Lecturer: Erik Demaine and David Karger

Scribes: Joungkeun Lim

Maximum flow algorithms

• augmenting path algorithm O(mf) time when f is max-flow value. For f$\leq$nU, running time is O(mnU) (pseudopolynomial)

• max-capacity augmenting path algorithm takes O(m^2 log f) = O(m^2lgnU) time (weak polynomial)

7.1 Scaling algorithm

•Scaling algorithm is to reduce general problem to 0-1 problem.

7.1.1 The idea

1. round-off flow problem.
2. solve the rounded part.
3. correcting round off errors.

-Gabow '85

-Dinic '73

7.1.2 algorithm

1. Start with all capacities 0
2. shift in high-order bit of each capacity, then each capacity is 0 or 1.
3. find max-flow in with given capacity.
4. repeat the general itertation for remaining bits till done.

7.1.3 General iteration for each phase

1. We have some max flow before shifting bit.

2. If we shift 1 bit of each capacity, then all capacities are doubled and have increment 0 or 1.

3. Double the previous maximum flow and add maximum residual flow to it.

4. Before the shifting bit, we had 0-cut in residual graph. So after shifting bit, the max-cut value is at most m.

5. Therefore maximum flow value is at most m in new residual graph.

6. Find maximum flow in the residual graph and add it to the doubled max-flow.

7.1.4 running time

In each phase, to find the max-flow in the residual graph takes O(m^2) time for each augmenting path takes O(m) time to find.

Capacity is at most O(lg U) bit number therefore O(lg U) phases total.

Total running time is O(m^2lg U).

Scaling algorithm is also weak polynomial algorithm.

Now we will show a Shortest-augmenting-path algorithm which is strongly polynomial algorithm.

7.2 Shortest augmenting path

•This algorithm is by augmenting shortest path (by edge nunmber) each time.

Definition 1 *d(x,y) is the shortest residual path length from x to y.*

Lemma 1 *Under shortest-augmenting-path algorithm, d(s,v) and d(v,t) are nondecreasing.*

Proof:

Suppose some d(s,v) decreases,

Consider v with decreased distance value.

It has previous point w in the shortest path from s to v such that d(s,w) has not decreased.

So path-augmenting process created edge (w,v) in residual graph.

It means that augmenting path sent flow from v to w.

Therefore before the path-augmenting process,w was farther from S than v

d(s,w)¿d(s,v) before the process. d(s,v) has decreased and d(s,w) has not decreased during the process.

Therefore d(s,w) has to be lareger than d(s,v) and it is contradiction.

■

Lemma 2 *The number of augmenting path is at most mn/2.*

Proof: Consider edge (v,w) saturated by path augmenting operation.

Before we can saturate the edge (v,w) again, we must "flip" the distances of v and w and push flow on (w,v).

So whenever (v,w) is saturated, d(s,v) or d(s,w) must increase by 2.

It implies that there can be only n/2 possible saturation of (v,w).

There is m edges in the graph, total saturation is at most mn/2 saturations i.e. mn/2 APs.

■

•Total running time for this algorithm is O(m^2n) time. It is strong polynomial algorithm.

7.3 Blocking Flows(Dinic)

7.3.1 The blocking flow algorithm for the unit-capacity graph

Definition 2 *When the vertices of a graph are arranged by the distances from sink, we call this graph as layered graph.*

Definition 3 *Admissible arc is the arc on shortest s-t paths. i.e. the arcs which traverse adjacent layers.*

The idea

1.simultaneously saturates more than one arc on every admissible-arc path.

2.key insight : when we saturate edges, we can delete the edges until future round.

3.increases augmenting path length.(never decreasing)

4.n iterations : n blocking flows suffice.

Blocking flow in O(m) time

Greedy algorithm

do DFS from s

if we reach t

-we augment the path.

-remove the edges in the path.

-continue DFS

if we get stuck

-retreat and burn the edges behind.

It takes O(m) time for each blocking flow steps.

Total running time

After each step, the minimum path length from s to t increases.

At most n blocking steps are possible.

Total time to find max-flow is O(mn) time

7.3.2 Better bound for unit capacity graph

Lemma 3 *We can find max flow in unit capacity graph by O($m^{\frac{3}{2}}$) time.*

Proof:

Suppose that we did k blocking flow steps before.

then d(s,t) $\geq$ k

We can decompose the residual max-flow into paths.

Each paths are edge-disjoint and contains at least k edges.

So the number of paths is at most m/k and the residual max-flow is at most m/k.

Blocking flow finds at least 1 path per phase, at most m/k blocking flow steps are needed.

Therefore overall running time is O($km + m^2/k$)=O($m^{\frac{3}{2}}$) when k=$\sqrt{m}$ ■

7.3.3 Bipartite matching

•Goal : Find maximum number of possible pairing in bipartite graph.

Definition 4 *Bipartite graph is the graph G s.t. V(G)=A $\cup$ B when A and B is disjoint vertex set in G and every edges in G have their one end vertex in A and another end vertex in B.*

Make starting point S which has edges to every vertices in A and make end point T which has edges to every vertices in B.

Make each edge to be a unit capacity edge.

We can easily see that maximum matching is equivalent to the maximum flow of new graph.

For the graph is unit graph, we can decompose the maximum flow into paths. Also each path is vertex disjoint.

Similarly to the argument before, total running time is O($km + mn/k$)=O($m\sqrt{n}$) when k=$\sqrt{n}$

7.3.4 The blocking flow algorithm for general capacity graph

The idea

1. We will combine blocking flow algorithm and scaling algorithm.
2. We will use scaling about the capacities of the edges.
3. For each scailing phase, use the blocking flow algorithm.
4. This will give better bound than the scaling algorithm.

Running time

Lemma 4 *We can bound each phase in O(mn) time by using blocking flow algorithm.*

Proof: In each phases, we need to consider 2 kinds of costs.

•cost for retreats

In each phase, retreat can happen at most the number of edges.

So the cost for retreat is O(m)

•cost for augments

At the start of phase, residual flow is at most m and Each augment decreases the flow by 1.

Thus during the phase, there is at most m augments.

Each augment takes O(n)time.

Therefore total augment cost is O(mn).

•Total cost per phase is O(mn).

■

•There is O(lg U) phases.

•Therefore total running time is O(mn lg U).