COMP5712 - Combinatorial Optimization

Taught by Sunil Arya

Notes by Aaron Wang

Contents

1 March 26th, 2019

1.1 Flow Decomposition

Lemma 1.1

Let $(G = (V, E), s, t, c)$ be a flow network. Let f be a flow in this network. Then there is a collection of feasible flows $f_1, f_2, \dots f_k$ and a collection of $s - t$ paths $p_1, \dots, p_k$.

- Value of f is equal to the sum of flows f_i.
- flow f_i sends positive flow along edge p_i.
- $k \leq |E|$. (look at lecture notes) for every path, at least one edge will drop off (since once edge gets zero'd, as such we can only have at most $|E|$ paths.

This means the average flow **for the first step** must have $\frac{|f|}{k}$, and the maximum must have more than this amount. This is greater than $\frac{OPT}{|E|}$.

Corollary 1.2

At the beginning, there is a augmenting path from s to t in which each edge has **capacity** $\frac{OPT}{|E|}$, where OPT is the value of the maximum flow.

This is because there must be a $s - t$ path on the residual network for which all edges on that path has capacity greater than equal to $\frac{OPT}{|E|}$ (as we can push this much flow).

Since OPT is changing, even though we need at most $|E|$ iterations, running time will be $|E| \log OPT$.

Theorem 1.3

Assuming a flow network with integer capacities, the fattest-path implementation of Ford-Fulkerson method runs in time at most $(|E| \log(OPT)\, |E| \log |V|)$
$= O(|E|^2 \log |V| \log(OPT)$ which is polynomial.

The time for one iteration to find the largest path is

- $|E|$ to construct the residual network
- $|E| \log |V|$ to run Dijkstra's algorithm to find the path

Proof. Let $m = |E|$ and f_i denote the flow value after i iterations. Let res_i denote the value of optimal flow in the residual network after i iterations.

$$res_i = OPT - f_i.$$

By 1.2, in the $(i+1)$-th iteration, we can find a flow of value greater than or equal to $\frac{res_i}{2|E|}$, as the residual network is a flow network with at most $2|E|$ edges (a forward edge and backward edge for each edge in G). Note that

$$res_{i+1} \leq res_i - \frac{res_i}{2|E|} = res_i \left(1 - \frac{1}{2|E|}\right).$$

As the fattest path has at least $\frac{res_i}{2|E|}$. Now we need to see how many iterations will it take for res_i to be less than or equal to 1 (since they are integer). Note that the factor it drops by is constant. As such we have

$$res_0 = OPT.$$

$$res_t \leq res_{t-1}\left(1 - \frac{1}{2|E|}\right) = \ldots = res_0\left(1 - \frac{1}{2|E|}\right)^t = OPT\left(1 - \frac{1}{2|E|}\right)^t.$$

After $2|E|\ln(OPT)$ iterations

$$res_t \approx OPT\left(1 - \frac{1}{2|E|}\right)^{2|E|\ln OPT} \approx \left(\frac{1}{e}\right)^{\ln OPT} = 1.$$

□

As such for $t \approx 2|E|\ln OPT + 1$, $res_t < 1 \implies res_t = 0$. Look at formal proof in notes.

The $\log(OPT)$ factor in the running time is awkward for many people, so we will try to remove it.

1.2 Edmonds-Karp Algorithm

Definition 1.4 (Strongly Polynomial vs Weakly Polynomial)**.** An algorithm runs in **strongly** poly time if, assuming unit time arithmetic operations (e.g. $+, -$), the running time is polynomial in the # of numerical operations given as input (polynomial only in $|V|, |E|$).

Note that our algorithm is not strongly polynomial - we say that it is weakly polynomial. (polynomial in the number of bits in the problem size).

This distinction is only applicable to problems dealing with integers. One problem where this is relevant is in linear programming. As of now, there are no strongly polynomial time algorithm known yet. All known poly-time algorithms are weakly polynomial (ellipsoid method, integer point methods are weakly polynomial). However, for NP-Complete perspective, both are polynomial in the input size.

Edmonds-Karp Algorithm is a:

- Strongly polynomial
- Specific implementation of FF method
- At each iteration, choose the path in the residual network with the smallest # of edges. Each iteration will take $|E|$ time (by BFS).

Theorem 1.5

If, at a certain iteration, the length of a shortest $s - t$ path is ℓ then at every subsequent iteration, it is $\geq \ell$. Furthermore, after at most $|E|$ iterations, then the length of the shortest $s - t$ path becomes $\geq \ell + 1$.

Note that ℓ can only stay the same or increase. In addition, at each ℓ you can only stay at the same length for $|E|$ iterations. On top of that, $\ell \leq |V| - 1$, since it is a simple path, as such:

$$\text{Total \# of iterations} \leq |E|\,(|V| - 1) = O(|E||V|).$$

Proof. Consider the residual network after T iterations. The length of the shortest $s - t$ path is ℓ. Edges in the graph of BFS from s that go downwards are called the "forward edges". Note that edges can only go down by one level and can connect vertices on the same level or upwards (cannot go more than one level down).

In iteration $T + 1$, we push additional flow to the shortest $s - t$ path, saturating at least

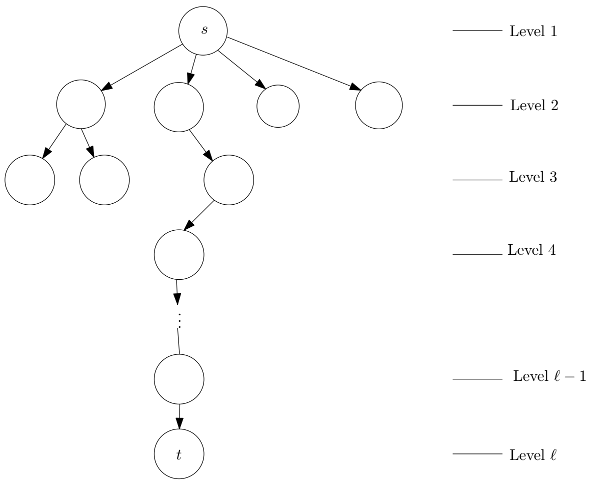

Figure 1

one of the edges on this path. As such the residual network will look as follows:

- All edges on P that are saturated disappear
- we may introduce backward edges connecting to edges in P

Note that it will not fall. Proving that ℓ is increasing. For ℓ to stay the same, you must only use forward edges, otherwise it would increase. Since you remove at least one forward edge during each iteration, you can only stay on the same level for $|E|$ iterations. □

2 March 28th, 2019

2.1 Bipartite Matching

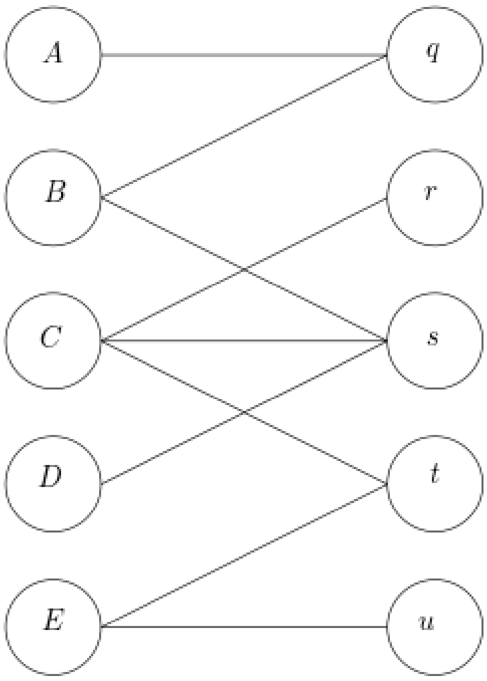

Figure 2: Example of Bipartite Graph

Definition 2.1. A **bipartite graph** is a graph with a bipartition, with all edges only crossing the bipartition.

- An edge represents a person willing to do a job
- Each person can be assigned to at most one job
- Each job can be occupied by at most one person
- QUESTION: is it possible to assign the employees such that every employee gets one job and each job is filled?

Remark 2.2 — A **maximal matching** is a matching for which if you add another edge, it would not be a matching. A **maximum matching** is the matching with maximum cardinality. A maximum matching is always a maximal, but a maximal is not always a maximum.

Definition 2.3. A **neighborhood** of a subset of vertices, A, is all the vertices they are connected to a vertex in A by an edge in E. e.g.

$$N(\{A, B, D\}) = \{q, s\}.$$

Remark 2.4 — The following are necessary conditions to have a perfect matching:

- The size of each side in this partition is the same.
- The size of the neighborhood of any subset is greater than or equal to the size of the subset. e.g. in the example above

$$|N(S)| \geq |S|, \ \forall S \subseteq L.$$

These will be proven to be sufficient later

From weak duality, we have that for any graph, the size of the max matching is less than or equal to the size of the min vertex cover.

2.2 Bipartite Matching as a Max Flow Problem

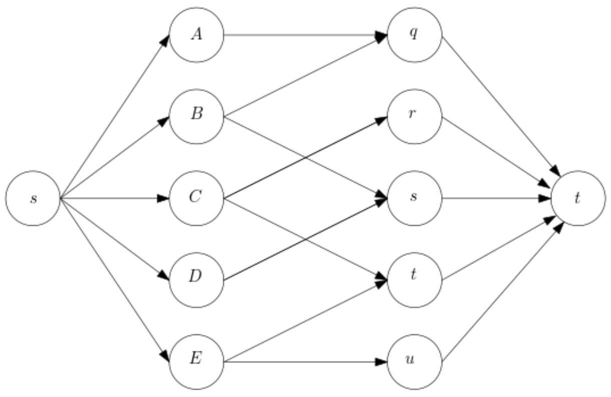

Figure 3: Matching as a Max Flow Problem (all $c(s, v)$ and $c(t, v)$ is 1)

Easy observations:

1. Suppose you have an integral flow of value k in G', the flow network. Then you have a matching of size k in G.
2. Suppose you have a matching os size k is G. Then you have a flow of value k.

Algorithm 2.5

We can solve the problem of finding a max matching in a bipartite graph in polynomial time by reducing it to max flow as follows:

- Let L, R denote the vertices on the left and right side of the bipartite graph respectively. Direct edges of the graph from left to right.
- Add 2 new nodes s and t, and add direct edges from s to all vertices in L and from all vertices in R to t.
- Set $c(s, v) = c(v, t) = 1$ and ∞ for edges from L to R.
- Find the max flow in the network, assuming that the flow is integral.
- Return the edges of the graph in which flow is 1.

This algorithm outputs a maximum matching.

Proof. Follows from (1) and (2) □

Remark 2.6 — Note that we can use any FF method, since the number of iteration is bounded by $|V|$. Since each iteration takes $O(|E|)$ time, we have a $O(|V||E|)$ algorithm.

We can run the FF method to find the maximum matching, as all capacities are integral values (all edges have capacity 1). If we have a flow, we will have a matching equal to $|f|$.

Remark 2.7 — This algorithm only works for bipartite graphs, but there is another polynomial time algorithm that can find the maximum matching of arbitrary graphs (albeit more complicated).

2.3 Perfect Matching in Bipartite Graphs

Definition 2.8. A **perfect matching** is a matching which "covers" all the vertices.

Theorem 2.9 (Hall's Theorem)
A bipartite graph $G(V,E)$ with bipartition (L,R) has a perfect matching if and only if:

1. $|L| = |R|$; and
2. $\forall A \subseteq L$, we have
$$|A| \leq |N(A)|.$$

These properties have been shown to be necessary conditions, but they are also sufficient conditions (using max-flow min-cut).

Proof. Suppose that G does not have a perfect matching and $|L| = |LR|$. Then we will prove that $\exists A \subseteq L$ such that $|A| > |N(A)|$, i.e. (2) will be violated. We will use the same reduction to the max-flow problem. Recall the flow network G' as before.

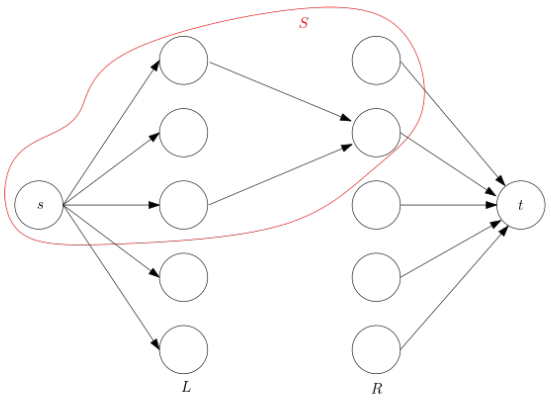

Figure 4: Example of a cut S

Consider the $s - t$ cut S of minimum capacity. The capacity of this cut is

$$|L - S| + |R \cap S|.$$

Note that there is no edges from $L \cap S$ that goes to $R \cap (V - S)$, since the capacity of the cup would be infinite, which would violate the max flow min cut thoerem, since the max flow is clearly finite. This means that

$$N(L \cap S) \subseteq R \cap S.$$

Since G does not have a perfect matching, if we let $|L| = |R| = n$

$$\text{Size of max matching} < n \implies \text{Value of max flow } < n \implies \text{Capacity of min cut } < n.$$

Note that

$$|L - S| + |R \cap S| < n = |L - S| + |L \cap S|,$$

which means

$$|L \cap S| > |R \cap S|.$$

$\square$

Index