CS 684: Approximation and Network Algorithms	10/03/2001
Minimum cost flow algorithms	Scribe: Martin Pál

1 Quick review

Last time we discussed two minimum cost flow algorithms, both of which have a terrible worst case running time (depending linearly on the maximum demand).

Given a directed graph $G = (V, E)$, each node v has *demand* b_v, each edge e has *capacity* u_e and unit cost c_e. Our goal is to find a flow f that satisfies

$$\begin{array}{ll} \forall e \in E & 0 \leq f_e \leq u_e \\ \forall v \in V & f^{\text{in}}(v) - f^{\text{out}}(v) = b_v \end{array}$$

to minimize

$$\sum_{e \in E} c_e f_e.$$

We introduced *prices* $p(v)$ for $v \in V$. For a set of prices, we defined the *reduced cost* of an edge $e = (v, w)$ to be $c_e^p = c_e + p(v) - p(w)$. We say that a flow f and prices p are in equilibrium, iff for every edge e

$$\begin{array}{lll} f_e < u_e & \Rightarrow & c_e^p \geq 0 \\ f_e > 0 & \Rightarrow & c_e^p \leq 0. \end{array}$$

We defined a *residual graph* $G_f = (V, E_f)$ to be a graph with vertices V and edges

- $(v, w) \in E_f$ iff $e = (v, w) \in E$ and $f_e < u_e$.
- $(w, v) \in E_f$ iff $e = (v, w) \in E$ and $f_e > 0$

Fact 1.1 *Flow f and prices p are at equilibrium, iff for all edges $(v, w) \in E_f$ we have $c_{vw}^p \geq 0$.*

Fact 1.2 *If f and p are at equilibrium, then f is minimum cost.*

2 The primal dual algorithm

Our algorithm will maintain a feasible "pre-flow" f and a set of prices p such that f and p are at equilibrium. By pre-flow we mean that f satisfies all capacity and non-negativity constraints, but not necessarily the flow conservation constraints. We repeatedly modify f and p, making f closer and closer to a feasible flow (i.e. satisfying flow conservation). See fig. 1.

Last time we proved that

start with a flow f and prices p at equilibrium
while f not feasible
  if a path P exists in G_f using zero cost edges (subject to costs c_e^p)
    from any excess node to any deficit node, augment f along P
  else
    let $W = \{v \mid v$ reachable from some excess node on a path
            consisting of edges with $c_e^p = 0\}$
    set $p(v) = p(v)$ if $v \in W$
        $p(v) = p(v) + \Delta$ if $v \notin W$
    where $\Delta = \min\{c_e^p \mid e = (u,v),\ u \in W,\ v \notin W\}$

Figure 1: Primal dual algorithm

1. At the end of each iteration, f and p are at equilibruim.
2. Upon termination, f is a flow.

These two facts together with fact 1.2 allow us to conclude that the resulting flow is a minimum cost flow. Unfortunately, the running time of the algorithm in fig. 1 is prohibitively large – it may be proportional to the sum of capacities of all edges. And even worse, if we drop the assumption of integral costs and capacities, on pathological instnces it may not even terminate. In this lecture we present two improvements to this algorithm. We still keep the assumption of integral costs and capacities.

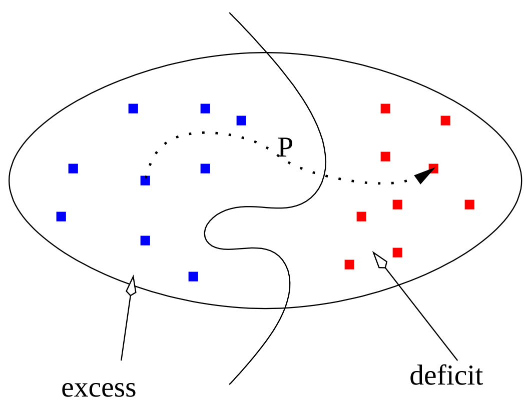

Figure 2:

3 Shortest paths

Let's look at what the primal dual algorithm does in a slightly different way. Suppose there is no zero cost path any excess vertex to any deficit vertex. Let W be the set of all vertices reachable from some excess vertex by a zero cost path. In each iteration, the algorithm increases the costs of vertices in $V - W$ to make the cheapest edge (call it $e = (v, w)$ for now) in the cut $(W, V - W)$ have zero cost. This step makes w, the endpoint of e reachable by a zero cost path, adding it to the set W. Compare with a run of the Dijkstra's shortest path algorithm. It also maintains a set W' of processed vertices. In each step, it examines the edges leaving W' and adds to W' the closest vertex not yet processed. Essentially, the primal dual algorithm is simulating a shortest path computation in the residual graph using the reduced prices c^p, but in an inefficient way. Our next improvement will be to do this shortest path computation explicitly, thus merging $O(|V|)$ iterations of our algorithm into a single iteration. The resulting algorithm is in fig. 3.

```
start with a flow f and prices p at equilibrium
while f not feasible
  let S be the set of vertices with excess
      T be the set of vertices with deficit
  find a minimum cost path P from a node in S to a node in T
  in the residual graph G_f subj. to costs c^p
  for v ∈ V, let d(v) be the cost of the minimum cost path
      from any node s ∈ S to v subj. to costs c^p
  for v ∈ V, set p(v) := p(v) + d(v).
  send as much flow along P as you can (note that P is now zero cost)
```

Figure 3: Faster primal dual algorithm

Note. Given a flow, what is the cost of shipping one more unit from an excess node to a deficit node? Clearly, it is the cost of the minimum cost path from some excess vertex to some deficit vertex in the residual graph, but subject to the nonreduced costs c. Thus it seems that instead of looking for shortest paths using reduced costs, we should use the true costs. However, it turns out that a shortest path under one cost is also a shortest path under the other cost. [1]

Since this is just a faster implementation of the first algorithm, all properties stated above (e.g. f and p always at equilibrium, upon termination f satisfies flow conservation) apply to the improved algorithm as well. As for running time: in every iteration we augment the flow by at least one unit, thus we stop after no more than $\sum_{e \in E} u_e$ iterations. Each iteration involves one one-to-all shortest path computation, thus the overall time is (Dijkstra) $\times$ $(\sum_{e \in E} u_e)$ (where (Dijkstra) stands for the running time of the shortest path computation, currently $O(m + n \log n)$).

[1] For all excess vertices v, $p(v) = 0$ and for all deficit vertices w, $p(w) = p_{\mathrm{def}}$, for some constant p_{def}. Thus the cost of any vw path subj. to c_e differs from its cost subj. to c_e^p only by the constant p_{def}.

4 Scaling

Our next goal will be to get rid of the unpleasant fact that the running time depends linearly on $(\sum_{e\in E} u_e)$. The trouble stems from the fact that in every iteration, we may be able to push only a single unit of flow along the minimum cost path. Thus we would like to introduce a Δ-scaling step. The idea is, that in a particular iteration, we are interested only in paths along which we can ship at least Δ units of flow.

The first attempt would be to focus only on excess nodes with excess at least Δ and deficit nodes with deficit at least Δ, and paths between them using only edges with residual capacity at least Δ. A moment's thought shows that we cannot simply augment the flow along such a path without messing up the equilibrium. As a workaround, we introduce the transshipment problem.

4.1 The transshipment problem

The transshipment problem is a simplification of the minimum cost flow problem. Namely, it is a min-cost flow problem satisfying the following constraints:

1. For all $e \in E$, $c_e \geq 0$.
2. For all $e \in E$, $u_e = \infty$.

It is easy to see that the first assumption can be made without loss of generality. In fact, in the next lecture we shall see that any min-cost flow problem can be effectively reduced to a transshipment problem. Thus from now on, it suffices to focus on solving the transshipment problem.

We will need one more technical assumption. We require that the graph is complete, i.e. there is an edge between any pair of vertices v, w. (One can achieve this by adding really expensive edges. See the next lecture.)

The algorithm is stated in fig. 4.

Fact 4.1 *f remains feasible throughout the algorithm, i.e. at all times and for all edges, $f(e) \geq 0$.*

Proof. The flow is certainly feasible at the beginning of the algorithm. Consider a phase for some particular Δ. In this phase, as well as in all previous phases, the amount of flow on every edge has been changed by only multiples of Δ, thus all flow values are multiples of Δ. This means that $f(e) > 0$ implies $f(e) \geq \Delta$. Since in each augment step we push back only on edges with positive flow and always push only Δ units, the flow in any edge never gets negative. ∎

Fact 4.2 *Upon termination of the algorithm, f is a valid flow (i.e. nonnegative and satisfies flow conservation).*

```
start with p(v) = 0, f(e) = 0
let Δ = 2^k, where k is the largest integer s.t. 2^k ≤ max b_v
repeat
  while a path exists
    let S(Δ) be the set of vertices with excess at least Δ
        T(Δ) be the set of vertices with deficit at least Δ
    find a minimum cost path P from S(Δ) to T(Δ)
      in the residual graph G_f subj. to costs c^p
    for v ∈ V, let d(v) be the cost of the minimum cost path
        from any node s ∈ S to v subj. to costs c^p
    for v ∈ V, set p(v) := p(v) + d(v).
    send Δ units of flow along P
  Δ := Δ/2
until Δ < 1
```

Figure 4: Scaling algorithm

Proof. In fact 4.1 we established that f is non-negative throughout the algorithm. It is also easy to check that the prices p and the flow f stay at equilibrium. This is due to the fact that we always augment along the shortest path. Finally, it is easy to see that f satisfies flow conservation. To see this, consider the last iteration, when $\Delta = 1$. The while loop terminates only if there is no path from any excess vertex to any demand vertex in the residual graph. Since the original graph was by assumption complete, this can happen only if there are no excess (or deficit) vertices. ■

Runtime: $(\log_2 max_v b_v)$(# of paths in one Δ-phase)(Dijkstra).

Lemma 4.3 *The number of paths for each particular value of Δ is at most $|V| - 1$.*

Proof. Consider the end of the (2Δ) phase. The (2Δ) phase ends when there is no path from $S(2\Delta)$ to $T(2\Delta)$. By our assumption, the graph is complete (and all capacities are infinite, hence G_f is also complete), hence the only way for this to happen is either $S(2\Delta) = \emptyset$ or $T(2\Delta) = \emptyset$. Consider the case when $S(2\Delta) = \emptyset$ (case $T(2\Delta) = \emptyset$ is similar). At the beginning of the Δ phase, $S(\Delta)$ contains at most $|V| - 1$ vertices, and from each of those vertices, at most one augmenting path can originate in that phase. Thus there are at most $|V| - 1$ augmentations in the phase. ■

Hence, we ended up (modulo the reduction of min-cost flows to transshipment problems) with the following theorem.

Theorem 4.4 *There is a minimum cost flow algorithm that runs in time $(\log_2 max_v b_v)(|V| - 1)$(Dijkstra).*