CS395T: Large Scale Data-Mining, Lecture 9: GraphPartitioning

1 Objective

G = (V,E) , where V is set of vertices and E is set of edges.
Informal Objective : Partition V into disjoint parts V1 and V2 such that

(i) No of crossing edges between V1 and V2 is minimized,
(ii)V1 and V2 are nearly equal in size.

The desire to get equal-sized partitions makes the problem hard - it is much harder than the min-cut problem. Problem is NP-complete.
One formal objective:

$$\min_{V1,V2} Cut(V1, V2)/|V1| + Cut(V1, V2)/|V2| - \text{ratio cut objective}$$

Many heuristic algorithms for this problem have been proposed. Three of the most successful algorithms are:

2 Kernighan-Lin Algorithm

Kernighan-Lin algorithm is a local descent algorithm, and it produces equal (or off-by-one) partitions.
The algorithm start with an edge weighted graph

$$G = (V, E, W_E)$$

AND a partitioning

$$V = A \cup B, |A| = |B|$$

Goal : Find X ⊂ A, Y ⊂ B, and swap(exchange) X and Y such that this swap reduces the total cost of edges from A to B.

Current cost = $\sum_{a \in A, b \in B} w(a,b)$, where w(a,b) is the weight of edge(a,b).
A_{new} = A - X ∪ Y, B_{new} = B - Y ∪ X
New cost =

$$\sum_{a \in A_{new}, b \in B_{new}} w(a,b)$$

To compute the new cost efficiently, maintain
E(a) = external cost of a = $\sum_{b \in B} w(a,b)$
I(a) = internal cost of a = $\sum_{a` \in A, a` \neq a} w(a,a`)$
C(a) = E(a) - I(a)
and analogously
E(b) = external cost of b = $\sum_{a \in A} w(a,b)$
I(b) = internal cost of b = $\sum_{b` \in B, b` \neq b} w(b,b`)$
C(b) = E(b) - I(b)
Swap a and b by first computing

$$\begin{aligned} NewCost &= oldCost + [I(a) - E(a) + w(a,b)] + [I(b) - E(b) + w(a,b)] \\ &= oldCost - [C(a) + C(b) - 2w(a,b)] \\ &= oldCost - Gain(a,b) \end{aligned}$$

C(a') and C(b') also changed for all a` ∈ A, a' ≠ a and b ` ∈ B, b` ≠b.

2.1 Kernighan-Lin

Initialize partitions A & B and compute total cost.
Repeat

- Compute C(x) for all vertices x.
- Unmark all vertices.
- While there are unmarked vertices –($|N|/2$) iterations
  - i = 1
  - Find unmarked pair(a_i, b_i) maximizing gain(a_i, b_i) – O($|N|^2$)
  - mark a_i and b_i (Don't swap)
  - Update C(x) ∀ unmarked x, as though a_i & b_i had been swapped.
  - i ← i + 1
- Pick j that maximizes Gain = $\sum_{i=1}^{j} gain(a_i, b_i)$

- If Gain > 0 then
  - Update

$$\begin{aligned} A &= A - a_1, a_2, \ldots a_k \cup b_1, \ldots b_k, \quad k = |N|/2 \\ B &= B - b_1, \ldots b_k \cup a_1, \ldots a_k \\ Cost &= Cost - Gain. \end{aligned}$$

Until Gain $\geq$ 0.
Cost = O($|N|^3$) assuming # iteration is a fixed constant. Note that some Gain(a_i, b_i) may be negative. Thus, it is possible to escape some "local minimum". But overall, K-L is still a local algorithm. A better implementation by Fiduccia & Matthyses, 1982. O($|E|$) cost per iteration.

3 Spectral Partitioning

Consider the node-edge incidence matrix E where E is an $|V| * |E|$ matrix where column e=(i,j) is zero except for the i^{th} and j^{th} entries which are +1 or -1 respectively.
Let $L = EE^T$ which is a $|V| * |V|$ matrix. It turns out that L(i,i) = degree of node i, and L(i,j) = -1, if there is an edge(i,j), i≠j.
Since $L = EE^T$ it is(a)symmetrix, (b) positive semi-definite matrix ⇒ all eigenvalues of L are real and non-negative.
If $e = [11 \ldots 1]^T$, ie. e is the vector of all ones, it is easy to see that Le = 0.
Many graph properties are expressed in terms of eigenvalues of L. For example,

- # if zero eigenvalues of L = # of connected components of G.
- Let λ and v be an eigenvalue & a normalized eigenvector of L.
  - $L \cdot v = \lambda \cdot v$
  - $v^T \cdot L \cdot v = \lambda$

The smallest eigenvalue of L is 0 since Le = 0.
Suppose G is connected, then λ2(G) (L's second smallest eigenvalue) is positive.
Let the vector X capture graph partitioning information, ie.,

$$x = \begin{cases} 1 & \text{if node i} \in V_1 \\ -1 & \text{if node i} \in V_2 \end{cases}$$

Then

$$X^T L x = x^T E E^T x = \sum_{(i,j) \in E} [x(i) - x(j)]^2$$

$$= 4 \cdot Cut(V_1, V_2)$$

Thus we are looking for a ± 1 vector x such that $x^T L x$ is minimized.
This is very hard, but real relaxation $\Rightarrow x \approx v_2 =$ second smallest eigenvector.

3.1 Simple Spectral Bisection Algorithm

- Compute eigenvector V_2 for eigenvalue $\lambda 2$ of L(G).
- For each node i $\in$ V,
  - If V_2(i) < 0
    - put node i in partition V_1
  - else
    - put node i in partition V_2
  - end if.

Many other extensions to this simple algorithm have been studied. Spectral partitioning gives good results that can be improved by K-L or FM heuristic. Spectral partitioning gives a global heuristic that can be improved by the "local-smoothing" of KL or FM.

4 Multilevel Methods

- Very Fast.
- Good partitioning results.
- e.g., METIS software.
- Multilevel KL or multilevel Spectral.
- Multilevel methods use a divide and conquer approach.

4.1 Steps

1. Coarsen the graph G.
2. Partition coarse graph(may do it by recursively going to step 1).
3. Refine the coarse partitioning to get partitioning for G.