Survivable Network Design

Mihir Kaskhedikar
Raghav Ajmera

IIT Delhi

COL754

Problem Statement

We are given as input an undirected graph $G = (V, E)$, costs $c_e \geq 0$ for all $e \in E$, and connectivity requirements r_{ij} for all pairs of vertices $i, j \in V$, where $i \neq j$. The connectivity requirements are nonnegative integers. The goal is to find a minimum-cost set of edges $F \subseteq E$ such that for all pairs of vertices i, j with $i \neq j$, there are at least r_{ij} edge-disjoint paths connecting i and j in (V, F).

Integer Program

The problem can be modelled by the following integer program:

$$\begin{aligned} \text{minimize} \quad & \sum_{e \in E} c_e x_e \\ \text{subject to} \quad & \sum_{e \in \delta(S)} x_e \geq \max_{i \in S, j \notin S} r_{ij}, \quad \forall S \subseteq V, \\ & x_e \in \{0, 1\}, \quad \forall e \in E. \end{aligned}$$

Solution of LP

- relax x_e as $0 \leq x_e \leq 1$
- We can find the optimal solution using ellipsoid method
- max flow instance between each pair of nodes serves as the polynomial time separation oracle

Weakly Supermodular

The function $f(S) = \max_{i \in S, j \notin S} r_{ij}$ is weakly supermodular.

- For any solution to LP if f is weakly super modular then there exists an edge $e \in E'$ such that $x_e \geq 1/2$
- This can be proved by the coin charging method done in class

Iterated Rounding

Algorithm 1 Pseudocode for Solving LP on Edge Set

$F \leftarrow \emptyset$
$i \leftarrow 1$
while F is not a feasible solution **do**
    Solve LP on edge set $E - F$ with function f_i, where
    $f_i(S) = f(S) - |\delta(S) \cap F|$, to obtain basic optimal solution x
    $F_i \leftarrow \{e \in E - F : x_e \geq 1/2\}$
    $F \leftarrow F \cup F_i$
    $i \leftarrow i + 1$
end while
return F

2−Approximation

- The above algorithm is a 2−approximation
- Intuitively in every iteration, the cost increases by less than 2 times the optimal value of the remaining LP
- Polynomial in time, at most $|E|$ iterations and in each iteration LP is solved in polynomial time using ellipsoid method

Implementation Challenges

- exponentially many constraints in LP
- we implemented ellipsoid method to solve this LP
- used Edmonds-Karp max flow algorithm as separation oracle

Ellipsoid Method

equation of ellipsoid at the k^{th} iteration

$$\mathcal{E}^{(k)} = \left\{ x \in \mathbb{R}^n : \left(x - x^{(k)} \right)^\top P_k^{-1} \left(x - x^{(k)} \right) \leq 1 \right\}.$$

Obtain a plane using separation oracle such that

$$g^{(k+1)\top} \left(x^* - x^{(k)} \right) \leq 0.$$

Ellipsoid Method

We therefore conclude that

$$x^* \in \mathcal{E}^{(k)} \cap \left\{ z : g^{(k+1)\top} \left(z - x^{(k)} \right) \leq 0 \right\}.$$

Update the equation of ellipse and jump the query point to new center

$$x^{(k+1)} = x^{(k)} - \frac{1}{n+1} P_k \tilde{g}^{(k+1)},$$

$$P_{k+1} = \frac{n^2}{n^2 - 1} \left(P_k - \frac{2}{n+1} P_k \tilde{g}^{(k+1)} \tilde{g}^{(k+1)\top} P_k \right),$$

where

$$\tilde{g}^{(k+1)} = \left(\frac{1}{\sqrt{g^{(k+1)\top} P_k g^{(k+1)}}} \right) g^{(k+1)}.$$

Ellipsoid Method

```
        feasible, hyperplane = separation_oracle(x, lamda, edges_found)
        if feasible:
            print(f"Solution found for {lamda}")
            solution_found = True
            found = True
            break

        a = hyperplane
        norm = np.sqrt(a.T @ P @ a)
        if(norm < tol_x):
            print("Norm value is too small")

        a = a / norm

        x = x - (1 / (m + 1)) * (P @ a)
        P = (m**2 / (m**2 - 1)) * (P - (2 / (m + 1)) * (P @ np.outer(a, a) @ P))

        if np.linalg.det(P) < tol_det:
            print(f"Ellipsoid volume too small, no solution found for lambda : {lamda}")
            break
```

Figure: Ellipsoid Implementation

Time Complexity

We performed binary search on the optimal solution as ellipsoid method only checks feasibility, ellipsoid method requires at most steps

$$2(n-1)n \cdot \ln\left(\frac{V(p)}{\epsilon}\right)$$

Each step computes flow $O(n^2)$ times, and in each step of iterated rounding we add at least one edge, therefore at most $|E|$ steps. Overall,

$$O\left(n^3 \cdot E^3 \cdot 2(n-1)n \cdot \ln\left(\frac{V(p)}{\epsilon}\right)\right)$$

Separation Oracle

```
#checking if each xi is in the range [0,1]
for i in range(m):
    if(x[i] < -tol_x) : #xi >= 0
        arr = np.zeros(m)
        arr[i] = -1
        return (False, arr)
    if(x[i] > 1 + tol_x) : #xi <= 1
        arr = np.zeros(m)
        arr[i] = 1
        return (False, arr)

#checking if sum of cost(i) * x(i) <= lamda
const = 0
for i in range(m) :
    const += (costs[i] * x[i])
if(const > lamda):
    arr = copy.deepcopy(costs)
    return (False, arr)

#checking if for edges already found, xi is 1
for e in edges_found :
    if(x[e] < 1 - tol_x) : #xe >= 1
        arr = np.zeros(m)
        arr[e] = -1
        return (False, arr)
```

Figure: Separation Oracle 1

Separation Oracle

```
for source in range(0, n, 1) :
    for sink in range(0, n, 1) :
        if(sink == source) :
            continue
        #computing the source-sink flow
        (flow_value, flow_dict) = nx.maximum_flow(G, source, sink, flow_func = nx.algorithms.flow.edmonds_karp)

        #creating residual graph of this flow
        flow_cap = dict()
        for key1, value1 in flow_dict.items():
            for key2, value2 in value1.items():
                flow_cap[(key1, key2)] = value2

        Gres = nx.DiGraph()
        for i in range(n):
            Gres.add_node(i)

        for e, cap in edge_cap.items():
            if((e not in flow_cap) or (flow_cap[e] < cap - tol_x)):
                Gres.add_edge(e[0], e[1], capacity = 1)

        #finding the reachable and non-reachable sets for this flow
        reachable = nx.descendants(Gres, source)
        reachable.add(source)
        non_reachable = set(Gres.nodes()) - reachable

        coeff = np.zeros(m)
        if(flow_value < req[source][sink] - 1e-2) : #checking if some (source,sink) flow is violating requirement
            for i in range(m) :
                if(((edge_list[0][i] in reachable) and (edge_list[1][i] in non_reachable))
                   or ((edge_list[1][i] in reachable) and (edge_list[0][i] in non_reachable))) :
                    if(not (i in edges_found)):
                        coeff[i] = -1
            return (False, coeff) #return the violated cut
```

Figure: Separation Oracle 2

Comparison with optimal

- To compare the output of the algorithm with the optimal value of the fractional LP, we generated random graphs $G = (V, E)$ where for each $i, j \in V$, $i \neq j$, (i, j) belongs to E with the same probability $edge\_prob$.

Comparison with optimal

- To compare the output of the algorithm with the optimal value of the fractional LP, we generated random graphs $G = (V, E)$ where for each $i, j \in V$, $i \neq j$, (i, j) belongs to E with the same probability $edge\_prob$.

- Each edge of G is given a cost in the range $[1, 10]$

Comparison with optimal

- To compare the output of the algorithm with the optimal value of the fractional LP, we generated random graphs $G = (V, E)$ where for each $i, j \in V$, $i \neq j$, (i, j) belongs to E with the same probability $edge\_prob$.
- Each edge of G is given a cost in the range $[1, 10]$
- We created a symmetric requirement matrix of size $|V| \times |V|$ where each entry is 0 with probability 0.3 and an equiprobable integer in the range $[1, max\_req]$ with probability 0.7.

Comparison with optimal

- To compare the output of the algorithm with the optimal value of the fractional LP, we generated random graphs $G = (V, E)$ where for each $i, j \in V$, $i \neq j$, (i, j) belongs to E with the same probability $edge\_prob$.
- Each edge of G is given a cost in the range $[1, 10]$
- We created a symmetric requirement matrix of size $|V| \times |V|$ where each entry is 0 with probability 0.3 and an equiprobable integer in the range $[1, max\_req]$ with probability 0.7.
- We kept $edge\_prob = 0.8$ for all test-cases.

Comparison with optimal

n	m	max_req	OPT	ALG	edges in OPT
20	147	2	89.27	120	26
30	350	2	170.72	240	72
40	540	3	260.70	396	135

Table: Performance of algorithm compared to the optimal solution