Networks, Flows and Cuts

Alexander Shen

LIRMM / CNRS, University of Montpellier. France

Outline

An Example

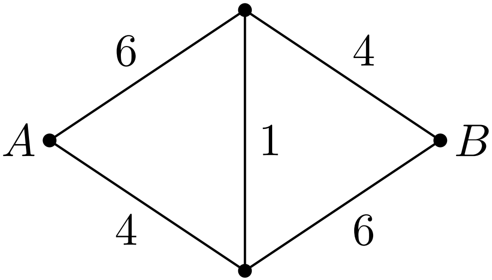
Framework

Ford and Fulkerson: Proof

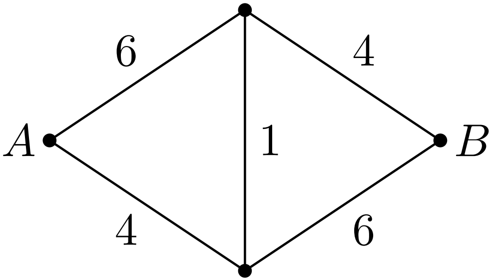
Application: Hall’s theorem

What Else?

Network: Pipes and Capacities

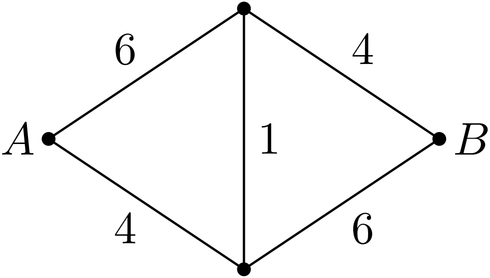


Network: Pipes and Capacities



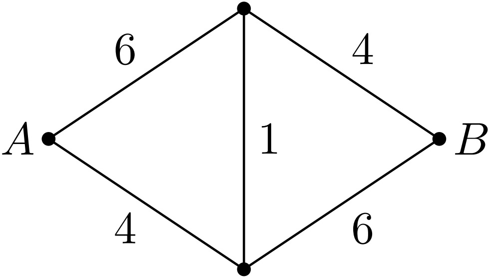
- edges = pipes

Network: Pipes and Capacities



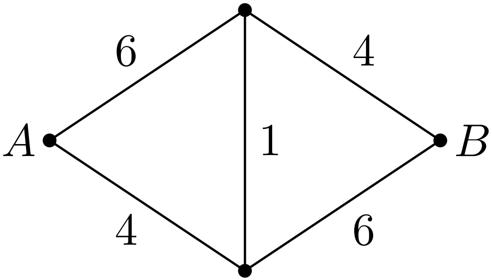
- edges = pipes
- numbers = capacities

Network: Pipes and Capacities



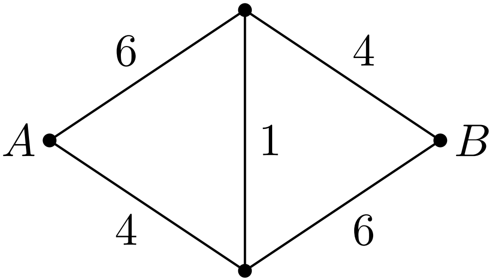
- edges = pipes
- numbers = capacities
- A: source, B: destination

Network: Pipes and Capacities



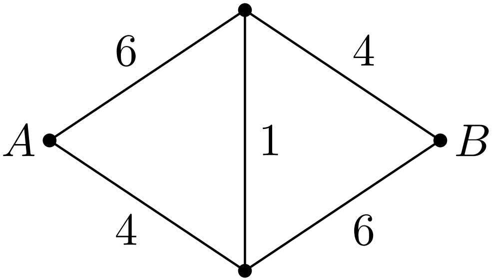
- edges = pipes
- numbers = capacities
- A: source, B: destination
- maximum flow?

Network: Pipes and Capacities



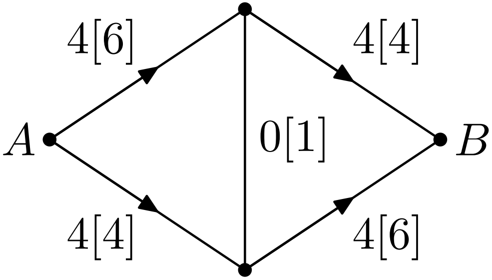
- edges = pipes
- numbers = capacities
- A : source, B : destination
- maximum flow? 10?

Network: Pipes and Capacities

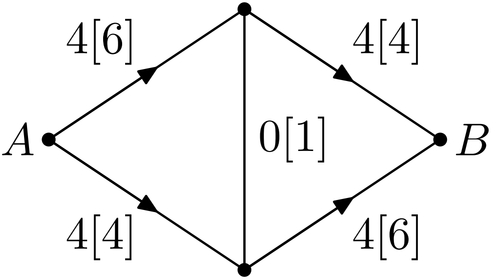


- edges = pipes
- numbers = capacities
- A: source, B: destination
- maximum flow? 10? not really

Flow: 8 Is Possible

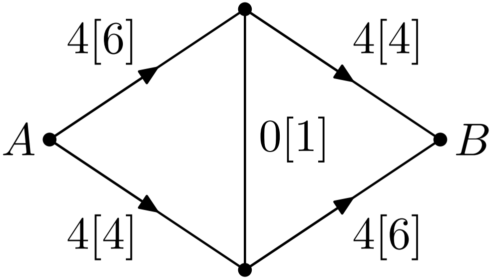


Flow: 8 Is Possible



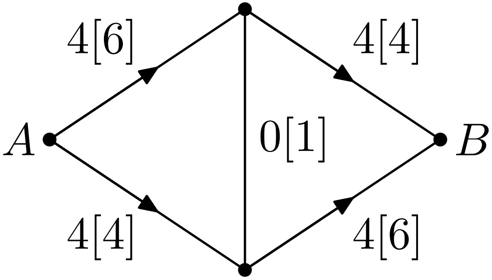
- flow[capacity]

Flow: 8 Is Possible



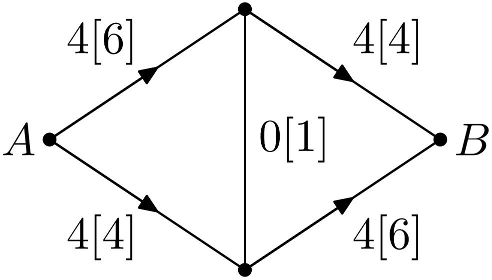
- flow[capacity]
- no oil is spilled (in = out)

Flow: 8 Is Possible



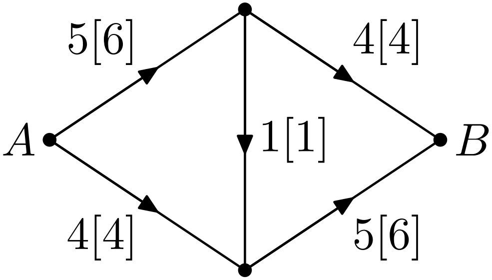
- flow[capacity]
- no oil is spilled (in = out)
- flow (from A to B): 8

Flow: 8 Is Possible

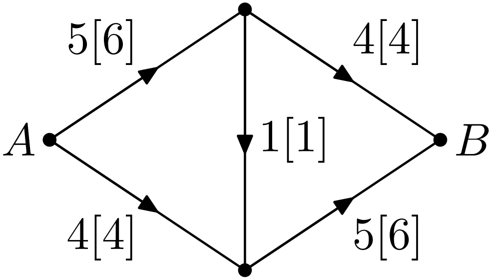


- flow[capacity]
- no oil is spilled (in = out)
- flow (from *A* to *B*): 8
- maximum flow?

Flow: 9 Is Possible

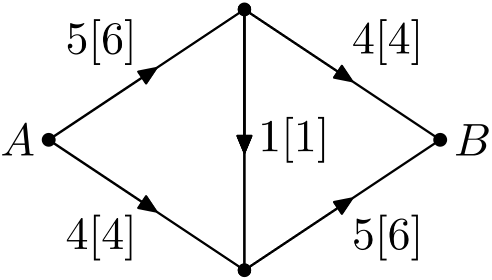


Flow: 9 Is Possible



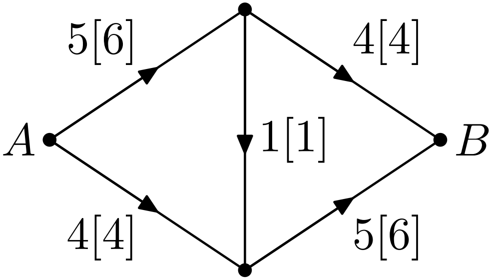
- flow (from A to B): 9

Flow: 9 Is Possible



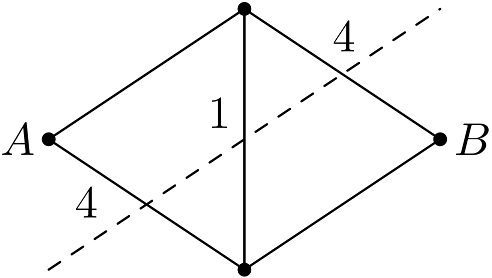
- flow (from *A* to *B*): 9
- maximum flow?

Flow: 9 Is Possible

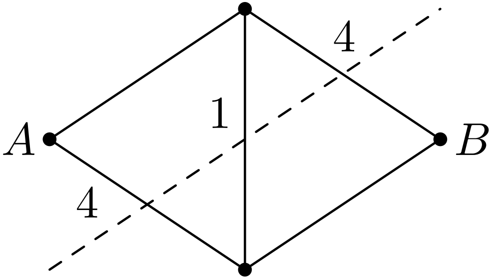


- flow (from *A* to *B*): 9
- maximum flow?
- 9 is possible, not more than 10

Why 9 Is Maximal

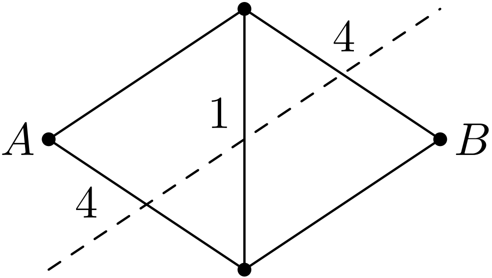


Why 9 Is Maximal



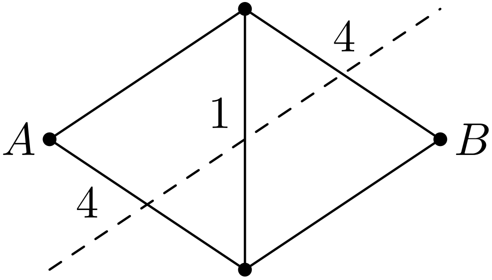
- dashed line is a *cut*

Why 9 Is Maximal



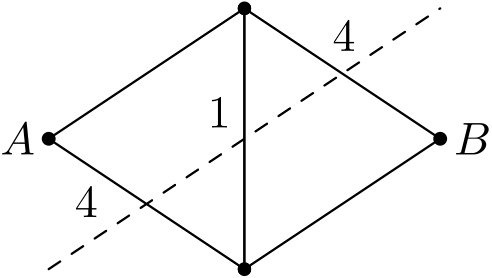
- dashed line is a *cut*
- pipes of capacities $4, 1, 4$ cross it

Why 9 Is Maximal



- dashed line is a *cut*
- pipes of capacities $4, 1, 4$ cross it
- no more than $4 + 1 + 4 = 9$

Why 9 Is Maximal



- dashed line is a *cut*
- pipes of capacities $4, 1, 4$ cross it
- no more than $4 + 1 + 4 = 9$
- 9 is indeed maximal

Our Goal

Our Goal

- network
  (=graph + edge capacities +
  source/destination)

Our Goal

- network
  (=graph + edge capacities +
  source/destination)
- maximal flow?

Our Goal

- network
  (=graph + edge capacities +
  source/destination)
- maximal flow?
- example

Our Goal

- network
  (=graph + edge capacities +
  source/destination)
- maximal flow?
- example
- maximality proof

Our Goal

- network
  (=graph + edge capacities +
  source/destination)
- maximal flow?
- example
- maximality proof (using cuts)

Our Goal

- network
  (=graph + edge capacities +
  source/destination)
- maximal flow?
- example
- maximality proof (using cuts)
- Ford and Fulkerson (1956)

Our Goal

- network
  (=graph + edge capacities +
  source/destination)
- maximal flow?
- example
- maximality proof (using cuts)
- Ford and Fulkerson (1956)
- simplification: integer capacities

Outline

An Example

Framework

Ford and Fulkerson: Proof

Application: Hall's theorem

What Else?

Network: Capacities

Network: Capacities

- vertices $1, 2, \ldots n$

Network: Capacities

- vertices $1, 2, \ldots n$
- source A, destination B

Network: Capacities

- vertices $1, 2, \ldots n$
- source A, destination B
- capacities $c[i, j]$ for $i \to j$

Network: Capacities

- vertices $1, 2, \ldots n$
- source A, destination B
- capacities $c[i, j]$ for $i \to j$
  integers; $c[i, j] \geq 0$

Network: Capacities

- vertices $1, 2, \ldots n$
- source A, destination B
- capacities $c[i,j]$ for $i \to j$
  integers; $c[i,j] \geq 0$; (no pipe = 0)

Network: Capacities

- vertices $1, 2, \dots n$
- source A, destination B
- capacities $c[i,j]$ for $i \to j$
  integers; $c[i,j] \geq 0$; (no pipe = 0)
- $c[i,j] \neq c[j,i]$ allowed

Network: Capacities

- vertices $1, 2, \ldots n$
- source A, destination B
- capacities $c[i, j]$ for $i \to j$
  integers; $c[i, j] \geq 0$; (no pipe = 0)
- $c[i, j] \neq c[j, i]$ allowed
  generalization needed for the analysis

Network: Capacities

- vertices $1, 2, \ldots n$
- source A, destination B
- capacities $c[i, j]$ for $i \to j$
  integers; $c[i, j] \geq 0$; (no pipe = 0)
- $c[i, j] \neq c[j, i]$ allowed
  generalization needed for the analysis
- two edges $i \to j$?

Network: Capacities

- vertices $1, 2, \ldots n$
- source A, destination B
- capacities $c[i, j]$ for $i \to j$
  integers; $c[i, j] \geq 0$; (no pipe = 0)
- $c[i, j] \neq c[j, i]$ allowed
  generalization needed for the analysis
- two edges $i \to j$? not allowed,
  but does not matter

Network: Capacities

- vertices $1, 2, \ldots n$
- source A, destination B
- capacities $c[i,j]$ for $i \to j$
  integers; $c[i,j] \geq 0$; (no pipe = 0)
- $c[i,j] \neq c[j,i]$ allowed
  generalization needed for the analysis
- two edges $i \to j$? not allowed,
  but does not matter
- $c[i,i] = 0$ for convenience

Network: Flows

Network: Flows

- flow $f[i, j]$ for $i \to j$

Network: Flows

- flow $f[i,j]$ for $i \to j$
- $f[i,j] \leq c[i,j]$: restriction

Network: Flows

- flow $f[i,j]$ for $i \to j$
- $f[i,j] \leq c[i,j]$: restriction
- $f[i,j] = -f[j,i]$

Network: Flows

- flow $f[i, j]$ for $i \to j$
- $f[i, j] \leq c[i, j]$: restriction
- $f[i, j] = -f[j, i]$ (electric current)

Network: Flows

- flow $f[i,j]$ for $i \to j$
- $f[i,j] \leq c[i,j]$: restriction
- $f[i,j] = -f[j,i]$ (electric current)
- implies $f[i,i] = 0$

Network: Flows

- flow $f[i,j]$ for $i \to j$
- $f[i,j] \leq c[i,j]$: restriction
- $f[i,j] = -f[j,i]$ (electric current)
- implies $f[i,i] = 0$
- no spill condition

Network: Flows

- flow $f[i,j]$ for $i \to j$
- $f[i,j] \leq c[i,j]$: restriction
- $f[i,j] = -f[j,i]$ (electric current)
- implies $f[i,i] = 0$
- no spill condition

  $\sum_j f[i,j] = 0$ for every i

Network: Flows

- flow $f[i,j]$ for $i \to j$
- $f[i,j] \leq c[i,j]$: restriction
- $f[i,j] = -f[j,i]$ (electric current)
- implies $f[i,i] = 0$
- no spill condition

  $\sum_j f[i,j] = 0$ for every i
- …except for A and B

Network: Flows

- flow $f[i,j]$ for $i \to j$
- $f[i,j] \leq c[i,j]$: restriction
- $f[i,j] = -f[j,i]$ (electric current)
- implies $f[i,i] = 0$
- no spill condition

  $\sum_j f[i,j] = 0$ for every i
- ...except for A and B
- total flow:

  $\sum_j f[A,j] = \sum_i f[i,B]$

Network: Cuts

Network: Cuts

- how to prove that no flow exceeds c?

Network: Cuts

- how to prove that no flow exceeds c?
- find a cut C of total capacity c

Network: Cuts

- how to prove that no flow exceeds c?
- find a cut C of total capacity c
- C: some set of nodes that contains A but not B

Network: Cuts

- how to prove that no flow exceeds c?
- find a cut C of total capacity c
- C: some set of nodes that contains A but not B
- *total capacity* of a cut: sum of capacities of all outgoing edges ($C \rightarrow$ not C)

Network: Cuts

- how to prove that no flow exceeds c?
- find a cut C of total capacity c
- C: some set of nodes that contains A but not B
- *total capacity* of a cut: sum of capacities of all outgoing edges ($C \rightarrow$ not C)
- obvious: any total flow never exceeds the total capacity of any cut

Network: Cuts

- how to prove that no flow exceeds c?
- find a cut C of total capacity c
- C: some set of nodes that contains A but not B
- *total capacity* of a cut: sum of capacities of all outgoing edges ($C \rightarrow$ not C)
- obvious: any total flow never exceeds the total capacity of any cut
- Ford–Fulkerson: the equality happens for some flow and some cut

What Ford and Fulkerson Say

What Ford and Fulkerson Say

obvious:

What Ford and Fulkerson Say

obvious:

What Ford and Fulkerson Say

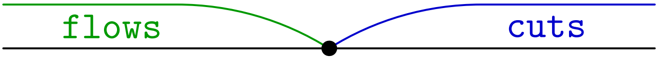
obvious:

Ford–Fulkerson:

What Ford and Fulkerson Say

obvious:

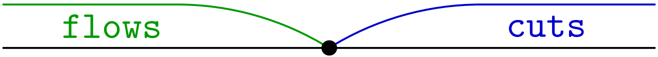
Ford–Fulkerson:



What Ford and Fulkerson Say

obvious:

Ford–Fulkerson:



Theorem: maximal flow = minimal cut

Outline

An Example

Framework

Ford and Fulkerson: Proof

Application: Hall's theorem

What Else?

Special Case

Special Case

- no flow $\Rightarrow$ cut of capacity 0

Special Case

- no flow $\Rightarrow$ cut of capacity 0
  (max flow = 0 $\Rightarrow$ min cut = 0)

Special Case

- no flow $\Rightarrow$ cut of capacity 0
  (max flow = 0 $\Rightarrow$ min cut = 0)
- why?

Special Case

- no flow $\Rightarrow$ cut of capacity 0
  (max flow = 0 $\Rightarrow$ min cut = 0)
- why?
- reachable node x: path from A to x with nonzero capacities

Special Case

- no flow $\Rightarrow$ cut of capacity 0
  (max flow = 0 $\Rightarrow$ min cut = 0)
- why?
- reachable node x: path from A to x with nonzero capacities
- B reachable $\Rightarrow$ non-zero flow

Special Case

- no flow $\Rightarrow$ cut of capacity 0
  (max flow = 0 $\Rightarrow$ min cut = 0)
- why?
- reachable node x: path from A to x with nonzero capacities
- B reachable $\Rightarrow$ non-zero flow
- B unreachable $\Rightarrow$ zero cut

Special Case

- no flow $\Rightarrow$ cut of capacity 0
  (max flow = 0 $\Rightarrow$ min cut = 0)
- why?
- reachable node x: path from A to x with nonzero capacities
- B reachable $\Rightarrow$ non-zero flow
- B unreachable $\Rightarrow$ zero cut
  (C = reachable nodes)

Proof Idea

Proof Idea

- “increase flow until max-flow=min-cut is achieved”

Proof Idea

- “increase flow until max-flow=min-cut is achieved”
- special case: a good start

Proof Idea

- "increase flow until max-flow=min-cut is achieved"
- special case: a good start
  (either min-cut=0 or a non-zero flow is possible)

Proof Idea

- "increase flow until max-flow=min-cut is achieved"
- special case: a good start
  (either min-cut=0 or a non-zero flow is possible)
- repeat: increase the flow by at least 1 unless a matching cut can be found

Proof Idea

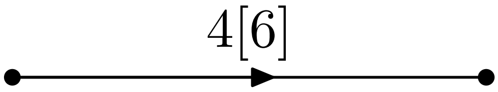
- “increase flow until max-flow=min-cut is achieved”
- special case: a good start
  (either min-cut=0 or a non-zero flow is possible)
- repeat: increase the flow by at least 1 unless a matching cut can be found
- why it is possible?

Proof Idea

- “increase flow until max-flow=min-cut is achieved”
- special case: a good start
  (either min-cut=0 or a non-zero flow is possible)
- repeat: increase the flow by at least 1 unless a matching cut can be found
- why it is possible?
- reduction to the special case: “residual network”

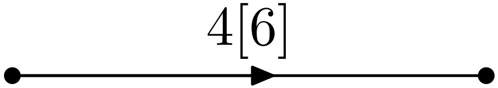
Edge Reserves

Edge Reserves



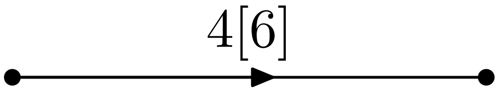
- edge that is not fully used

Edge Reserves



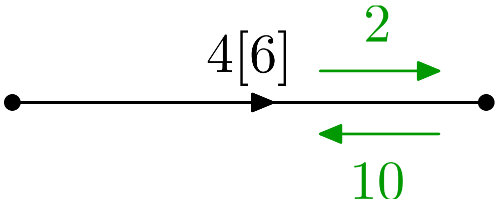
- edge that is not fully used
- more flow (2 $=$ 6 $-$ 4)

Edge Reserves



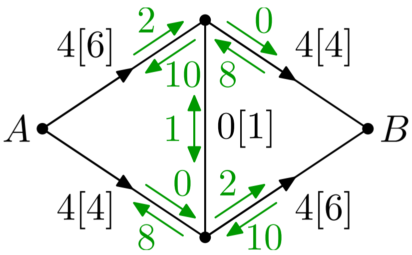
- edge that is not fully used
- more flow ($2 = 6 - 4$)
- back flow ($10 = 6 + 4$)

Edge Reserves



- edge that is not fully used
- more flow ($2 = 6 - 4$)
- back flow ($10 = 6 + 4$)
- new network with residual capacities

Residual Network



- flow in the residual network ⇔ increase for the current flow in the original network
- zero cut in the residual network ⇔ matching cut in the original network
- special case + residual network

Collecting the Pieces

Collecting the Pieces

flow = zero

Collecting the Pieces

flow = zero
repeat
    compute residual network wrt flow

Collecting the Pieces

```
flow = zero
repeat
    compute residual network wrt flow
    apply special case to it:
```

Collecting the Pieces

flow = zero
repeat
    compute residual network wrt flow
    apply special case to it:
        augmenting path: increase flow

Collecting the Pieces

```
flow = zero
repeat
    compute residual network wrt flow
    apply special case to it:
        augmenting path: increase flow
        zero residual cut: break
```

Collecting the Pieces

```
flow = zero
repeat
    compute residual network wrt flow
    apply special case to it:
        augmenting path: increase flow
        zero residual cut: break
// maximal flow = minimal cut found
```

Collecting the Pieces

```
flow = zero
repeat
    compute residual network wrt flow
    apply special case to it:
        augmenting path: increase flow
        zero residual cut: break
// maximal flow = minimal cut found
```

- termination?

Collecting the Pieces

```
flow = zero
repeat
    compute residual network wrt flow
    apply special case to it:
        augmenting path: increase flow
        zero residual cut: break
// maximal flow = minimal cut found
```

- termination?
- integer capacities

Collecting the Pieces

```
flow = zero
repeat
    compute residual network wrt flow
    apply special case to it:
        augmenting path: increase flow
        zero residual cut: break
// maximal flow = minimal cut found
```

- termination?
- integer capacities
- flow += 1 (at least)

Outline

An Example

Framework

Ford and Fulkerson: Proof

Application: Hall’s theorem

What Else?

The Setting

The Setting

- warning: “marriage terminology”

The Setting

- warning: “marriage terminology”
- n men, n women

The Setting

- warning: “marriage terminology”
- n men, n women
- some pairs (m, w) agree to marry

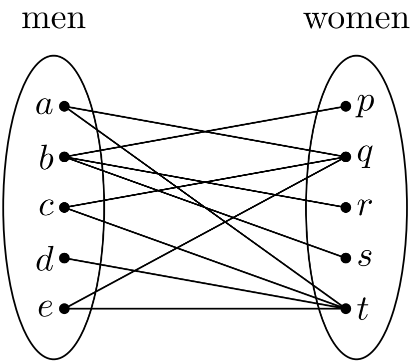
The Setting

- warning: “marriage terminology”
- n men, n women
- some pairs (m, w) agree to marry
- is there a perfect matching (each person is in one pair)?

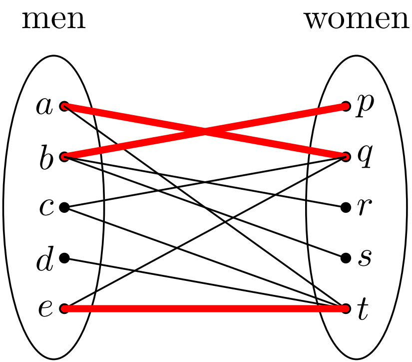
The Setting

- warning: “marriage terminology”
- n men, n women
- some pairs (m, w) agree to marry
- is there a perfect matching (each person is in one pair)?
- bipartite graph

Bipartite Graph

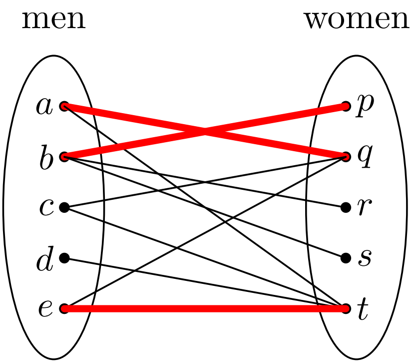


Bipartite Graph



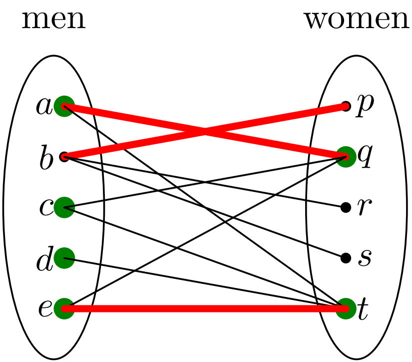
- easy to find 3 pairs

Bipartite Graph



- easy to find 3 pairs
- not more:

Bipartite Graph



- easy to find 3 pairs
- not more:
  a, *c*, *d*, *e* agree to marry only *q* and *t*

Hall’s Theorem: the Statement

Hall’s Theorem: the Statement

A bipartite graph (n left and n right vertices)

Hall's Theorem: the Statement

A bipartite graph (n left and n right vertices)
no perfect matching $\Leftrightarrow$

Hall's Theorem: the Statement

A bipartite graph (n left and n right vertices)
no perfect matching $\Leftrightarrow$
there exist some k
and k left vertices on the left
that have less than k right neighbors (in total)

Hall's Theorem: the Statement

A bipartite graph (n left and n right vertices)
no perfect matching $\Leftrightarrow$
there exist some k
and k left vertices on the left
that have less than k right neighbors (in total)

- a set of k left vertices with less than k neighbors is an obstacle

Hall's Theorem: the Statement

A bipartite graph (*n* left and *n* right vertices)
no perfect matching $\Leftrightarrow$
there exist some *k*
and *k* left vertices on the left
that have less than *k* right neighbors (in total)

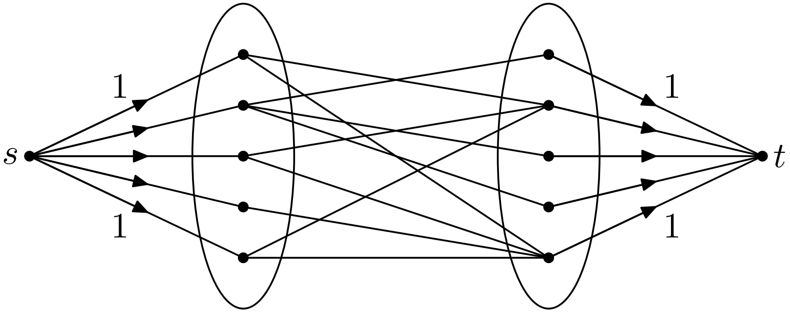
- a set of *k* left vertices with less than *k* neighbors is an obstacle
- obvious: obstacle is really an obstacle

Hall's Theorem: the Statement

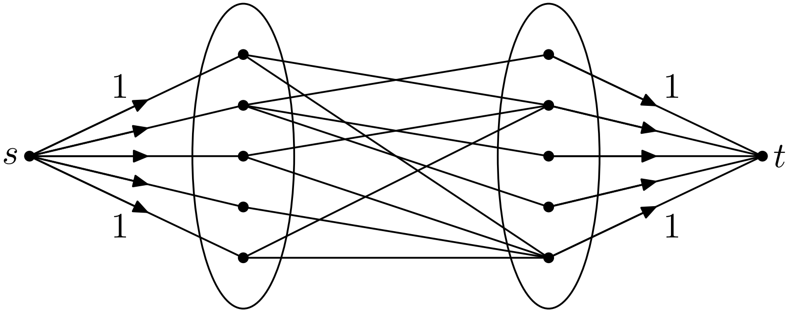
A bipartite graph (n left and n right vertices)
no perfect matching $\Leftrightarrow$
there exist some k
and k left vertices on the left
that have less than k right neighbors (in total)

- a set of k left vertices with less than k neighbors is an obstacle
- obvious: obstacle is really an obstacle
- serious: only these obstacles matter

Reduction to Flows: The Network

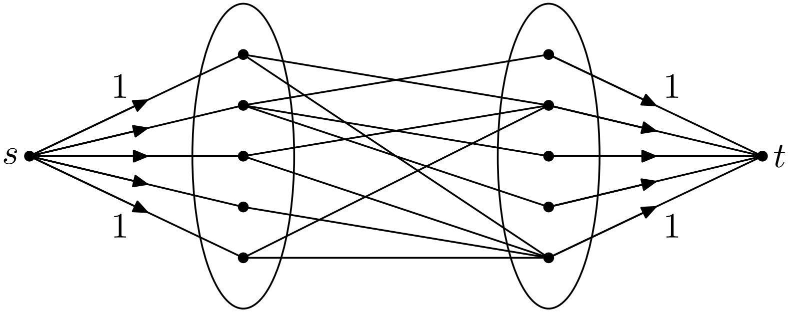


Reduction to Flows: The Network



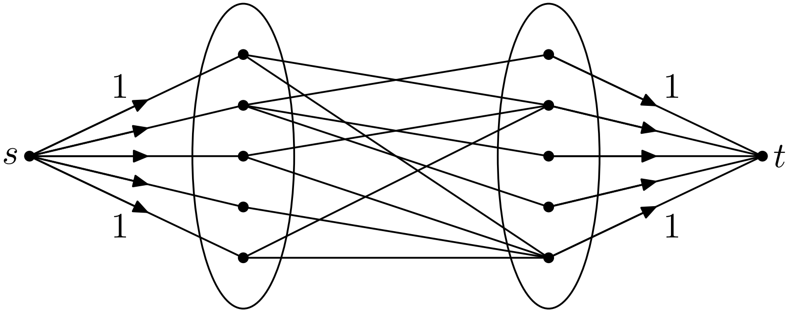
- all pipes only from left to right

Reduction to Flows: The Network



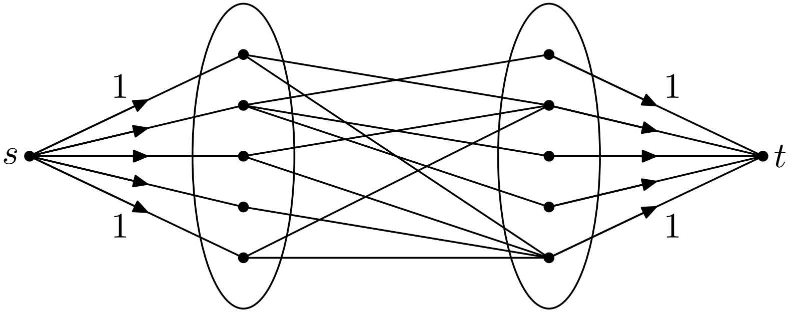
- all pipes only from left to right
- very large capacity in the middle

Reduction to Flows: The Network



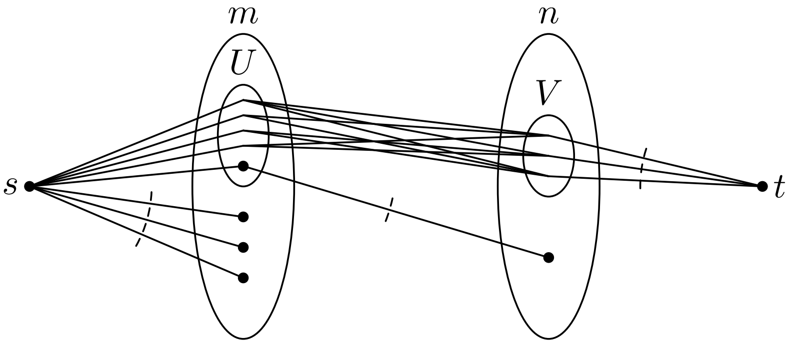
- all pipes only from left to right
- very large capacity in the middle
- matching = integer flow of size *n*

Reduction to Flows: The Network

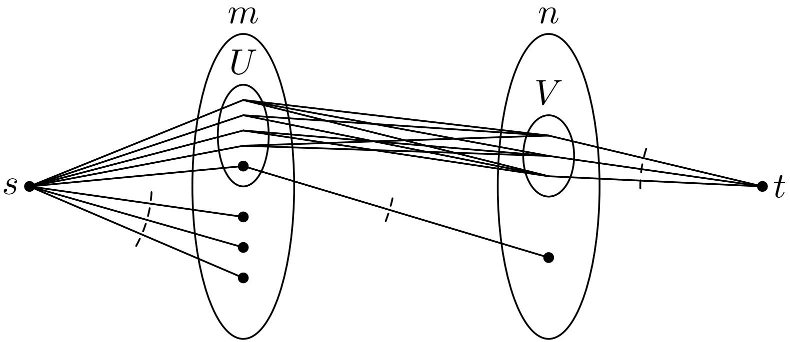


- all pipes only from left to right
- very large capacity in the middle
- matching = integer flow of size n
- no matching $\Rightarrow$ cut $< n$

What Is A Cut?

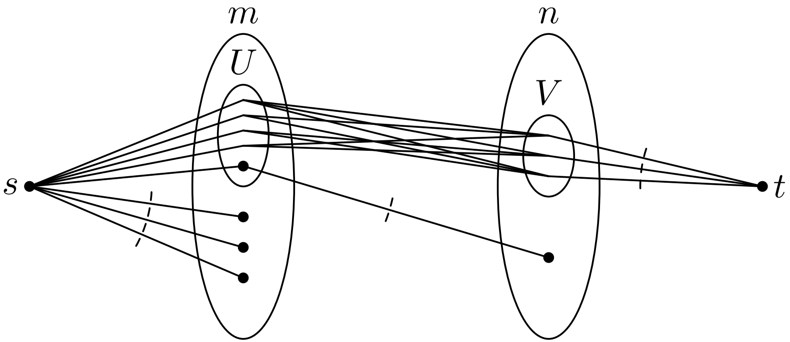


What Is A Cut?



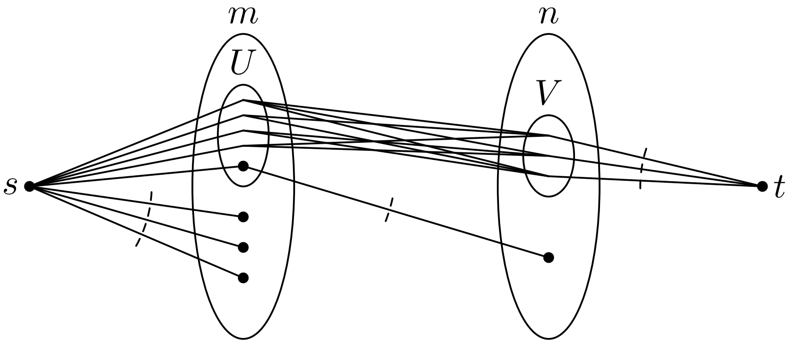
- cut C: A, some left and some right

What Is A Cut?



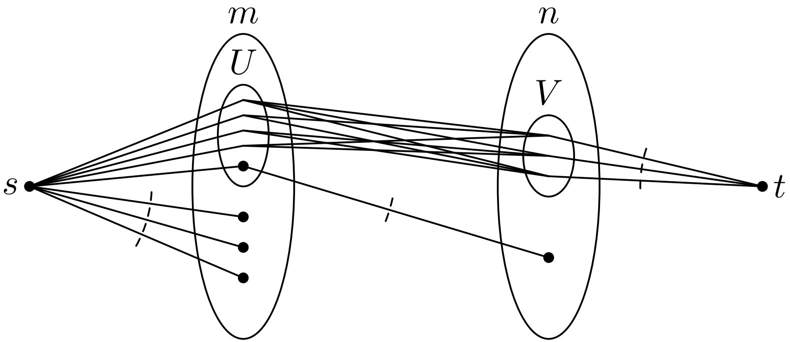
- cut *C*: *A*, some left and some right
- outgoing edges of three types

What Is A Cut?



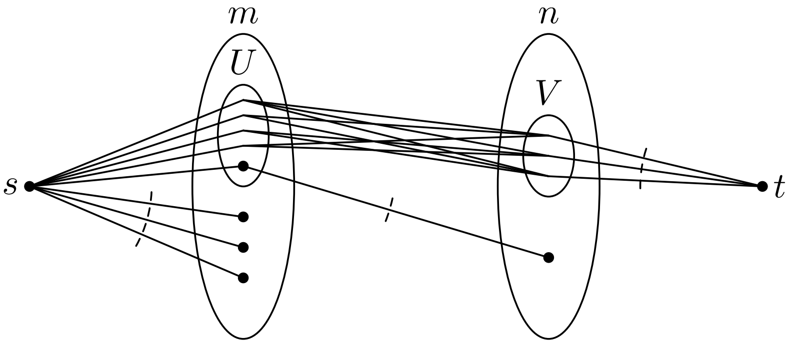
- cut *C*: *A*, some left and some right
- outgoing edges of three types
- second type impossible (large capacity)

What Is A Cut?



- cut C: A, some left and some right
- outgoing edges of three types
- second type impossible (large capacity)
- all neighbors of left C are in right C

What Is A Cut?



- cut C: A, some left and some right
- outgoing edges of three types
- second type impossible (large capacity)
- all neighbors of left C are in right C
- cut $< n \Rightarrow$ left $C >$ right C

Outline

An Example

Framework

Ford and Fulkerson: Proof

Application: Hall's theorem

What Else?

MaxFlow algorithms

MaxFlow algorithms

- MaxFlow is really important

MaxFlow algorithms

- MaxFlow is really important
- better algorithms

MaxFlow algorithms

- MaxFlow is really important
- better algorithms
- better algorithms for matching

MaxFlow algorithms

- MaxFlow is really important
- better algorithms
- better algorithms for matching
- more difficult: non-bipartite
  "matching for same-sex marriages"

MaxFlow algorithms

- MaxFlow is really important
- better algorithms
- better algorithms for matching
- more difficult: non-bipartite
  “matching for same-sex marriages”
- special case of *linear programming*

MaxFlow algorithms

- MaxFlow is really important
- better algorithms
- better algorithms for matching
- more difficult: non-bipartite
  “matching for same-sex marriages”
- special case of *linear programming*
- max solution = min obstacle: “duality”

Tilings Revisited

Tilings Revisited

- domino problem:
  tile a region with dominos

Tilings Revisited

- domino problem:
  tile a region with dominos
- tool: reduction
  to a matching problem in a graph

Tilings Revisited

- domino problem:
  tile a region with dominos
- tool: reduction
  to a matching problem in a graph
- left = white, right = black

Tilings Revisited

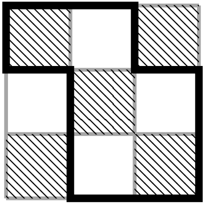
- domino problem:
  tile a region with dominos
- tool: reduction
  to a matching problem in a graph
- left = white, right = black
- edges = neighbors

Tilings Revisited

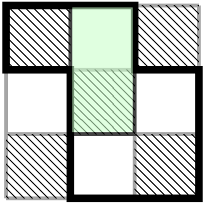
- domino problem:
  tile a region with dominos
- tool: reduction
  to a matching problem in a graph
- left = white, right = black
- edges = neighbors
- matching = tiling

The Reduction in Pictures

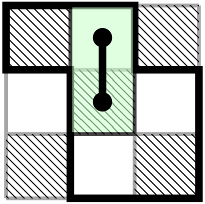
The Reduction in Pictures



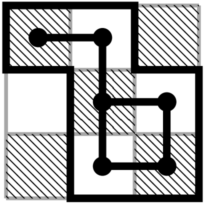
The Reduction in Pictures



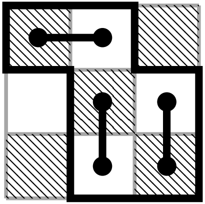
The Reduction in Pictures



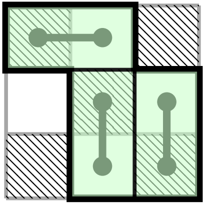
The Reduction in Pictures



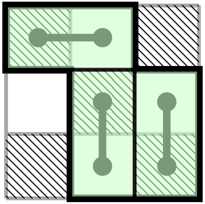
The Reduction in Pictures



The Reduction in Pictures

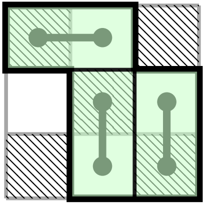


The Reduction in Pictures



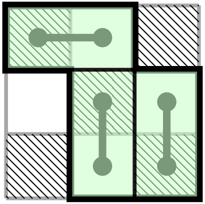
Ford–Fulkerson algorithm:

The Reduction in Pictures



Ford–Fulkerson algorithm:
finds a tiling if it exists

The Reduction in Pictures



Ford–Fulkerson algorithm:

finds a tiling if it exists

finds an obstacle if no tiling