Probabilistic Min-Cut

CSE 421 Fall 22
Lecture 24

Probabilistic Algorithms 101

What is a Probabilistic Algorithm

- Deterministic algorithms take input and produce some deterministic output
- Probabilistic algorithms take input and a source of random bits and make random choices during execution, so output is non-deterministic

Motivation for Probabilistic Algorithms

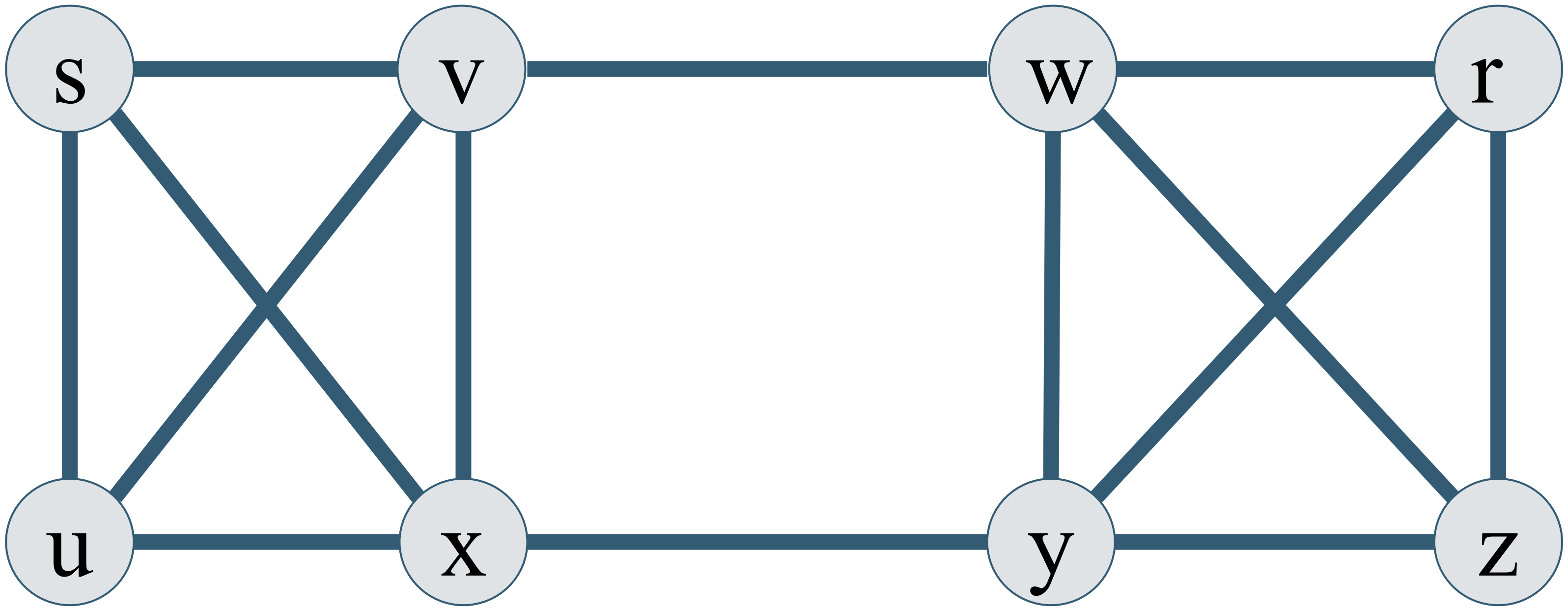
- Probabilistic algorithm may be more faster or more simple (or both)
- In some cases, the probabilistic algorithm is faster or more space efficient than the best-known deterministic algorithm
- For some use cases, if we don't care about "the best" answer and we're willing to tolerate some error in exchange for the above benefits (take CSE 422 and CSE 521 to learn more)

Today's Topic

- **Finding the minimum cut of an *undirected unweighted* graph using a probabilistic algorithm**
- What is a min cut
- Key idea of algorithm [Edge contraction]
- Correctness

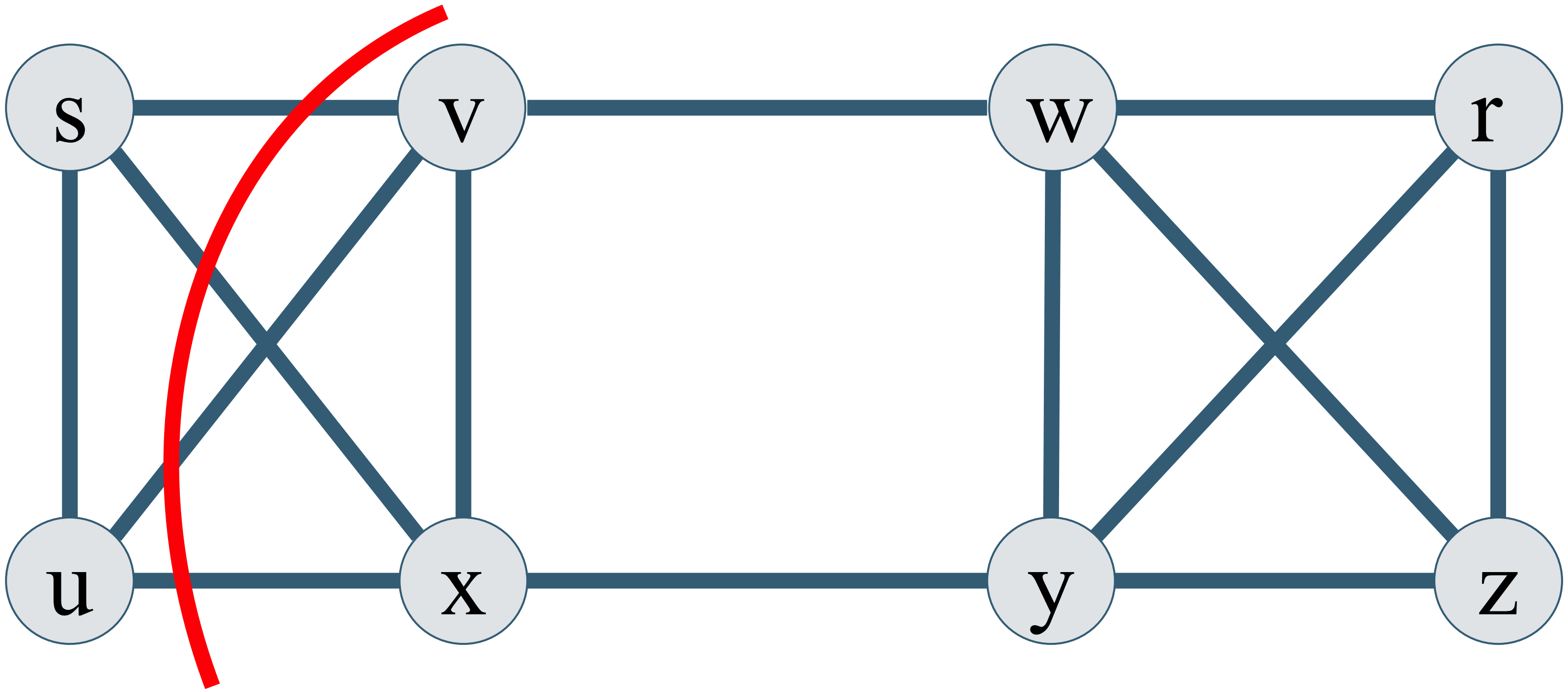
Today’s Topic

- A cut is defined as a partition of vertices into two disjoint sets
- The size of a cut is the number of edges in the graph with one endpoint in each set spanning the cut
- The minimum cut is the minimum collection of such edges



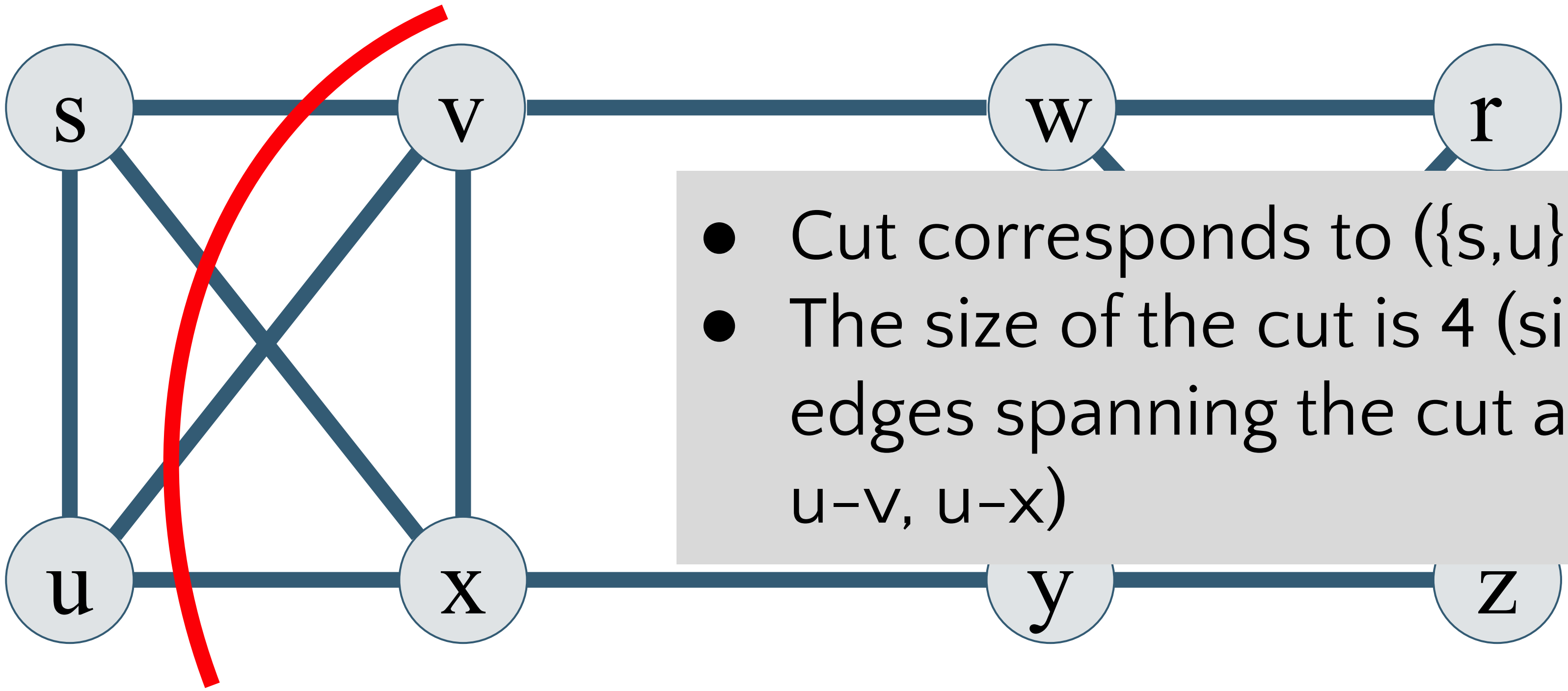
Today's Topic

- A cut is defined as a partition of vertices into two disjoint sets
- The size of a cut is the number of edges in the graph with one endpoint in each set spanning the cut
- The minimum cut is the minimum collection of such edges



Today's Topic

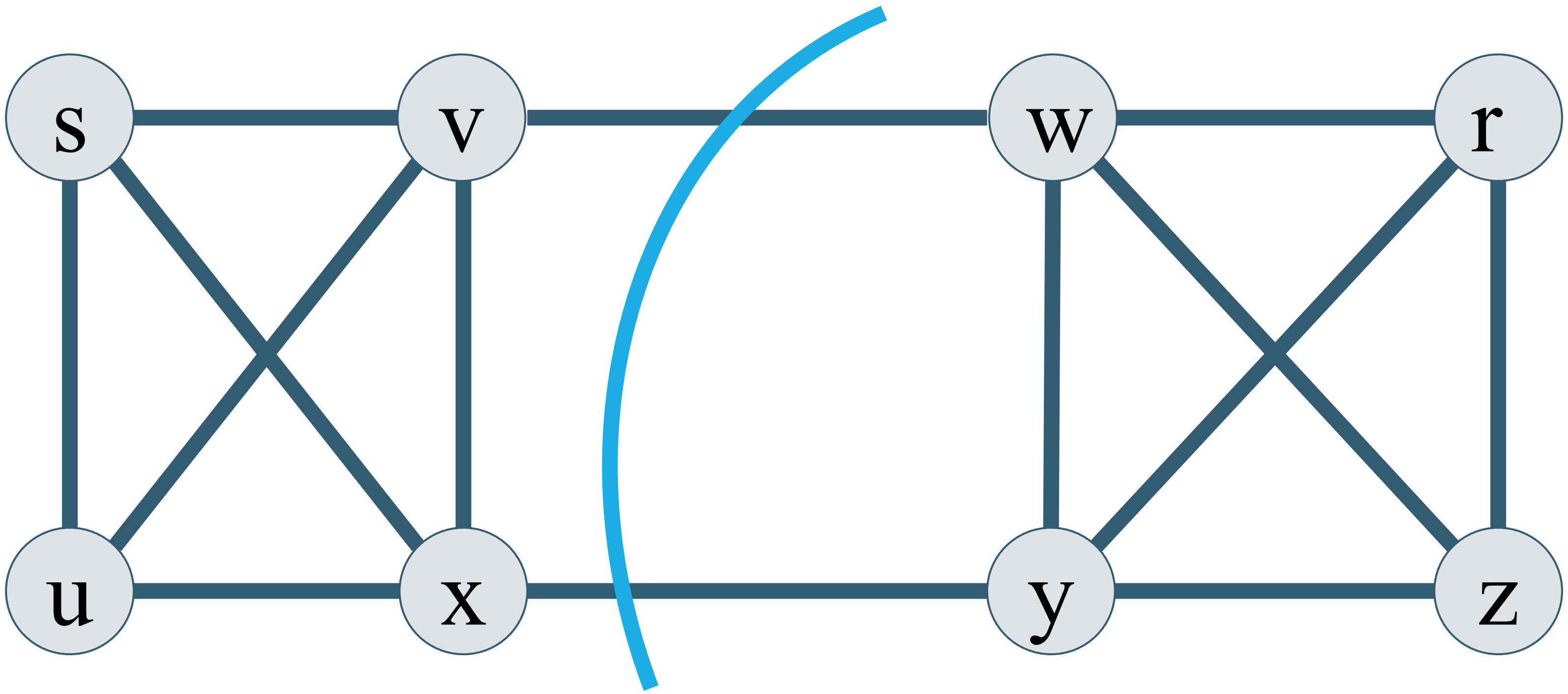
- A cut is defined as a partition of vertices into two disjoint sets
- The size of a cut is the number of edges in the graph with one endpoint in each set spanning the cut
- The minimum cut is the minimum collection of such edges



- Cut corresponds to ({s,u}, {v,x,w,r,y,z})
- The size of the cut is 4 (since the edges spanning the cut are s–v, s–x, u–v, u–x)

Today's Topic

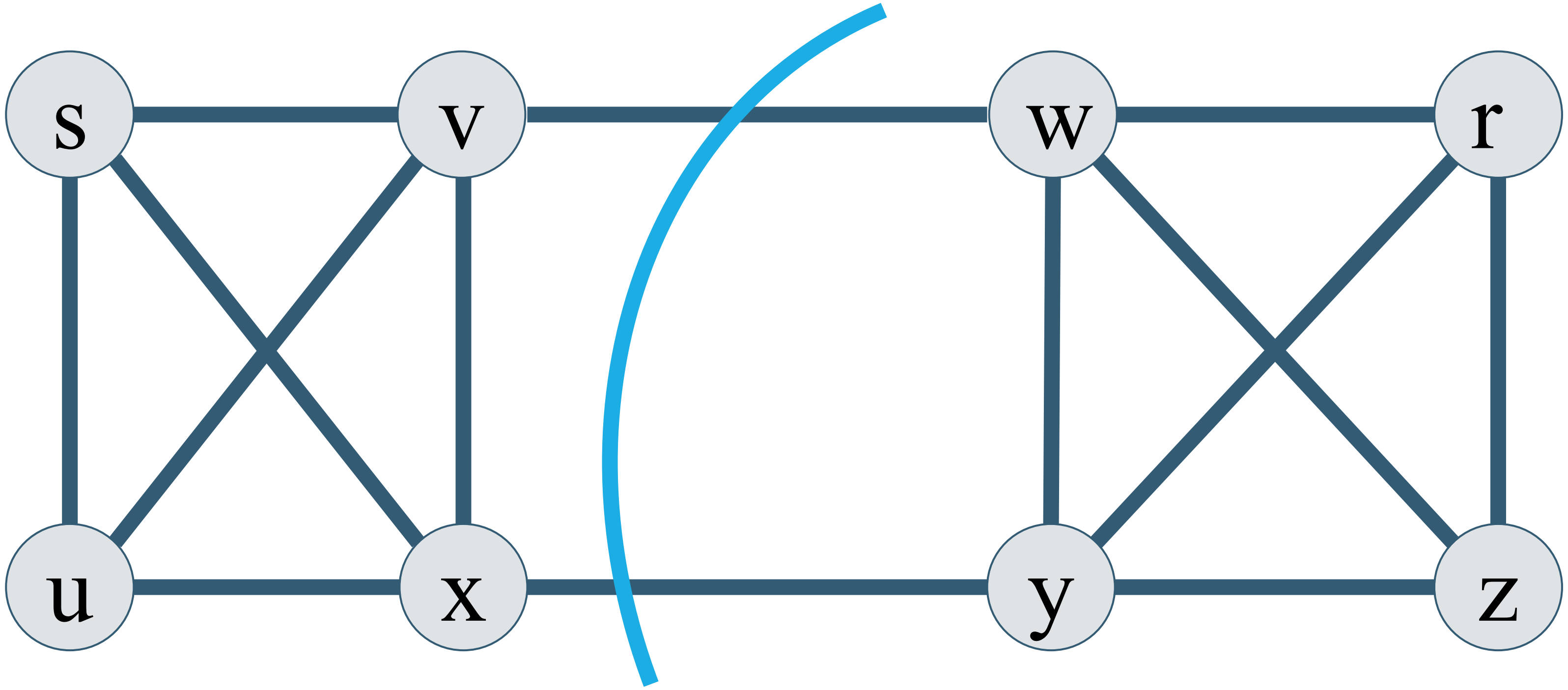
- A cut is defined as a partition of vertices into two disjoint sets
- The size of a cut is the number of edges in the graph with one endpoint in each set spanning the cut
- The minimum cut is the minimum collection of such edges



Today's Topic

- Cut corresponds to ({s,v,u,x}, {w,r,y,z})
- The size of the cut is 2 (since the edges spanning the cut are v–w, x–y)

- A cut is defin… disjoint sets
- The size of a … aph with one endpoint in e…
- The minimum … uch edges



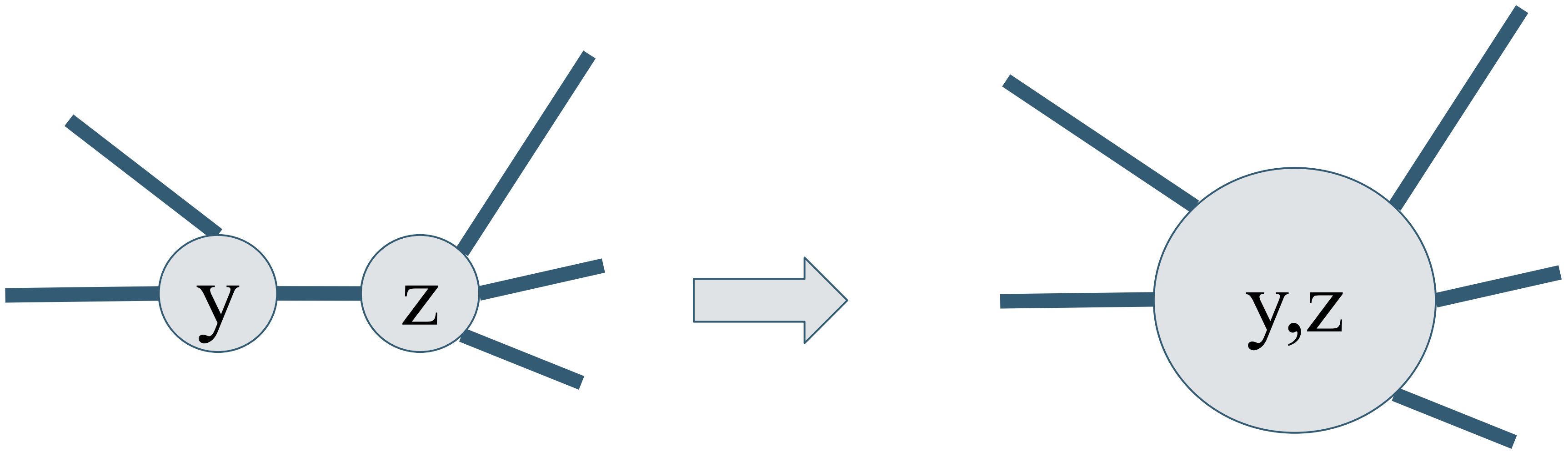
Karger’s Min Cut Algorithm

Intuition for Contractions

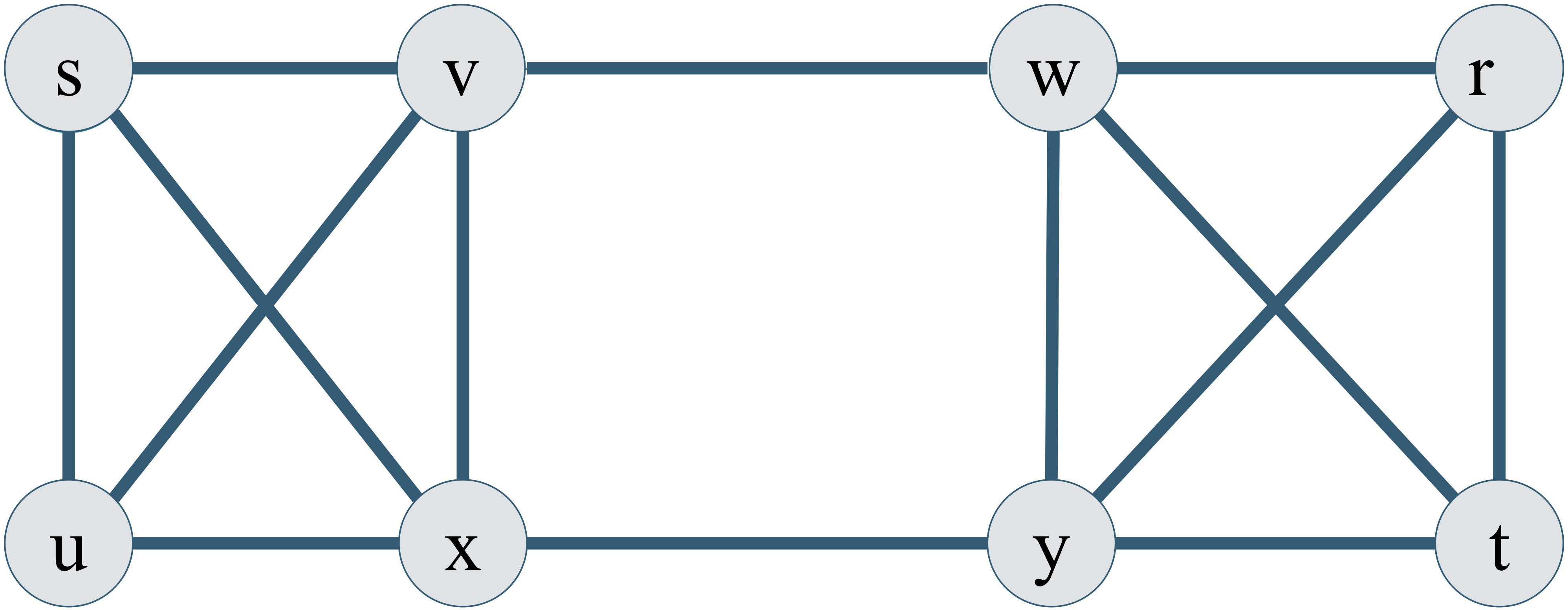
- The min-cut corresponds to the area in the graph that is “least dense” with respect to the number of edges
- If we can minimize the number of edges while ensuring that the “most dense” areas stay the most dense, we are able to transform this graph to a smaller graph (and smaller graphs are easier to deal with!)

What is a Contraction

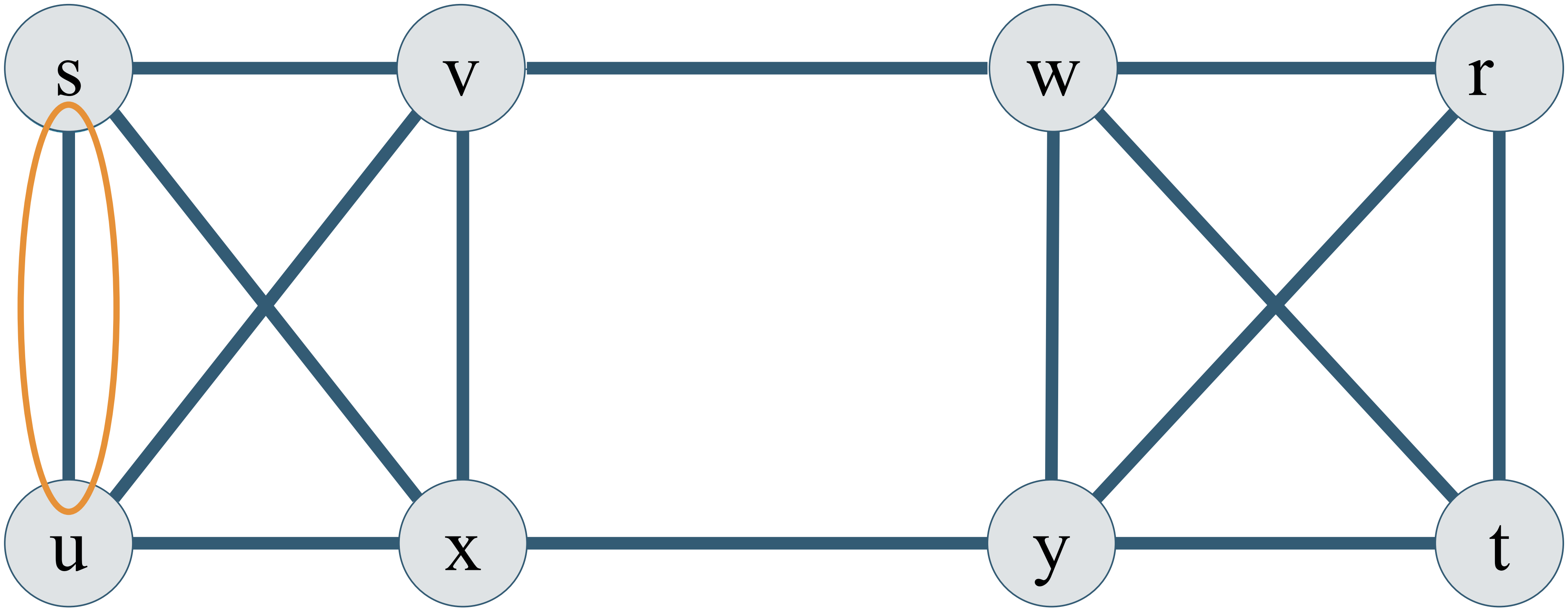
- Contractions: Merging the endpoints (u,v) of an edge into a supernode, reattach edges that were attached to u,v to the supernode
- We allow multi-edges but do not allow self-loops



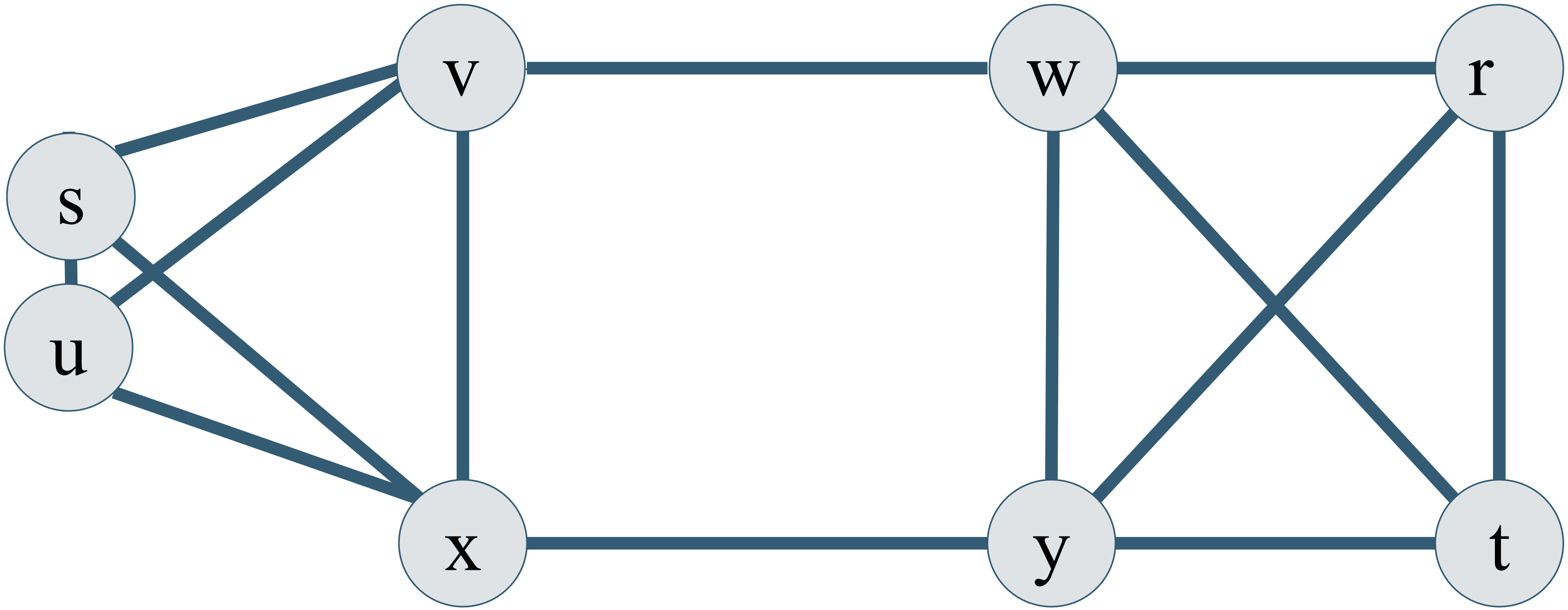
Contraction Example



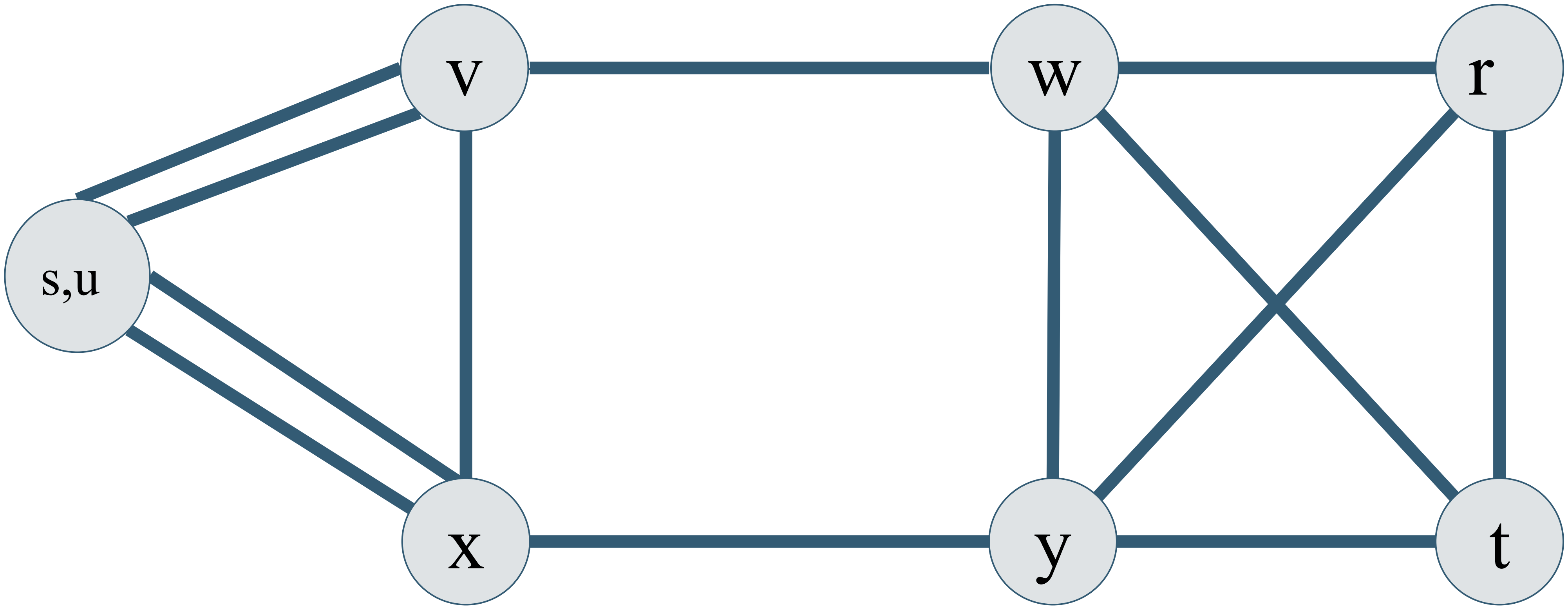
Contraction Example



Contraction Example



Contraction Example



Algorithm Basics

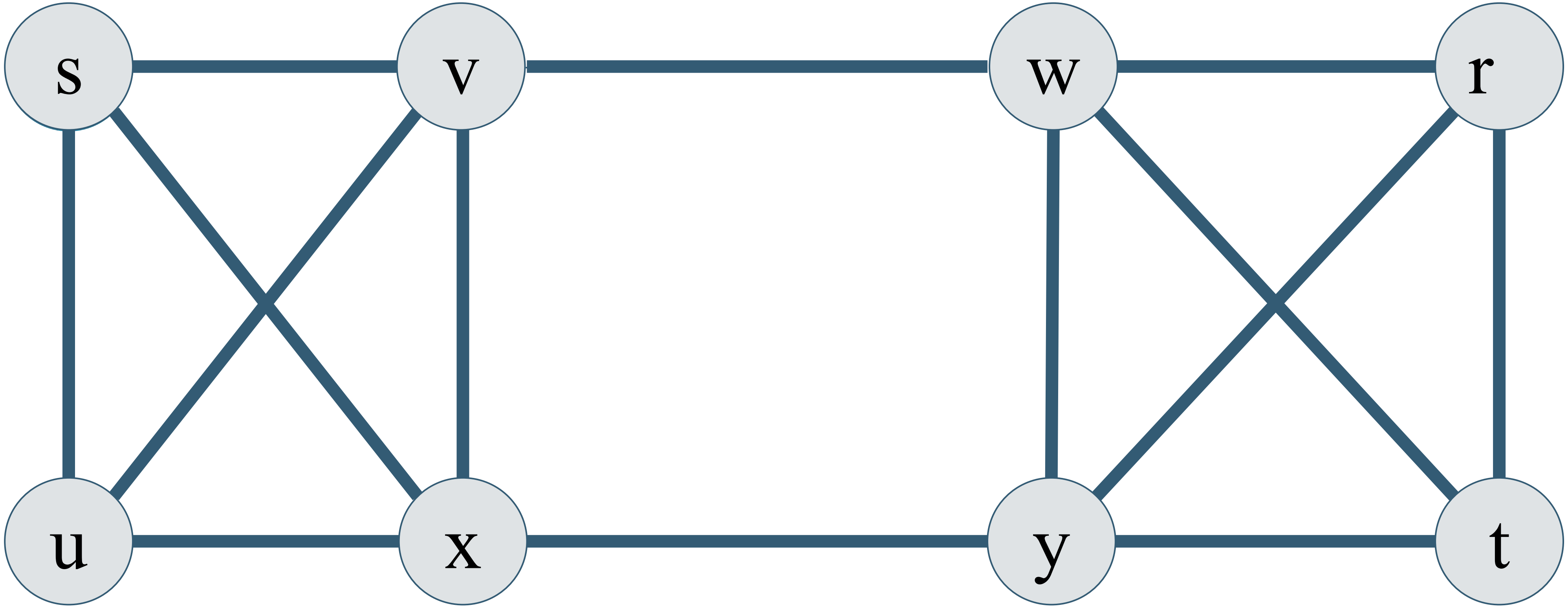
- Choose an edge uniformly at random to contract
- Repeat this until you have only two supernodes, these two supernodes represent the two halves of *a* cut (if we're lucky this is the minimum cut)
- The size of the cut is the number of edges between the two supernodes

Pseudocode

```
int FindMinCut(Edge[1,...,e], Vertex[1,...,v])
   while ( // there are more than two vertices
      edge -> Choose edge randomly from the list
      ContractEdge(edge)
   Return number of edges between the vertices

void ContractEdge(Edge e) // e.u = one vertex of the edge, e.v =
second vertex
   Create new vertex: SuperNode
   Reattach all edges from e.u and e.v to SuperNode
   Delete e
```

Karger's Algorithm Example

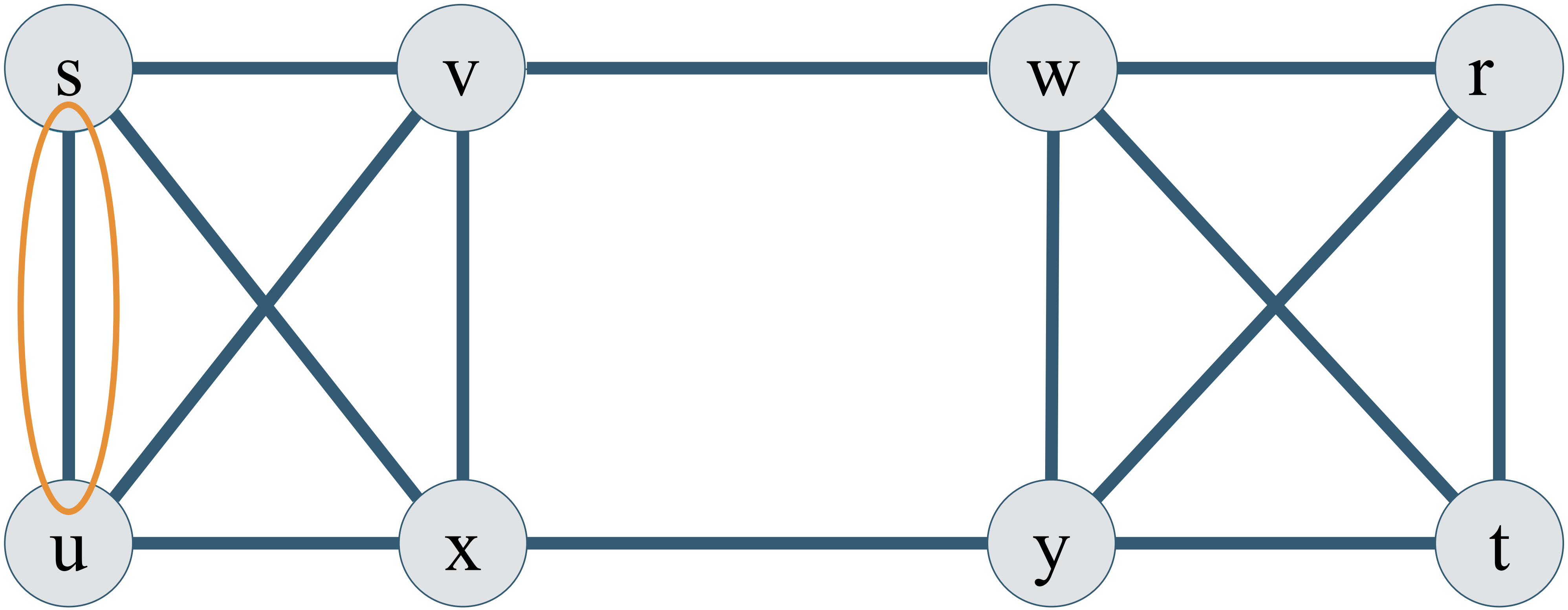


Edges:

- 1 – sv
- 2– su
- 3– sx
- 4 – vx
- 5 – vu
- 6 – ux
- 7 – vw
- 8 – xy
- 9 – wr
- 10 – wt
- 11 – wy
- 12 – ry
- 13 – rt
- 14 – yt

Random Sequence of Numbers:

Karger's Algorithm Example

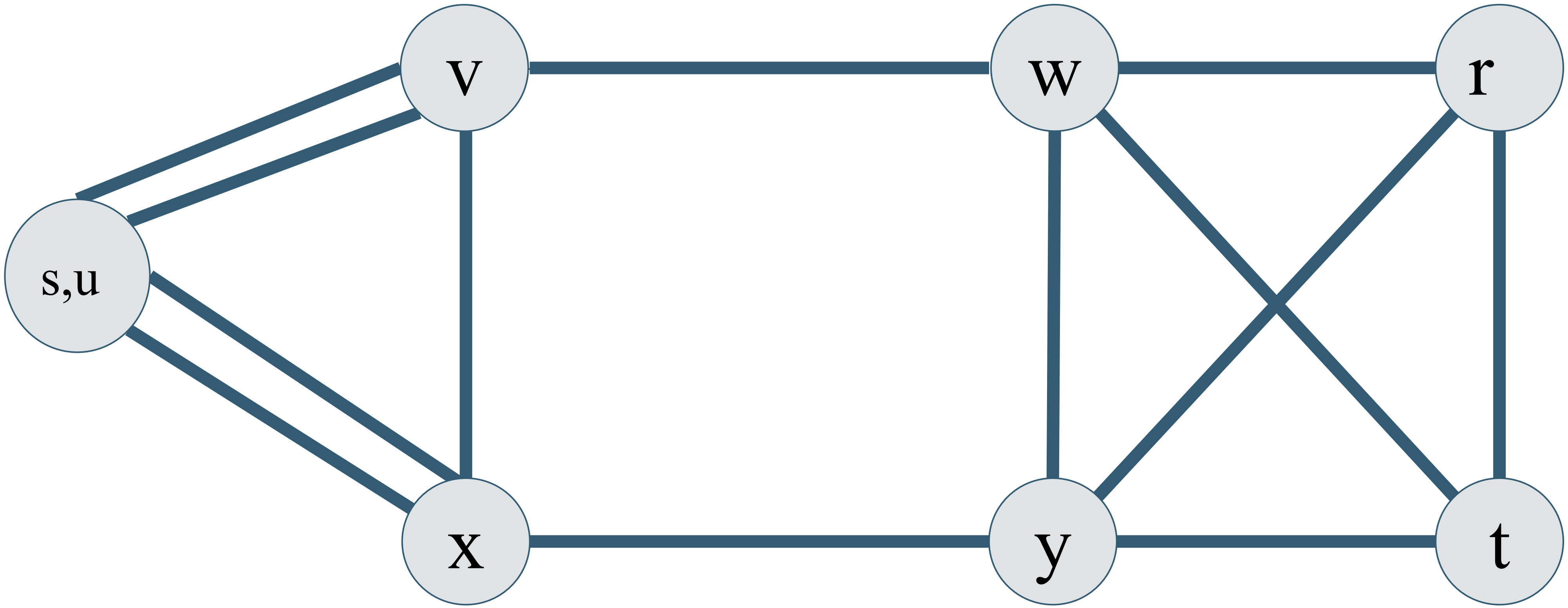


Edges:

- 1 – sv
- 2– su
- 3– sx
- 4 – vx
- 5 – vu
- 6 – ux
- 7 – vw
- 8 – xy
- 9 – wr
- 10 – wt
- 11 – wy
- 12 – ry
- 13 – rt
- 14 – yt

Random Sequence of Numbers: 2

Karger's Algorithm Example

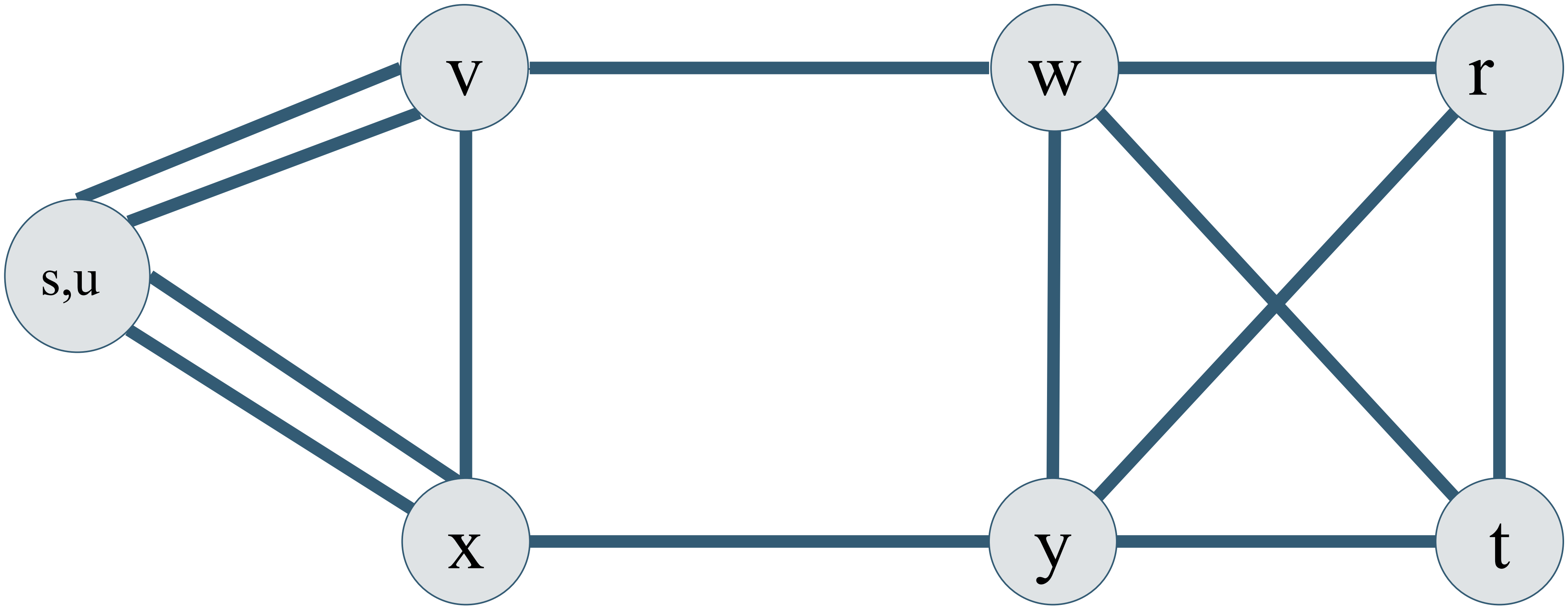


Edges:

- 1 – sv
- ~~2- su~~
- 3- sx
- 4 – vx
- 5 – vu
- 6 – ux
- 7 – vw
- 8 – xy
- 9 – wr
- 10 – wt
- 11 – wy
- 12 – ry
- 13 – rt
- 14 – yt

Random Sequence of Numbers: 2

Karger’s Algorithm Example

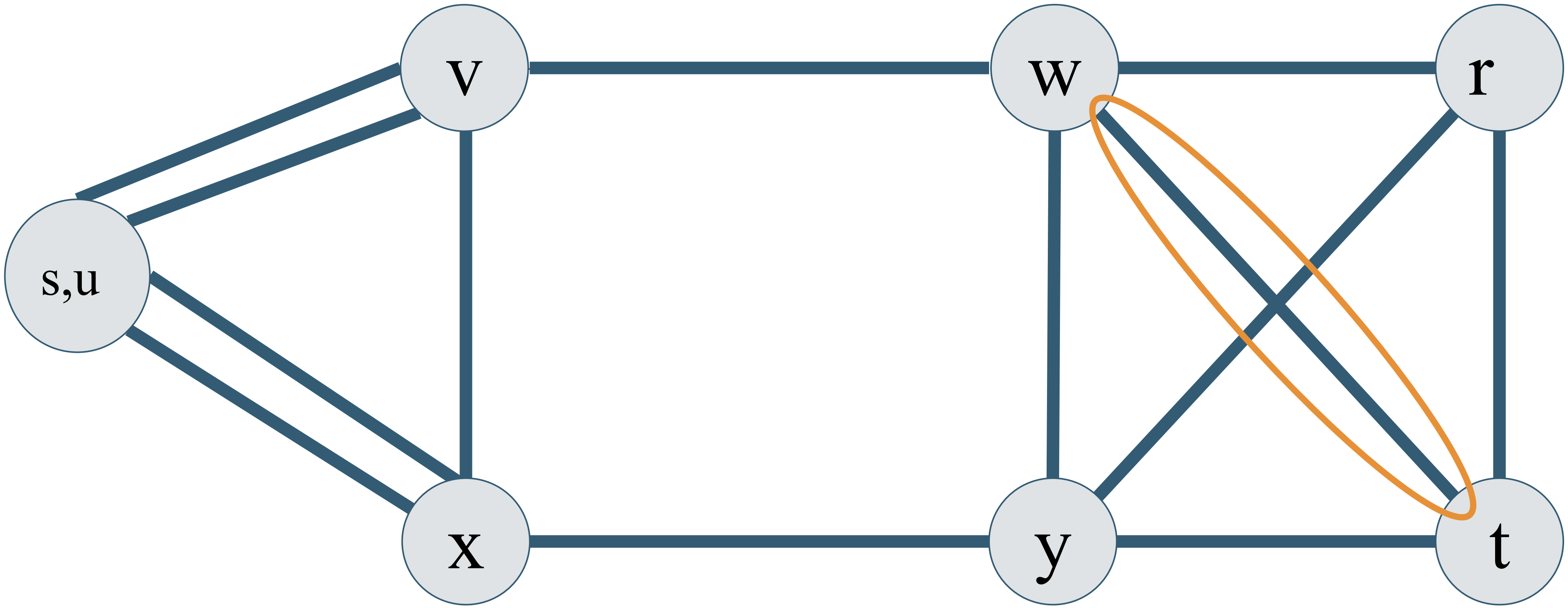


Edges:

- 1 – (us)v
- ~~2– su~~
- 3– (us)x
- 4 – vx
- 5 – v(us)
- 6 – (us)x
- 7 – vw
- 8 – xy
- 9 – wr
- 10 – wt
- 11 – wy
- 12 – ry
- 13 – rt
- 14 – yt

Random Sequence of Numbers: 2

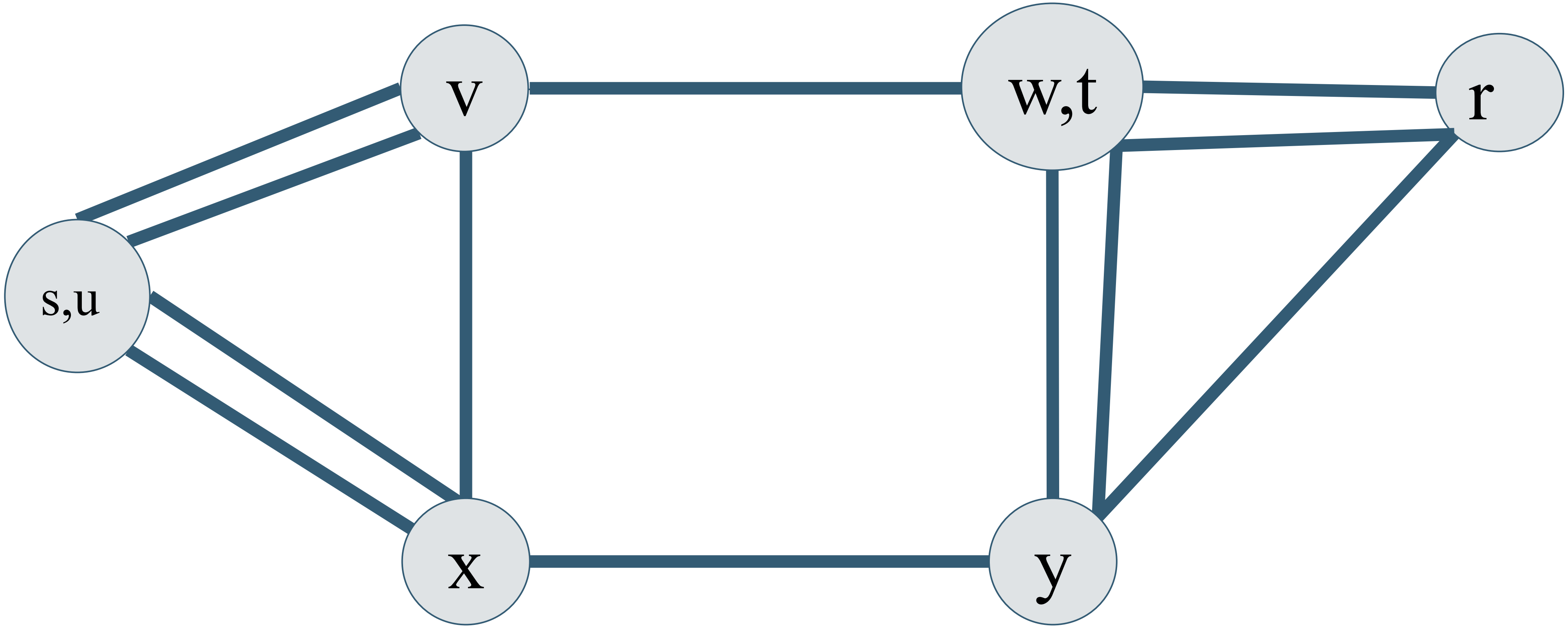
Karger's Algorithm Example



Edges:
- 1 – (us)v
- ~~2– su~~
- 3– (us)x
- 4 – vx
- 5 – v(us)
- 6 – (us)x
- 7 – vw
- 8 – xy
- 9 – wr
- 10 – wt
- 11 – wy
- 12 – ry
- 13 – rt
- 14 – yt

Random Sequence of Numbers: 2, 10

Karger's Algorithm Example

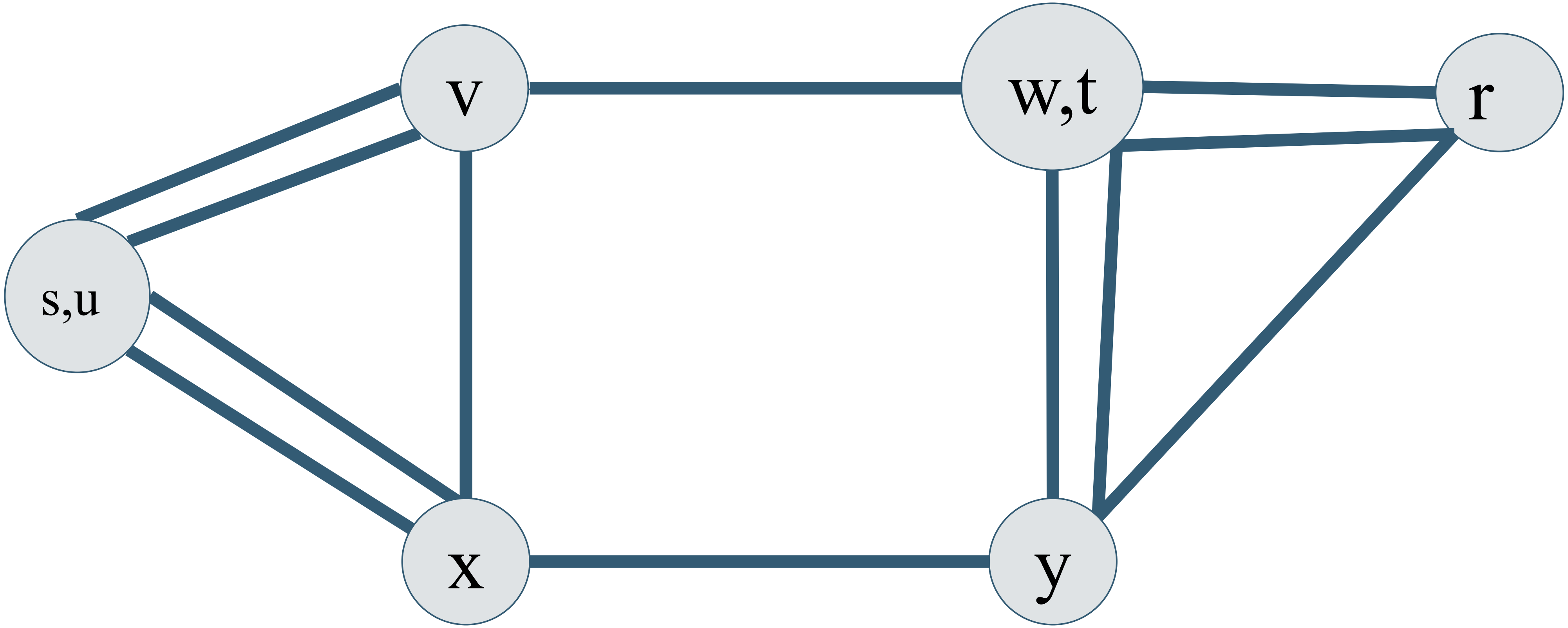


Edges:

- 1 – (us)v
- ~~2- su~~
- 3- (us)x
- 4 – vx
- 5 – v(us)
- 6 – (us)x
- 7 – vw
- 8 – xy
- 9 – wr
- ~~10 – wt~~
- 11 – wy
- 12 – ry
- 13 – rt
- 14 – yt

Random Sequence of Numbers: 2, 10

Karger's Algorithm Example

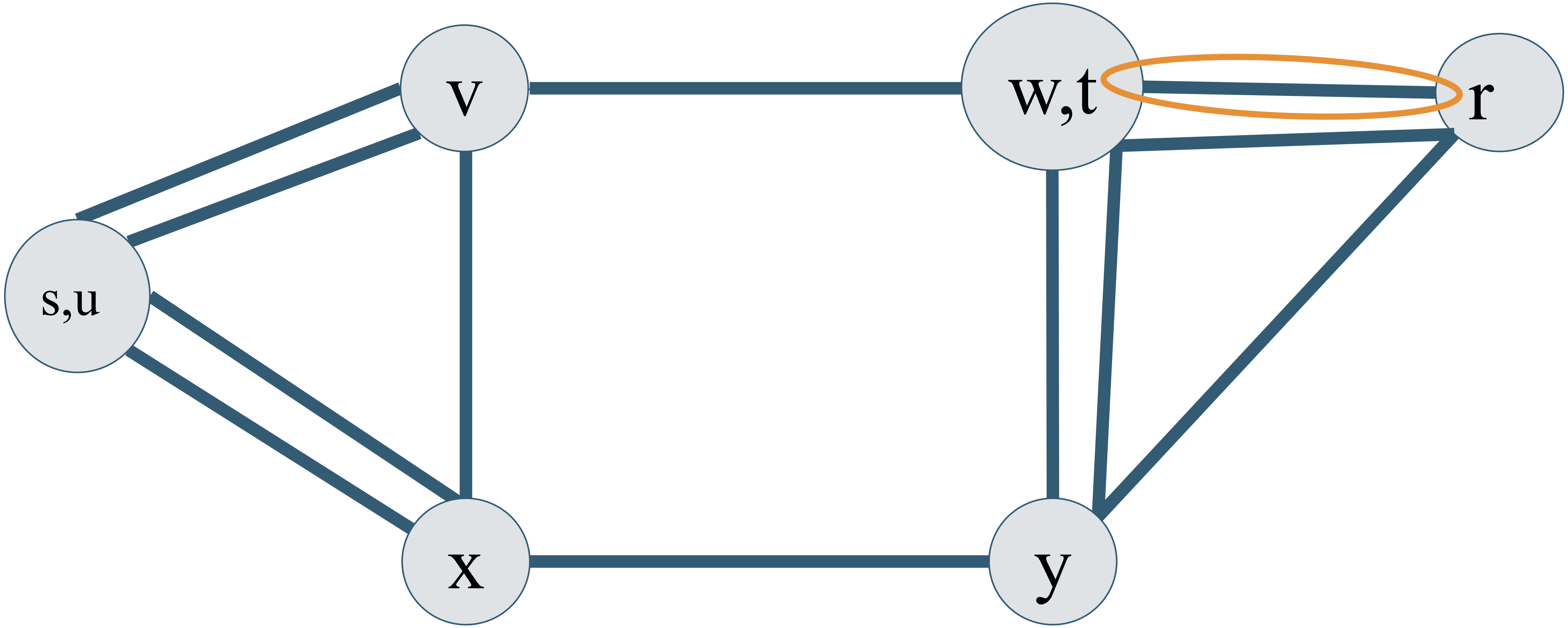


Edges:

- 1 – (us)v
- ~~2- su~~
- 3- (us)x
- 4 – vx
- 5 – v(us)
- 6 – (us)x
- 7 – v(wt)
- 8 – xy
- 9 – (wt)r
- ~~10 – wt~~
- 11 – (wt)y
- 12 – ry
- 13 – r(wt)
- 14 – y(wt)

Random Sequence of Numbers: 2, 10

Karger's Algorithm Example

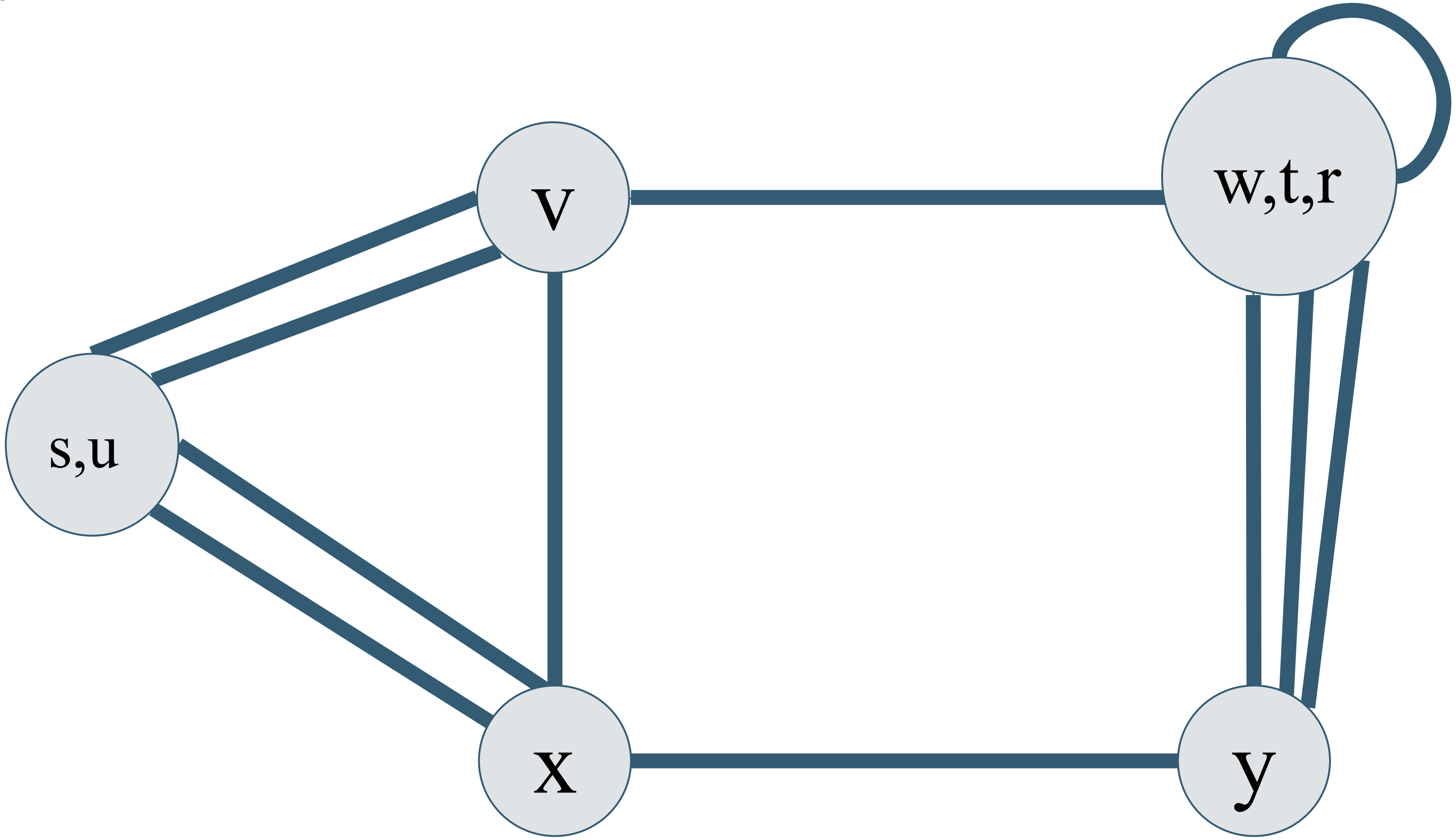


Edges:

- 1 – (us)v
- ~~2– su~~
- 3– (us)x
- 4 – vx
- 5 – v(us)
- 6 – (us)x
- 7 – v(wt)
- 8 – xy
- 9 – (wt)r
- ~~10 – wt~~
- 11 – (wt)y
- 12 – ry
- 13 – r(wt)
- 14 – y(wt)

Random Sequence of Numbers: 2, 10, 9

Karger's Algorithm Example

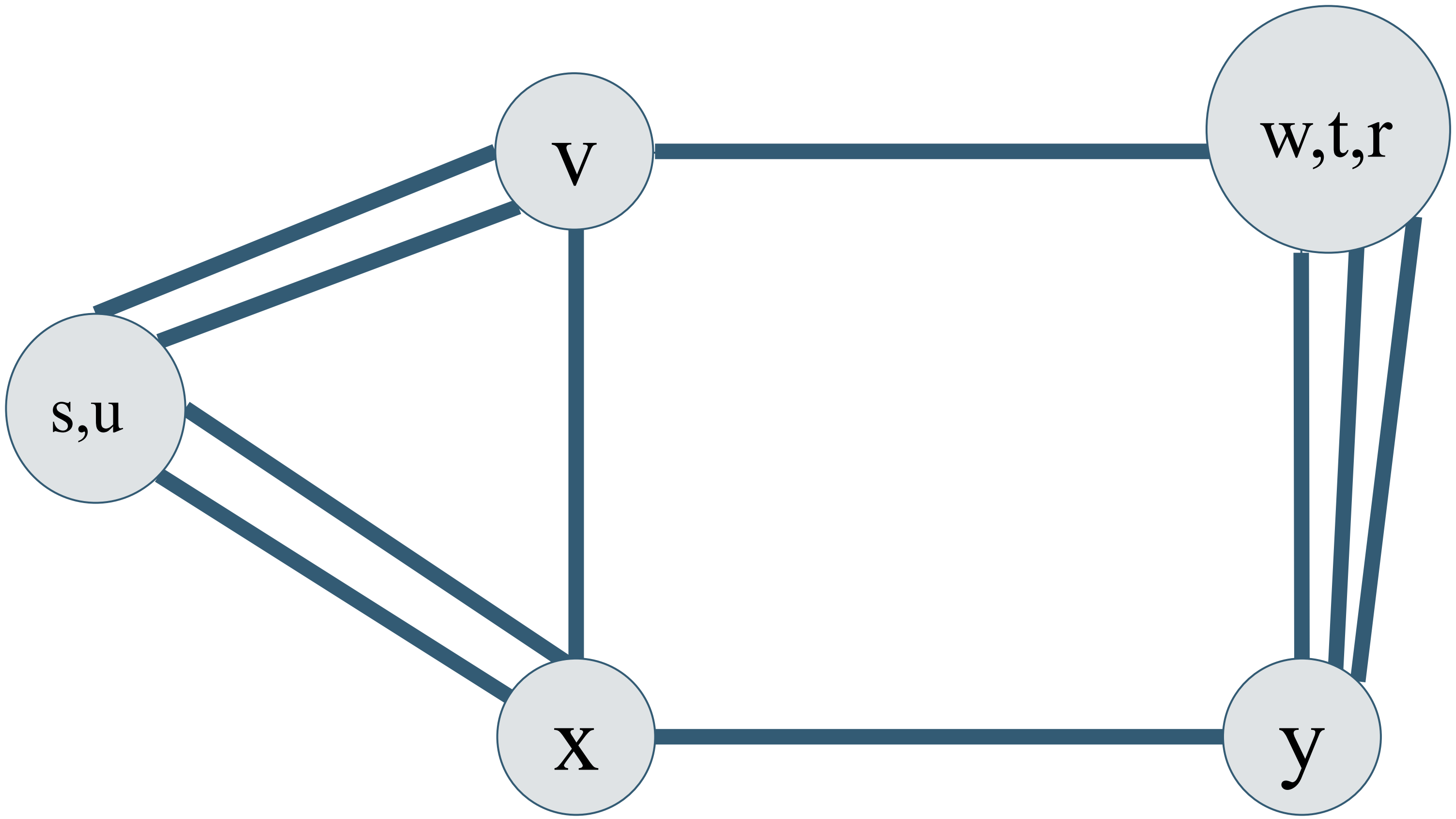


Edges:

- 1 – (us)v
- ~~2 – su~~
- 3– (us)x
- 4 – vx
- 5 – v(us)
- 6 – (us)x
- 7 – v(wt)
- 8 – xy
- ~~9 – (wt)r~~
- ~~10 – wt~~
- 11 – (wt)y
- 12 – ry
- 13 – r(wt)
- 14 – y(wt)

Random Sequence of Numbers: 2, 10, 9

Karger's Algorithm Example

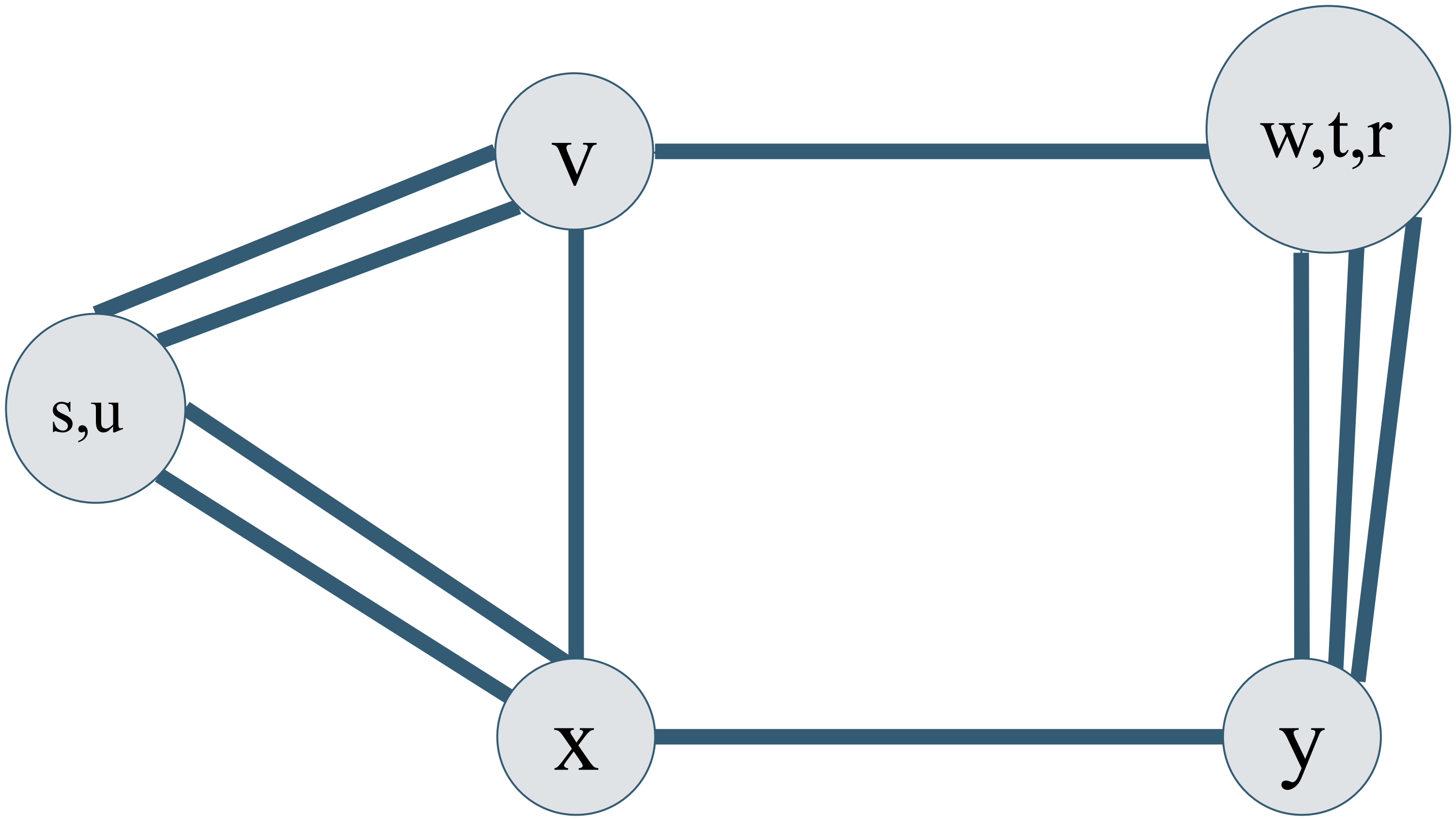


Edges:

- 1 – (us)v
- ~~2- su~~
- 3- (us)x
- 4 – vx
- 5 – v(us)
- 6 – (us)x
- 7 – v(wt)
- 8 – xy
- ~~9 – (wt)r~~
- ~~10 – wt~~
- 11 – (wt)y
- 12 – ry
- ~~13 – r(wt)~~
- 14 – y(wt)

Random Sequence of Numbers: 2, 10, 9

Karger's Algorithm Example

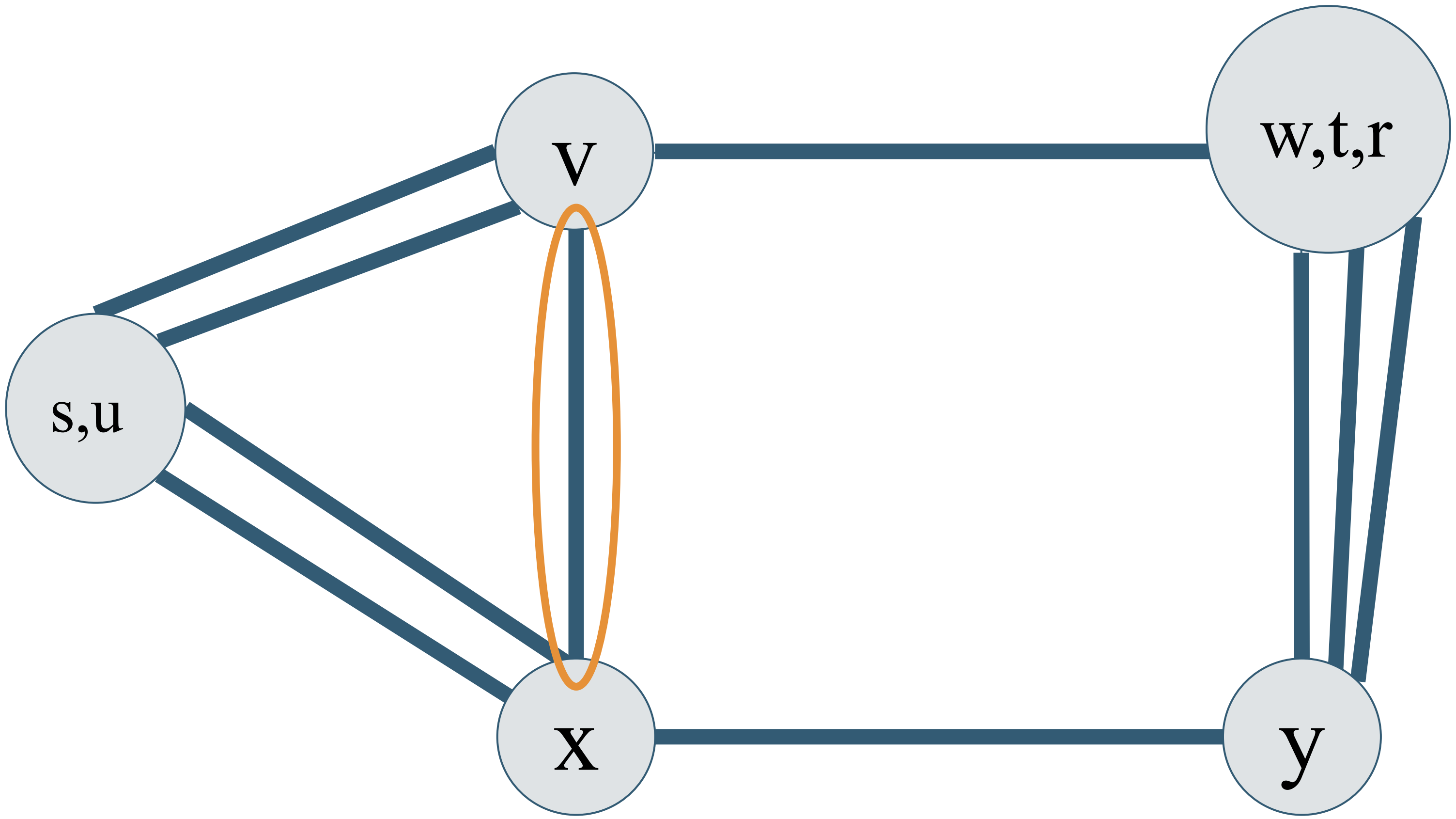


Edges:

- 1 – (us)v
- ~~2- su~~
- 3- (us)x
- 4 – vx
- 5 – v(us)
- 6 – (us)x
- 7 – v(wtr)
- 8 – xy
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- 11 – (wtr)y
- 12 – (wtr)y
- ~~13 – r(wt)~~
- 14 – y(wtr)

Random Sequence of Numbers: 2, 10, 9

Karger's Algorithm Example

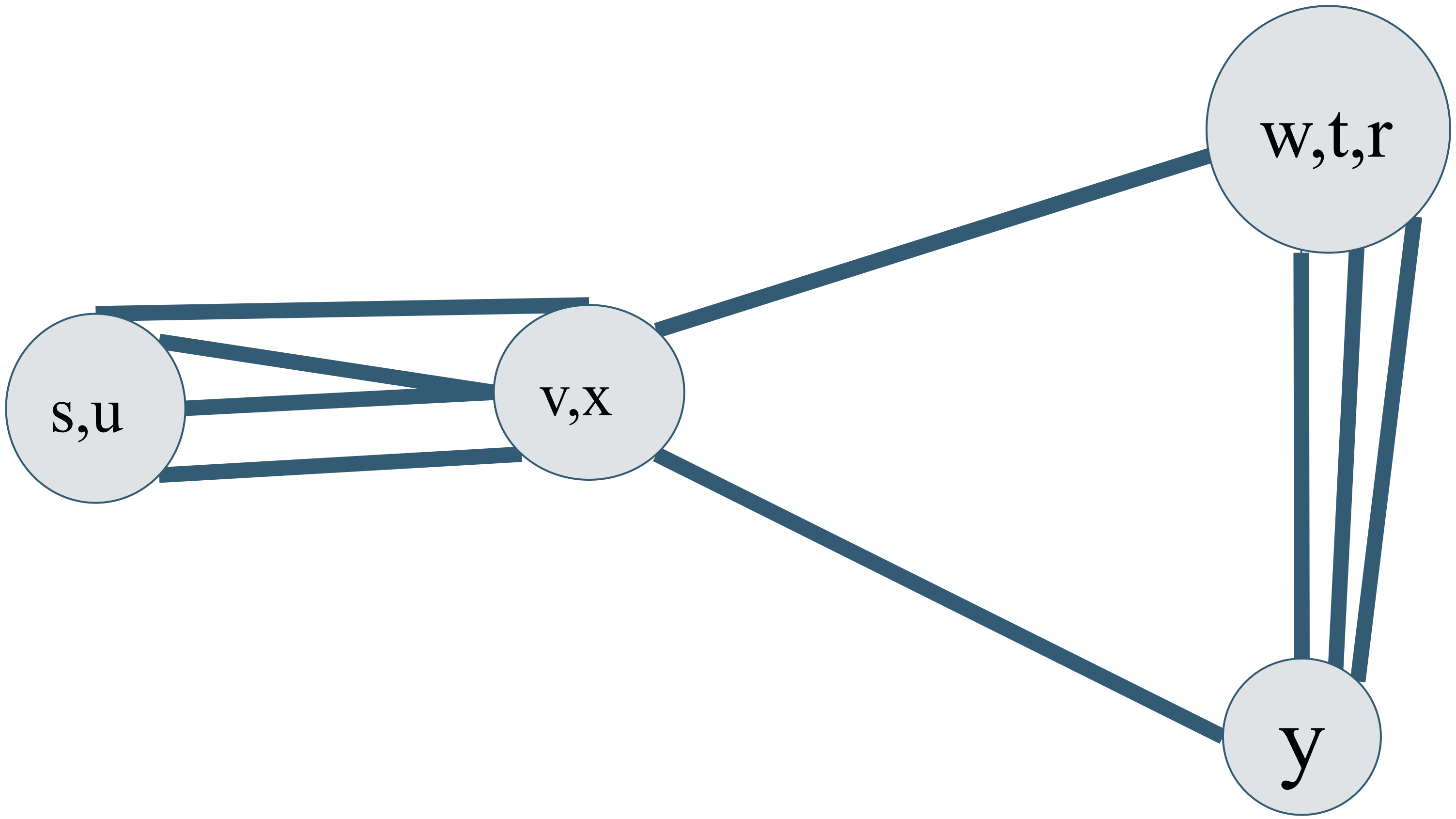


Edges:

- 1 – (us)v
- ~~2– su~~
- 3– (us)x
- 4 – vx
- 5 – v(us)
- 6 – (us)x
- 7 – v(wtr)
- 8 – xy
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- 11 – (wtr)y
- 12 – (wtr)y
- ~~13 – r(wt)~~
- 14 – y(wtr)

Random Sequence of Numbers: 2, 10, 9, 4

Karger's Algorithm Example

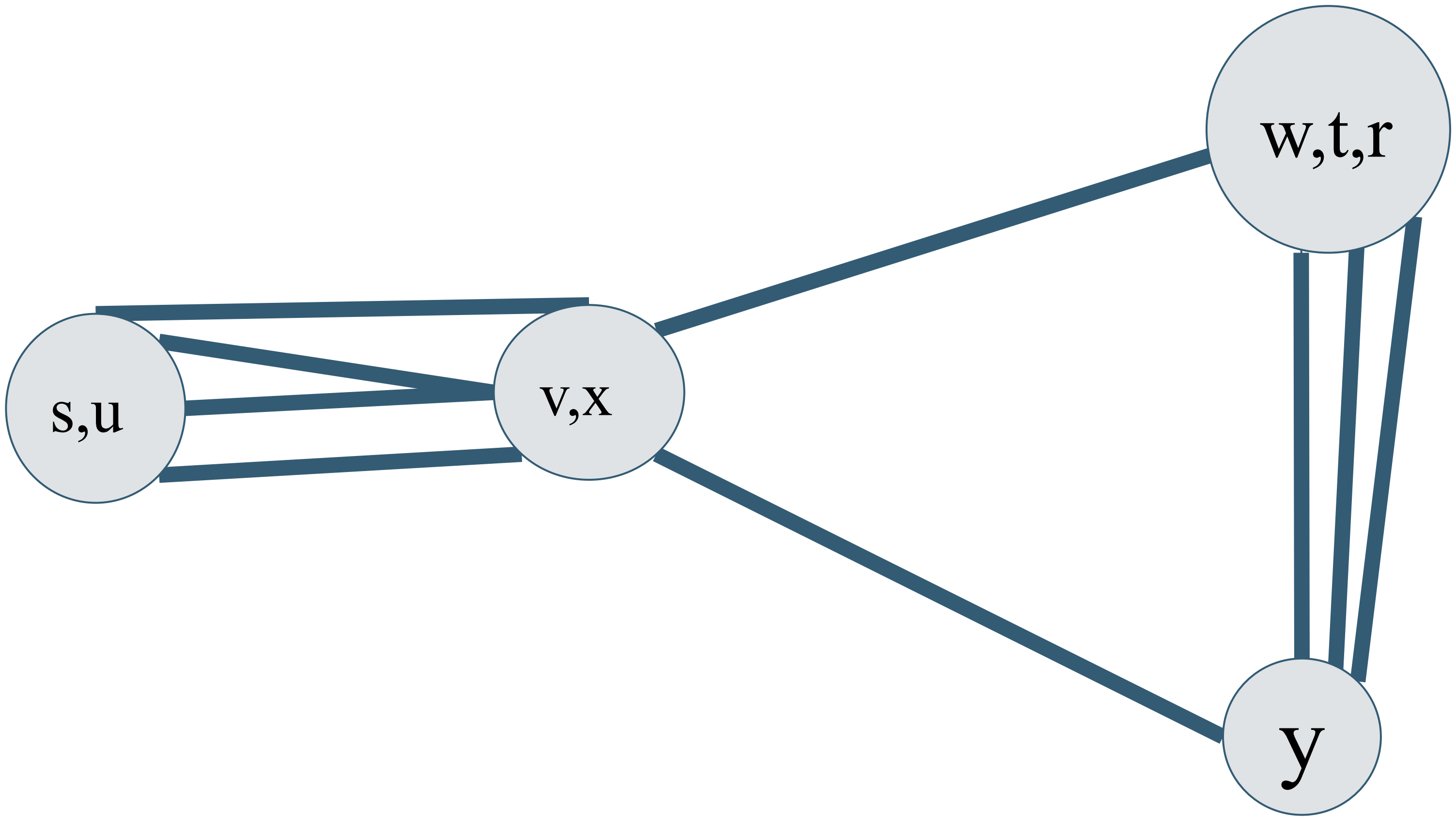


Edges:

- 1 – (us)v
- ~~2- su~~
- 3- (us)x
- ~~4 – vx~~
- 5 – v(us)
- 6 – (us)x
- 7 – v(wtr)
- 8 – xy
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- 11 – (wtr)y
- 12 – (wtr)y
- ~~13 – r(wt)~~
- 14 – y(wtr)

Random Sequence of Numbers: 2, 10, 9, 4

Karger's Algorithm Example

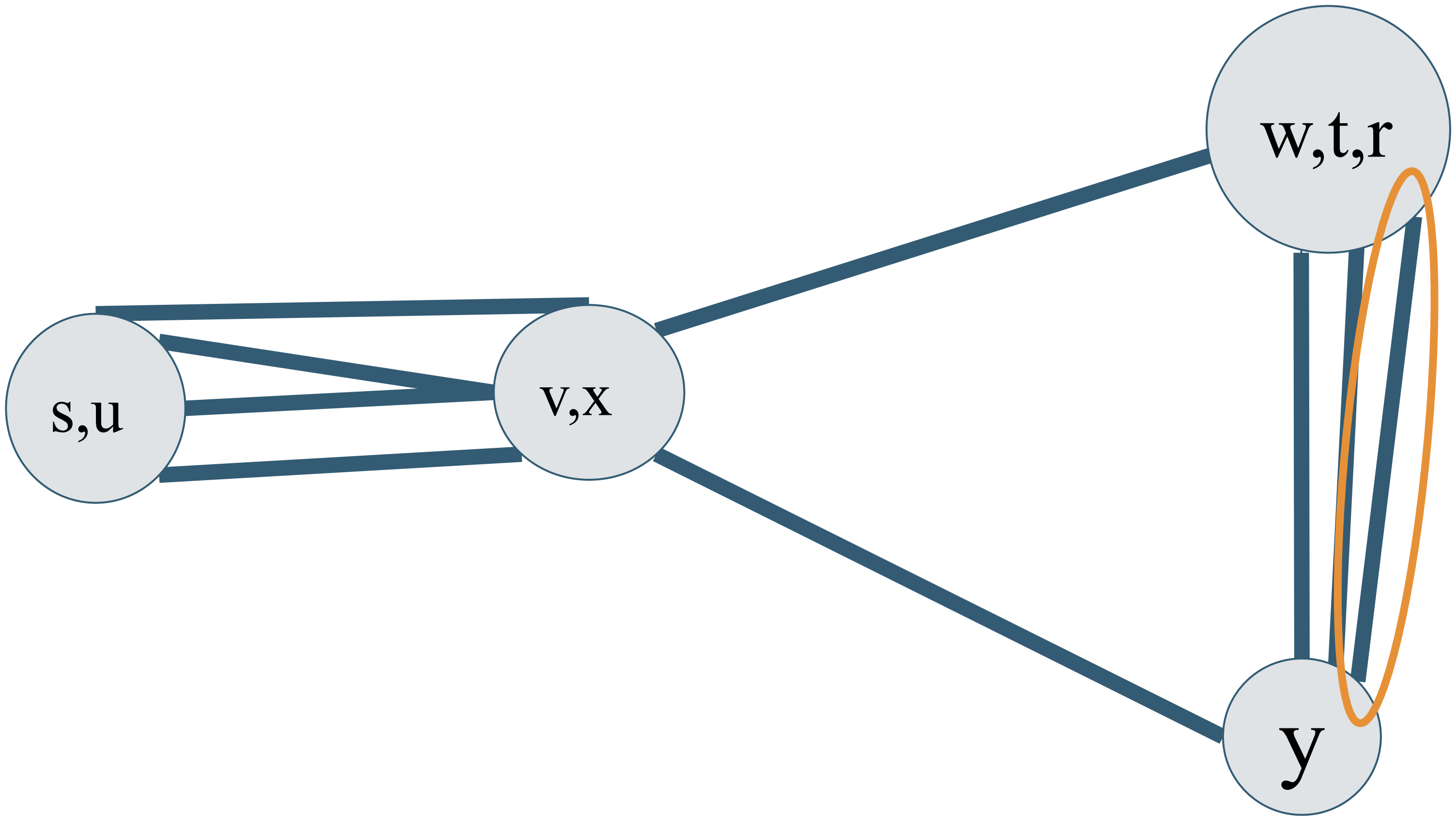


Edges:

- 1 – (us)(vx)
- ~~2 - su~~
- 3– (us)(vx)
- ~~4 - vx~~
- 5 – (vx)(us)
- 6 – (us)(vx)
- 7 – (vx)(wtr)
- 8 – (vx)y
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- 11 – (wtr)y
- 12 – (wtr)y
- ~~13 – r(wt)~~
- 14 – y(wtr)

Random Sequence of Numbers: 2, 10, 9, 4

Karger's Algorithm Example

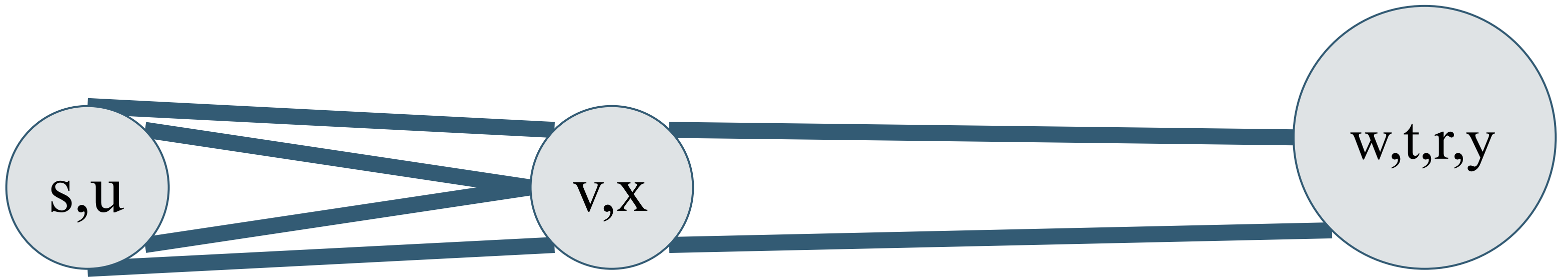


Edges:

- 1 – (us)(vx)
- ~~2- su~~
- 3– (us)(vx)
- ~~4 – vx~~
- 5 – (vx)(us)
- 6 – (us)(vx)
- 7 – (vx)(wtr)
- 8 – (vx)y
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- 11 – (wtr)y
- 12 – (wtr)y
- ~~13 – r(wt)~~
- 14 – y(wtr)

Random Sequence of Numbers: 2, 10, 9, 4, 14

Karger's Algorithm Example

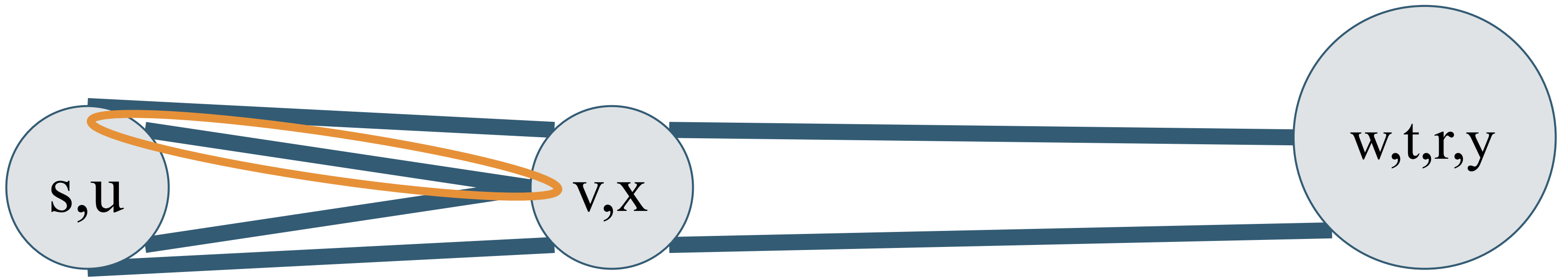


Edges:

- 1 – (us)(vx)
- ~~2– su~~
- 3– (us)(vx)
- ~~4 – vx~~
- 5 – (vx)(us)
- 6 – (us)(vx)
- 7 – (vx)(wtry)
- 8 – (vx)(wtry)
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- ~~11 – (wtr)y~~
- ~~12 – (wtr)y~~
- ~~13 – r(wt)~~
- ~~14 – y(wtr)~~

Random Sequence of Numbers: 2, 10, 9, 4, 14

Karger's Algorithm Example



Edges:

- 1 – (us)(vx)
- ~~2– su~~
- 3– (us)(vx)
- ~~4 – vx~~
- 5 – (vx)(us)
- 6 – (us)(vx)
- 7 – (vx)(wtry)
- 8 – (vx)(wtry)
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- ~~11 – (wtr)y~~
- ~~12 – (wtr)y~~
- ~~13 – r(wt)~~
- ~~14 – y(wtr)~~

Random Sequence of Numbers: 2, 10, 9, 4, 14, 5

Karger's Algorithm Example

Edges:

- ~~1 – (us)(vx)~~
- ~~2– su~~
- ~~3– (us)(vx)~~
- ~~4 – vx~~
- ~~5 – (vx)(us)~~
- ~~6 – (us)(vx)~~
- 7 – (vxus)(wtry)
- 8 – (vxus)(wtry)
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- ~~11 – (wtr)y~~
- ~~12 – (wtr)y~~
- ~~13 – r(wt)~~
- ~~14 – y(wtr)~~

Random Sequence of Numbers: 2, 10, 9, 4, 14, 5

Karger's Algorithm Example

- Since we're left with only two nodes, we terminate the edge deletion process and count the number of edges between them.
- There are two edges so we conclude that the a cut is 2 and the min-cut *may be* 2

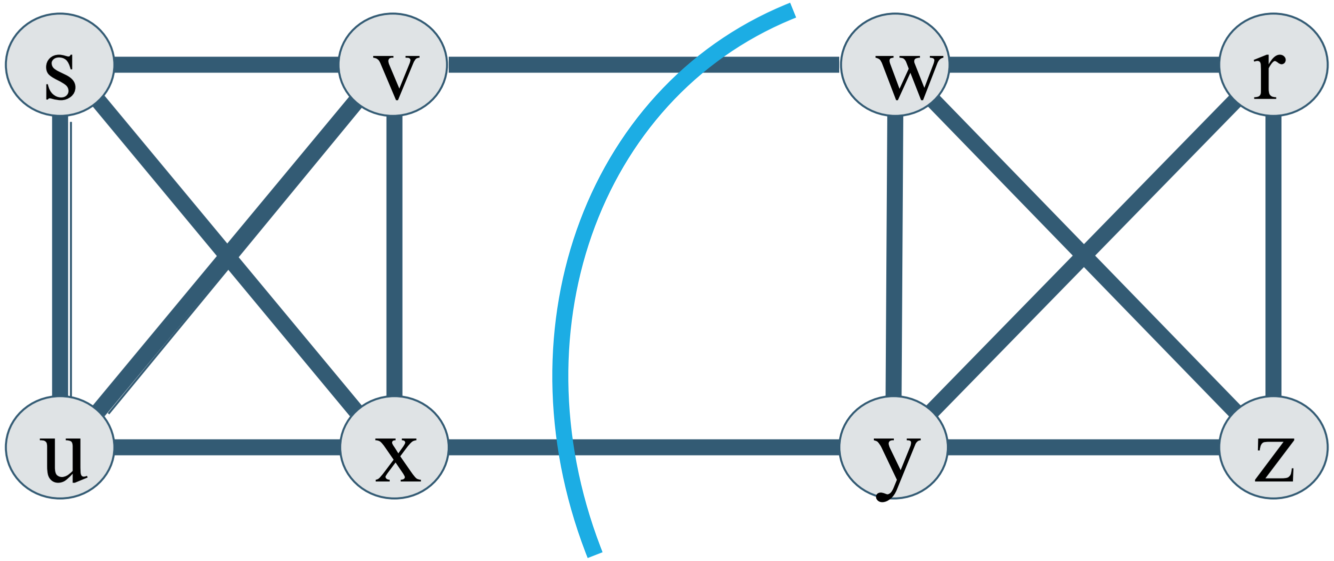
Edges:

- ~~1 – (us)(vx)~~
- ~~2– su~~
- ~~3– (us)(vx)~~
- ~~4 – vx~~
- ~~5 – (vx)(us)~~
- ~~6 – (us)(vx)~~
- 7 – (vxus)(wtry)
- 8 – (vxus)(wtry)
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- ~~11 – (wtr)y~~
- ~~12 – (wtr)y~~
- ~~13 – r(wt)~~
- ~~14 – y(wtr)~~

Random Sequence of Numbers: 2, 10, 9, 4, 14, 5

Karger's Algorithm Example

- This corresponds to the cut we saw earlier



Random Sequence of Numbers: 2, 10, 9, 4, 14, 5

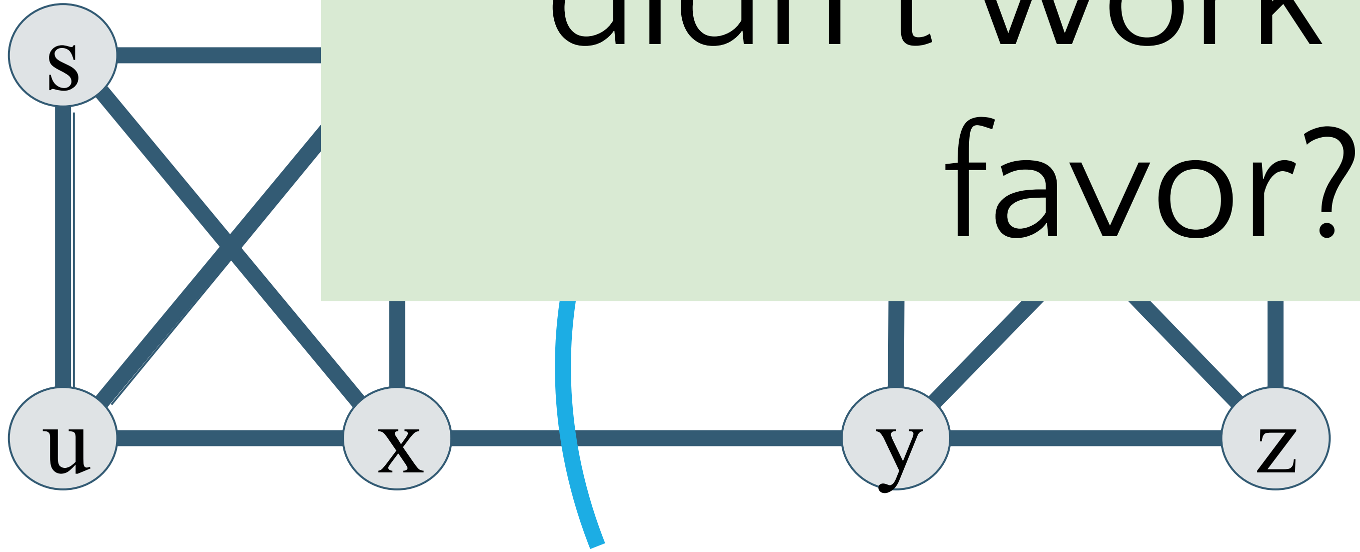
Edges:

- ~~1 – (us)(vx)~~
- ~~2 - su~~
- ~~3 - (us)(vx)~~
- ~~4 – vx~~
- ~~5 – (vx)(us)~~
- ~~6 – (us)(vx)~~
- 7 – (vxus)(wtry)
- 8 – (vxus)(wtry)
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- ~~11 – (wtr)y~~
- ~~12 – (wtr)y~~
- ~~13 – r(wt)~~
- ~~14 – y(wtr)~~

Karger's Algorithm Example

- This co
the cut
earlier

What if randomness didn't work in our favor?

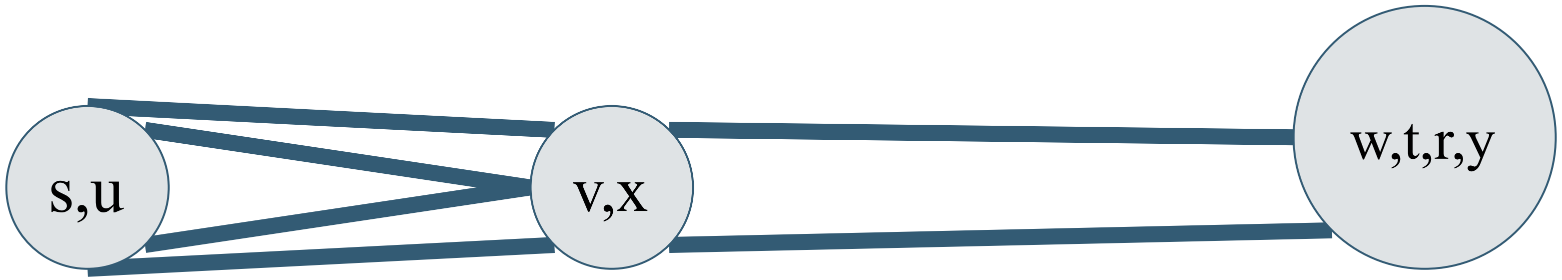


Random Sequence of Numbers: 2, 10, 9, 4, 14, 5

Edges:

- ~~1 – (us)(vx)~~
- ~~2 – su~~
- ~~3 – (us)(vx)~~
- ~~4 – vx~~
- ~~5 – (vx)(us)~~
- ~~6 – (us)(vx)~~
- 7 – (vxus)(wtry)
- 8 – (vxus)(wtry)
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- ~~11 – (wtr)y~~
- ~~12 – (wtr)y~~
- ~~13 – r(wt)~~
- ~~14 – y(wtr)~~

Karger's Algorithm Example 2

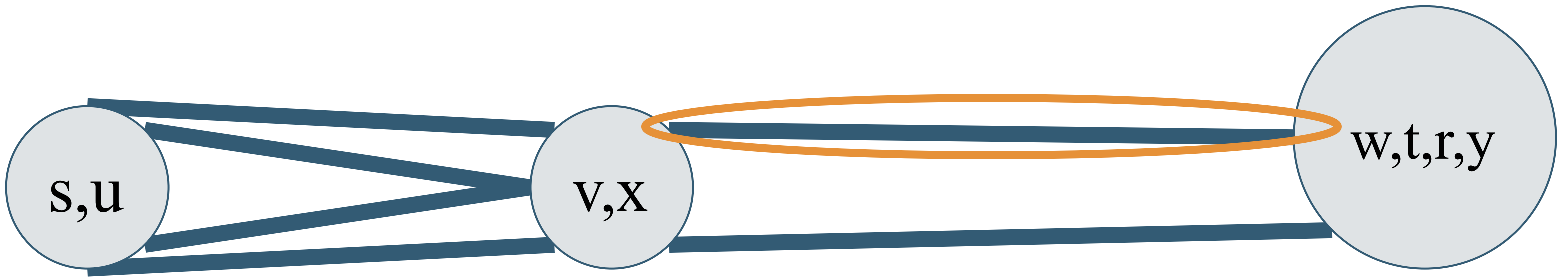


Edges:

- 1 – (us)(vx)
- ~~2 – su~~
- 3– (us)(vx)
- ~~4 – vx~~
- 5 – (vx)(us)
- 6 – (us)(vx)
- 7 – (vx)(wtry)
- 8 – (vx)(wtry)
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- ~~11 – (wtr)y~~
- ~~12 – (wtr)y~~
- ~~13 – r(wt)~~
- ~~14 – y(wtr)~~

Random Sequence of Numbers: 2, 10, 9, 4, 14

Karger's Algorithm Example 2

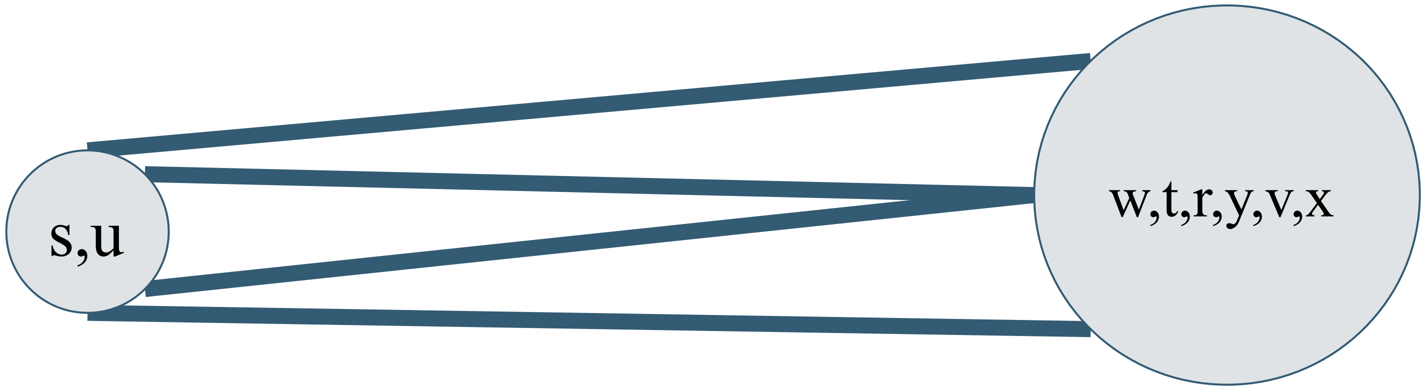


Edges:

- 1 – (us)(vx)
- ~~2 – su~~
- 3– (us)(vx)
- ~~4 – vx~~
- 5 – (vx)(us)
- 6 – (us)(vx)
- 7 – (vx)(wtry)
- 8 – (vx)(wtry)
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- ~~11 – (wtr)y~~
- ~~12 – (wtr)y~~
- ~~13 – r(wt)~~
- ~~14 – y(wtr)~~

Random Sequence of Numbers: 2, 10, 9, 4, 14, 8

Karger's Algorithm Example 2



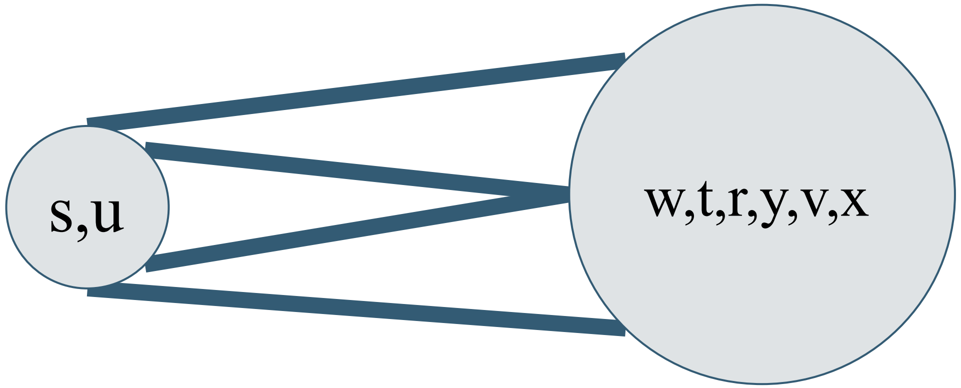
Edges:

- 1 – (us)(vx)
- ~~2– su~~
- 3– (us)(vx)
- ~~4 – vx~~
- 5 – (vx)(us)
- 6 – (us)(vx)
- ~~7 – (vx)(wtry)~~
- ~~8 – (vx)(wtry)~~
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- ~~11 – (wtr)y~~
- ~~12 – (wtr)y~~
- ~~13 – r(wt)~~
- ~~14 – y(wtr)~~

Random Sequence of Numbers: 2, 10, 9, 4, 14, 8

Karger's Algorithm Example 2

- Since we're left with only two nodes, we terminate the edge deletion process and count the number of edges between them.
- There are four edges so we conclude that the a cut is 4 and the min-cut *may be* 4

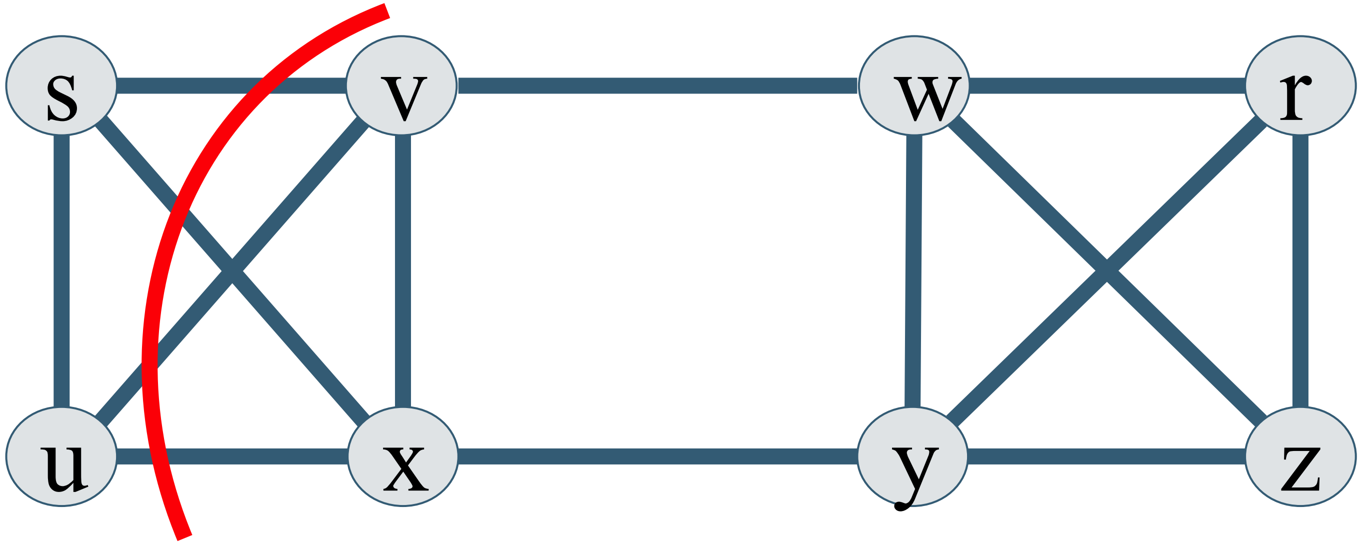


Edges:

- 1 – (us)(vx)
- ~~2– su~~
- 3– (us)(vx)
- ~~4 – vx~~
- 5 – (vx)(us)
- 6 – (us)(vx)
- ~~7 – (vx)(wtry)~~
- ~~8 – (vx)(wtry)~~
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- ~~11 – (wtr)y~~
- ~~12 – (wtr)y~~
- ~~13 – r(wt)~~
- ~~14 – y(wtr)~~

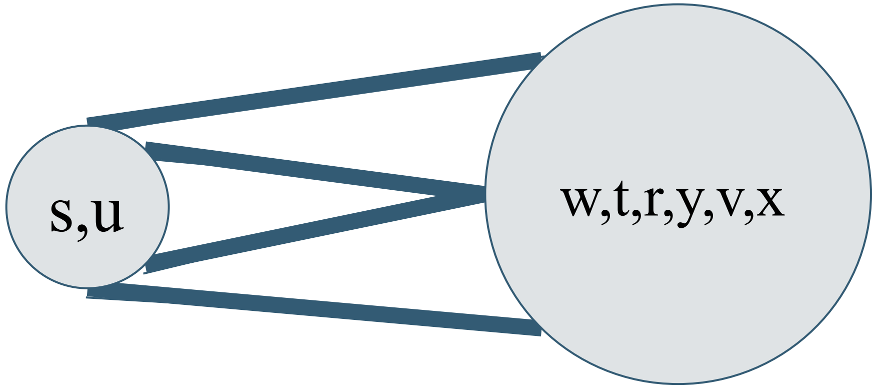
Random Sequence of Numbers: 2, 10, 9, 4, 14, 8

Karger's Algorithm Example 2



Edges:

- 1 – (us)(vx)
- ~~2– su~~
- 3– (us)(vx)
- ~~4 – vx~~
- 5 – (vx)(us)
- 6 – (us)(vx)
- ~~7 – (vx)(wtry)~~
- ~~8 – (vx)(wtry)~~
- ~~9 – (w,t)r~~
- ~~10 – wt~~
- ~~11 – (wtr)y~~
- ~~12 – (wtr)y~~
- ~~13 – r(wt)~~
- ~~14 – y(wtr)~~



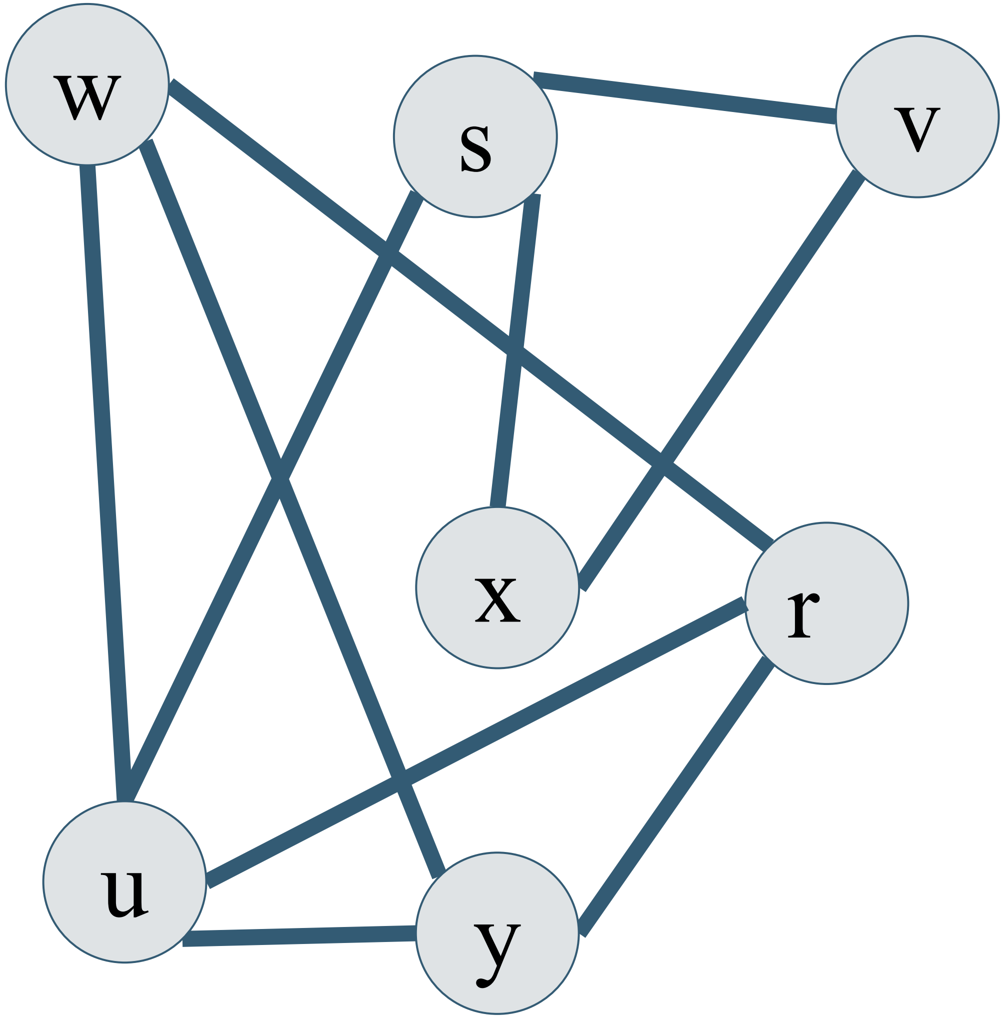
- Corresponds to the given cut
- Since we've seen an iteration of this algorithm with a smaller cut, we know 4 is not the minimum, but we don't know if 2 is the minimum

Random Sequence of Numbers: 2, 10, 9, 4, 14, 8

Intuition

- When we pick a random edge, it's more likely to come from somewhere in the graph with more edges.
- After every contraction, the min-cut stays minimum
- Since the min-cut is minimum, it's smaller than all possible other cuts, so probability of contracting it is smaller than contracting any other cut
- Maybe if we're unlucky in one iteration, we can run this algorithm enough times to ensure that there's some iteration in which we're lucky

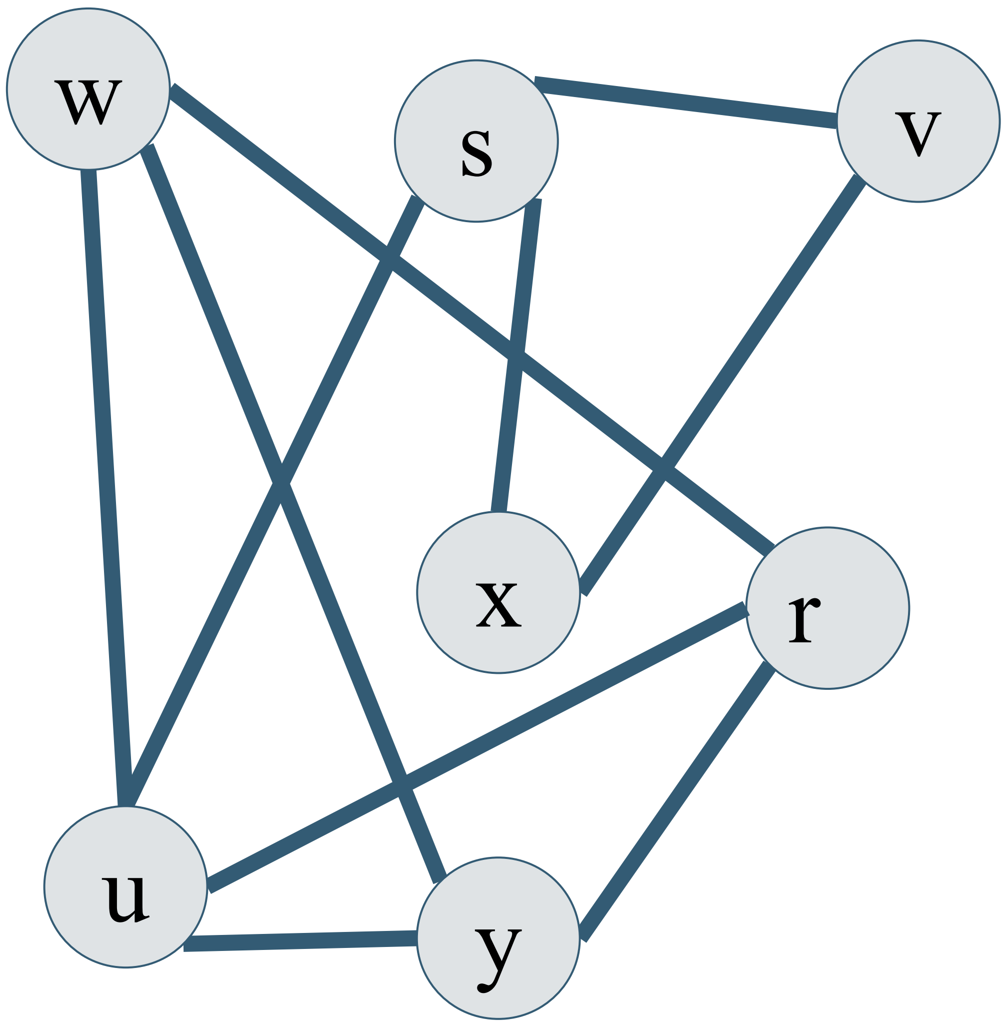
Run Karger's Algorithm yourself




```
int FindMinCut(Edge[1,...,e],
Vertex[1,...,v])
    while ( // there are more than two
vertices
        edge -> Choose edge randomly from
    the list
        ContractEdge(edge)
    Return number of edges between the
    vertices

void ContractEdge(Edge e) // e.u = one
vertex of the edge, e.v = second vertex
    Create new vertex: SuperNode
    Reattach all edges from e.u and e.v
to SuperNode
    Delete e
```

Run Karger's Algorithm yourself

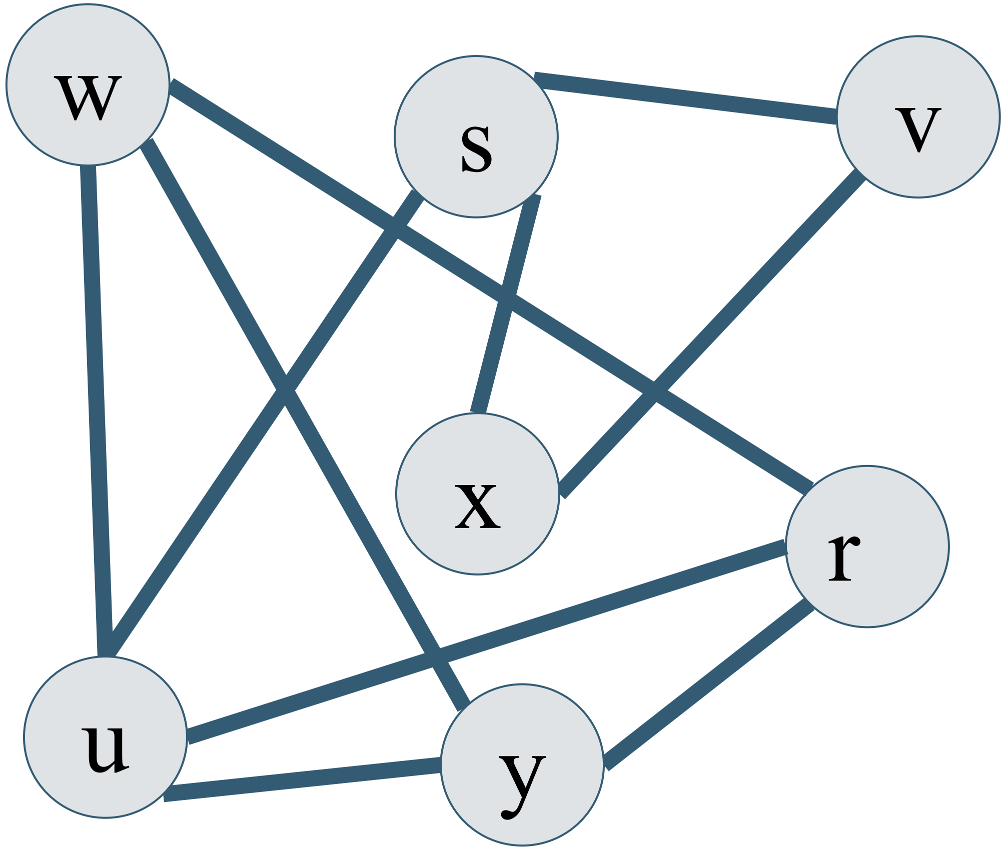



```
int FindMinCut(Edge[1,...,e], Vertex[1,...,v])
    while ( // there are more than two vertices
        edge -> Choose edge randomly from the
    list
        ContractEdge(edge)
    Return number of edges between the vertices

void ContractEdge(Edge e) // e.u = one vertex of
the edge, e.v = second vertex
    Create new vertex: SuperNode
    Reattach all edges from e.u and e.v to
SuperNode
    Delete e
```

- Work on running Karger's algorithm on this graph
- Use the dice, or phone/computer or brain to generate random numbers
- Compare your cut with your neighbors

Run Karger's Algorithm yourself: Run–Through

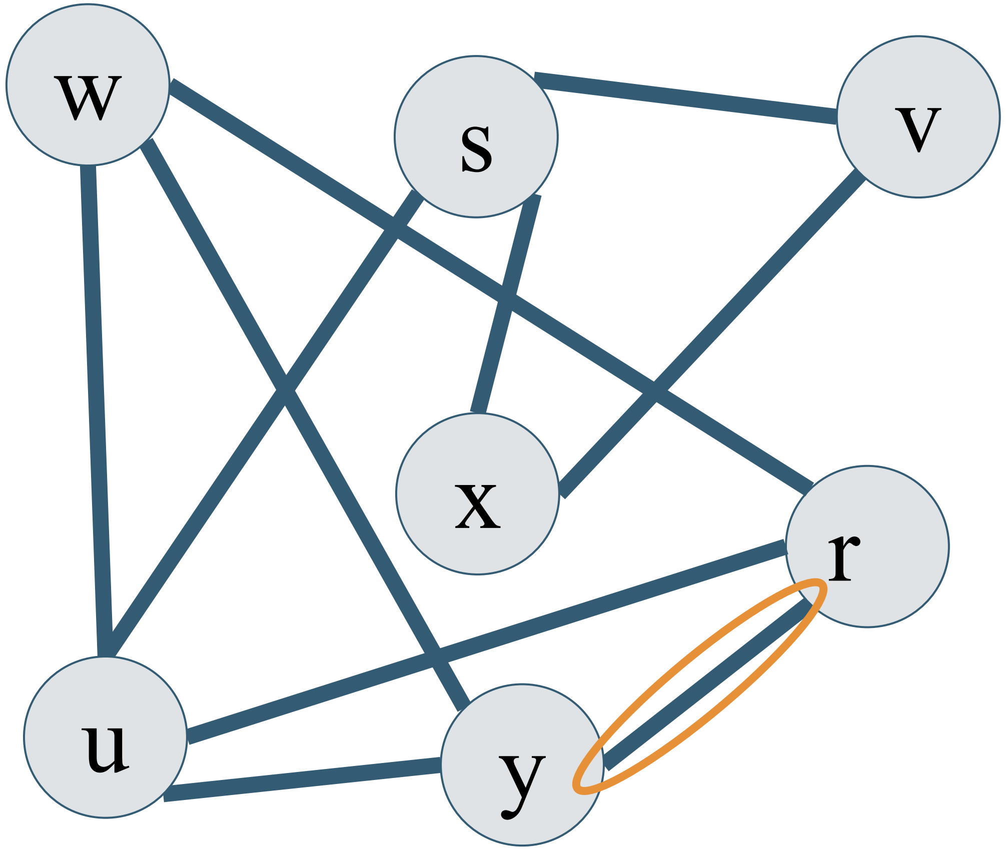


Edges:

- 1 – sv
- 2– vx
- 3– sx
- 4 – su
- 5 – uw
- 6 – wy
- 7 – yr
- 8 – rw
- 9 – uy
- 10 – ur

Random Sequence of Numbers:

Run Karger's Algorithm yourself: Run–Through

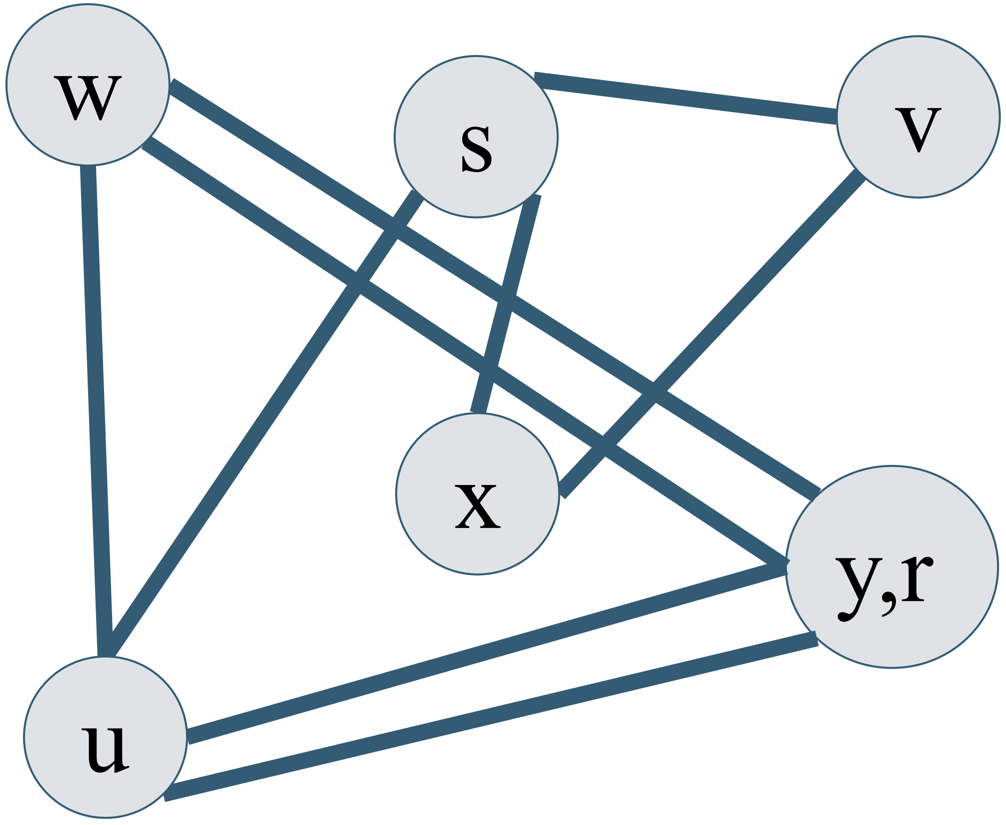


Edges:

- 1 – sv
- 2– vx
- 3– sx
- 4 – su
- 5 – uw
- 6 – wy
- 7 – yr
- 8 – rw
- 9 – uy
- 10 – ur

Random Sequence of Numbers: 7,

Run Karger's Algorithm yourself: Run-Through

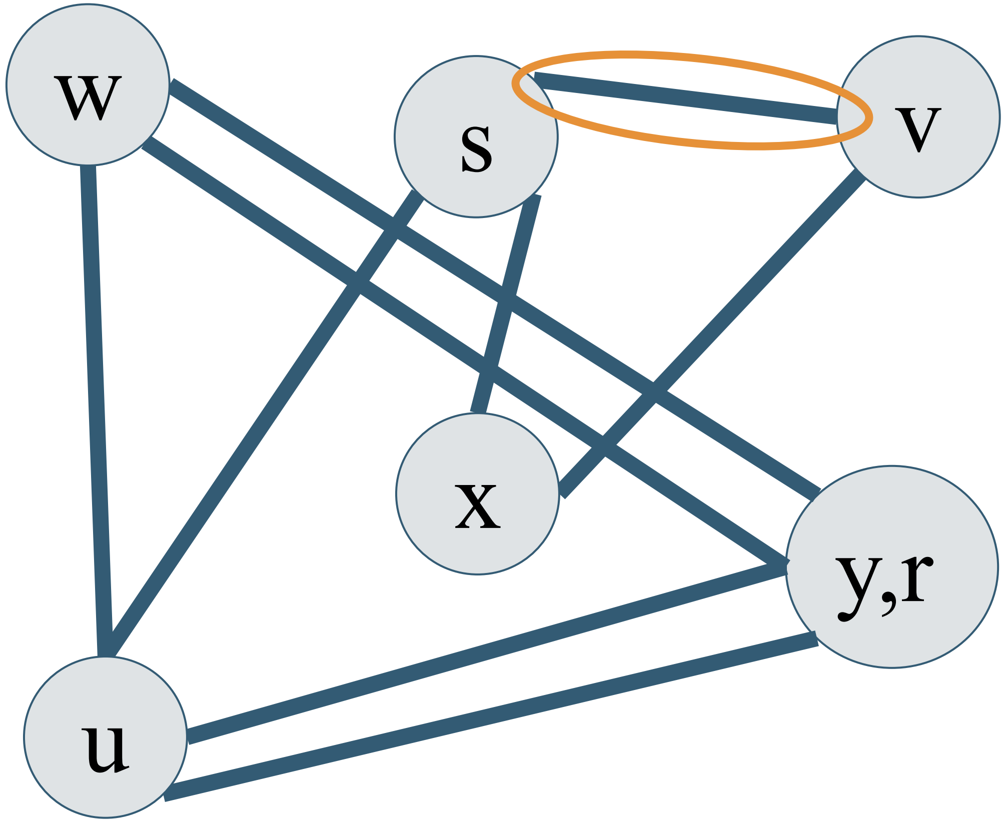


Edges:

- 1 – sv
- 2– vx
- 3– sx
- 4 – su
- 5 – uw
- 6 – w(yr)
- ~~7 – yr~~
- 8 – (yr)w
- 9 – u(yr)
- 10 – u(yr)

Random Sequence of Numbers: 7,

Run Karger's Algorithm yourself: Run-Through

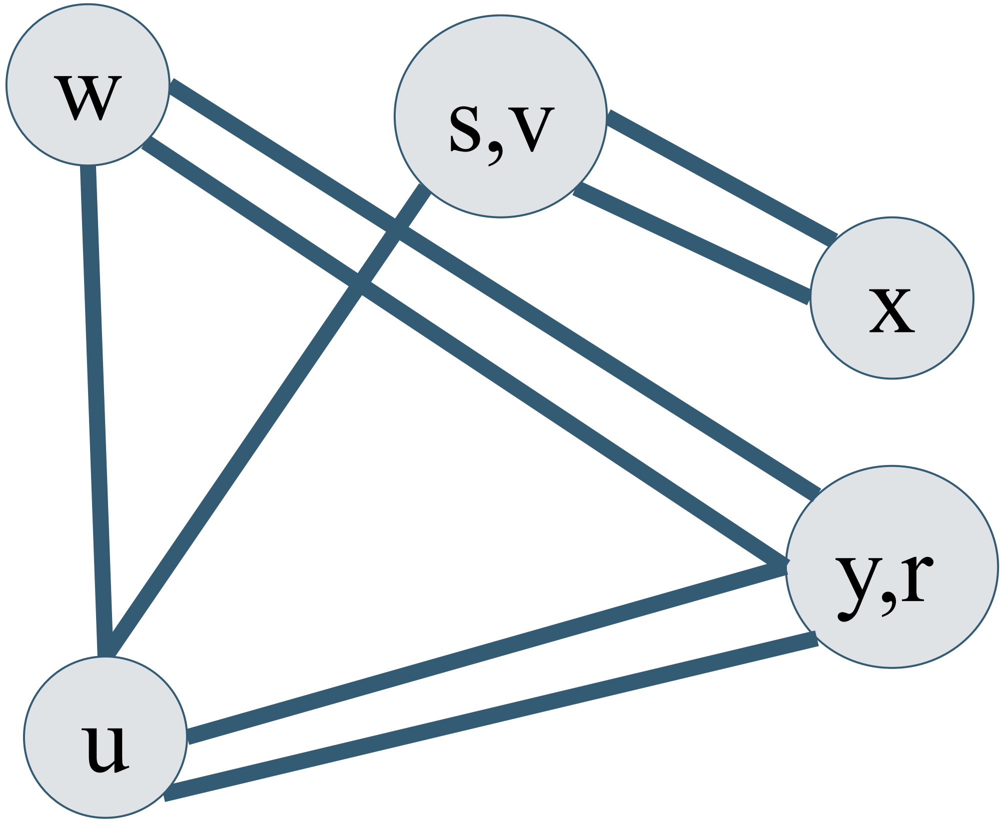


Edges:

- 1 – sv
- 2– vx
- 3– sx
- 4 – su
- 5 – uw
- 6 – w(yr)
- ~~7 – yr~~
- 8 – (yr)w
- 9 – u(yr)
- 10 – u(yr)

Random Sequence of Numbers: 7, 1

Run Karger's Algorithm yourself: Run-Through

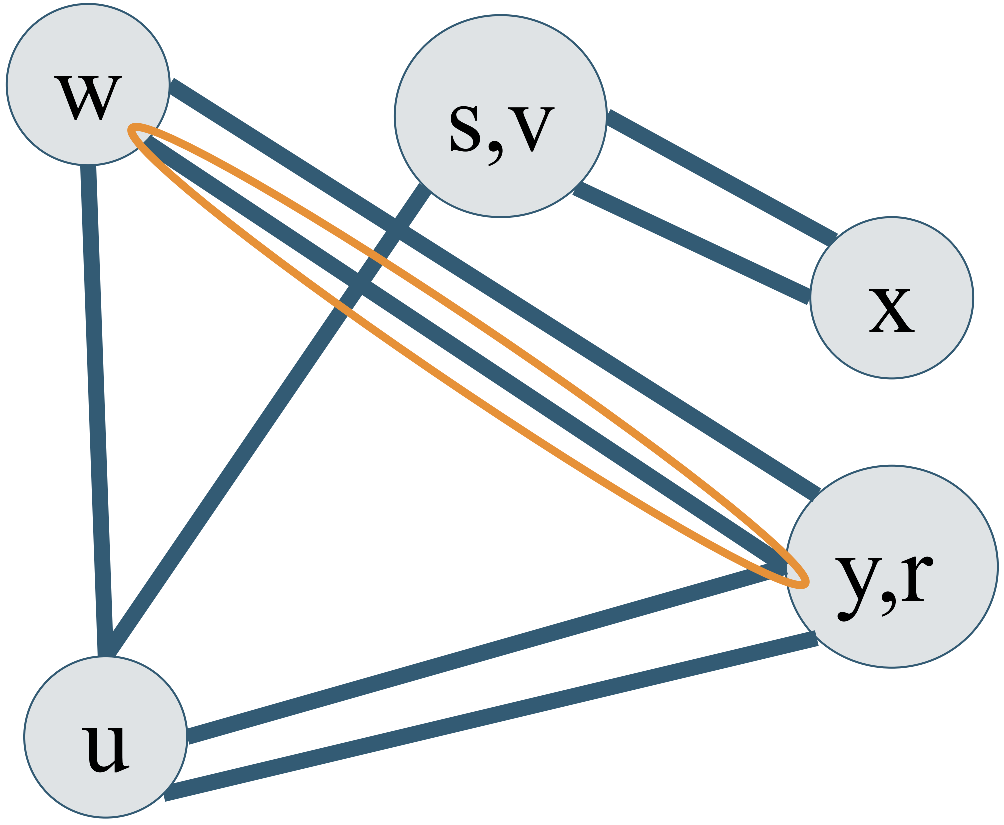


Edges:

- ~~1 – sv~~
- 2– (sv)x
- 3– (sv)x
- 4 – (sv)u
- 5 – uw
- 6 – w(yr)
- ~~7 – yr~~
- 8 – (yr)w
- 9 – u(yr)
- 10 – u(yr)

Random Sequence of Numbers: 7, 1

Run Karger's Algorithm yourself: Run-Through

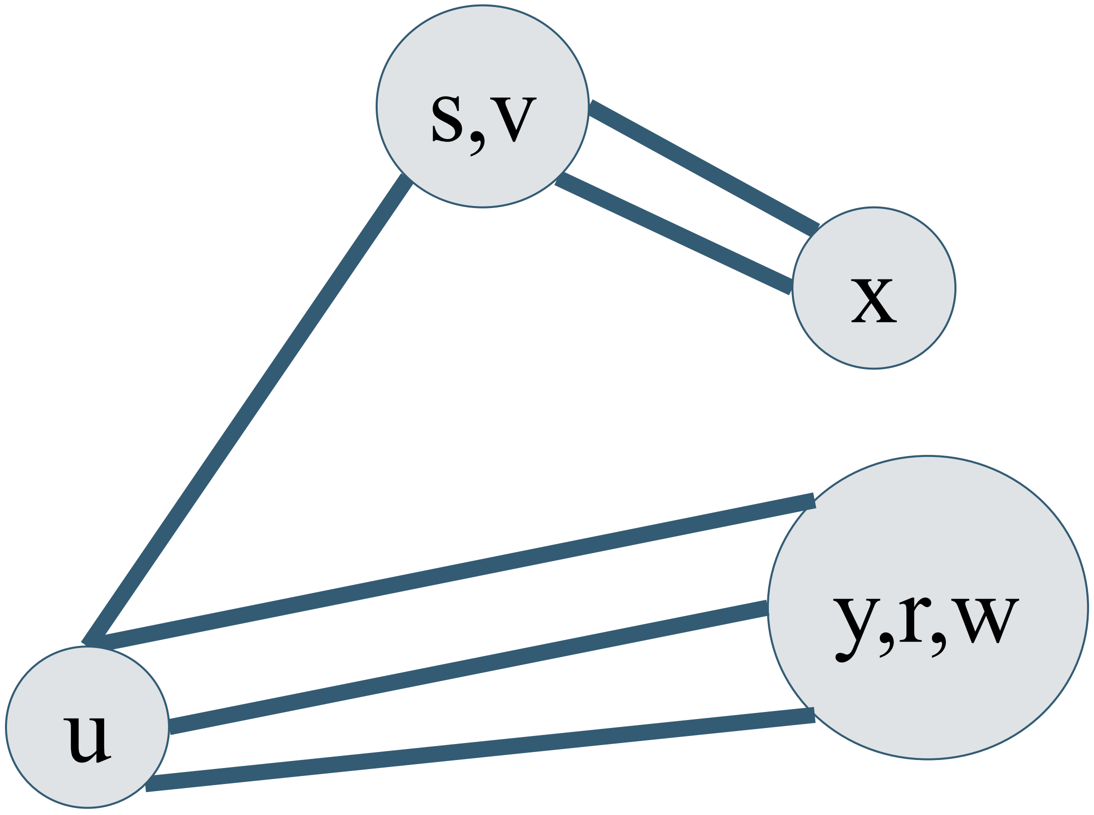


Edges:

- ~~1 – sv~~
- 2– (sv)x
- 3– (sv)x
- 4 – (sv)u
- 5 – uw
- 6 – w(yr)
- ~~7 – yr~~
- 8 – (yr)w
- 9 – u(yr)
- 10 – u(yr)

Random Sequence of Numbers: 7, 1, 6

Run Karger's Algorithm yourself: Run-Through

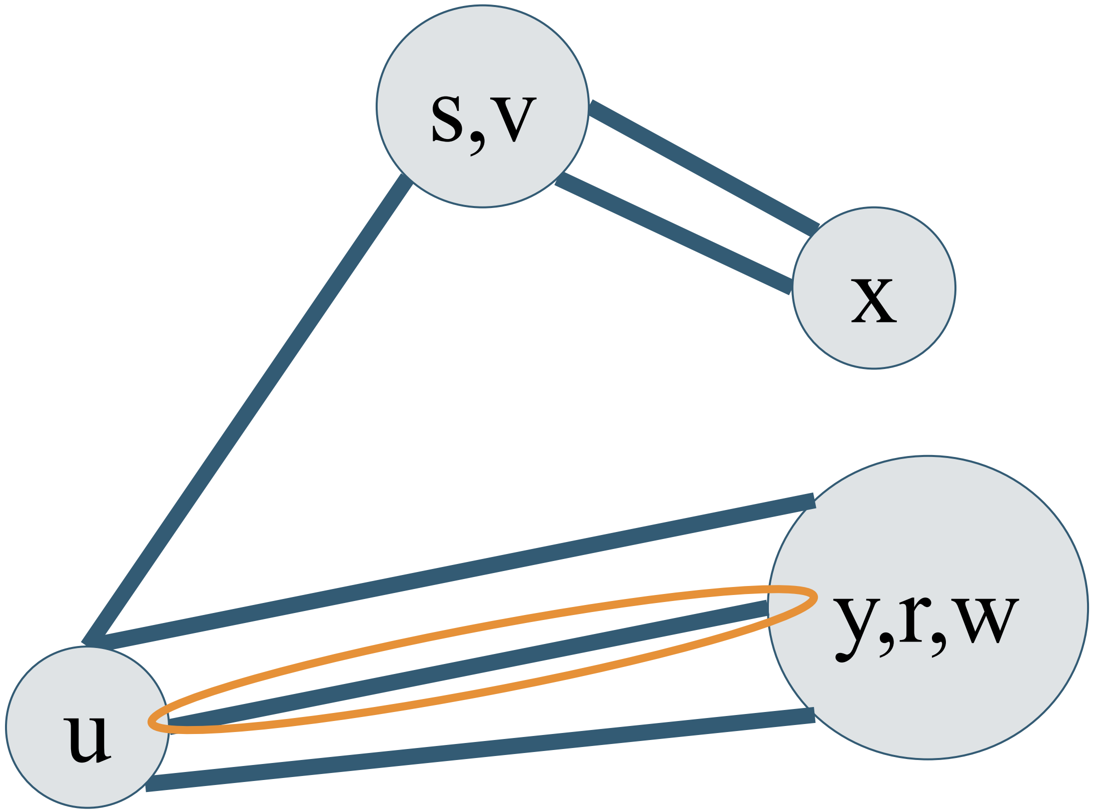


Edges:

- ~~1 – sv~~
- 2– (sv)x
- 3– (sv)x
- 4 – (sv)u
- 5 – u(yrw)
- ~~6 – w(yr)~~
- ~~7 – yr~~
- ~~8 – (yr)w~~
- 9 – u(yrw)
- 10 – u(yrw)

Random Sequence of Numbers: 7, 1, 6

Run Karger's Algorithm yourself: Run-Through

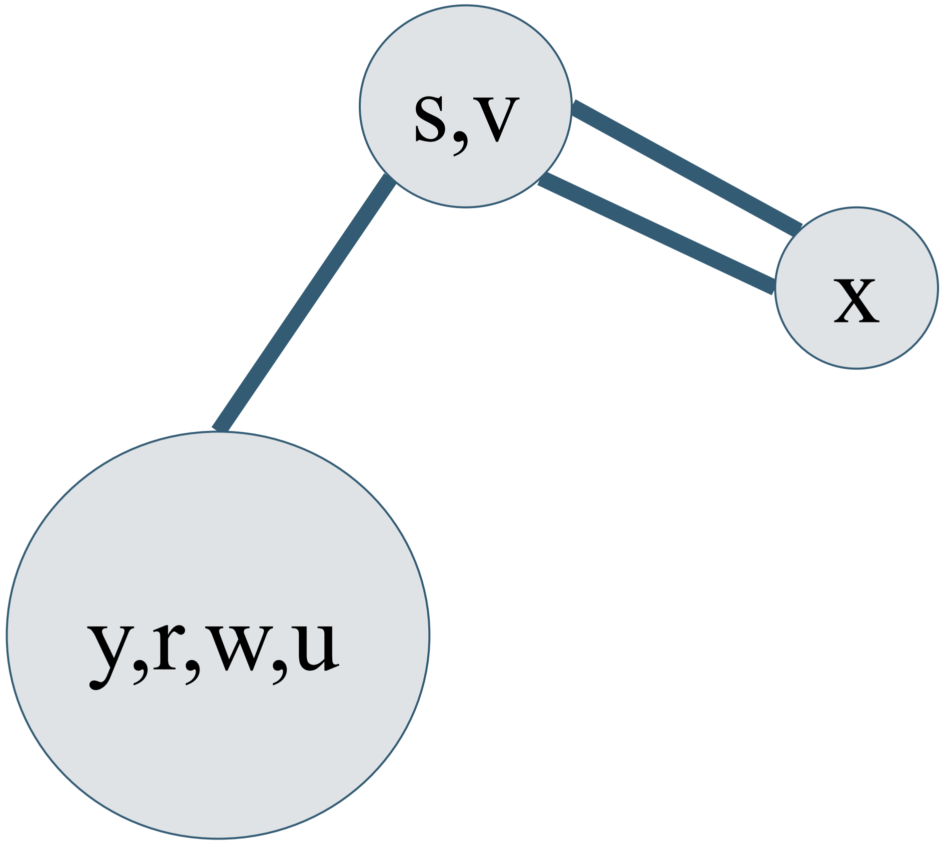


Edges:

- ~~1 – sv~~
- 2– (sv)x
- 3– (sv)x
- 4 – (sv)u
- 5 – u(yrw)
- ~~6 – w(yr)~~
- ~~7 – yr~~
- ~~8 – (yr)w~~
- 9 – u(yrw)
- 10 – u(yrw)

Random Sequence of Numbers: 7, 1, 6, 9

Run Karger's Algorithm yourself: Run-Through

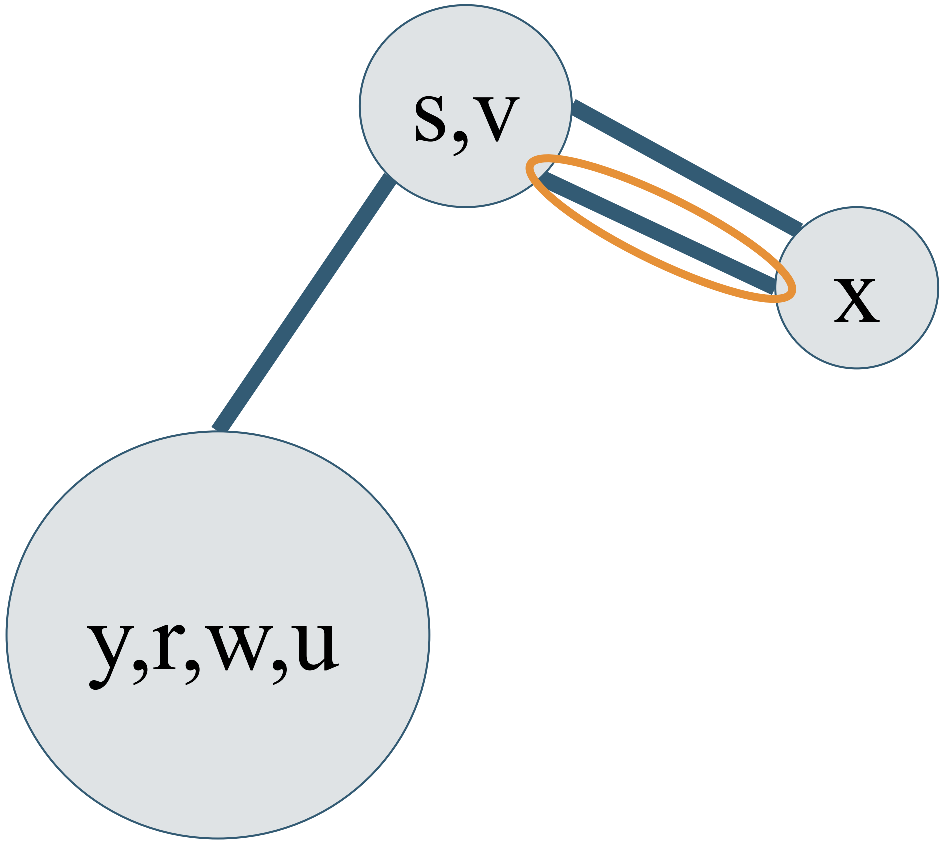


Edges:

- ~~1 – sv~~
- 2– (sv)x
- 3– (sv)x
- 4 – (sv)(yrwu)
- ~~5 – u(yrw)~~
- ~~6 – w(yr)~~
- ~~7 – yr~~
- ~~8 – (yr)w~~
- ~~9 – u(yrw)~~
- ~~10 – u(yrw)~~

Random Sequence of Numbers: 7, 1, 6, 9

Run Karger's Algorithm yourself: Run-Through

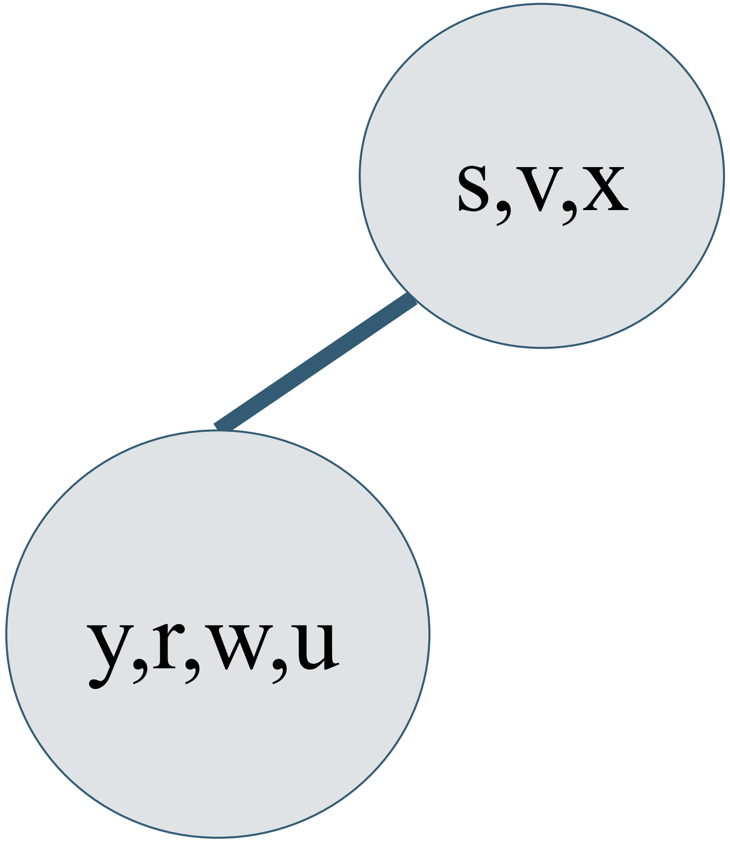


Edges:

- ~~1 – sv~~
- 2– (sv)x
- 3– (sv)x
- 4 – (sv)(yrwu)
- ~~5 – u(yrw)~~
- ~~6 – w(yr)~~
- ~~7 – yr~~
- ~~8 – (yr)w~~
- ~~9 – u(yrw)~~
- ~~10 – u(yrw)~~

Random Sequence of Numbers: 7, 1, 6, 9, 3

Run Karger's Algorithm yourself: Run-Through

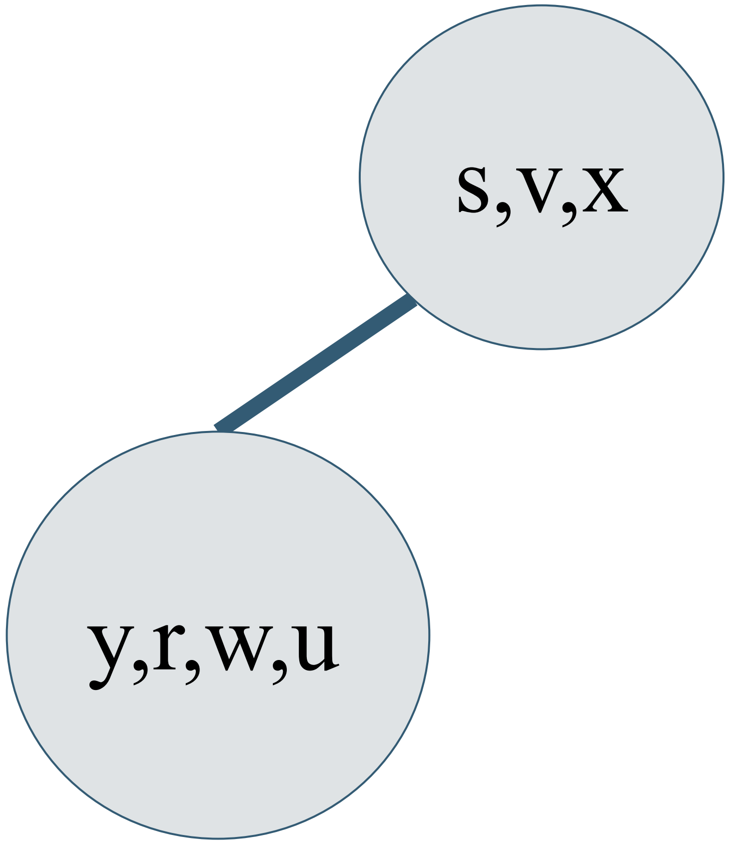


Edges:

- ~~1 – sv~~
- ~~2– (sv)x~~
- ~~3– (sv)x~~
- 4 – (sv)(yrwu)
- ~~5 – u(yrw)~~
- ~~6 – w(yr)~~
- ~~7 – yr~~
- ~~8 – (yr)w~~
- ~~9 – u(yrw)~~
- ~~10 – u(yrw)~~

Random Sequence of Numbers: 7, 1, 6, 9, 3

Run Karger's Algorithm yourself: Run–Through



Since there are two nodes, we terminate the algorithm since we have found a possible cut
The size of the cut is 1 so it must be the minimum cut (if we were not lucky, we may have found a larger cut

Edges:

- ~~1 – sv~~
- ~~2 – (sv)x~~
- ~~3 – (sv)x~~
- 4 – (sv)(yrwu)
- ~~5 – u(yrw)~~
- ~~6 – w(yr)~~
- ~~7 – yr~~
- ~~8 – (yr)w~~
- ~~9 – u(yrw)~~
- ~~10 – u(yrw)~~

Random Sequence of Numbers: 7, 1, 6, 9, 3

Karger’s Min Cut Analysis

Analyzing Probabilistic Algorithms

- Whenever we analyze deterministic algorithms, we argue that the output is necessarily correct
- With probabilistic algorithms, since we're not guaranteed the same output after multiple iterations, this type of analysis fails
- Instead, for probabilistic algorithms, we analyze the *probability* that the output is correct

Analyzing Probabilistic Algorithms

- Whenever we analyze deterministic algorithms, we argue that the output is necessarily correct
- With probabilistic algorithms, since we're not guaranteed the same output after multiple iterations, this type of analysis fails
- Instead, for probabilistic algorithms, we analyze the *probability* that the output is correct

In the next section, we will aim to show that the probability of contracting an edge in some fixed min-cut is 2/n (where n is the number of vertices)

Probability that output isn't min-cut

- **Fact 0**: If d(u) represents the degree of vertex u, then $\sum_{u \in V}$ d(u) = 2|E|

Probability that output isn't min-cut

- **Fact 0**: If d(u) represents the degree of vertex u, then $\sum_{u \in V}$ d(u) = 2|E|

Proof: We count every edge twice, once from one vertex (u), once from another vertex (v).
Since every edge has two vertices, we know we double count all the edges exactly twice.

Probability that output isn't min-cut

- **Fact 0**: If d(u) represents the degree of vertex u, then $\sum_{u \in V}$ d(u) = 2|E|
- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n

Probability that output isn't min-cut

- **Fact 0**: If d(u) represents the degree of vertex u, then $\sum_{u \in V}$ d(u) = 2|E|
- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
  - Proof: E[d(u)] = $\sum_{u \in V}$ Pr(X = u)d(u)

Probability that output isn't min-cut

- **Fact 0**: If d(u) represents the degree of vertex u, then $\sum_{u \in V}$ d(u) = 2|E|
- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
  - Proof: E[d(u)] = $\sum_{u \in V}$ Pr(X = u)d(u)

    = $\sum_{u \in V}$(1/n) d(u)

Probability that output isn't min-cut

- **Fact 0**: If d(u) represents the degree of vertex u, then $\sum_{u \in V}$ d(u) = 2|E|
- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
  - Proof: E[d(u)] = $\sum_{u \in V}$ Pr(X = u)d(u)

    = $\sum_{u \in V}$(1/n) d(u)

    = (1/n) $\sum_{u \in V}$ d(u)

Probability that output isn't min-cut

- **Fact 0**: If d(u) represents the degree of vertex u, then $\sum_{u \in V} d(u)$ = 2|E|
- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
  - Proof: E[d(u)] = $\sum_{u \in V} Pr(X = u) d(u)$

    = $\sum_{u \in V} (1/n)\, d(u)$

    = $(1/n) \sum_{u \in V} d(u)$ = (1/n)2|E| = 2|E|/n

Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is $2|E|/n$

Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
  - Proof: E[d(u)] = $\mathbb{E}_u \left[\sum_{e \in E} \mathbf{1}[\text{u is endpoint of e}] \right]$

Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
  - Proof: $E[d(u)] = \mathop{E}_{u} [\sum_{e\in E} \mathbf{1}[u \text{ is endpoint of } e]]$

    $= \sum_{e\in E} \mathop{E}_{u} [\mathbf{1}[u \text{ is endpoint of } e]]$

Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
  - Proof: $E[d(u)] = \mathbb{E}_u \left[\sum_{e \in E} \mathbf{1}[u \text{ is endpoint of } e] \right]$

$$= \sum_{e \in E} \mathbb{E}_u \left[\mathbf{1}[u \text{ is endpoint of } e] \right]$$

$$= \sum_{e \in E} P[u \text{ is endpoint of } e]$$

Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
  - Proof: $E[d(u)] = \mathop{E}_{u}\left[\sum_{e\in E} \mathbf{1}[u \text{ is endpoint of } e]\right]$

$$= \sum_{e\in E} \mathop{E}_{u}\left[\mathbf{1}[u \text{ is endpoint of } e]\right]$$

$$= \sum_{e\in E} P[u \text{ is endpoint of } e]$$

$$= \sum_{e\in E} 2/n = 2|E|/n$$

Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
- **Fact 2**: Size of min-cut is upper-bounded by 2|E|/n

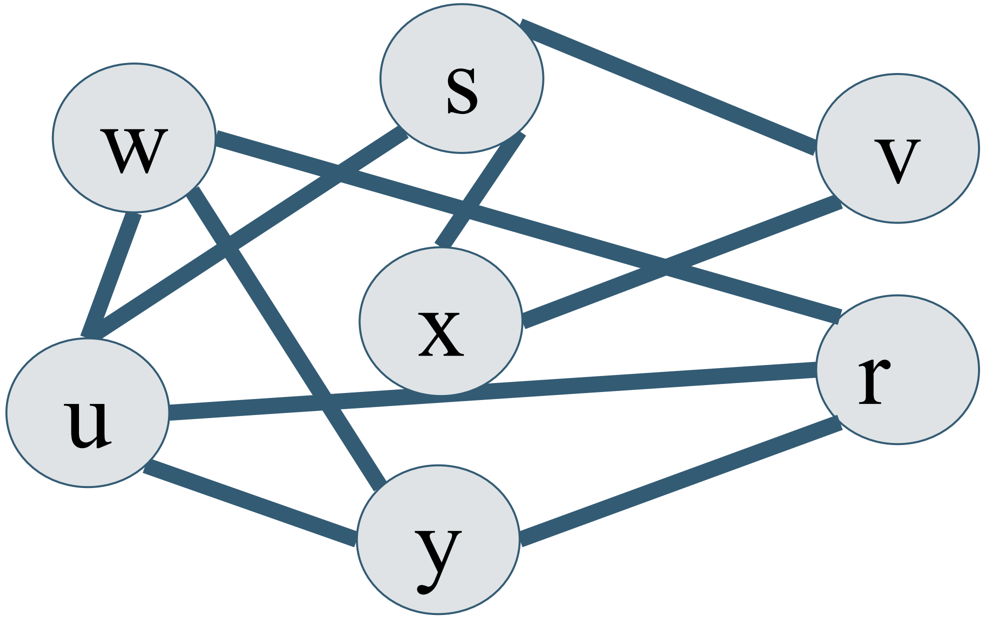
Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
- **Fact 2**: Size of min-cut is upper-bounded by 2|E|/n
  - Proof: One *valid* cut is separating one vertex from the rest of the n-1 vertices

Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
- **Fact 2**: Size of min-cut is upper-bounded by 2|E|/n
  - Proof: One *valid* cut is separating one vertex from the rest of the n-1 vertices

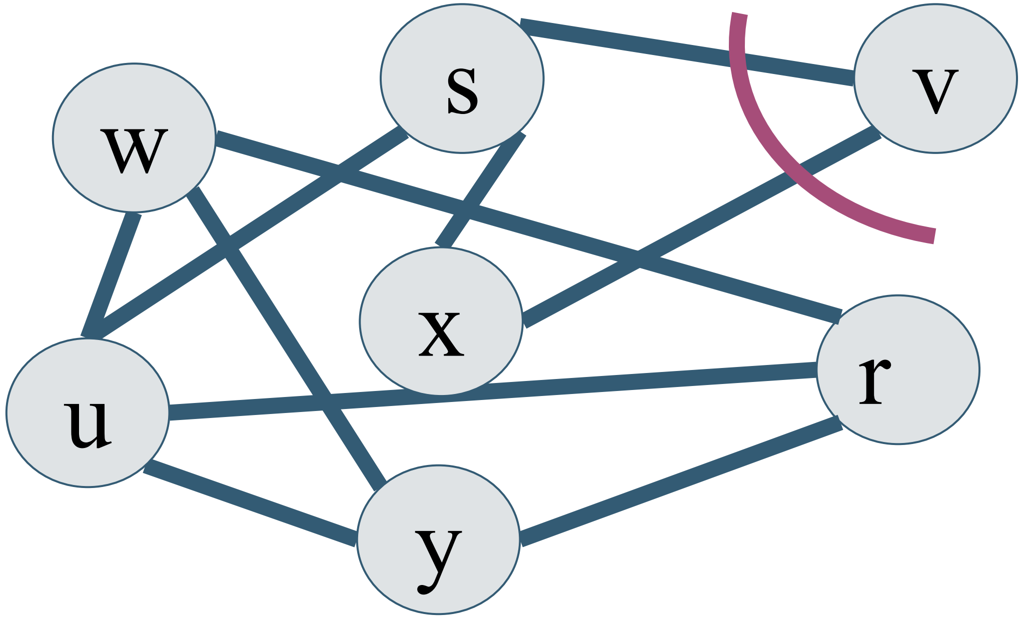
Example: 2*10/7 = 2.85...



Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
- **Fact 2**: Size of min-cut is upper-bounded by 2|E|/n
  - Proof: One *valid* cut is separating one vertex from the rest of the n-1 vertices

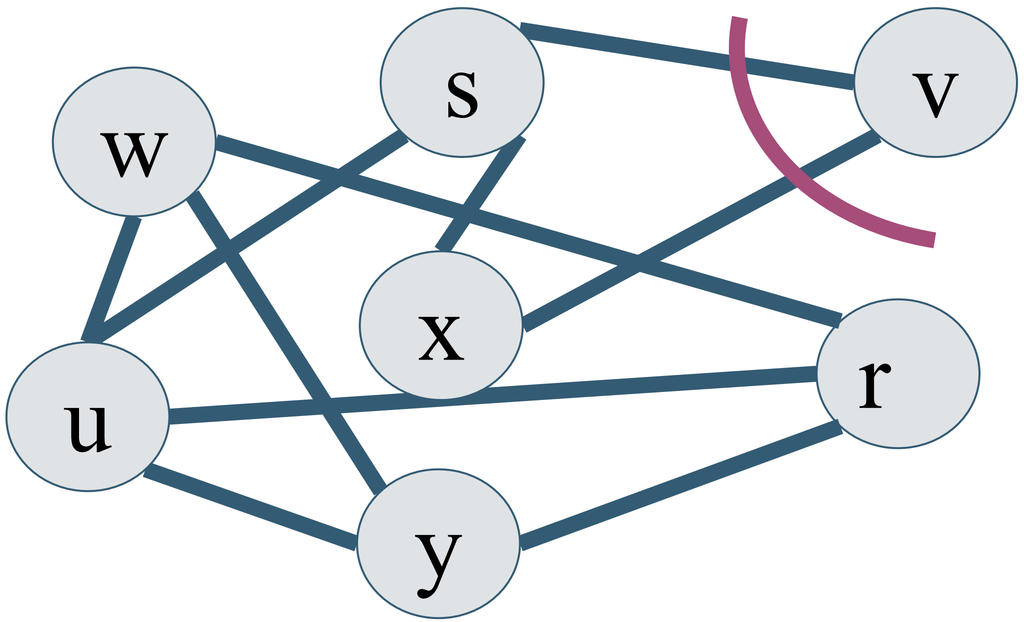
Example: 2*10/7 = 2.85...



Probability that output isn't min-cut

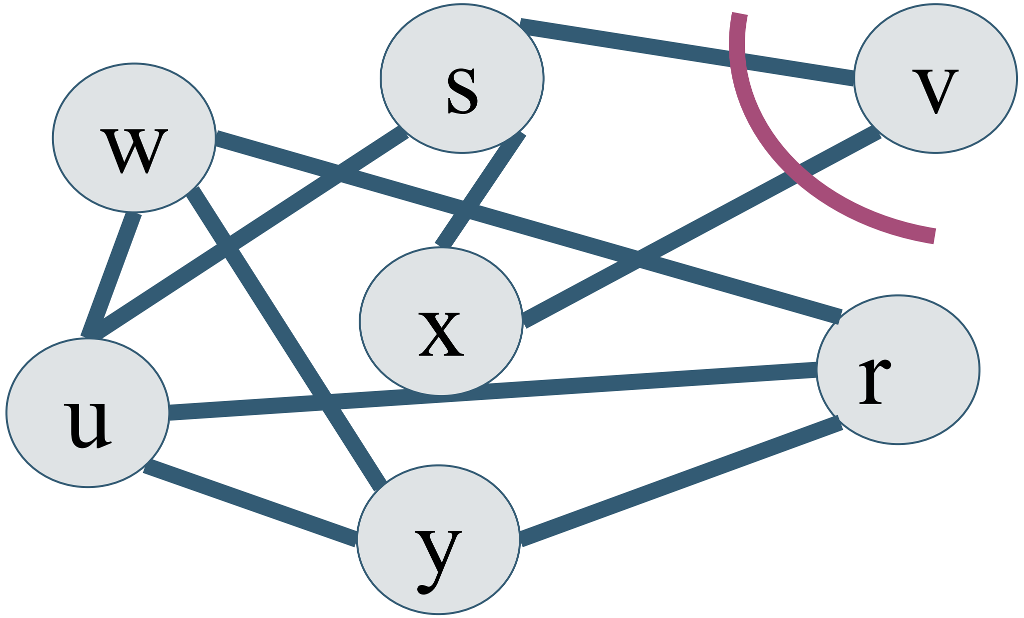
- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
- **Fact 2**: Size of min-cut is upper-bounded by 2|E|/n
  - Proof: One *valid* cut is separating one vertex from the rest of the n-1 vertices
  - The expected number of edges crossing this cut is 2|E|/n (**Fact 1**)

Example: 2*10/7 = 2.85...



Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
- **Fact 2**: Size of min-cut is upper-bounded by 2|E|/n
  - Proof: One *valid* cut is separating one vertex from the rest of the n-1 vertices
  - The expected number of edges crossing this cut is 2|E|/n (**Fact 1**)
  - Thus, since we know such a cut always exists, the min cut is no larger than 2|E|/n



Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
- **Fact 2**: Size of min-cut is upper-bounded by 2|E|/n
- **Fact 3**: A randomly picked edge crosses the min cut with probability at most 2/n

Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
- **Fact 2**: Size of min-cut is upper-bounded by 2|E|/n
- **Fact 3**: A randomly picked edge crosses the min cut with probability at most 2/n
  - Proof: **Work on this with the people around you**

Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
- **Fact 2**: Size of min-cut is upper-bounded by 2|E|/n
- **Fact 3**: A randomly picked edge crosses the min cut with probability at most 2/n
    - Proof:

Probability that output isn't min-cut

- **Fact 1**: If there are n vertices, the expected value of a vertex's degree is 2|E|/n
- **Fact 2**: Size of min-cut is upper-bounded by 2|E|/n
- **Fact 3**: A randomly picked edge crosses the min cut with probability at most 2/n
  - Proof: Since we start off with |E| edges to choose to contract and at most 2|E|/n (**Fact 3)** are in the min-cut, then we choose a min-cut edge with probability (2|E|/n)/(|E|) = 2/n

Karger's Algorithm Analysis

- We note that Karger's ALG returns the correct answer as long as none of the contracted edges are in the min-cut
- This can be represented by
  - *Pr(final cut is min cut) = Pr(None of the min cut edges were contracted before we were left with only 2 supernodes)*

Karger's Algorithm Analysis

- We note that Karger's ALG returns the correct answer as long as none of the contracted edges are in the min-cut
- This can be represented by
  - *Pr(final cut is min cut) = Pr(None of the min cut edges were contracted before we were left with only 2 supernodes)*

$$\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\ldots\left(1-\frac{2}{4}\right)\left(1-\frac{2}{3}\right)$$

Karger's Algorithm Analysis

- We note that Karger's ALG returns the correct answer as long as none of the contracted edges are in the min-cut
- This can be represented by
  - *Pr(final cut is min cut) = Pr(None of the min cut edges were contracted before we were left with only 2 supernodes)*

$$\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\ldots\left(1-\frac{2}{4}\right)\left(1-\frac{2}{3}\right)$$

$$= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\ldots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right)$$

Karger's Algorithm Analysis

- We note that Karger's ALG returns the correct answer as long as none of the contracted edges are in the min-cut
- This can be represented by
  - *Pr(final cut is min cut) = Pr(None of the min cut edges were contracted before we were left with only 2 supernodes)*

$$\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\ldots\left(1-\frac{2}{4}\right)\left(1-\frac{2}{3}\right)$$

$$= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\ldots\ldots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right)$$

Karger's Algorithm Analysis

- We note that Karger's ALG returns the correct answer as long as none of the contracted edges are in the min-cut
- This can be represented by
  - *Pr(final cut is min cut) = Pr(None of the min cut edges were contracted before we were left with only 2 supernodes)*

$$\geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\ldots\left(1-\frac{2}{4}\right)\left(1-\frac{2}{3}\right)$$

$$= \left(\frac{n-2}{n}\right)\left(\frac{n-3}{n-1}\right)\left(\frac{n-4}{n-2}\right)\ldots\ldots\left(\frac{2}{4}\right)\left(\frac{1}{3}\right) = \frac{2}{n(n-1)}$$

Karger's Algorithm Analysis

- We know that Karger's succeeds with probability at least $\frac{2}{n(n-1)}$
- If we run Karger's twice then the probability of success is:

$$1 - (1 - \frac{2}{n(n-1)})^2$$

Similarly, running Karger 'k' times yields a success probability of:

$$1 - (1 - \frac{2}{n(n-1)})^k$$

e^{-p}

Karger's Algorithm Analysis

- We can use the approximation that 1-p is about e^{-p}
- We note that if we run this algorithm $\frac{n(n-1)}{2}\ln(n)$ times then

$$1-(1-\frac{2}{n(n-1)})^{\frac{n(n-1)\ln(n)}{2}} \approx 1-(e^{-\frac{2}{n(n-1)}})^{\frac{n(n-1)\ln(n)}{2}}$$

$$=1-e^{-\ln(n)}=1-\frac{1}{n}$$

e^{-p}

Karger's Algorithm Analysis

- We can use the approximation that 1-p is about e^{-p}
- We note that if we run this algorithm $\frac{n(n-1)}{2}\ln(n)$ times then

$$1-(1-\frac{2}{n(n-1)})^{\frac{n(n-1)\ln(n)}{2}} \approx 1-(e^{-\frac{2}{n(n-1)}})^{\frac{n(n-1)\ln(n)}{2}}$$

$$= 1-e^{-\ln(n)} = 1-\frac{1}{n}$$

- To get this probability of success, we need to run this ALG $\frac{n(n-1)}{2}\ln(n)$ times, but we note we go through 'm' edges meaning the runtime of the ALG is O($\frac{n(n-1)\ln(n)m}{2}$) or O(n²mlog(n))

Karger-Stein

Intuition

- In the beginning, low chance of contracting a min-cut edge
- As we contract more edges, the ratio of “min cut edges” to “non min cut edges” becomes higher
- However, as we contract more edges, there are less total edges
- Perhaps instead of contracting all the way, we choose some point where we stop contracting and just process the remaining graph more carefully

Pseudocode

```
int FindMinCut(Edge[1,...,e], Vertex[1,...,v])
while ( // there are more than n - n/sqrt(2) edges
    edge -> Choose edge randomly from the list
    ContractEdge(edge)

Call FindMinCut on this graph twice and return the minimum of the two cuts

void ContractEdge(Edge e) // e.u = one vertex of the edge, e.v = second
vertex
    Create new vertex: SuperNode
    Reattach all edges from e.u and e.v to SuperNode
Delete e
```

Pseudocode

```
int FindMinCut(Edge[1,...,e], Vertex[1,...,v])
while ( // there are more than n - n/sqrt(2) edges
   edge -> Choose edge randomly from the list
   ContractEdge(edge)
```

If we go through the math, we contract edges until the probability of contracting a min cut edge is greater than 1/2

```
Call FindMinCut on this graph twice and return the minimum of the two cuts

void ContractEdge(Edge e) // e.u = one vertex of the edge, e.v = second vertex
   Create new vertex: SuperNode
   Reattach all edges from e.u and e.v to SuperNode
Delete e
```

Pseudocode

```
int FindMinCut(Edge[1,...,e], Vertex[1,...,v])
while ( // there are more than n - n/sqrt(2) edges
    edge -> Choose edge randomly from the list
    ContractEdge(edge)
```

If we go through the math, we contract edges until the probability of contracting a min cut edge is greater than 1/2

```
Call FindMinCut on this graph twice and return the minimum of the two cuts
```

```
void ContractEdge(Edge e) // e.u = one vertex of the edge, e.v = second vertex
    Create new vertex: SuperNode
    Reattach all edges from e.u and e.v
Delete e
```

Here, the probability of returning a min cut is 1/log(n) (the proof involves some induction)

Pseudocode

```
int FindMinCut(Edge[1,...,e], Vertex[1,...,v])
while ( // there are more than n - n/sqrt(2) edges
    edge -> Choose edge randomly from the list
    ContractEdge(edge)
```

If we go through the math, we contract edges until the probability of contracting a min cut edge is greater than 1/2

```
Call FindMinCut on this graph twice and return the minimum of the two cuts
```

```
void ContractEdge(Edge e) // e.u = one vertex of the edge, e.v = second
```

To achieve an error probability of O(1/n) we need to run this algorithm $\log^2(n)$ amount of times for a total runtime of $O(n^2\log^3(n))$

```
nd e.v
```

Here, the probability of returning a min cut is 1/log(n) (the proof involves some induction)

Karger-Stein: Termination Vertex

- Denote 'm' as the vertex at which we terminate the contraction process (i.e. after contraction of this vertex, the probability of contracting a min-cut edge becomes more than ½). We see:

$$\frac{n-2}{n} * \frac{n-3}{n-1} * \frac{n-4}{n-2} * \dots \frac{m-1}{m+1} = \frac{m(m-1)}{n(n-1)} = \frac{\binom{m}{2}}{\binom{n}{2}} \geq \frac{1}{2}$$

$$\implies m \geq \frac{n}{\sqrt{2}}$$

Thus we terminate the contraction process at the n/sqrt(2) 'th vertex. (To be more exact, you could terminate at n/sqrt(2) , but the asymptotic arguments are identical in either case, and for simplicity we'll use the former)

Karger-Stein: Runtime

- We note that first we need to perform n-n/sqrt(2) contractions and then perform two iterations of Min-Cut on the remaining graph with n/sqrt(2) nodes .
- Using an adjacency list implementation, the contractions take $O(n^2)$ to perform
- From here we need to perform this algorithm on the smaller graph yielding the following run-time

$$T(n) = 2T(\frac{n}{\sqrt{2}}) + O(n^2)$$

- From here, we use Master's Theorem to yield a runtime of $O(n^2 \log(n))$

Karger-Stein: Correctness

- We argue that probability of success is dependent on getting at least one success in one of these smaller graphs, which has a ½ chance of not having the min-cut already contracted (this argument comes from the fact that the last contracted edge ensures that further contracted edges have that probability to be a min-cut edge)
- With this argument, we can yield the following recurrence:

$$P(n) \geq 1 - (1 - \frac{1}{2}P(\frac{n}{\sqrt{2}}))^2$$

An inductive argument shows that P(n) >= 1/(n+2), and we finish with the argument that 1/O(n) >= O(log n). Thus, we need at least log(n) to get a constant probability of success and $\log(n)^2$ iterations to get a 1-1/poly(n) probability of success , yielding a $O(n^2\log^3(n))$ runtime

References

- Here are some of the notes I consulted and good resources to broaden your understanding of these algorithms
  - http://www.cs.toronto.edu/~anikolov/CSC473W20/Lectures/Karger-Stein.pdf
  - https://courses.cs.washington.edu/courses/cse521/16sp/521-lecture-1.pdf
  - https://www.cs.princeton.edu/courses/archive/fall16/cos521/Lectures/lec2.pdf