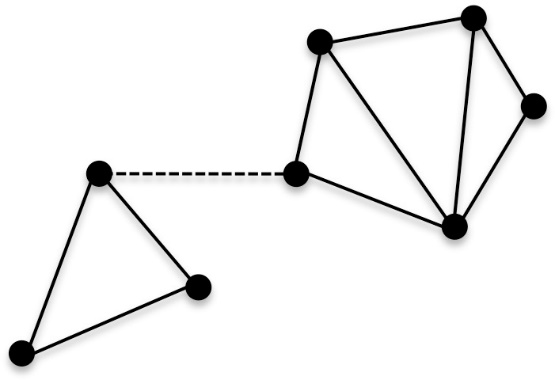
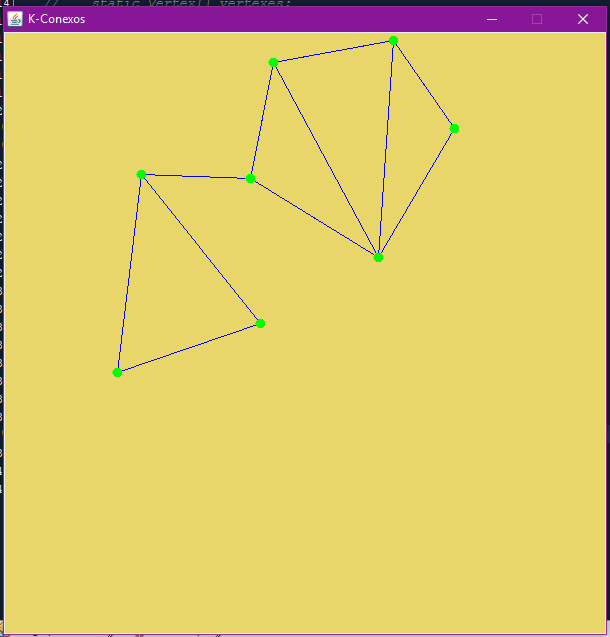
changes++;

}

} else {

donePaths[goalID] = true;

}



G =

|  |  |  |
| --- | --- | --- |
| Pruebas | Tiempo Original (microsegundos) | Tiempo Mejorado (microsegundos) |
| 1 | 1262.44 | 841.623 |
| 2 | 1241.91 | 894.726 |
| 3 | 1426.65 | 1033.955 |
| 4 | 1231.64 | 580.568 |
| 5 | 1637.06 | 880.446 |