CS 341: Algorithms

Lecture 18: Applications of flows and cuts

Éric Schost

based on lecture notes by many other CS341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

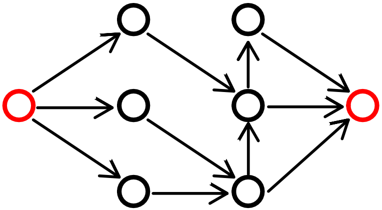
Fall 2024

Edge disjoint paths

Edge disjoint paths

Problem:

- **input:** a directed graph, with source $\boldsymbol{s}$ and sink $\boldsymbol{t}$
- **output:** the maximum number of **edge-disjoint paths** $\boldsymbol{s} \to \boldsymbol{t}$



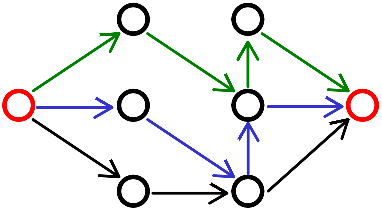
Algorithm

- all edges get capacity 1
- Ford-Fulkerson outputs a 0/1 flow
  (0/1 flow = flow which only takes values 0 and 1)
  proof: flow values are integers ≤ 1

Edge disjoint paths

Problem:

- **input:** a directed graph, with source s and sink t
- **output:** the maximum number of **edge-disjoint paths** $s \to t$



Algorithm

- all edges get capacity 1
- Ford-Fulkerson outputs a 0/1 flow

  (0/1 flow = flow which only takes values 0 and 1)

  proof: flow values are integers ≤ 1

Edge disjoint paths

Claim

There is a $0/1$ flow with value k **iff** there are k edge-disjoint paths $s \to t$ in G (using the edges with flow 1)

Paths $\to$ flow: set flow to 1 on edges covered by paths

Flow $\to$ paths by induction: given a $0/1$ flow f with N **flow-1 edges**, can find $\mathsf{Val}(f)$ edge-disjoint paths $s \to t$ in G using only flow-1 edges

- OK for $N = 0$
- induction assumption: true for $0, 1, \ldots, N-1$
- start from s, follow edges with flow 1
- **if we loop**, set flow to 0 on the cycle $\to$ flow with same value and smaller N
- **if we get to t**, set flow to 0 on the path $\to$ flow with value $\mathsf{Val}(f) - 1$ and smaller N

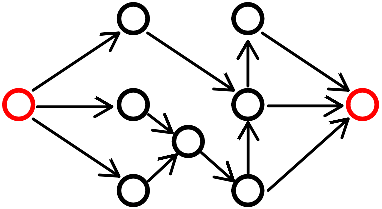
Remarks

1. Runtime:

- value of the flow is at most $\boldsymbol{n}$ so runtime $\boldsymbol{O(mn)}$ if no isolated vertex, $\boldsymbol{O((m+n)n)}$ otherwise

2. Edge version of Menger's theorem (from max flow = min cut)

- max number of edge-disjoint paths $s \to t$ = min number of edges to remove if we want to ensure there is no path $s \to t$
- exercise: fill in the details



Remarks

1. Runtime:

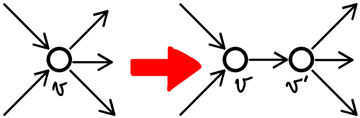
- value of the flow is at most $\boldsymbol{n}$ so runtime $\boldsymbol{O(mn)}$ if no isolated vertex, $\boldsymbol{O((m+n)n)}$ otherwise

2. Edge version of Menger's theorem (from max flow = min cut)

- max number of edge-disjoint paths $s \to t$ = min number of edges to remove if we want to ensure there is no path $s \to t$
- exercise: fill in the details

3. Vertex-disjoint paths

- transform the graph and find edge-disjoint paths



- $n' \le 2n$, $m' \le m + n$

Bipartite matching

Example

Context: students and professors:

- each professor offers **one** internship, but not all students may be eligible;
- each student wants **one** internship, and is ready to go for any of them.

Example: 3 students and 3 professors.

$$
\begin{aligned}
&P_1 \text{ will only consider } S_1 \text{ and } S_2\\
&P_2 \text{ will only consider } S_2\\
&P_3 \text{ will only consider } S_1 \text{ and } S_3
\end{aligned}
$$

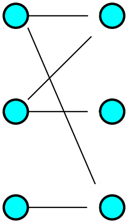
How to find the best matching?

Bipartite graphs

Definition

a (symmetric) graph G whose vertices are split into two groups S_i and P_j, with no edge between S_i's, or between P_j's.

Can be turned into a weighted directed graph G', adding a source and a sink

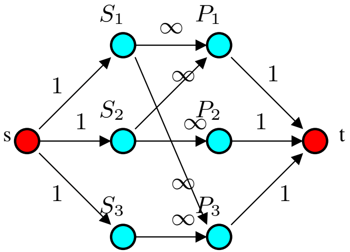


Bipartite graphs

Definition

a (symmetric) graph G whose vertices are split into two groups S_i and P_j, with no edge between S_i's, or between P_j's.

Can be turned into a weighted directed graph G', adding a source and a sink



In general

Bipartite matching:

- a set of r edges (between the S_i and the P_j) with no common vertex, *i.e.*,
- vertices $S_{m_1}, \ldots, S_{m_r}$ and $P_{\ell_1}, \ldots, P_{\ell_r}$ such that S_{m_i} is connected to P_{ℓ_i}

How to find one: set up a flow problem as before

- a matching of size $\boldsymbol{r}$ gives a $0/1$ flow of value $\boldsymbol{r}$
- a $0/1$ flow of value $\boldsymbol{r}$ gives a matching of size $\boldsymbol{r}$ as in edge-disjoint paths (using 1s instead of ∞s, this is edge-disjoint paths)
- Ford-Fulkerson's algorithm returns a $0/1$ flow

Runtime: $n + 2$ vertices, $m + n$ edges so $O((m + n)n)$

Minimum vertex cover

Definition

- $G = (V, E)$ is a **symmetric graph**
- a **vertex cover** C is a subset of **vertices** that contains an extremity of **every edge**
- C vertex cover iff $V - C$ **independent set** (no vertices in it are directly connected)
- want C as small as possible

Complexity

- **trees:** dynamic programming (for independent sets)
- **bipartite graphs:** min cut
- **general graphs:** NP-hard

König's theorem

Thm

in a **bipartite graph**, maximum size of a matching $=$ minimum size of a vertex cover

take a matching of maximum size $\boldsymbol{r}$

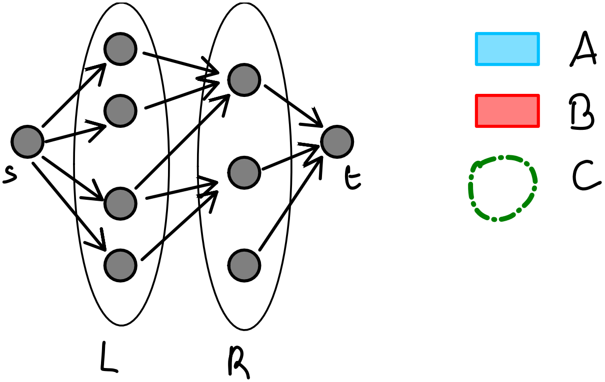
All vertex covers have size at least r

- need at least $\boldsymbol{r}$ vertices to cover these edges

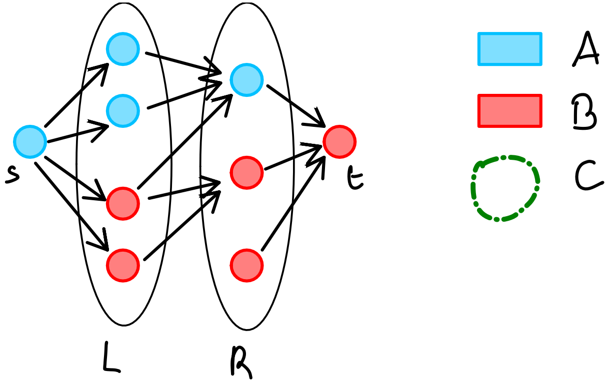
Proof of $=$

- max flow has value $\boldsymbol{r}$ in G'
- max flow $=$ min cut: there is a cut $A, B = V - A$ of capacity $\boldsymbol{r}$ in G'
- use it to find a vertex cover of size $\boldsymbol{r}$

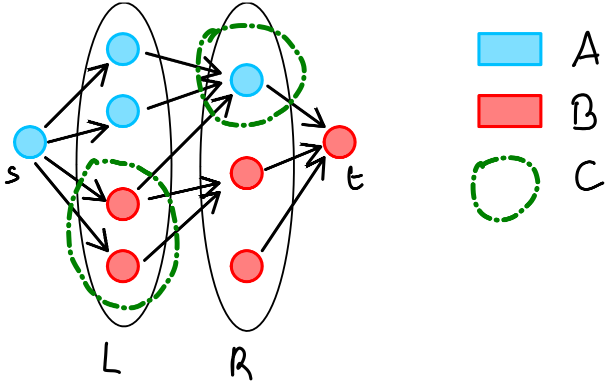
König's theorem



König's theorem

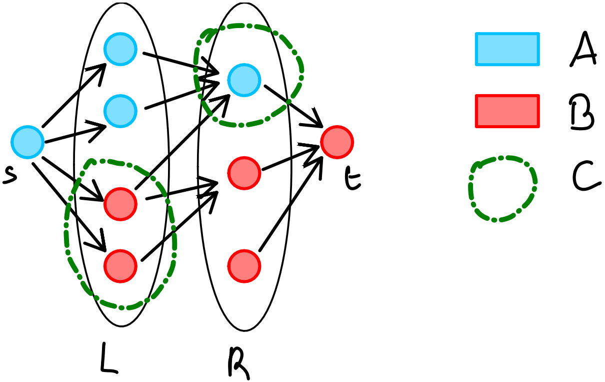


König's theorem



Define $\boldsymbol{C} = (\boldsymbol{L} \cap \boldsymbol{B}) \ \cup \ (\boldsymbol{R} \cap \boldsymbol{A})$

König's theorem



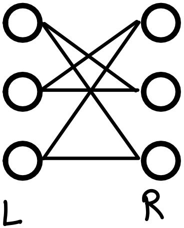
Define $\boldsymbol{C = (L \cap B) \ \cup \ (R \cap A)}$

- there are no edges **blue in $\boldsymbol{L}$** $\to$ **red in $\boldsymbol{R}$** (they have infinite capacity)
- so C is a vertex cover in G
- and $r = c(A) = |(\text{edges } s \to C)| + |(\text{edges } C \to t)| = |C|$

Special case: d-regular bipartite graphs (bonus)

d-regular:

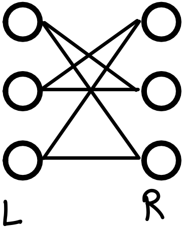
- all vertices have d incident edges
- if G is also **bipartite**, then $\boldsymbol{m = d|L| = d|R|}$ and so $\boldsymbol{|L| = |R|}$



Special case: d-regular bipartite graphs (bonus)

d-regular:

- all vertices have d incident edges
- if G is also **bipartite**, then $\boldsymbol{m = d|L| = d|R|}$ and so $\boldsymbol{|L| = |R|}$



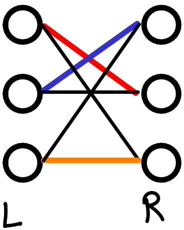
Claim

In a d-regular bipartite graph, there is a **perfect matching** (all vertices are matched)

Special case: d-regular bipartite graphs (bonus)

d-regular:

- all vertices have d incident edges
- if G is also **bipartite**, then $\boldsymbol{m} = \boldsymbol{d}|\boldsymbol{L}| = \boldsymbol{d}|\boldsymbol{R}|$ and so $|\boldsymbol{L}| = |\boldsymbol{R}|$



Claim

In a d-regular bipartite graph, there is a **perfect matching** (all vertices are matched)

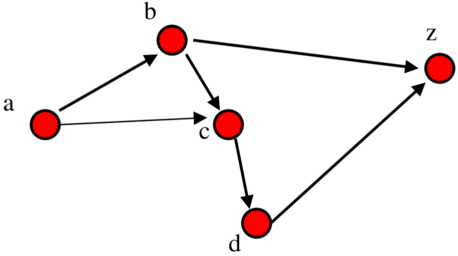
Proof

König's theorem: maximum size of a matching = minimum size of a vertex cover

- call $r = |L| = |R|$
  - 1 vertex covers d edges
  - 2 vertices cover at most $2d$ edges
  - $\ldots$
  - $r - 1$ vertices cover at most $(r-1)d$ edges
- we have **rd** edges, so $r - 1$ vertices are not sufficient
- so r vertices are necessary, and also sufficient (easy), and min vertex cover has size r

A shipping problem

Suppliers $a, b, c, d, \ldots$ want to send stuff to a buyer z, through a network



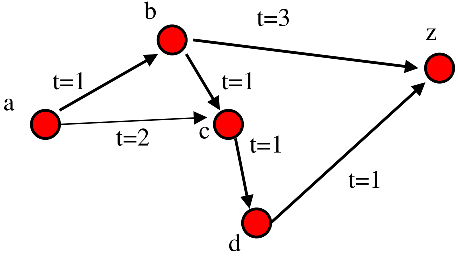
Initially, there are $s_a, s_b, \ldots$ units of stuff in $a, b, \ldots$.

We take **shipping time** into account: a maximum of stuff should arrive before $t = T$.

- the amount of stuff that can leave a toward b **per time unit** is $c_{(a,b)}$, and the same for $c_{(a,c)}, \ldots$
- the **traversal time** from a to b is $t_{(a,b)}$, and the same for $t_{(a,c)}, \ldots$

A shipping problem

Suppliers $a, b, c, d, \ldots$ want to send stuff to a buyer z, through a network



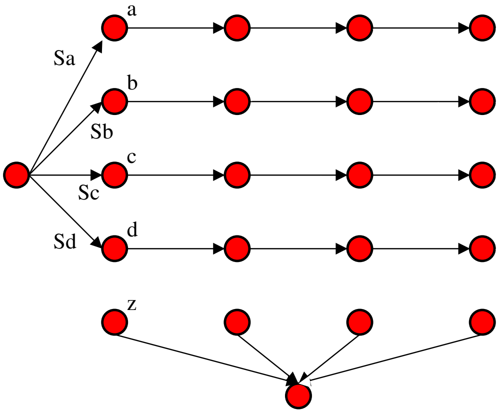
Initially, there are $s_a, s_b, \ldots$ units of stuff in $a, b, \ldots$.

We take **shipping time** into account: a maximum of stuff should arrive before $t = T$.

- the amount of stuff that can leave a toward b **per time unit** is $c_{(a,b)}$, and the same for $c_{(a,c)}, \ldots$
- the **traversal time** from a to b is $t_{(a,b)}$, and the same for $t_{(a,c)}, \ldots$

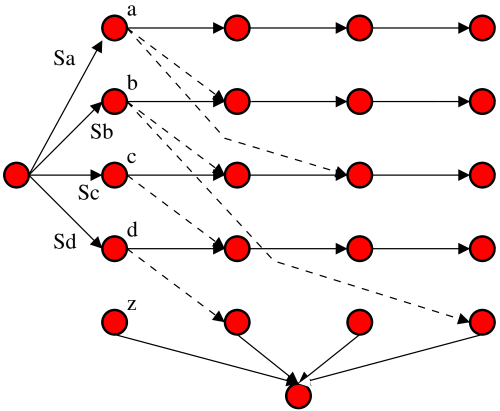
A shipping problem

- previous graph is copied $T+1$ times: one copy for each time step
- edges are arranged to match the time contraints
- capacities are the $c_{(u,v)}$
- super-source and super-sink



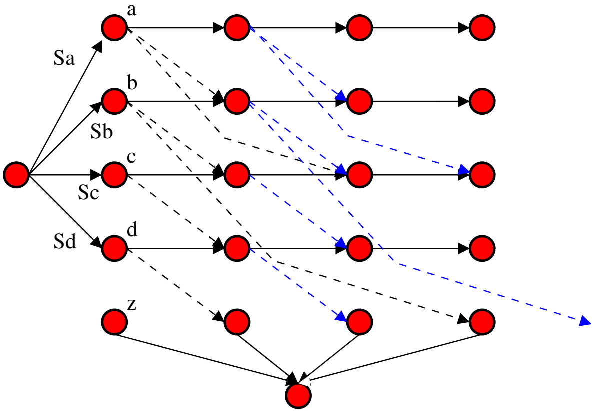
A shipping problem

- previous graph is copied $T + 1$ times: one copy for each time step
- edges are arranged to match the time contraints
- capacities are the $c_{(u,v)}$
- super-source and super-sink



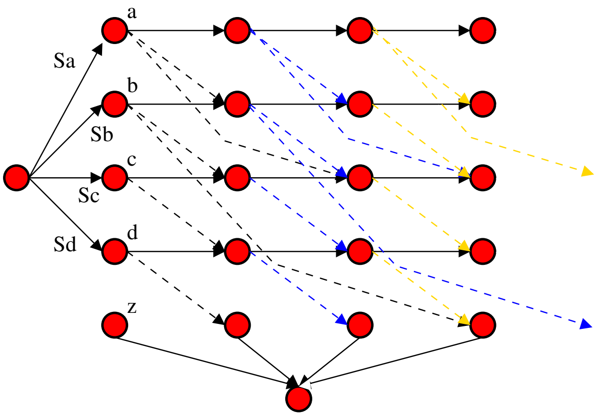
A shipping problem

- previous graph is copied $T + 1$ times: one copy for each time step
- edges are arranged to match the time contraints
- capacities are the $c_{(u,v)}$
- super-source and super-sink



A shipping problem

- previous graph is copied $T+1$ times: one copy for each time step
- edges are arranged to match the time contraints
- capacities are the $c_{(u,v)}$
- super-source and super-sink



A shipping problem

- previous graph is copied $T+1$ times: one copy for each time step
- edges are arranged to match the time contraints
- capacities are the $c_{(u,v)}$
- super-source and super-sink

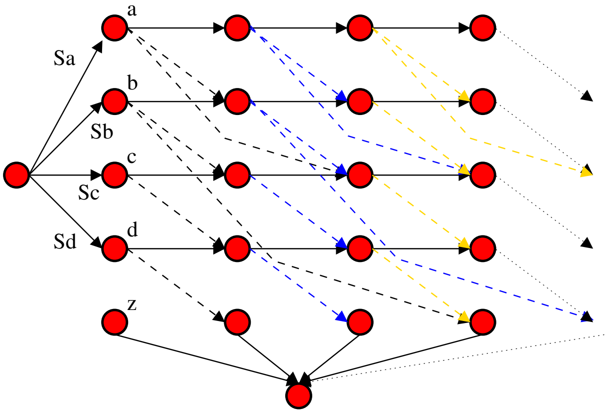


Image segmentation

Segmenting an image: separating a grid P of pixels into **background** and **foreground**.

Constraints:

- **single-pixel:** each pixel v in P comes with integers f_v and b_v
  - f_v large: v should be in the **foreground**
  - b_v large: v should be in the **background**
- **connectivity:** for adjacent pixels v, w, there is a **penalty** $p_{v,w}$ to pay if they are **not** in the same component.

Finding $f_v, b_v, p_{v,w}$ takes some work. Assuming we know them, want to find subset F **(foreground)** and $B = P - F$ **(background)** that maximizes

$$W(F) = \sum_{v \in F} f_v + \sum_{v \in B} b_v - \sum_{v \in F,\ w \in B,\ (v,w)\ \text{connected}} p_{v,w}.$$

Making it a min-cut problem

Step 1: turning a max into a min.

Let $K = \sum_v f_v + \sum_v b_v$ (independent of the choice of F and B).

Then

$$K = \sum_{v \in F} f_v \; + \sum_{v \in B} f_v + \sum_{v \in F} b_v \; + \sum_{v \in B} b_v .$$

So

$$W(F,B) \; = \; K \; - \; \sum_{v \in B} f_v \; - \; \sum_{v \in F} b_v \; - \sum_{v \in F,\ w \in B,\ (v,w) \text{ connected}} p_{v,w} .$$

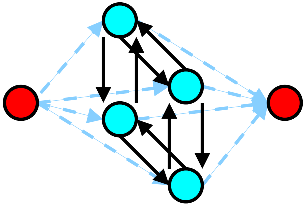
Since K does not depend on F and B, **maximizing** W is the same thing as **minimizing**

$$\sum_{v \in B} f_v \; + \; \sum_{v \in F} b_v \; + \sum_{v \in F,\ w \in B,\ (v,w) \text{ connected}} p_{v,w} .$$

Making it a min-cut problem

Step 2: setting up the graph G:

- vertices of G are **all pixels**, plus a source s and a sink t;
- edges of G:
  - for all v, an edge (s, v), with capacity b_v;
  - for all v, an edge (v, t), with capacity f_v;
  - for all neighbours v, w, edges (v, w) and (w, v), with capacities $p_{v,w}$.



Making it a min-cut problem

Step 3: understanding the cuts.

A **cut** $(U, V = P - U)$ in G gives a **partition** of the pixels:

- the background B is $U - \{s\}$,
- the foreground F is $V - \{t\}$.

What is its capacity? The edges $U \to V$ come into 3 categories:

- edges (v, t), for v in B, **contributes f_v to the capacity**,
- edges (s, v), for v in F, **contributes b_v to the capacity**,
- edges (v, w), for v in B and w in F, **contributes $p_{v,w}$ to the capacity**.

The capacity of the cut is

$$\sum_{v \in B} f_v + \sum_{v \in F} b_v + \sum_{v \in F,\ w \in B,\ (v,w) \text{ connected}} p_{v,w},$$

so solving min-cut solves the problem.