Contents

1 Problem description

Given a 2n-node undirected graph G(V,E); positive integer $k \leq G.E$. Can the nodes of G be partitioned into 2 disjoint sets U and W each of size n and such that the total number of distinct edges in G.E that connect a node u in U to a node w in W is at most k?

The problem described above is the decision version of the well-known MINIMUM BISECTION PROBLEM. So let us call our problem K-BISECTION problem. A brief overview of MINIMUM BISECTION PROBLEM is given below.

MINIMUM BISECTION PROBLEM

The MINIMUM BISECTION PROBLEM is a well-known NP-hard problem, which is intended to partition the vertices of a given graph into two equal halves so as to minimize the cost where cost is defined as the number of edges with two ends in 2 different halves. MINIMUM BISECTION was shown to be NP-hard by Garey, Johnson and Stockmeyer[1].

2 Problem formulation

K-BISECTION PROBLEM

INPUT : $< G, k >$, where G is a graph encoding and k is a positive integer.

OUTPUT : YES, if $< G, k > \in L$
NO, otherwise

where language L is defined as follows,
L = $\{< G, k > ||G.V| = 2n$, $n \in I$ and G.V can be partitioned into U and V such that $U \cap V = \emptyset$ and $U \cup V = G.V$ and number of edges $(x, y) \in G.E$ such that $x \in U$ and $y \in V$ is at most k $\}$.

3 Solving MINIMUM BISECTION PROBLEM using K-BISECTION as a subroutine

Let ML be a deterministic Turing machine that determines if an encoded input $< G, k >$ belongs to language L or not. We use this machine to build a machine MBP which finds the min cost bisection of the input graph. The construction of this machine is shown in the figure below.

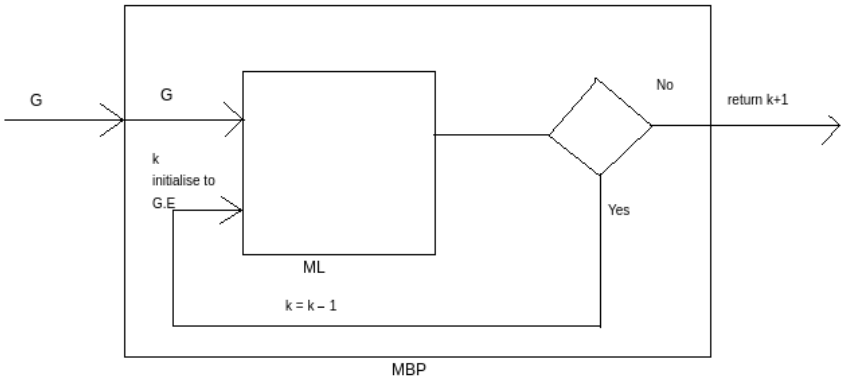

Figure 1: Machine to find min cost bisection

4 Proof that K-BISECTION is NP-Complete

To prove that the given problem is NP-Complete ,we have to show that it is both NP and NP-Hard.

4.1 Proof that K-BISECTION is NP

To prove that the problem is NP, we give a polynomial time verification algorithm for a certificate to the problem. We have as the certificate an encoding of graph G , two sets of vertices V1 and V2 and a positive integer k. The algorithm is as follows

VERIFY$(G, V1, V2, k)$

```
1  if (|V1| ≠ |V2|)
2        return FALSE
3  if (V1 ∪ V2 ≠ G.V) ∨ (V1 ∩ V2 ≠ ∅)
4        return FALSE
5  count = 0
6  for ∀u ∈ V1.
7        for ∀v ∈ V2.
8              if (u, v) ∈ G.E
9                    count = count + 1
10 if count ≤ k
11       return TRUE
12 return FALSE
```

4.2 Proof that K-BISECTION is NP-Hard.

To prove that this problem is NP-Hard, we give a reduction from K-CUT problem ,the decision version of MAX-CUT problem, which is already proved to be NP-Hard.

4.2.1 MAX-CUT PROBLEM

The MAX-CUT problem asks us to find a subset S of the vertex set of the given graph such that the number of edges between S and the complementary subset is as large as possible.

4.2.2 K-CUT PROBLEM

The K-CUT problem takes as input a graph G and a positive integer k and determines whether G contains a cut of size at most k or not.
K-CUT = $\{< G, k > \mid$ Graph G has a cut of size at most k$\}$.

4.2.3 Reduction from K-CUT to K-BISECTION

Let the input to K-CUT problem be $< G, k >$ where G is a graph and k is a positive integer.

- Add $|G.V|$ isolated vertices to G to yield G'.
- G' now has $2|G.V|$ vertices.
- As the newly added vertices have no edges, moving them around contributes nothing to the cut.
- Every cut $(S, V - S)$ of G can be made into a bisection by appropriately allocating the new nodes between S and V - S.
- Hence each cut of G can be made a cut of G' of the same size, and vice versa.
- So the presence of a k-bisection in G' implies the presence of a k-cut in G.

Hence proved that K-BISECTION is NP-Hard.

As K-BISECTION is both NP and NP-Hard, it belongs to the NP-Complete class of problems.

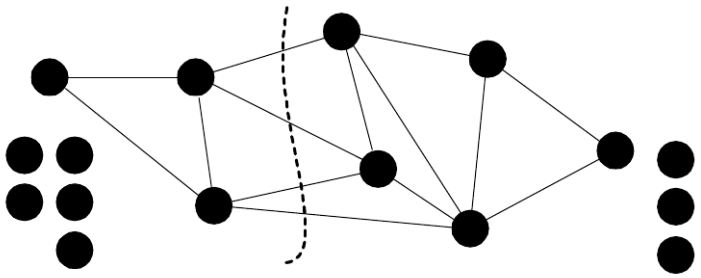

Figure 2: Graph G'

5 Applicability of classical strategies

Since we have shown that K-BISECTION belongs to NP-C, any of the classical strategies like divide and conquer, greedy strategy, dynamic programming etc are unlikely to give a polynomial time solution.
An exhaustive brute force search which tries all possible bisections of the given graph can give a solution in exponential time.

5.1 A brute force algorithm for K-BISECTION problem

In this section we give a brute force algorithm for K-BISECTION problem[1]

BRUTE-FORCE(G, k)

```
1  count = 0
2  for ∀U, V ⊂ G.V such that |U| = |V|  &  U ∩ V = Ø  &  U ∪ V = G.V :
3        for ∀(x, y) ∈ G.E
4              if (x ∈ U  &  y ∈ V) ∨ (x ∈ V  &  y ∈ U)
5                    count = count + 1
6        if count ≤ k
7              return TRUE
8  return FALSE
```

5.2 Analysis of time complexity of brute force algorithm

Let $|G.V| = n$ and $|G.E| = m$
Step 2 tries all possible partitions of G.V. We know that maximum no of possible partitions is equal to $\binom{n}{n/2}$.

[1]Implementation of this algorithm in C++ programming language(brute_force.zip) has been submitted through eduserver page.

$\binom{n}{n/2} = \frac{n!}{(n/2)!(n/2)!} = \frac{\sqrt{2\pi n}(n/e)^n}{[\sqrt{2\pi(n/2)}(\frac{n/2}{e})^{\frac{n}{2}}]^2} = \sqrt{\frac{2}{\pi n}}2^n$

Step 3 checks every edge of G for each possible partition.
So this algorithm runs in $O(m(2^n))$ steps.

6 Study of approximability

6.1 An approximate algorithm for MIN-BISECTION problem

In this section we present an approximate algorithm for MIN-BISECTION problem.The approximation factor of best algorithm known till date for MIN-BISECTION is polynomial in number of vertices of the graph.Here we present an elementary algorithm known as longest path algorithm, which shows good experimental results. [2]

The algorithm takes as input a graph G=(V,E) and return two sets of vertices X and Y such that $|X| = |Y|$ and $X \cup Y = G.V$ and $X \cap Y = \emptyset$

LONGEST-PATH(G) [2]

Input Graph G = (V, E) with $|G.V| = n$.

1. Choose uniformly at random a vertex z.
2. Z = {z}.
3. List all neighbors of vertices from Z and add it to Z.
4. Repeat step 3, until $Z = G.V$.
5. Choose one of the vertices, added in the last execution of step 3, and denote it with x.
6. Repeat steps 2 to 5 with x instead of z. The resulting vertex is denoted with y.
7. X = {x}, Y = {y}.
8. List all neighbors of vertices from X and add each neighbor to X, except for $|X| \geq \lceil \frac{n}{2} \rceil$
9. Repeat step 8, if $|X| \leq \lceil \frac{n}{2} \rceil$.

[2]Implementation of this algorithm in C++ programming language(longest_path.zip) has been submitted through eduserver page.

10. List all neighbors of vertices from Y and add each neighbor to Y, except for $|Y| \geq \lceil \frac{n}{2} \rceil$
11. Repeat step 10, if $|X| \leq \lceil \frac{n}{2} \rceil$.
12. return X,Y.

.

The algorithm initially picks a vertex z from G.V at random and adds it to the set Z. In step 3 it adds all neighbours of z to set Z. Again it picks vertices from set Z and add their neighbours to it. It keeps on doing this until Z=G.V. In other words it visits the vertices of the graph layer by layer starting from vertex z. Then it selects one of the vertices added in the last step (x in our case) and repeat steps 2 to 5 starting with x.This time the last vertex added is denoted by y. Now include x in a set X and repeat the same procedure until $|X| = \lceil \frac{n}{2} \rceil$. Include y in a set Y and repeat the procedure with this set until $|Y| = \lceil \frac{n}{2} \rceil$.

The resulting sets X and Y are the approximate min bisection of the given graph G returned by longest path algorithm. It should be noted that this algorithm works only for connected graphs.

We can use this algorithm to create an approximate algorithm for K-BISECTION problem as described below.

1. Take $< G, k >$ as input.
2. Run LONGEST-PATH on G to obtain two sets of vertices X and Y.
3. Count the number of edges in G.E having endpoints in two different sets.
4. If the number of such edges is less than k, return TRUE.
5. Else, return FALSE.

The above mentioned algorithm need not always give correct output. The correctness of this algorithm depends on how close the approximate min cost produced by LONGEST-PATH algorithm is to the actual min cost of bisection.

6.2 Analysis of time complexity of approximate algorithm

In step 3,8 and 10 of the algorithm we see that, the size of set of neighbouring vertices for any vertex can be $O(n)$. Execution of step 3 increases the size of Z by at least one. Similarly step 8 and 10 increases the size of X and Y by at least one respectively. So executing these steps for all the vertices would mean that the overall time complexity will be $O(n^3)$.

A good implementation of this algorithm can result in a better time complexity of $O(n^2)$. This can be achieved by keeping track of visited vertices and thus avoiding processing them again and again.

7 Applications of MIN-BISECTION problem

MIN-BISECTION problems arise in fields such as VLSI design, image processing, parallel computing, task scheduling etc.[3]

References

[1] Garey, Johnson and Stockmeyer. *Some simplified NP-Complete graph problems.*
https://www.sciencedirect.com/science/article/pii/0304397576900591

[2] Gerold Jäger. *An Efficient Algorithm for Graph Bisection of Triangularizations.*
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.8603rep=rep1type=pdf

[3] Ureil Feige and Robert Kruthgamer. *A POLY LOGARITHMIC APPROXIMATION OF THE MINIMUM BISECTION.*
http://www.wisdom.weizmann.ac.il/ robi/papers/FK-bisection2-SIGEST.pdf