Washington University in St. Louis

Engineering

# All-Pairs Shortest Paths and Faster Min-Cost Flows

Jon Turner
Computer Science & Engineering
Washington University

www.arl.wustl.edu/~jst

# Floyd's Algorithm

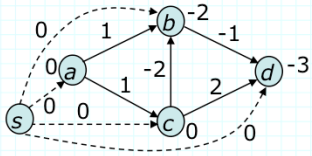
- Dynamic programming algorithm for all-pairs problem
  - » initialize *dist*(*u*,*v*) to *length*(*u*,*v*) if there is an edge from *u* to *v*, zero if *u*=*v* and ∞ otherwise
  - » apply the following step to each vertex *v*

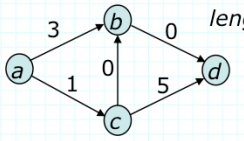
    *Labeling Step*: if *dist*(*v*,*v*)<0, abort: there is a negative cycle; otherwise for each pair *u*,*w* with *dist*(*u*,*w*)>*dist*(*u*,*v*)+*dist*(*v*,*w*), replace *dist*(*u*,*w*) by *dist*(*u*,*v*)+*dist*(*v*,*w*)
- Store path information in separate *midpoint* mapping
  - » values initialized to *null* and *midpoint*(*u,w*) assigned value *v* whenever *dist*(*u,w*) is assigned the value *dist*(*u*,*v*)+*dist*(*v*,*w*)
  - » construct paths with a recursive procedure that uses *midpoint*
- Runs in $\Theta(n^3)$ time regardless of number of edges
  - » also requires $\Theta(n^2)$ space; for very dense graphs, its simplicity makes it faster than Dijkstra's algorithm

2

# Dijkstra's Algorithm for All Pairs

- For non-negative edge lengths use Dijkstra's algorithm to solve all pairs problem in $O(mn+n^2 \log n)$ time
  - » can get same time bound with negative edge lengths by first transforming edge lengths
  - » first, add vertex *s* to *G* and zero length edge [*s*,*v*] to every *v*




*length*(*a*,*d*) = *length'*(*a*,*d*)
+*distance*(*d*)
–*distance*(*a*)
= 1+(–3)–0=–2

  - » use breadth-first algorithm to compute *distance*(*v*)=length of shortest path from *s* to *v* in augmented graph
  - » apply Dijkstra's algorithm to *G* with a new length function, *length'*(*u*,*v*)=*length*(*u*,*v*)+*distance*(*u*)–*distance*(*v*)
  - » to get path length in original graph, apply reverse transform

3

# Validity of Edge Length Transform

- *Theorem 7.8*. For any edge $[u,v]$, $length'(u,v) \geq 0$ and for every path $p$ from a vertex $x$ to a vertex $y$, $length'(p) = length(p) + distance(x) - distance(y)$.

  *Proof*. First part of the theorem is implied by Theorem 7.3

  Second part is proved by induction on number of edges in $p$

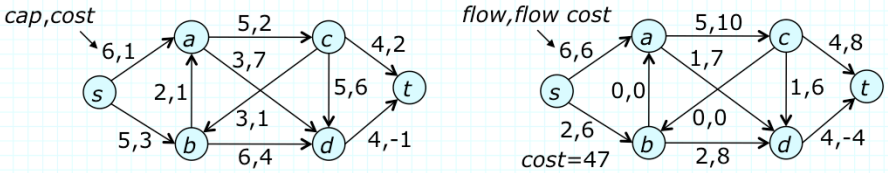
  Basis: $|p|=1$. Immediate from the definition of *length'*

  Induction: $|p|=k+1$. Let $q$ be path containing first $k$ edges of $p$ and let $z$ be last vertex on $q$. Then,

$$\begin{aligned} length'(p) &= length'(q) + length'(z,y) \\ &= length(q) + distance(x) - distance(z) \\ &\quad + length(z,y) + distance(z) - distance(y) \\ &= length(p) + distance(x) - distance(y) \end{aligned}$$

  ■

# Minimum Cost Flows




- Max flow can be generalized by adding edge costs
  - costs are skew symmetric; $cost(v,w)=-cost(w,v)$
  - for flow $f$, $cost(f)=\Sigma_{f(v,w)>0} cost(v,w)f(v,w)=\Sigma_{v,w}\ cost(v,w)f(v,w)/2$
  - cost of path is defined as sum of edge costs and residual graph is defined as before; $cost(v,w)$ in $R$ is same as in $G$
  - A flow $f$ has *minimum cost* if there is no flow $g$ with $|g|=|f|$ that has lower cost than $f$
  - min-cost, max-flow problem: seek min cost flow with max value
- General strategies for finding minimum cost flows
  - cost reduction – add flow to negative cost cycles
  - min cost augmentation – add flow to min cost augmenting paths

# Min-Cost Augmenting Path Algorithm

- Find min cost flow using augmenting path algorithm and selecting min cost paths
  - » $O$((# of steps)(time to find a min cost augmenting path))
  - » for integer capacities, number of steps is at most $|f^*|$
  - » since residual graphs may have negative cost edges, we must use an algorithm that can handle negative cost edges
  - » using breadth-first scanning, running time is $O(mn|f^*|)$
- Can improve by transforming edge costs to be $\geq 0$
  - » transformed edge costs preserve relative lengths of paths, while eliminating negative edges
  - » this allows use of Dijkstra's algorithm to find shortest paths, reducing running time to $(mn+|f^*|(m+n\log n))$
  - » as flow is added, edge costs transformations must be updated to reflect changes in residual graph

6

# Maintaining Transformed Edge Costs

- Let $cost_0(u,v) = cost(u,v)$
  - » construct an augmented graph $G'$ from $G$ by adding new vertex $s'$ with zero cost edge to each vertex in $G$
  - » compute $dist_0(u)$=length of shortest path from $s'$ to $u$ in $G'$ (using breadth-first scanning)
  - » for all edges $[u,v]$ in $G$, define

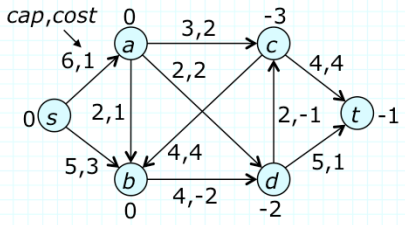
    $cost_1(u,v) = cost_0(u,v) + dist_0(u) - dist_0(v) \quad cost_1(v,u) = -cost_1(u,v)$
- Repeat the following steps until no augmenting path
  - » let $dist_k(u)$ be length of shortest path from $s$ to $u$ in current residual graph, where cost of an edge $[u,v]$ in residual graph is given by $cost_k(u,v)$ (compute $dist_k$ using Dijkstra's algorithm)
  - » saturate selected path; for all edges $[u,v]$ in $G$ compute,

    $$cost_{k+1}(u,v) = cost_k(u,v) + dist_k(u) - dist_k(v)$$
  - » will show that new costs are non-negative and shortest paths in residual graph are the same for new cost function as for old

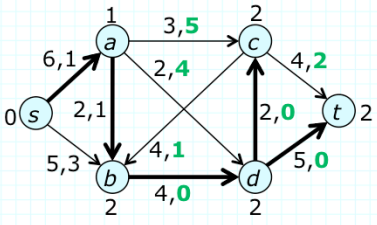
7

# Example

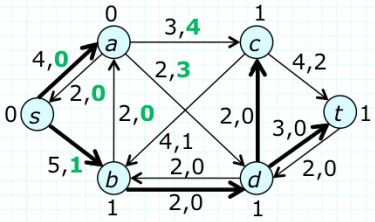
*Original Network with $dist_0$*



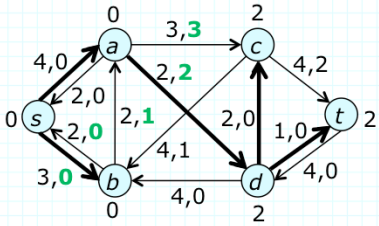

*Step 1. showing $cost_1$ $dist_1$, spt and path*




*Step 2. showing $cost_2$ $dist_2$, spt and path*




*Step 3. showing $cost_3$ $dist_3$, spt and path*




8

# Correctness of Algorithm

- *Invariant*: if $R_k$ is residual graph at start of step $k$,
  - » $cost_k(u,v) \geq 0$ for all $(u,v)$ in $R_k$
  - » for any $s$-$x$ path $p$ in $R_k$

    $$cost_k(p) = cost_0(p) - (dist_0(x) + \cdots + dist_{k-1}(x))$$

    so, a min-cost $s$-$x$ path with respect to $cost_k$ is a min-cost path with respect to the original costs

*Proof*. For $k=1$, claim follows from Theorem 7.8; so assume that in $R_{k-1}$, $cost_{k-1}(u,v) \geq 0$, $cost_{k-1}(p) = cost_0(p) - (dist_0(x) + \cdots + dist_{k-2}(x))$ for all $s$-$x$ paths, $p$

In step $k-1$, $dist_{k-1}(x)$ is computed using $cost_{k-1}$ and min cost path is selected using these distances; for any edge $(u,v)$ in $R_{k-1}$,

$$cost_{k-1}(u,v) \geq dist_{k-1}(v) - dist_{k-1}(u)$$

and for $(u,v)$ on the path,

$$cost_{k-1}(u,v) = dist_{k-1}(v) - dist_{k-1}(u)$$

9

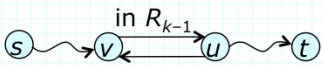
# Correctness of Algorithm

After saturating path, we compute $cost_k(u,v)$ for all $(u,v)$ using

$$cost_k(u,v)=cost_{k-1}(u,v)+dist_{k-1}(u)-dist_{k-1}(v)$$
$$=cost_{k-1}(u,v)-(dist_{k-1}(v)-dist_{k-1}(u))$$

For any edge $(u,v)$ in $R_{k-1}$, the right side of the expression is $\geq 0$, so in this case, $cost_k(u,v)\geq 0$

Suppose $(u,v)$ is in an edge $R_k$, but not in $R_{k-1}$; in this case, it is the reverse of some edge on the augmenting path; since the right side of the above expression is zero for augmenting path edges, skew symmetry implies $cost_k(u,v)=0$




*augmenting path*

Last part follows from induction hypothesis and definition of $cost_k$

$$cost_k(p)=cost_{k-1}(p)+dist_{k-1}(s)-dist_{k-1}(x)=cost_{k-1}(p)-dist_{k-1}(x)$$
$$=(cost_0(p)-(dist_0(x)+\cdots+dist_{k-2}(x)))-dist_{k-1}(x)$$

■

10

# Better Min Cost Augmentation

- Recap of version using transformed edge costs
  - » transform edge costs to make them non-negative

    $cost_1(u,v)=cost(u,v)+dist(u)-dist(v)$  $cost_1(v,u)=-cost_1(u,v)$
  - » repeat the following steps until no augmenting path
    - let $dist_k(u)$ be length of shortest path from $s$ to $u$ in current residual graph, where cost of an edge $[u,v]$ in residual graph is given by $cost_k(u,v)$ (compute $dist_k$ using Dijkstra's algorithm)
    - saturate selected path; for all edges $[u,v]$ in $G$ compute,

      $$cost_{k+1}(u,v)=cost_k(u,v)+dist_k(u)-dist_k(v)$$
- Alternate approach to cost transform
  - » define current costs in terms of vertex labels
  - » transform labels to effectively transform costs
- Can also use vertex labels for *scaling algorithm*
  - » allows use of higher capacity augmenting paths

11

# Using Vertex Labels to Transform Costs

- Cost transform for least-cost augmenting path algorithm

$$cost_{k+1}(u,v)=cost_k(u,v)+dist_k(u)-dist_k(v)$$

and consequently,

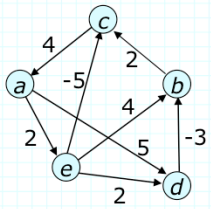
$$cost_{k+1}(u,v)=cost_0(u,v)+\underbrace{(dist_0(u)+\cdots+dist_k(u))}_{\lambda(u)}-\underbrace{(dist_0(v)+\cdots+dist_k(v))}_{\lambda(v)}$$

- This enables an alternate viewpoint
  - »define *current cost* in terms of original cost and vertex labels $\lambda$
  - »transform labels as flow changes to effectively transform edges
- Revised algorithm
  - »initialize $\lambda(u)$=distance from *s'* to *u* in *augmented* graph
  - »define *current cost*=$cost_\lambda(u,v)=cost(u,v)+\lambda(u)-\lambda(v)$
  - »during each augmenting step,
    - let $dist_\lambda(u)$ be length of shortest path (w.r.t. $cost_\lambda$) from *s* to *u* in current residual graph; saturate shortest *s*-*t* path
    - for each vertex *u*, let $\lambda(u)=\lambda(u)+dist_\lambda(u)$

12

# Exercises

1. The figure below shows the state in the execution of Floyd's algorithm for the all-pairs shortest path problem, after the first three labeling steps. The distance array is shown at right.




| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0 | ∞ | ∞ | 5 | 2 |
| b | 6 | 0 | 2 | 11 | 8 |
| c | 4 | ∞ | 0 | 9 | 6 |
| d | 3 | -3 | -1 | 0 | 5 |
| e | -1 | 4 | -5 | 2 | 0 |

Show the distance array after each of the last two labeling steps.

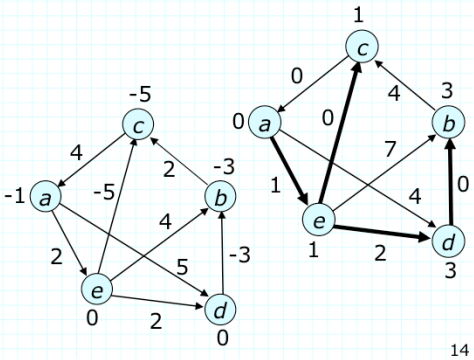
| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0 | **2** | **4** | 5 | 2 |
| b | 6 | 0 | 2 | 11 | 8 |
| c | 4 | **6** | 0 | 9 | 6 |
| d | 3 | -3 | -1 | 0 | 5 |
| e | -1 | **-1** | -5 | 2 | 0 |

| | a | b | c | d | e |
|---|---|---|---|---|---|
| a | 0 | **1** | **-3** | **4** | 2 |
| b | 6 | 0 | 2 | **10** | 8 |
| c | 4 | **5** | 0 | **8** | 6 |
| d | 3 | -3 | -1 | 0 | 5 |
| e | -1 | -1 | -5 | 2 | 0 |

2. Apply the edge transformation described in the notes to the graph in the previous problem to obtain a set of non-negative edge lengths. Use Dijkstra's algorithm to compute shortest path trees rooted at *a*. Compare the distances from *a* to all other vertices in your computed shortest path tree to those obtained in the previous problem. Explain the correspondence between the two sets of distances. Compile the program for Dijkstra's algorithm with transformed edge lengths that you will find on the web site. Familiarize yourself with the program and use it to compute shortest path distances for several graphs.

*The left hand figure below shows the graph with the original edge weights and distances from a source vertex s with a zero length edge to each original vertex (s and its edges are not shown).*

*The right hand figure shows the graph with the transformed edge costs and the distances from a computed using these edge costs. The distances computed in the last problem from a to the other four vertices are 1, -3, 4 and 2. The costs shown at right above can be obtained from these costs by adding the label shown at left above for a (-1) and subtracting the labels for b, c, d and e (these labels are -3, -5, 0 and 0).*



3. Suppose we assign labels $\lambda(u)$ to each vertex of a directed graph with a *length* function defined on its edges. Let $length'(u,v)=length(u,v)+(\lambda(u)-\lambda(v))$. For a path $p=u_1,u_2,\ldots,u_r$, let $length(p)=length(u_1,u_2)+length(u_1,u_2)+\ldots+length(u_{r-1},u_r)$ and let $length'(p)$ be defined similarly. Show that for two paths $p$ and $q$ joining the same pair of vertices, that $length(p)\le length(q)$ if and only if $length'(p)\le length'(q)$.

*For any such path p,*

$$\begin{aligned} length'(p) &= length(u_1,u_2)+length(u_2,u_3)+\ldots+length(u_{r-1},u_r) \\ &\quad +(\lambda(u_1)-\lambda(u_2))+(\lambda(u_2)-\lambda(u_3))+\ldots+(\lambda(u_{r-1})-\lambda(u_r)) \\ &= length(p)+\lambda(u_1)-\lambda(u_r) \end{aligned}$$

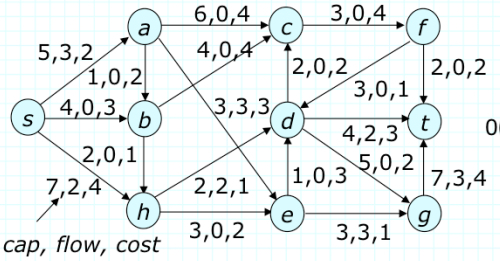
*So, the following lines are equivalent.*

$$length'(p) \le length'(q)$$

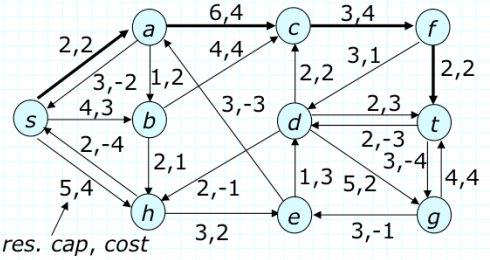
$$length(p)+\lambda(u_1)-\lambda(u_r) \le length(q)+\lambda(u_1)-\lambda(u_r)$$

$$length(p) \le length(q)$$

4. The figures below show an instance of the minimum cost, maximum flow problem, with some flow, and the residual graph corresponding to this flow.

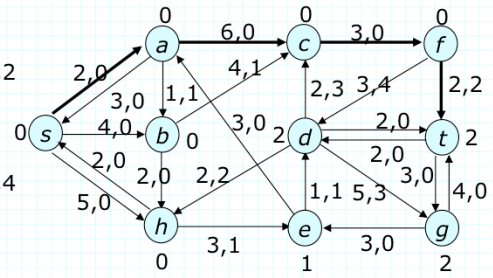



*cap, flow, cost*




*res. cap, cost*

Transform the edge costs in the residual graph to make them non-negative. Also, show the distances from *s* using the transformed costs.



Find the shortest augmenting path using the transformed costs, add flow to that path and show the new residual graph with the new transformed edge costs and new distance values. Identify the shortest augmenting path in this residual graph.