CS403: Algorithm Design and Analysis

Feb-Jun 2017

Indian Institute of Technology Mandi

Assignment 6

Priyansh Saxena

B14118

Computer Science and Engineering

CS 403 – Algorithm Design and Analysis

Assignment 6

Problem 1: Implement the max-flow problem as a linear programming problem.

In this problem we are given a graph G = (V , E) and associated with each edge is a capacity. We need to find the maximum flow from a source node to a sink node. We use flow constraints and conservation laws to construct a series of inequalities. We then use linear programming to solve a standard maximization problem in which the equation to be maximized is the total flow coming out of the source node. The algorithm is given below:

find_max_flow(G,k)
    Construct the inequalities using flow constraint and conservation laws
    Apply Simplex Method to solve the problem
end

The main computation for the problem is done to solve the augmented matrix. The augmented matrix is of size $O(E^2)$. Solving the augmented matrix requires O(E) computation and each computation takes $O(E^2)$. Therefore, the total time of the algorithm is $O(E^3)$.

Problem 2: Implement the vertex cover problem as an integer programming problem.

Given a graph G(V,E) and associated with each vertex having a weight, find a minimum weight vertex cover using Integer Linear Programming.

Assign a decision variable x_i for each node $i \in V$ to indicate whether to include node i in the vertex cover or not, $x_i = 1$ being that node i is in the vertex cover, and $x_i = 1$ being that it is not. We use linear inequalities to encode the requirement that the selected nodes form a vertex cover and the objective function to encode the goal of minimizing the total weight.

For each edge $(i, j) \in E$, it must have one end in the vertex cover, and we write this as the inequality $x_i + x_j \geq 1$, for all pairs of nodes (i,j).

The minimization problem is then to minimize the value $w^T x$, where w denotes the set of node weights as an n-dimensional vector and x denotes the n-dimensional vector formed by assigning x_i values for all nodes to its i^{th} component.

The main computation for the problem is done to solve the augmented matrix. The augmented matrix is of size $O(E^2)$. Solving the augmented matrix requires O(E) computation and each computation takes $O(E^2)$. Therefore, the total time of the algorithm is $O(E^3)$.

Problem 3: Implement the randomized contraction algorithm to find global min-cut with probability of correct answer at least $1 - (1/n^3)$.

Karger's Contraction algorithm is used to solve this problem. This is a randomized method in which a random edge e(u,v) is picked and merged into a single node w, thereby producing a new graph G'. Edges between u and v in the original graph G are not present in G'. Any other edge in G with one end as u or v is updated to w.

The algorithm is to perform this contraction until only two nodes are left. The number of edges that connect the two supernodes is the size of the cut.

The pseudo code for the algorithm is:

```
karger(multigraph G(V,E)) {
        for each node v, let S(v) be the set of nodes that have been contracted into v
        initially S(v) = {v} for each v
        if G has two nodes v1 and v2
                then return cut (S(v1), S(v2))
        else
                choose an edge e = (u, v) of G uniformly at random
                let G' be the graph resulting from the contraction of e,
                  with a new node zuv replacing u and v
                define S(zuv ) = S(u) ∪ S(v)
                return karger(G')
        endif
}
```

A single run of this algorithm returns a global min cut with probability atleast $1/({}^{n}C_{2})$. If we run this algorithm for N iterations where $N = c*({}^{n}C_{2})*\ln(n)$,then the probability of success becomes atleast $1- 1/(n^{c})$.

The contraction algorithm takes atmost O(E) operations, each of O(1) because of union by rank and path compression heuristics used in the disjoint set. The contraction algorithm is run for $c*({}^{n}C_{2})*\ln(n)$ times where n = V. So the total running time for this algorithm is $O(c.n^{2}.\ln(n).E)$.