MIN-CUT ALGORITHMS

Hakki Can Karaimer
HU Sixing
Pan An
Philipp Keck
Taehoon Kim

AGENDA

- Introduction to Minimum Cuts
- Karger's Algorithm
- Improvement by Karger and Stein
- Parallelized Version
- Applications

INTRODUCTION

Hakki Can Karaimer

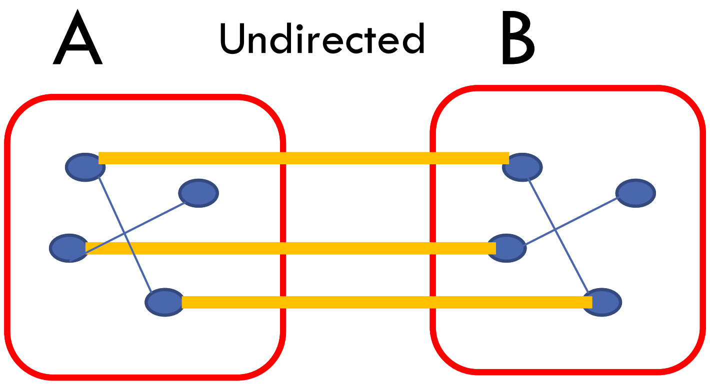
GRAPHS REFRESHER

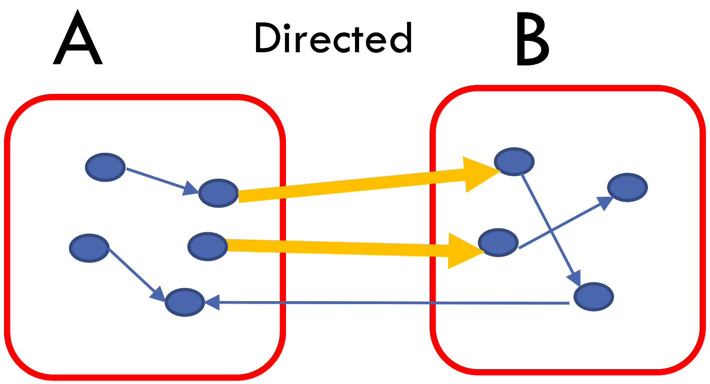
- Two ingredients
  - Vertices (singular vertex) a.k.a. nodes. (V)
  - Edges (E) = pairs of vertices
    - Can be undirected (unordered pair)

      or directed (ordered pair) (a.k.a arcs)
- $G = (V, E),\ \ n = |V|,\ \ m = |E|$

CUT PROBLEM

- Definition: a cut of a graph (V, E) is a partition of V into non-empty sets A and B.
- Definition: the crossing edges of a cut (A, B) are those with:
  - One endpoint in each of (A, B) (undirected)
  - Tail in A, head in B (directed)



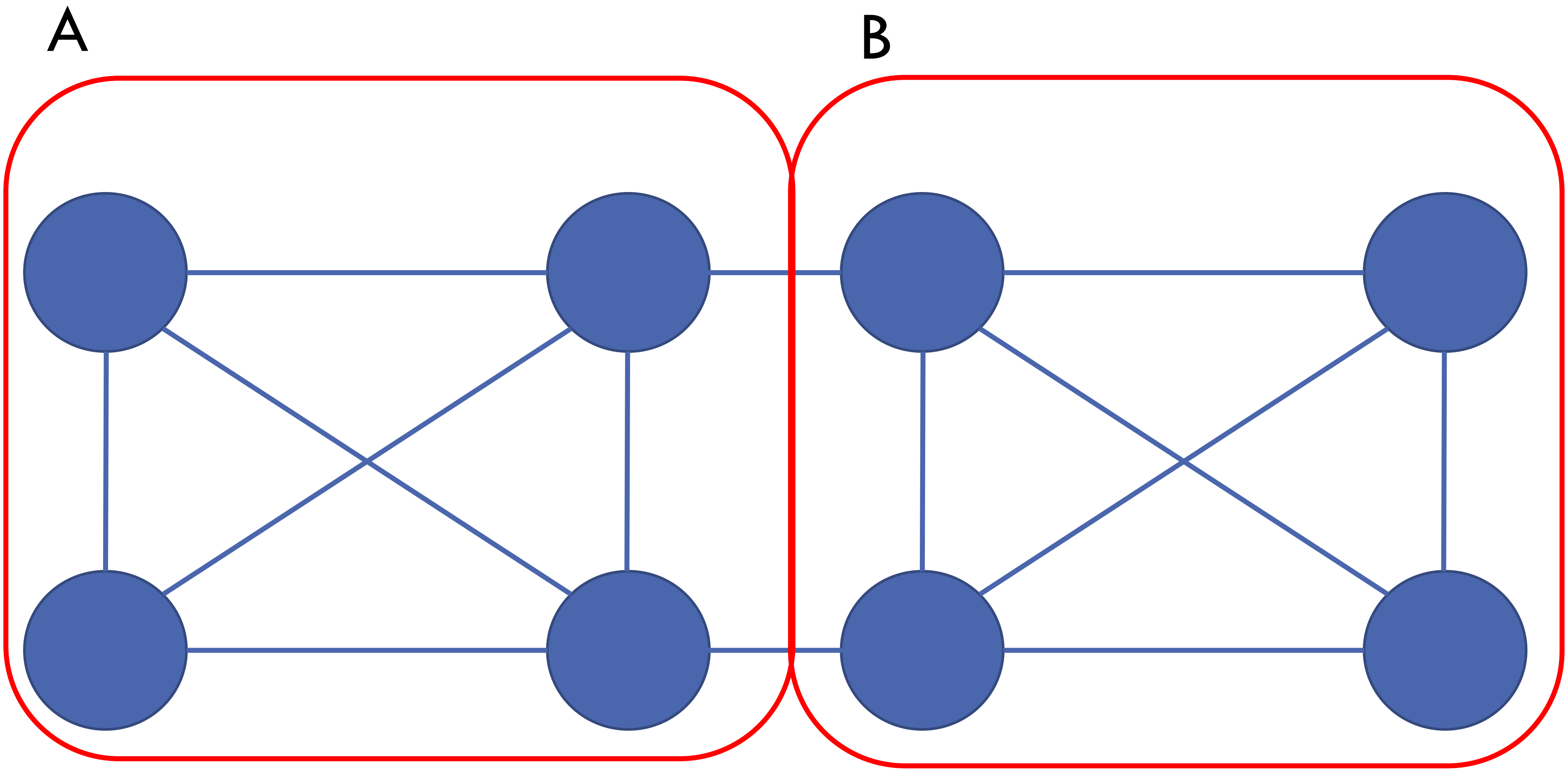


- If the graph has n vertices
  - There are $2^n - 2$ possible cuts.

MINIMUM CUT PROBLEM

- Definition: the minimum cut of an undirected graph $G = (V, E)$ is a partition of the nodes into two groups A and B (that is, $V = A \cup B$ and, $A \cap B = \emptyset$), so that the number of edges between A and B is minimized.
- Input: an undirected graph $G = (V, E)$
  - Parallel (multiple) edges are allowed
- Goal: compute a cut with fewest number of crossing edges (a min-cut).

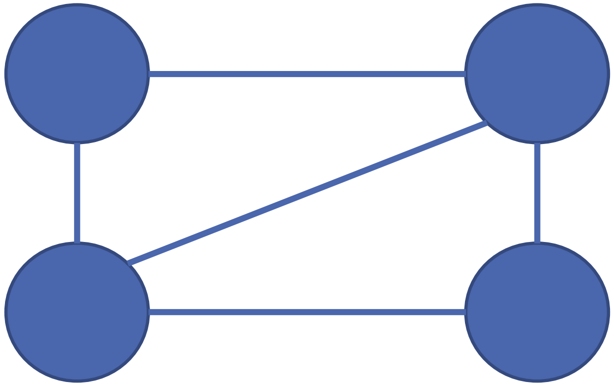
MINIMUM CUT PROBLEM



RANDOM CONTRACTION ALGORITHM

- David Karger, early 90’s
- While there are more than 2 vertices:
  - Pick a remaining edge (u, v) uniformly at random
  - Merge (or “contract”) u and v into a single vertex
  - Remove self-loops
- Return cut represented by final 2 vertices

EXAMPLE



While there are more than 2 vertices:

- Pick a remaining edge (u,v) uniformly at random
- Merge (or "contract") u and v into a single vertex
- Remove self loops

ANALYSIS OF KARGER‘S ALGORITHM

HU Sixing

ANALYSIS OF KARGER'S ALGORITHM

- What is the probability of success
  - Karger's Algorithm succeeds with probability $p \geq \frac{2}{n^2}$

- The time complexity of Karger's algorithm is $O(n^2)$

ANALYSIS OF KARGER'S ALGORITHM

- Fact 1 (Handshaking Lemma).

$$\sum_{u \in V} degree(u) = 2m$$

- $degree(u)$: the degree of a vertex (u) of a graph is the number of edges incident to the vertex.
- Proof:
  - Each edge contributes two to the total degree. All edges together contribute $2m$ to the graph's degree.

ANALYSIS OF KARGER'S ALGORITHM

- Fact 2. The average degree of a node is $\frac{2m}{n}$
- Proof:
  - $\mathbb{E}[degree(X)] = \sum_{u \in V} Pr(X = u) degree(u)$

    Randomly pick

    $= \sum_{u \in V} \frac{1}{n} degree(u)$

    $= \frac{1}{n} \sum_{u \in V} degree(u)$

    Fact 1: $\sum_{u \in V} degree(u) = 2m$

    $= \frac{2m}{n}$
  - $\mathbb{E}$ is the mathematical expectation
  - X is a random variable representing a vertex of the graph, u is the specific vertex

ANALYSIS OF KARGER'S ALGORITHM

- Fact 3. The size of the minimum cut is at most $\frac{2m}{n}$
- Proof:
  - Let f denote the size of minimum cut
  - $f \leq degree(u),\ \forall u \in V$
  - $nf \leq \sum_{u \in V} \ degree(u)$
  - $f \leq \frac{\sum_{u \in V} \ degree(u)}{n} = \frac{2m}{n}$

ANALYSIS OF KARGER'S ALGORITHM

- Fact 4. If an edge is picked at random, the probability that it lies across the minimum cut is at most $\frac{2}{n}$
- Proof:
  - Let the probability that an edge lies across the minimum cut be p
  - $p = \frac{size\ of\ minimum\ cut}{total\ \text{number}\ of\ edges}$

$$\leq \frac{\frac{2m}{n}}{m}$$

$$= \frac{2}{n}$$

Fact 3: The size of the minimum cut is at most $\frac{2m}{n}$

ANALYSIS OF KARGER'S ALGORITHM

- Karger's Algorithm succeeds with probability $p \geq \frac{2}{n^2}$
- Fact 4. If an edge is picked at random, the probability that it lies across the minimum cut is at most $\frac{2}{n}$
- Proof:
  - Karger's algorithm returns the right answer as long as it never picks an edge across the minimum cut.
  - $\Pr(success) \geq Pr(finding\ the\ mincut)$

$$= Pr(first\ selected\ edge\ is\ not\ in\ mincut) \times$$
$$Pr(second\ selected\ edge\ is\ not\ in\ mincut) \times \cdots$$
$$Pr(last\ selected\ edge\ is\ not\ in\ mincut)$$
$$\geq \left(1 - \frac{2}{n}\right)\left(1 - \frac{2}{n-1}\right) \dots \left(1 - \frac{2}{3}\right) = \frac{2}{n(n-1)} \Leftarrow \binom{n}{2}^{-1}$$

k-combination of a set S which has n elements $\binom{n}{k} = \frac{n(n-1)\dots(n-k+1)}{k(k-1)\dots 1}$

ANALYSIS OF KARGER'S ALGORITHM

- If we run the algorithm $l\binom{n}{2}$ (l is a constant) times, and let p denote the probability that at least succeed once, then we get
- $p = 1 - \Pr(\text{fail in all } l\binom{n}{2} \text{ runs})$

$$\geq 1 - \left(1 - \binom{n}{2}^{-1}\right)^{l\binom{n}{2}}$$

$$= 1 - e^{-l}$$

- Let $l = c\ln n$(c is a constant), then $p \geq 1 - \frac{1}{n^c}$
- If we run the algorithm $c\ln n\binom{n}{2}$times ,the probability of finding the minimum cut is larger than $1 - \frac{1}{n^c}$; or the error probability is less than $\frac{1}{n^c}$

ANALYSIS OF KARGER'S ALGORITHM

While there are more than 2 vertices:
- Pick a remaining edge (u, v) uniformly at random
- Merge u and v into a single vertex
- Remove self loops

Return cut represented by final 2 vertices

- The time complexity of Karger's algorithm is $O(n^2)$
  - Every iteration two vertices are merged to one, need (n-2) times -- $O(n)$
  - In each iteration, select an edge (u, v) randomly
    - [We maintain a vector $D(u)$ of degree of each node u, $degree(u)$, a matrix $W(u, v)$ of weight of edge (u, v)]
    - Choose endpoint u with probability proportional to $D(u)$ -- $O(n)$
    - Then choose another endpoint v with probability proportional to $W(u, v)$ -- $O(n)$
    - Contract u and v
- The time complexity after boosting is $O(n^4 \log n)$

ANALYSIS OF KARGER'S ALGORITHM

While there are more than 2 vertices:
- Pick a remaining edge (u, v) uniformly at random
- Merge u and v into a single vertex
- Remove self loops

Return cut represented by final 2 vertices

- Contract u and v -- $O(n)$
  - Update vector D
    - $D(u) \coloneqq D(u) + D(v) - 2W(u, v)$
    - $D(v) \coloneqq 0$
  - Update matrix W
    - $W(u, v), W(v, u) \coloneqq 0$
    - For each vertex w except u, v
      - $W(u, w) \coloneqq W(u, w) + W(v, w)$
      - $W(w, u) \coloneqq W(w, u) + W(w, v)$
      - $W(w, v), W(v, w) \coloneqq 0$

ANALYSIS OF KARGER'S ALGORITHM

While there are more than 2 vertices:
- ➢ Pick a remaining edge (u, v) uniformly at random
- ➢ Merge u and v into a single vertex
- ➢ Remove self loops

Return cut represented by final 2 vertices

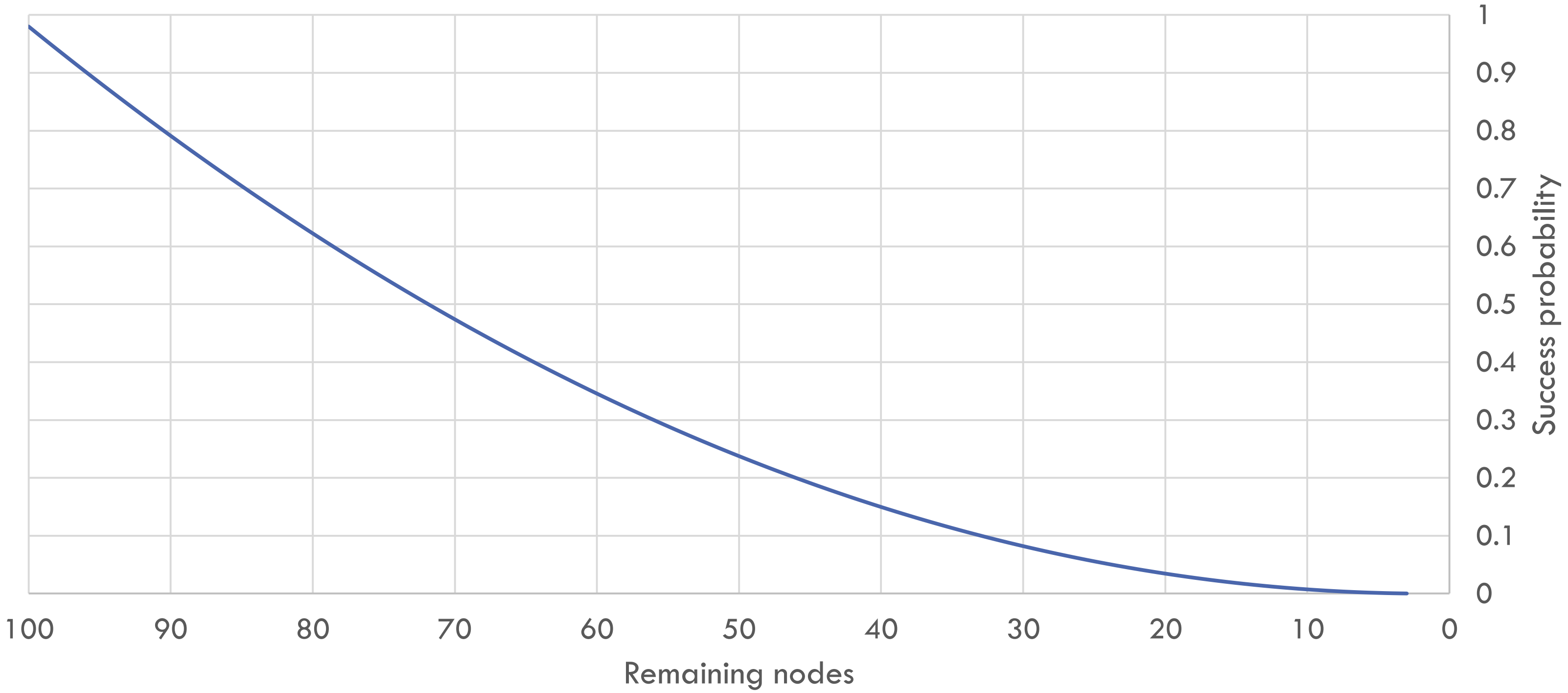
- The time complexity of Karger's algorithm is $O(n^2)$
  - Every iteration two vertices are merged to one, need (n-2) times -- $O(n)$
  - In each iteration, select an edge (u, v) randomly
    - [We maintain a vector $D(u)$ of degree of each node u, $degree(u)$, a matrix $W(u, v)$ of weight of edge (u, v)]
    - Choose endpoint u with probability proportional to $D(u)$ -- $O(n)$
    - Then choose another endpoint v with probability proportional to $W(u, v)$ -- $O(n)$
    - Contract u and v -- $O(n)$
- The time complexity after boosting is $O(n^4 \log n)$

IMPROVED VERSION BY KARGER AND STEIN

Philipp Keck

SUCCESS DURING RUNTIME

- $\left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\cdots\left(1-\frac{2}{4}\right)\left(1-\frac{2}{3}\right)$
- Good in the beginning, worse towards the end



IMPROVING THE RUNTIME

- $\left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\cdots\left(1-\frac{2}{4}\right)\left(1-\frac{2}{3}\right)$
- Good in the beginning, worse towards the end
- Improving by repeating takes a long time
- Idea: Use recursion to share partial results among repeats
  - Share the better parts
  - Retry more on the worse parts to improve those
- Good cut-off: $\mathrm{k}=\frac{n}{\sqrt{2}}+1\approx 70\%\cdot n$
  - $k=$ remaining nodes, i.e., 30% contracted already

IMPROVED ALGORITHM

Recursive−Contract(Graph G of size n)

if $n > 6$ then

$k \leftarrow \frac{n}{\sqrt{2}} + 1$

$G_1 \leftarrow$ Contract G down to k nodes

$G_2 \leftarrow$ Contract G down to k nodes

$Cut_1 \leftarrow$ Recursive−Contract(G_1)

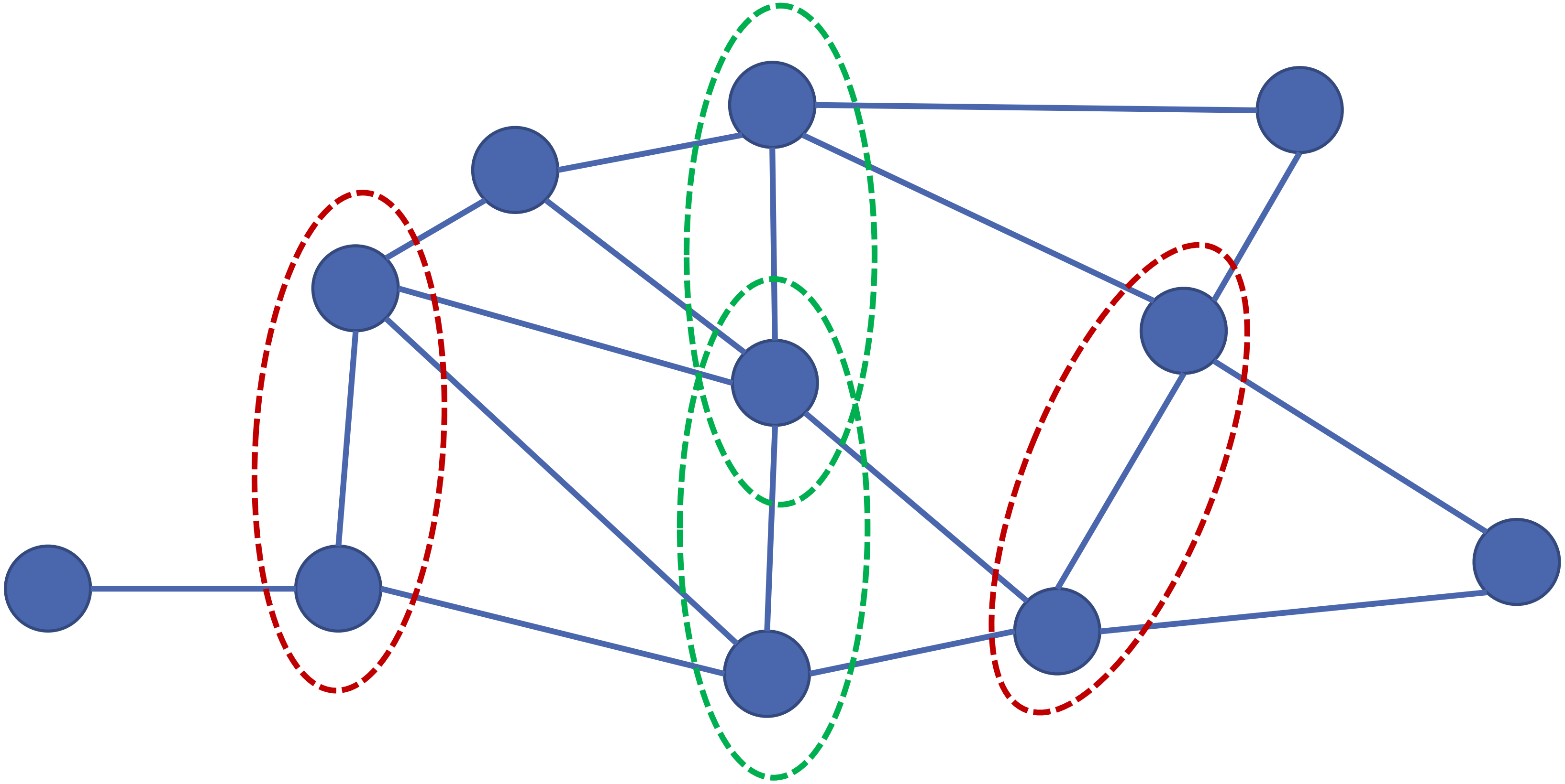
$Cut_2 \leftarrow$ Recursive−Contract(G_2)

return min(Cut_1, Cut_2)

else

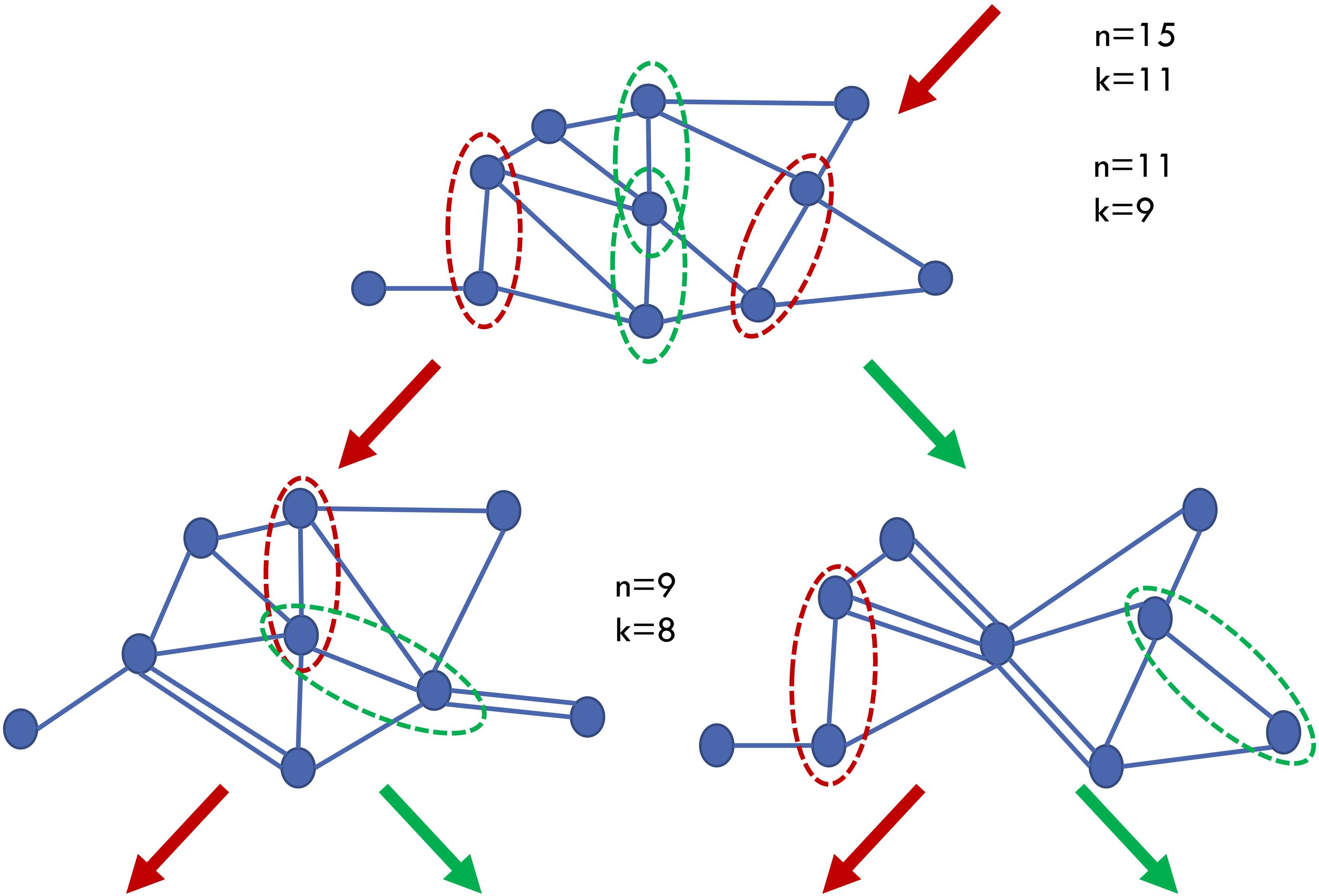
return Some−Algorithm(G)

SHARING RESULTS BY RECURSION



$$n = 11 \Rightarrow k = 9 \Rightarrow \text{Contract two edges}$$

SHARING RESULTS BY RECURSION



IMPROVED ALGORITHM

Recursive−Contract(Graph G of size n)

if $n > 6$ then

$k \leftarrow \frac{n}{\sqrt{2}} + 1$

$G_1 \leftarrow$ Contract G down to k nodes

$G_2 \leftarrow$ Contract G down to k nodes

$Cut_1 \leftarrow$ Recursive−Contract(G_1)

$Cut_2 \leftarrow$ Recursive−Contract(G_2)

return min(Cut_1, Cut_2)

else

return Some−Algorithm(G)

IMPROVED ALGORITHM – RUNTIME

Recursive−Contract(Graph G of size n)	$T(n)$
if $n > 6$ then	
$\quad k \leftarrow \frac{n}{\sqrt{2}} + 1$	$O(1)$
$\quad G_1 \leftarrow$ Contract G down to k nodes	$O(n^2)$
$\quad G_2 \leftarrow$ Contract G down to k nodes	$O(n^2)$
$\quad Cut_1 \leftarrow$ Recursive−Contract(G_1)	$T(k)$
$\quad Cut_2 \leftarrow$ Recursive−Contract(G_2)	$T(k)$
$\quad$ return $\min(Cut_1, Cut_2)$	$O(1)$
else	
$\quad$ return Some−Algorithm(G)	$O(1)$

$$T(n) = O(n^2) + 2 \cdot T\left(\frac{n}{\sqrt{2}}\right) = O(n^2 \log n)$$

Master-Theorem: $\log_{\sqrt{2}} 2 = 2$

SUCCESS PROBABILITY DOWN TO k

- Stopping at $k < n$ remaining nodes preserves fixed min-cut with probability

$$\left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\left(1-\frac{2}{n-2}\right)\cdots\left(1-\frac{2}{k+1}\right)$$

$$=\frac{\left(1-\frac{2}{n}\right)\cdots\left(1-\frac{2}{3}\right)}{\left(1-\frac{2}{k}\right)\cdots\left(1-\frac{2}{3}\right)}=\frac{\binom{n}{2}^{-1}}{\binom{k}{2}^{-1}}=\frac{\binom{k}{2}}{\binom{n}{2}}$$

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

SUCCESS PROBABILITY DOWN TO k

- Plugging in $k = \frac{n}{\sqrt{2}} + 1$

$$\cdots = \frac{k(k-1)}{n(n-1)} = \frac{\left(\frac{n}{\sqrt{2}} + 1\right)\left(\frac{n}{\sqrt{2}} + 1\ - 1\right)}{n(n-1)}$$

$$= \frac{\frac{n^2}{2} + \frac{n}{\sqrt{2}}}{n^2 - n} \overset{!}{\geq} \frac{1}{2}$$

$$\Leftrightarrow \frac{n^2}{2} + \frac{n}{\sqrt{2}} \overset{!}{\geq} \frac{1}{2}(n^2 - n)$$

$$\Leftrightarrow n^2 + \sqrt{2}n \overset{!}{\geq} n^2 - n$$

$$\Leftrightarrow \sqrt{2} \overset{!}{\geq} -1$$

SUCCESS PROBABILITY RECURSION

- Success probability of a single run (including all recursion):

$$P(n) \geq 1 - \left(1 - \frac{1}{2} \cdot P\left(\frac{n}{\sqrt{2}} + 1\right)\right)^2$$

$$\Rightarrow (\ldots \text{lots of math} \ldots)$$

$$\Rightarrow P(n) = \Omega\left(\frac{1}{\log n}\right)$$

HERE IS THAT MATH

- $P(n) \geq 1 - \left(1 - \frac{1}{2} \cdot P\left(\frac{n}{\sqrt{2}} + 1\right)\right)^2$
- $p_0 = \frac{2}{6(6-1)} = \frac{1}{15}; p_{i+1} \geq 1 - \left(1 - \frac{1}{2} p_i\right)^2$
- $z_i \coloneqq \frac{4}{p_i} - 1 \Leftrightarrow p_i = \frac{4}{z_i+1}$
- $z_0 = 59$
- $z_{i+1} = \frac{4}{p_{i+1}} - 1 \leq \frac{4}{1-\left(1-\frac{1}{2}p_i\right)^2} - 1 = \frac{4}{1-\left(1-\frac{2}{z_i+1}\right)^2} - 1 = \frac{4}{1-\left(1-\frac{4}{z_i+1}+\frac{4}{(z_i+1)^2}\right)} -$
  $1 = \frac{1}{\left(\frac{1}{z_i+1}-\frac{1}{(z_i+1)^2}\right)} - 1 = \frac{1}{\left(\frac{z_i+1-1}{(z_i+1)^2}\right)} - 1 = \frac{z_i^2+2z_i+1}{z_i} - 1 = z_i + 1 + \frac{1}{z_i}$
- $\Rightarrow i < z_i \leq 59 + 2i \Rightarrow z_i = \Theta(i) \Rightarrow p_i = \Theta\left(\frac{1}{i}\right)$
- Recursion depth i $= O(\log n) \Rightarrow Success = \Theta\left(\frac{1}{\log n}\right)$

SUCCESS PROBABILITY REPETITION

- One run succeeds with $\Omega\left(\frac{1}{\log n}\right)$ probability.
- We run $\log^2 n$ times.
- $\Pr(\text{At least one run succeeds})$

$$= 1 - \left(1 - \frac{1}{\log n}\right)^{\log^2 n}$$

$$= 1 - \left(1 + \frac{1}{-\log n}\right)^{(-\log n)\cdot(-\log n)}$$

$= 1 - e^{-\log n} = 1 - \frac{1}{n} \Rightarrow$ Error probability in $O\left(\frac{1}{n}\right)$

COMPARISON

Algorithm	Runtime	Success	Implementation
Brute Force	$O(2^n \cdot m)$	1	easy
Max-flow based	$\tilde{O}(nm)$	1	hard
Karger's	$O(n^4 \log n) = \tilde{O}(n^4)$	$1 - O(1/n^c)$	easy
Karger+Stein	$O(n^2 \log^3 n) = \tilde{O}(n^2)$	$1 - O(1/n)$	still easy

- K+S is Monte Carlo (might return sub-optimal)
- Usual conversion to Las Vegas (might take longer) by checking and repeating is not possible

PARALLELIZATION

Pan An

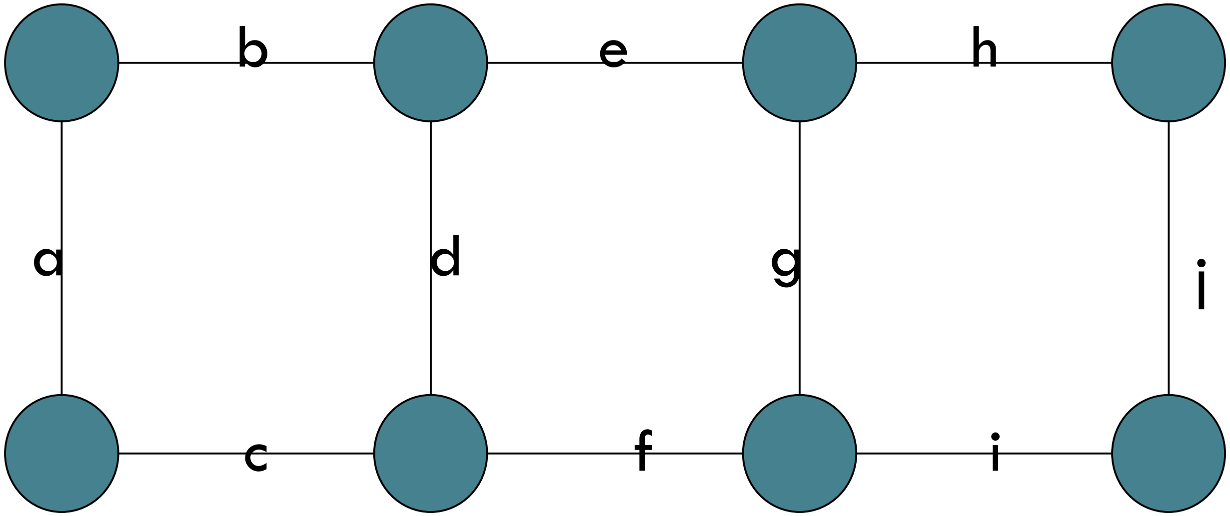
PARALLELISM - COMPACT

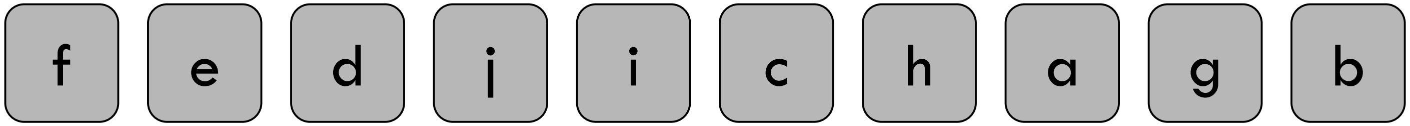
- Definitions:
  - L: an ordered sequence of all edges $l_1, l_2, \dots, l_n$;
  - V: set that contains all vertices;
  - L' : prefix of L;
  - $H(V, L')$: graph composed by edge set L' and vertex set V;
  - L^α: prefix of L, $l_1, l_2, \dots, l_\alpha$ where $\alpha \leq n$;
  - $f_c(G)$: number of connected components in G;
  - L_1/L_2: edges in L_1 after contraction of all edges in L_2

Compact is a method to find a prefix $L^\alpha = l_1, l_2, \dots, l_\alpha$ where:

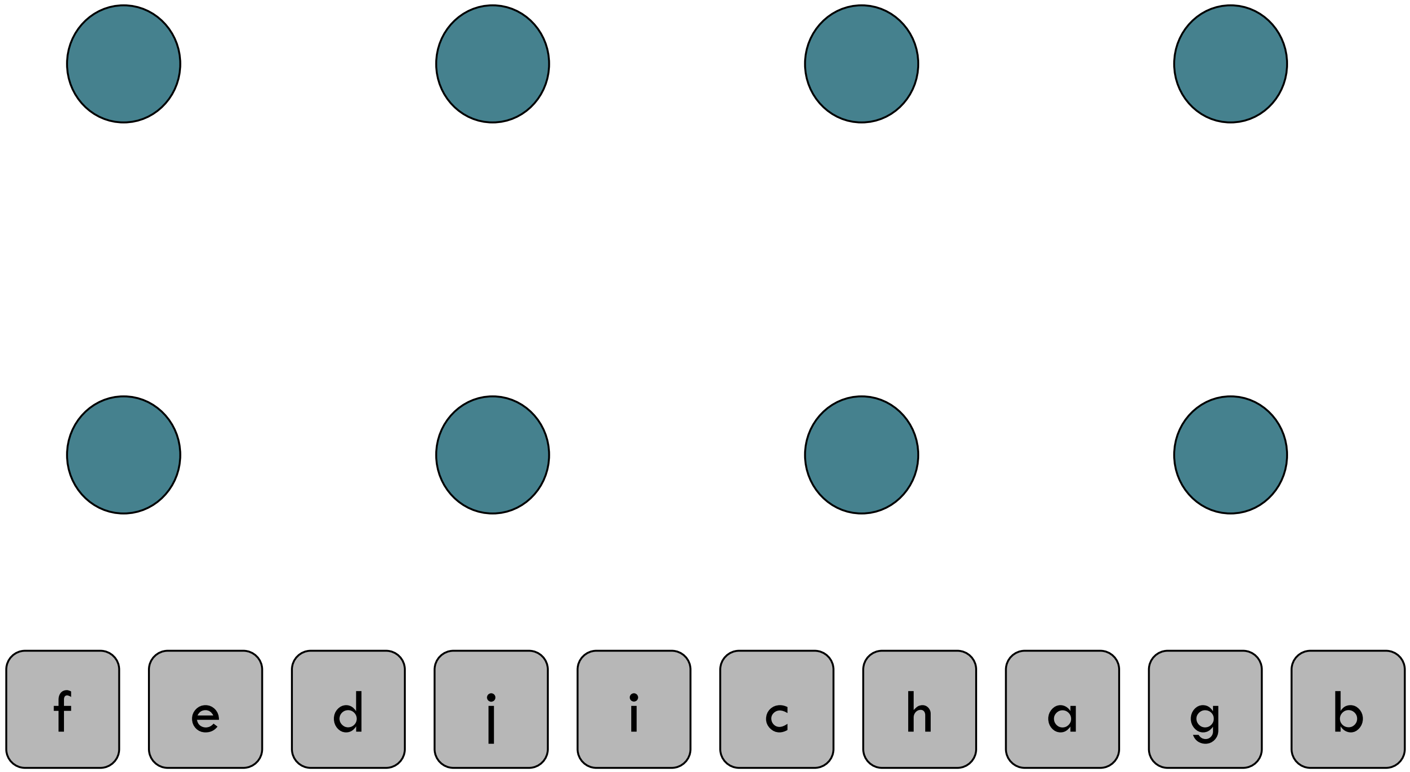
$$f_c\big(H(V, L^\alpha)\big) = k \text{ and } f_c\big(H(V, L^{\alpha-1})\big) < k$$

CONTRACT = FINDING PREFIX

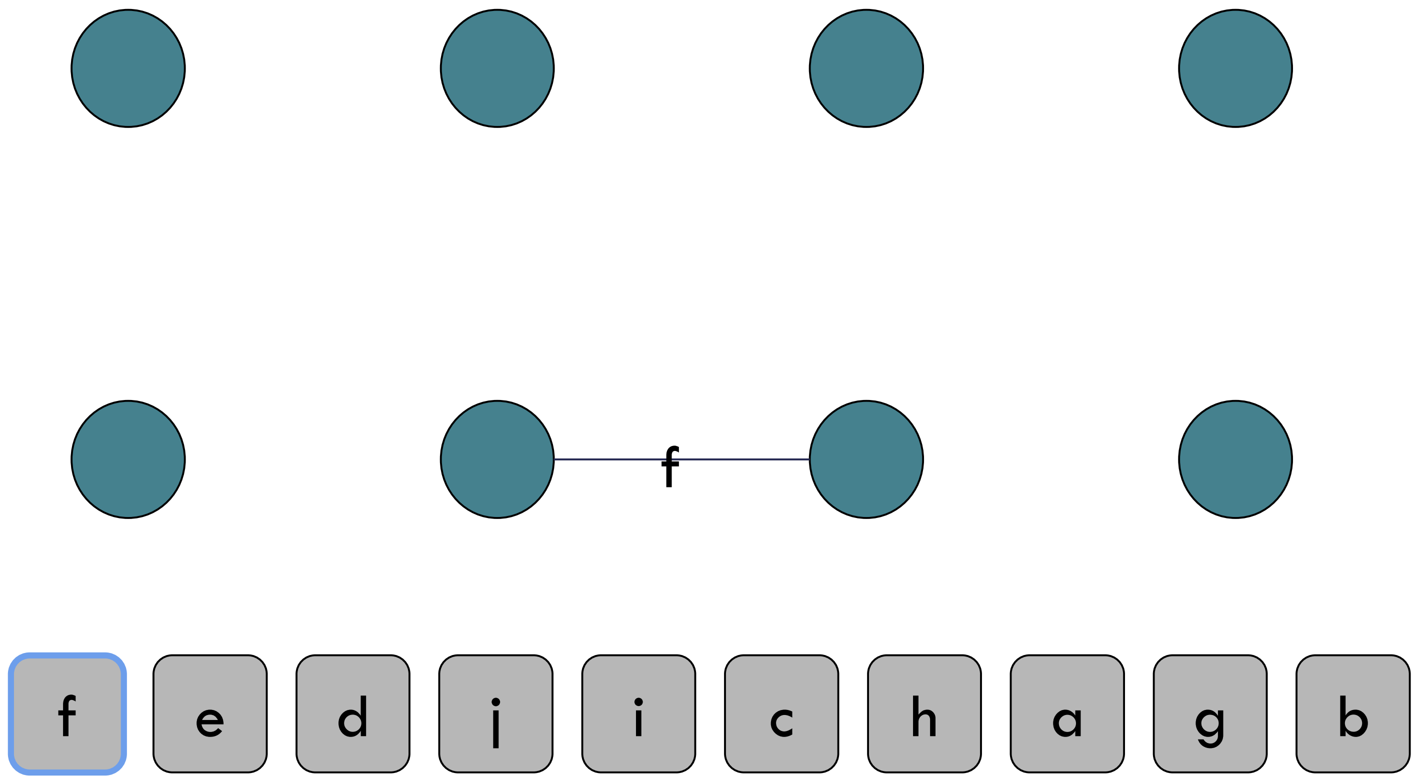




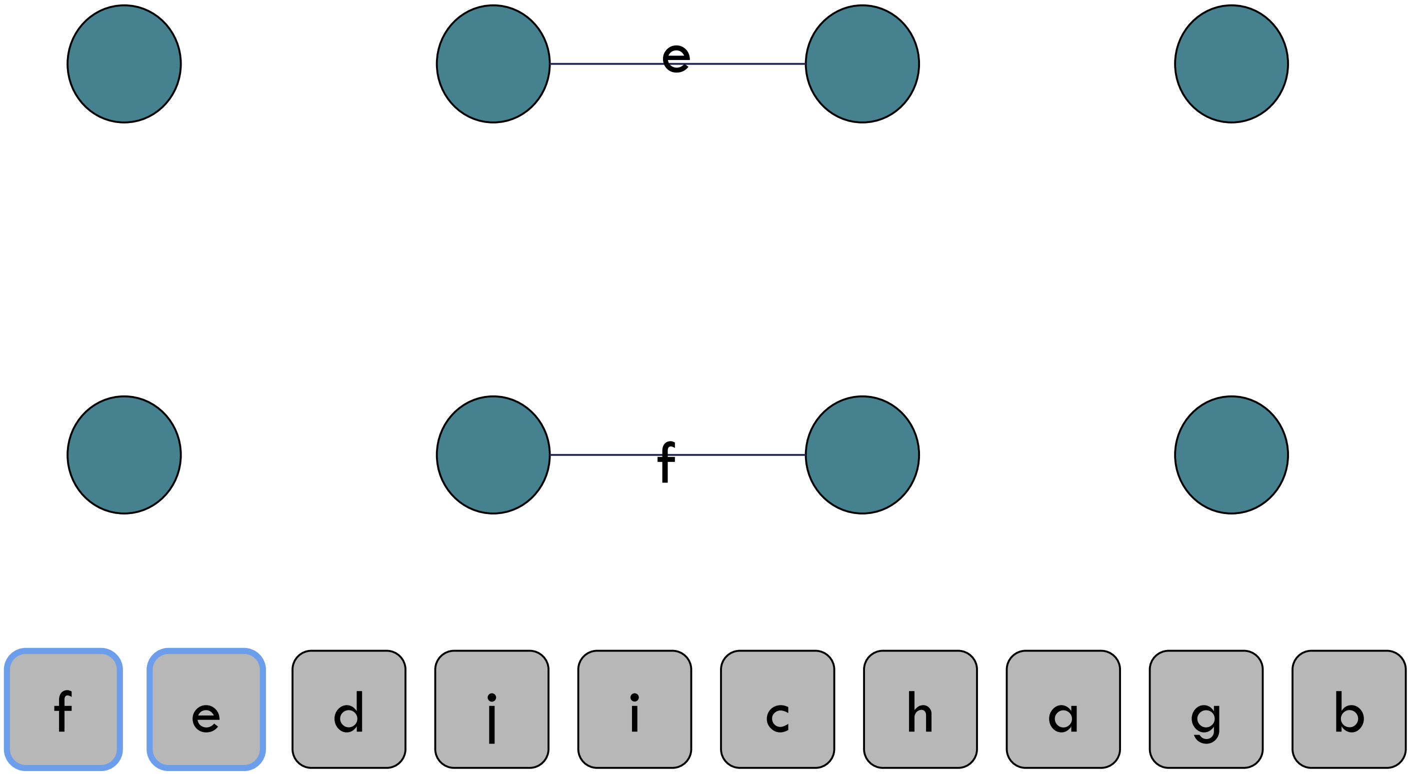
CONTRACT = FINDING PREFIX



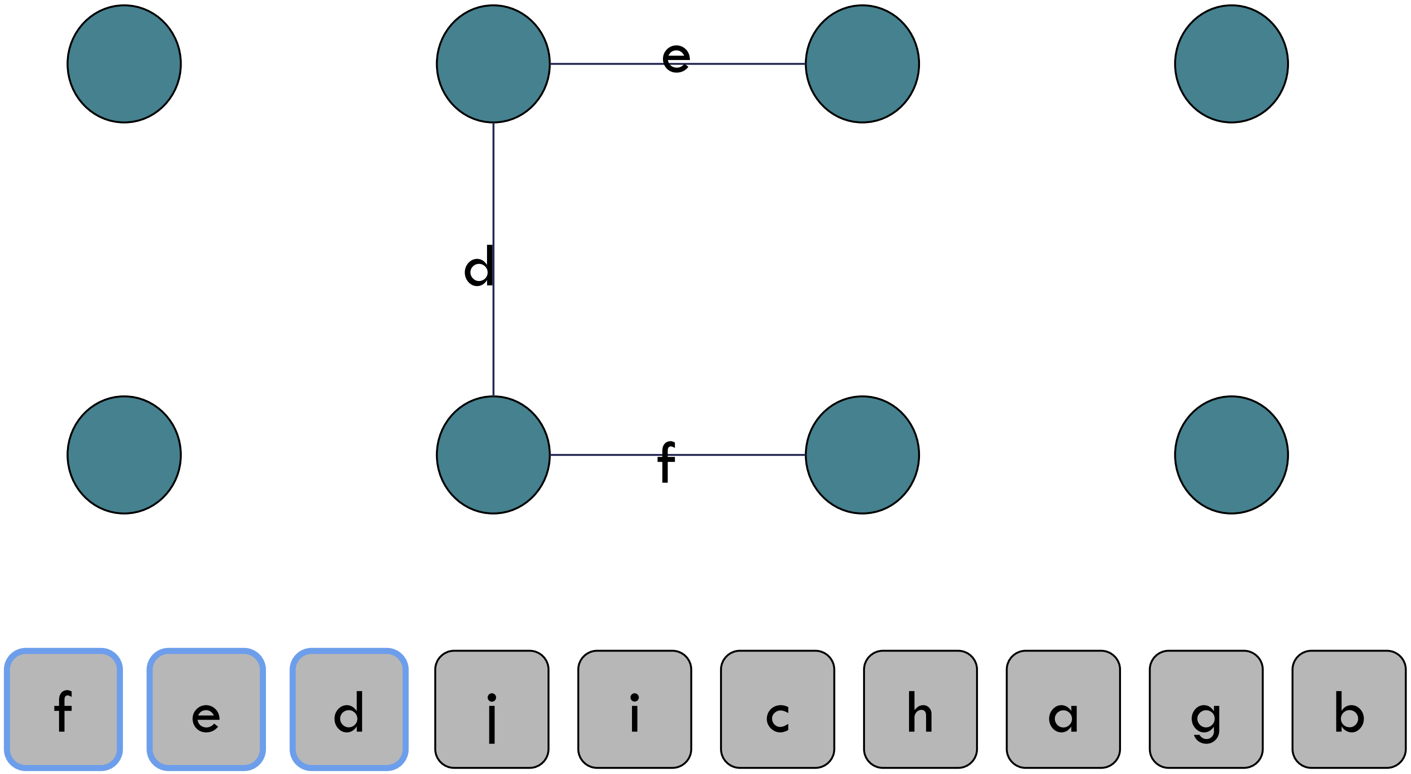
CONTRACT = FINDING PREFIX



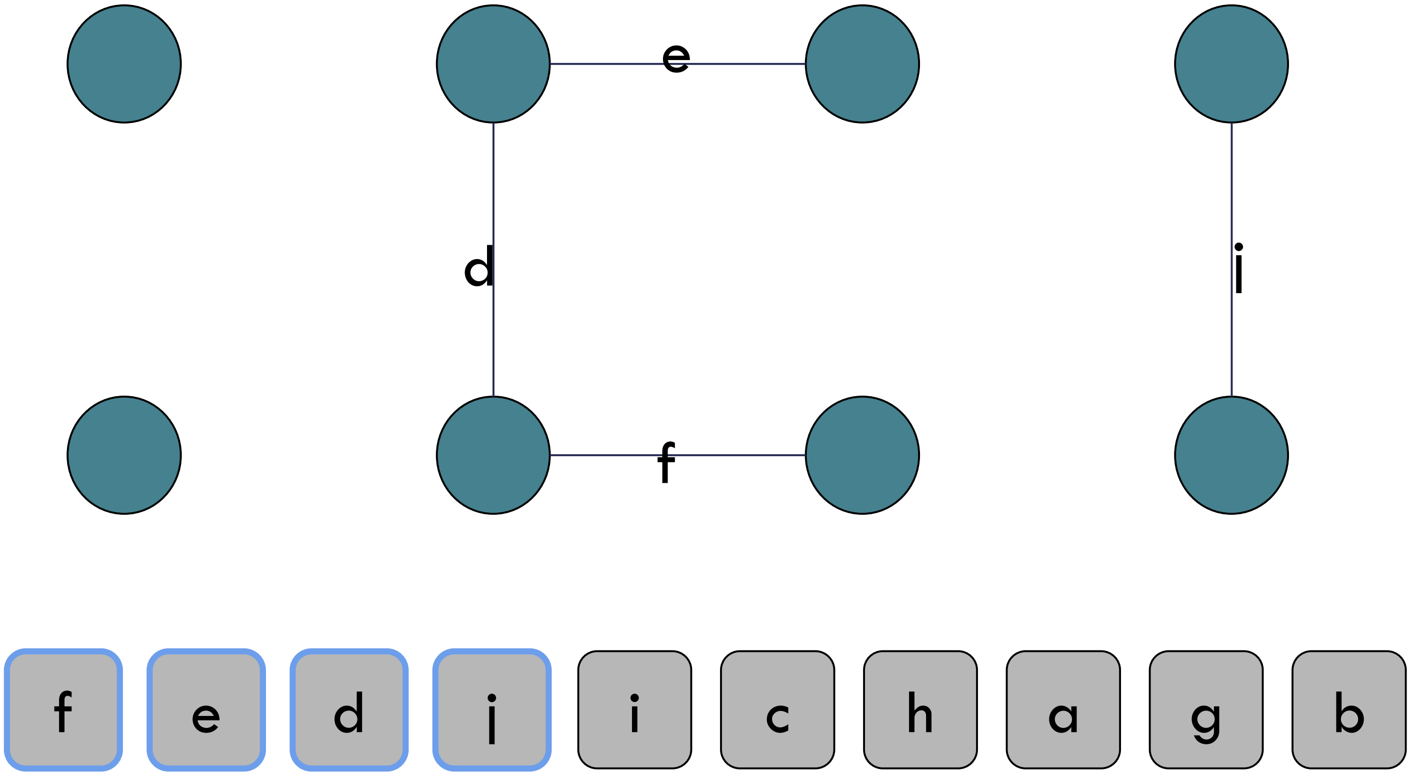
CONTRACT = FINDING PREFIX



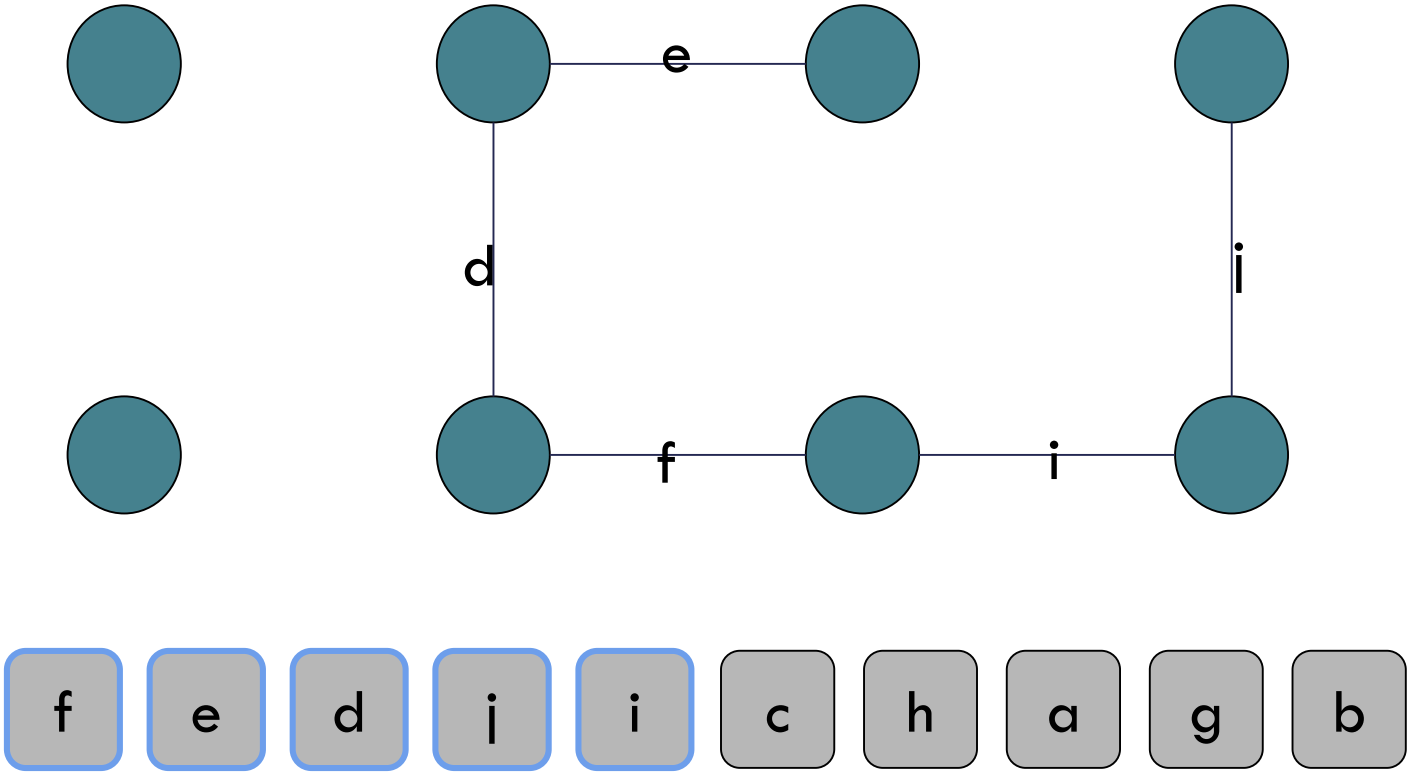
CONTRACT = FINDING PREFIX



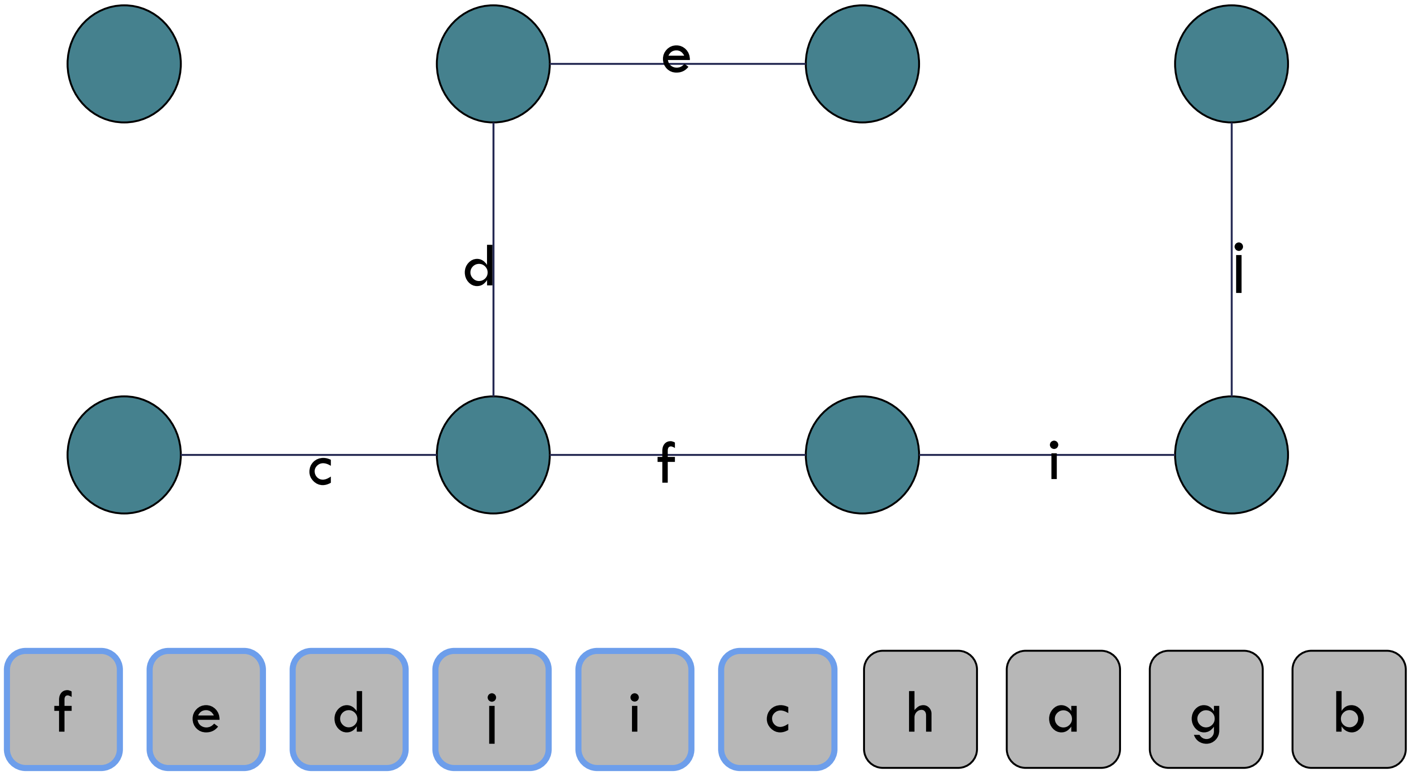
CONTRACT = FINDING PREFIX



CONTRACT = FINDING PREFIX



CONTRACT = FINDING PREFIX



PARALLELISM – COMPACT

- Definitions:
  - L: an ordered sequence of all edges $l_1, l_2, \ldots, l_n$;
  - V: set that contains all vertices;
  - L' : prefix of L;
  - $H(V, L')$: graph composed by edge set L' and vertex set V;
  - L^α: prefix of L, $l_1, l_2, \ldots, l_\alpha$ where $\alpha \leq n$;
  - $f_c(G)$: number of connected components in G;
  - L_1/L_2: edges in L_1 after contraction of all edges in L_2

Compact is a method to find a prefix $L^\alpha = l_1, l_2, \ldots, l_\alpha$ where:

$$f_c\big(H(V, L^\alpha)\big) = k \text{ and } f_c\big(H(V, L^{\alpha-1})\big) < k$$

COMPACT – OVERVIEW

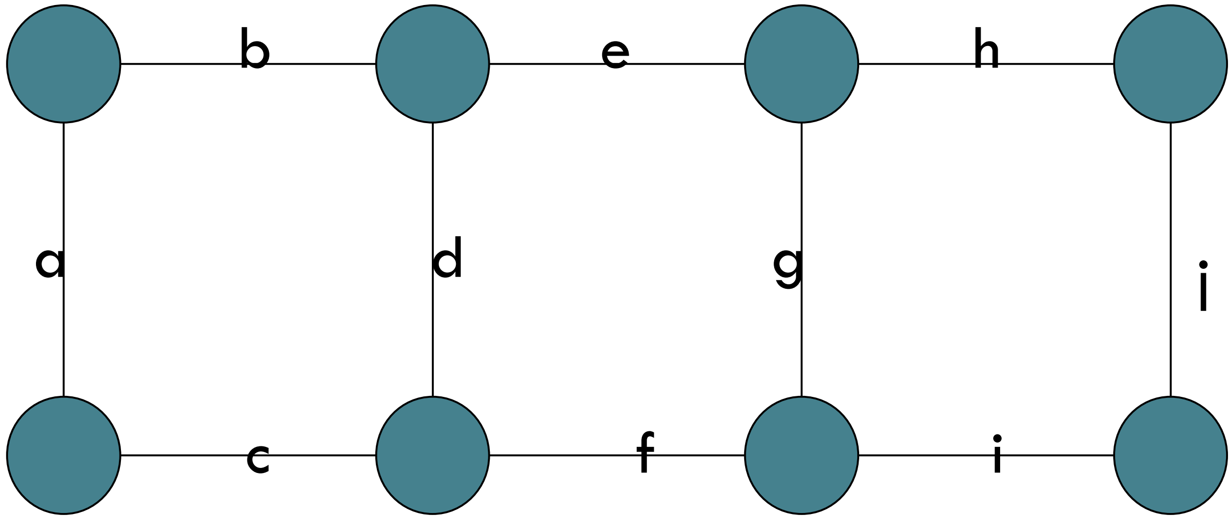
- Using binary search, the correct prefix can be determined using $O(\log m)$ **connected component computations**, where m is the number of edges;
- Each connected component computation requires $O(m+n)$ time;
- Only 1 processor used so far.
- Running time of this algorithm is $O(m \log m)$;
- This can be further reduced to $O(m)$ by reusing information between iterations.

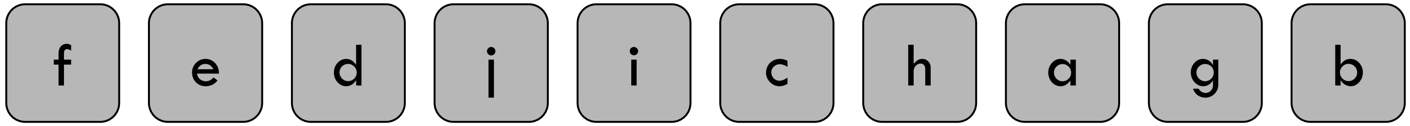
COMPACT – ALGORITHM

Parallel Algorithm:

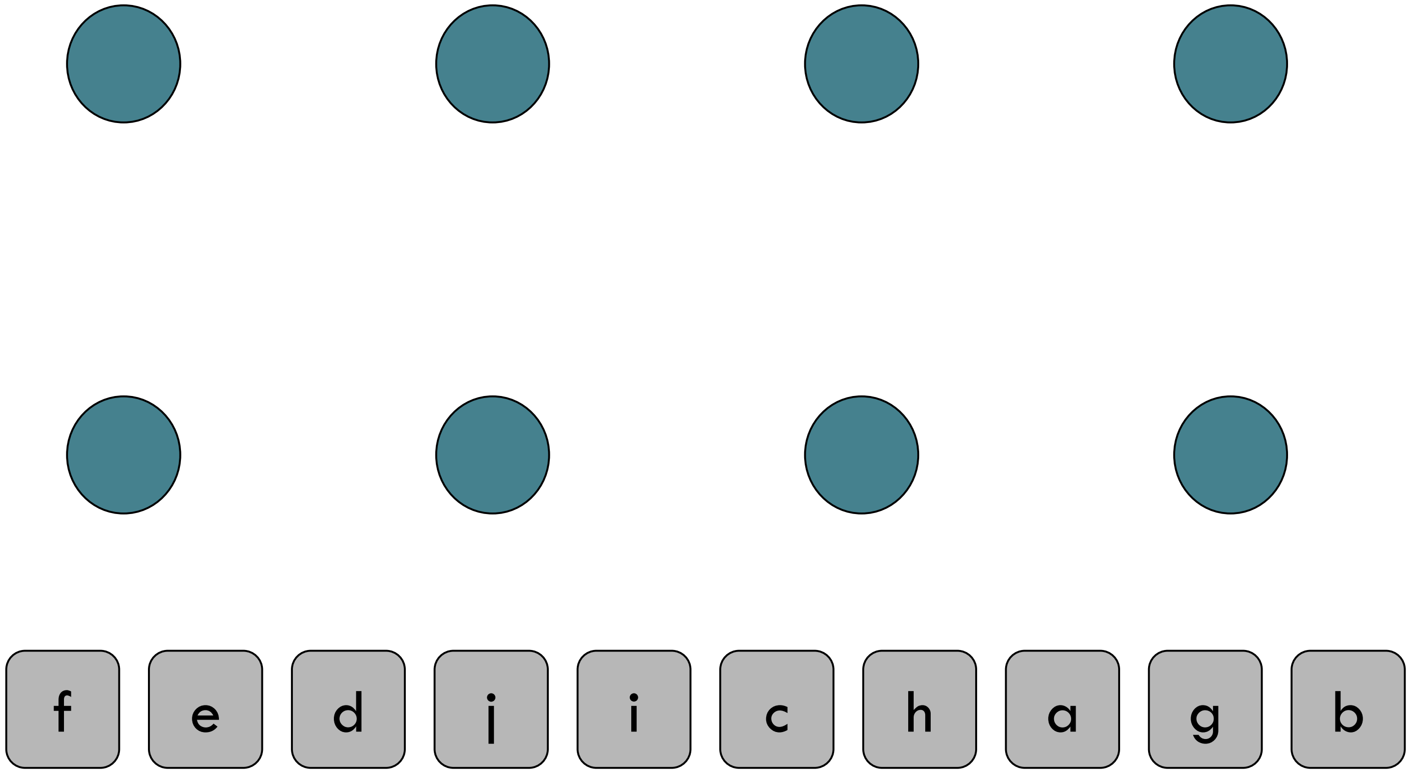
```
COMPACT(G, L, k)
Data: A graph G, list of edges L, and parameter k
if G has k vertices or L = ϕ (empty) then
|   return G
else
|   Let L1 and L2 be the first and second half of L
|   if H has fewer than k connected components then
|   |   return COMPACT(G, L1, k)
|   else
|   |   return COMPACT(G/L1, L2/L1, k).
|   end
end
```

COMPACT – EXAMPLE

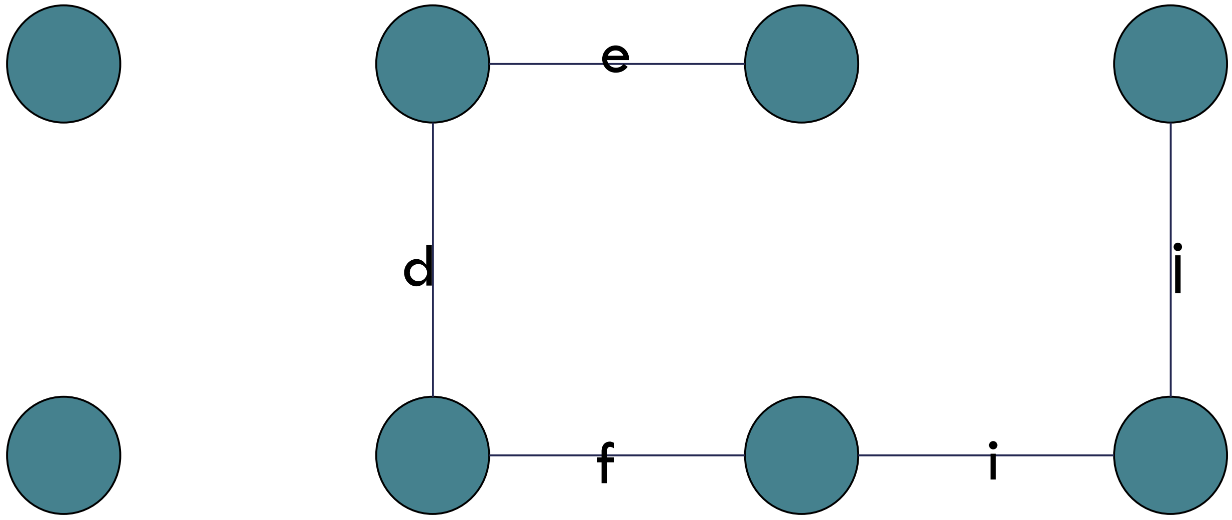


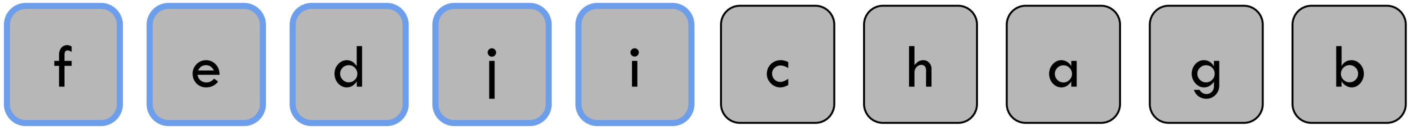


COMPACT – EXAMPLE

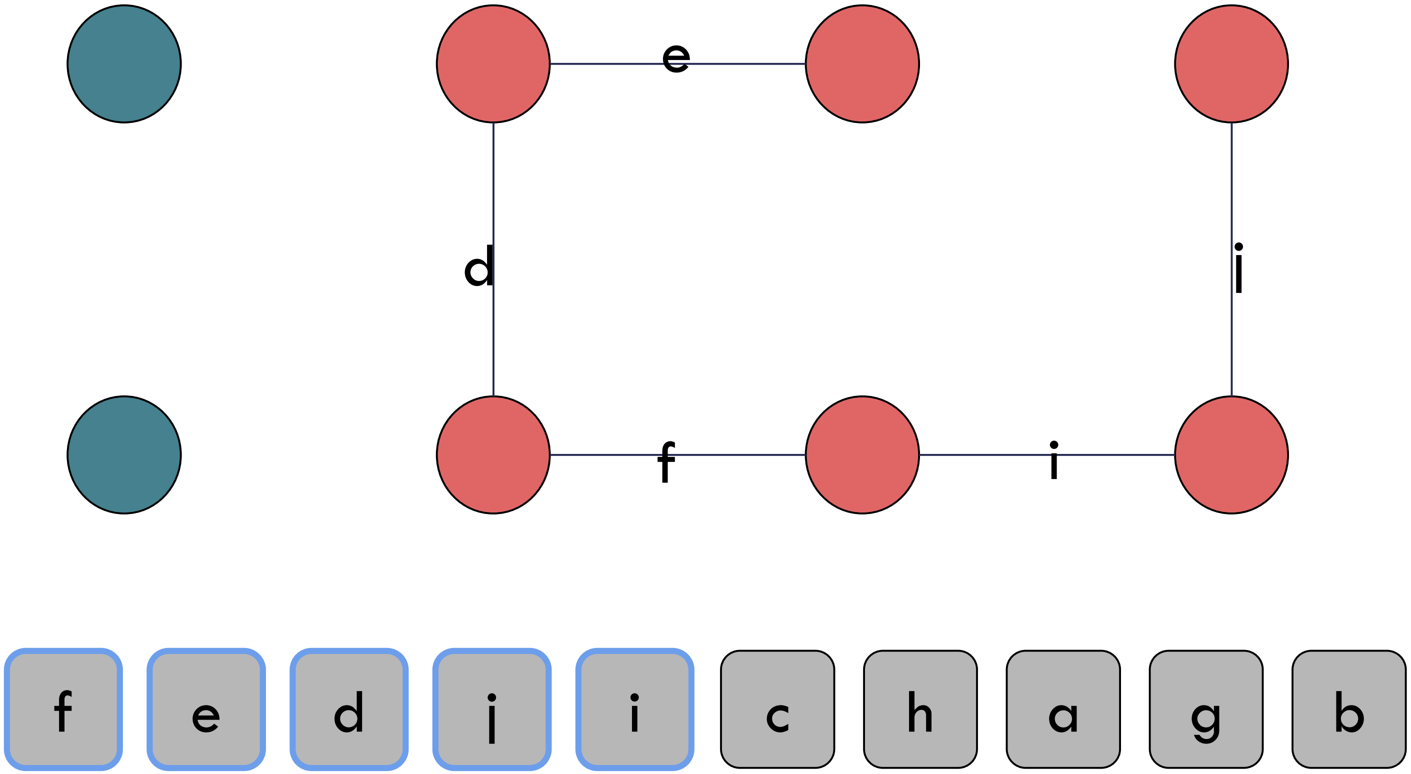


COMPACT – EXAMPLE

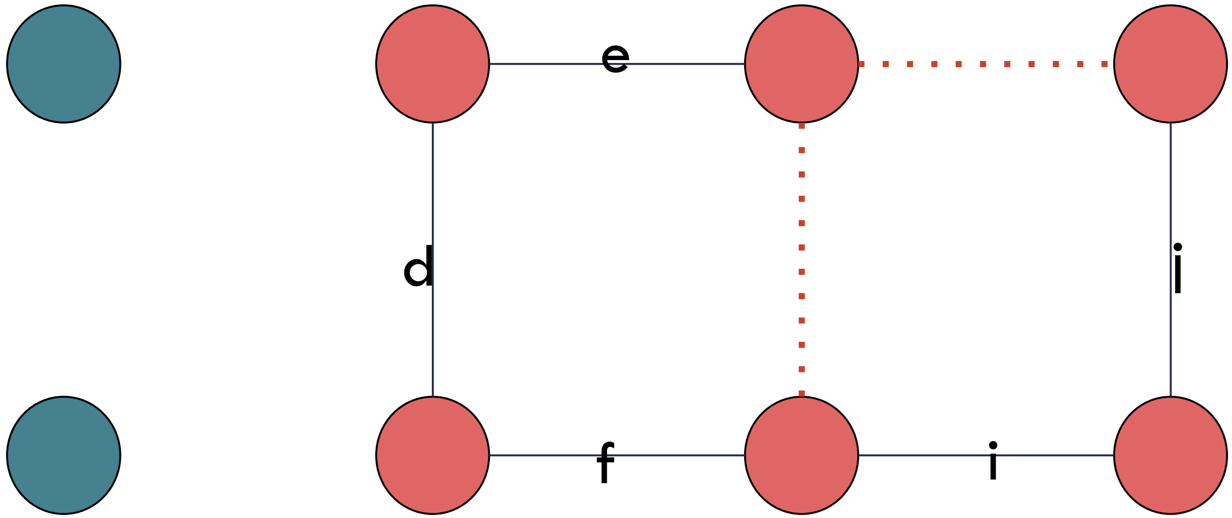


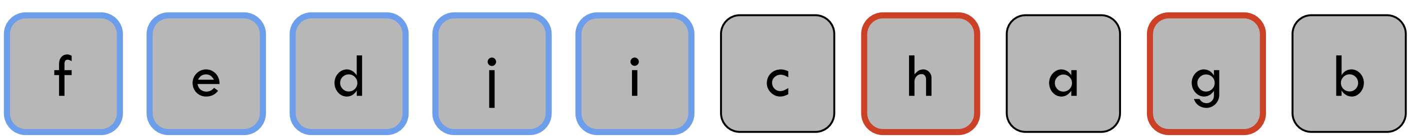


COMPACT – EXAMPLE

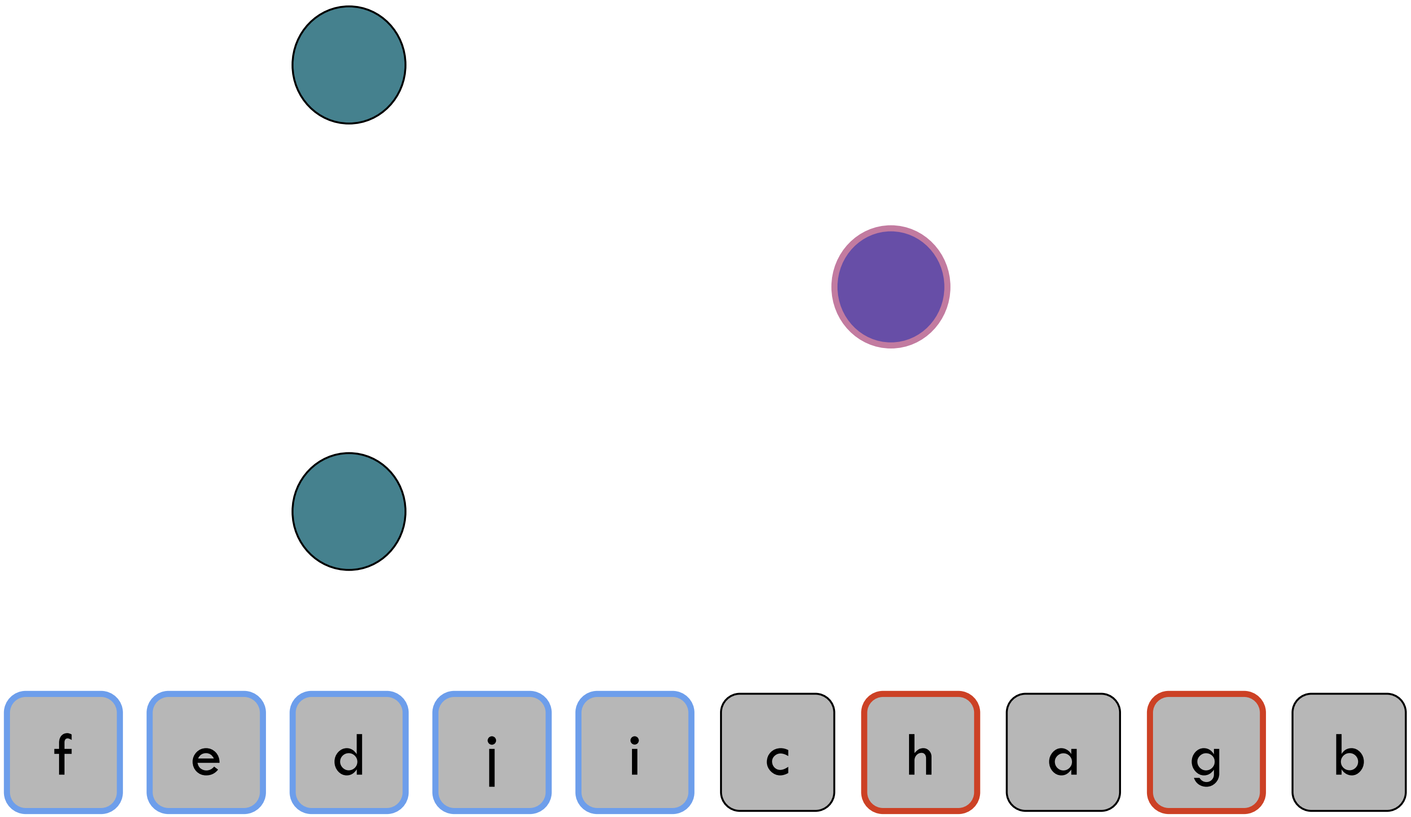


COMPACT – EXAMPLE

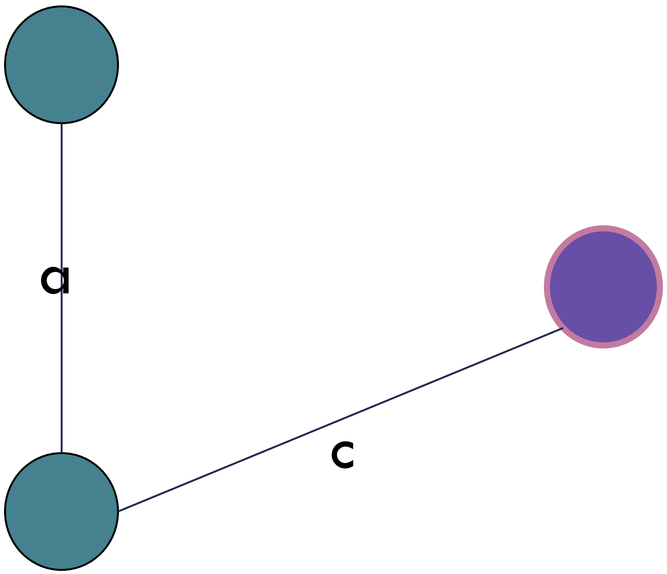




COMPACT – EXAMPLE

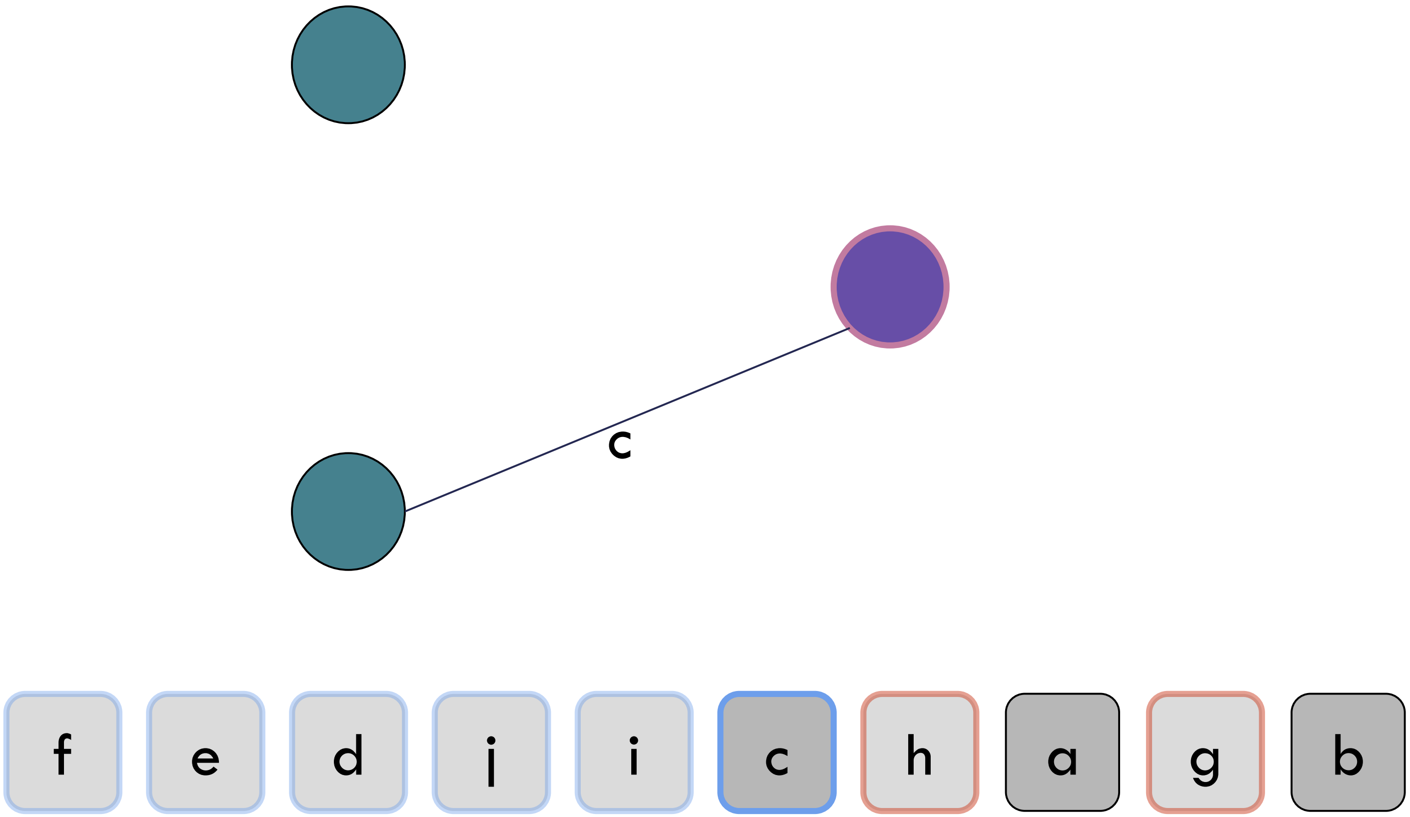


COMPACT – EXAMPLE

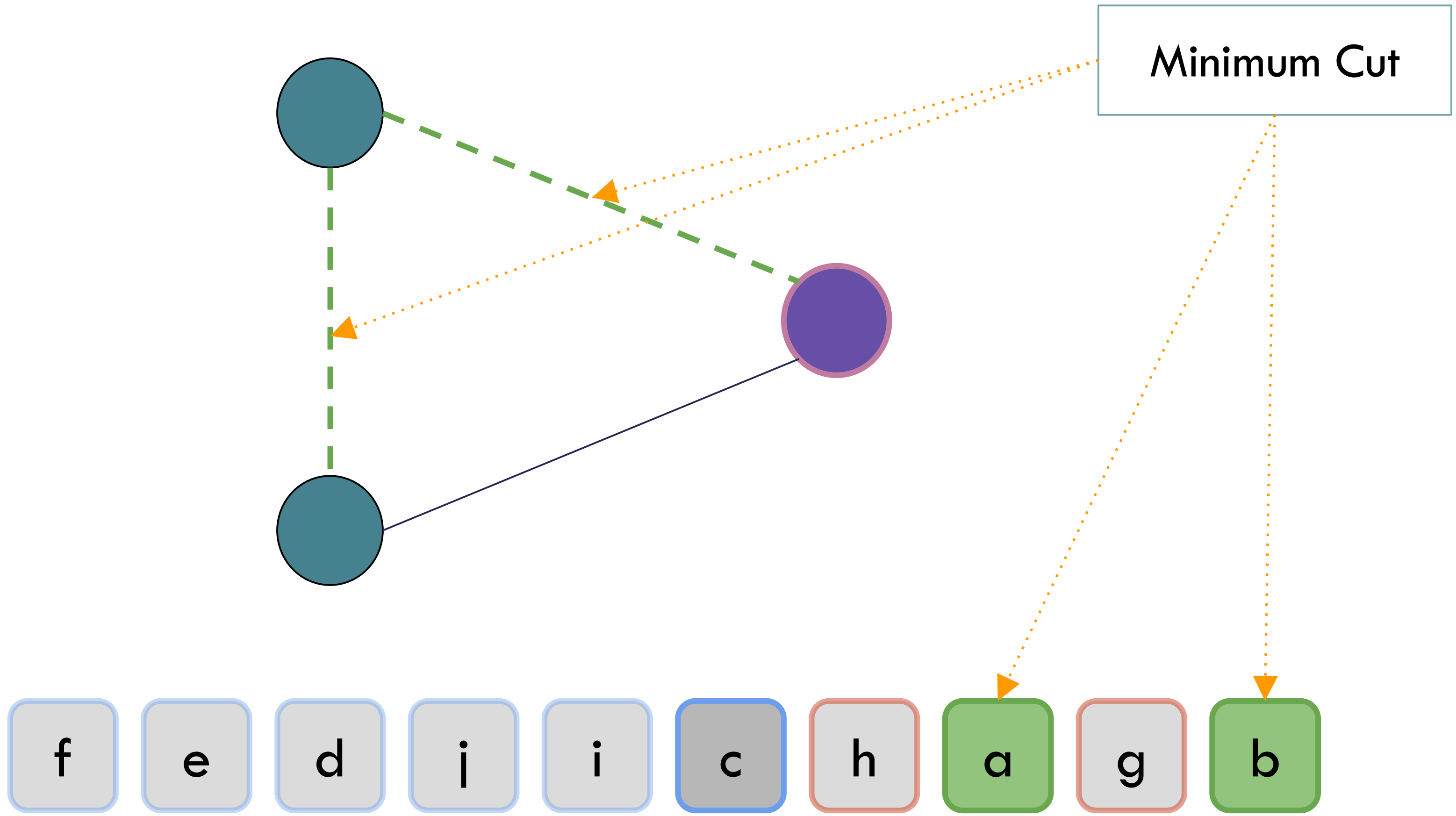


f	e	d	j	i	c	h	a	g	b

COMPACT – EXAMPLE



COMPACT – EXAMPLE



COMPACT – SEQUENTIAL

- E1. Creation of random sequence L → $O(m)$
- E2. Binary search → $\mathrm{O}(\log m)$ rounds
- E3. Connected components → $O(m)$
- E4. Contraction → $O(m)$

Time complexity is $O(\log m) \times O(m) = O(m \log m)$

COMPACT – PARALLELIZING THE PERMUTATION

- Permutation generation time should be $O(1)$;
- If G is unweighted, uniform sampling can be used for random number generation;
- For a weighted graph we need to achieve the following distribution on $I_r = [0, r]$:

$$\Pr[X > t] = \left(1 - \frac{t}{r}\right)^{wr}$$

As when r becomes insanely big: $\Pr[X > t] = e^{-wt}$.

This must be achieved at $O(1)$ time!

COMPACT – PARALLELIZING THE PERMUTATION

- Definitions:
  - U : random variable uniformly distributed on [0, 1];
  - U': approximated variable of U;
  - $R^{O(1)}$: random number generated with constant time(and bits);

We need to generate X:

$$\underline{\Pr[X > t] = e^{-wt} \rightarrow X = -(\ln U)/w}$$

Obstacles:

1. Uniform distribution on [0, 1] is not possible in real machine;
2. Computing $\underline{\ln U}$ might take time;

COMPACT – RANDOM NUMBER GENERATION

Method – Exponentially Distributed Random Variable:

$R^{O(1)}$: random number generated with constant time (and bits)

A1. Choose an integer $M = R^{O(1)}$

A2. Select an integer N from [1, M] using $O(\log R)$ random bits

A3. $U' = \frac{N}{M}$, ; U' is then the approximation of U

A4. Compute $\mathrm{X} = -\frac{\ln \mathrm{U}'}{\mathrm{w}}$ where we use the first $O(\log R)$ terms of the Taylor expansion of $\ln \mathrm{U}'$;

COMPACT – RANDOM NUMBER GENERATION

- If we let $x = U' - 1$:

$$\ln(1+x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} x^n = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots \quad \text{for} \quad |x| \le 1,$$

COMPACT – PARALLELIZATION

- Parallel:
  - Generation of random sequence L → $O(1)$
  - Assigning each node a processor. Each processor assigns a random number to its edge at the beginning of each round.
  - Do binary search with parallelism:
    - The algorithm chooses a value t
    - a processor returns its edge for next contraction if X > t.

Step E3., E4. can also be parallelized. For E3, a paper has been posted to the IVLE forum, showing connected component detection in $O(\log n)$ time.

COMPACT – PARALLELIZATION

E1. is the only step that is explained in detail in the original paper.

- E1. Creation of random sequence L → $O(1)$
- E2. Binary search → $\mathrm{O}(\log m)$ rounds
- E3. Connected components → $O(\log n)$
- E4. Contraction → $O(1)$

Time complexity is

$$O(\log m) \cdot \big(O(1) + O(\log n)\big) = O(\log^2 n)$$

using $m = O(n^2)$ processors

COMPACT – THEOREMS

- RNC (Randomized Nick's Class): Solvable in $O(\log^c n)$ time with $O(n^d)$ processors (for some c, d).
- Compact method is RNC because it takes $O(\log^2 n)$ time using $m = O(n^2)$ processors.
- Minimum cut problem is RNC because the recursion tree (logarithmic depth) can be processed breadth-first and because the $O(\log^2 n)$ retries can be run at the same time in parallel.
- Similarly, algorithms can be found to solve the minimum k-cut problem in RNC.

APPLICATIONS

Taehoon Kim

APPLICATIONS

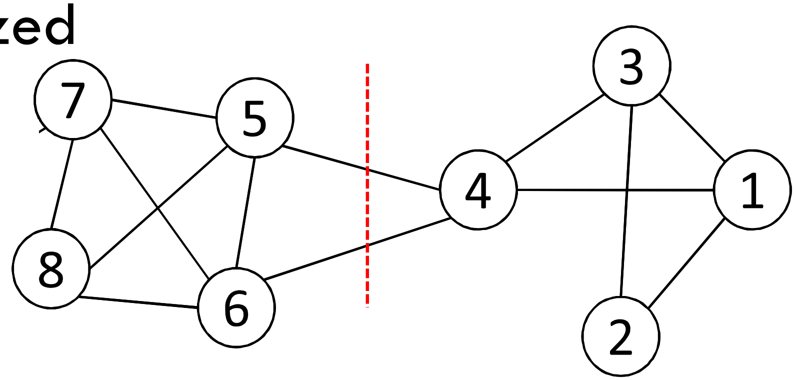
- Splitting large graphs
- Community detection
- Weakness on a network
- Detecting weak ties

APPLICATIONS – SPLITTING LARGE GRAPHS

- Real world graphs are large
  - Sometimes they are too large to compute
- Objective:
  - Less computation
  - Better understanding of the data
    - Even after the graph is divided, the graph still maintains its structural characteristics
- Use min-cut to divide one large graph into several smaller graphs

APPLICATIONS – COMMUNITY DETECTION

- Community on social media:
  - Formed by individuals
  - Individuals within the same community interact more frequently
- Community detection:
  - Discovering groups in a social network
- Min-cut on community detection:
  - Find a graph partition such that the number of edges between the two sets is minimized



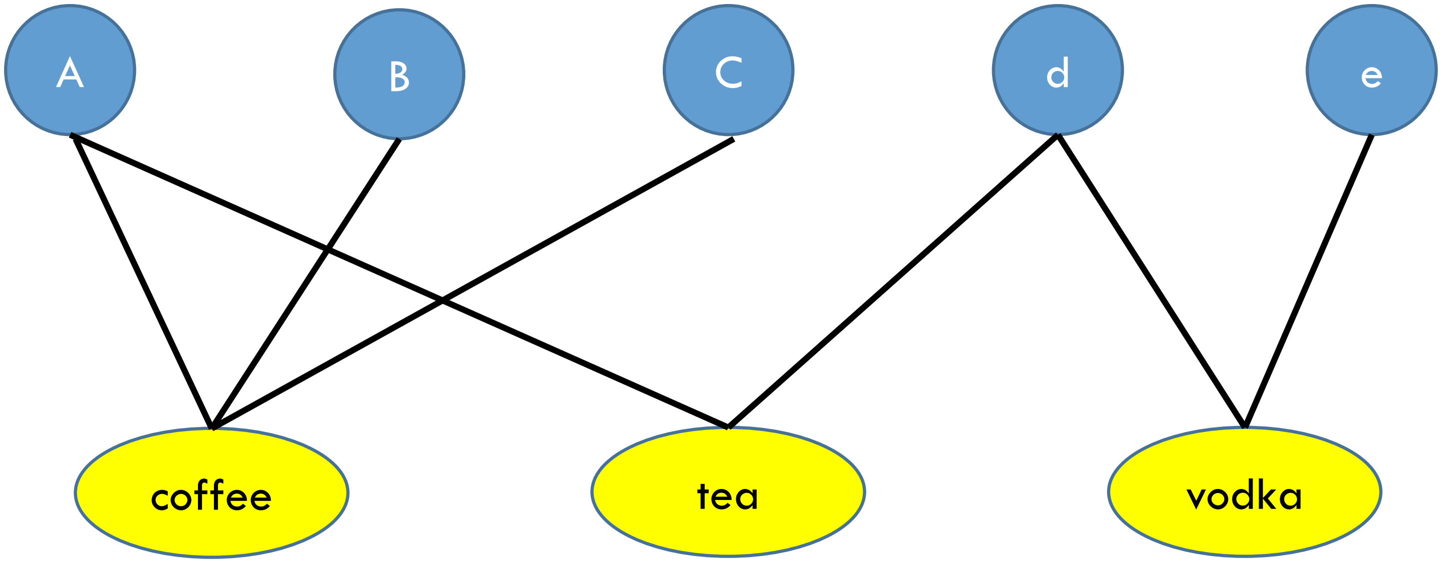
APPLICATIONS – COMMUNITY DETECTION

- Edges: Interaction counts
  - Location
    - user communications in Twitter exhibit strong geographic locality (Zhang *et al. CNS, IEEE 2015*)
  - Closeness

- Applications:
  - Localized Marketing
  - Friend recommendation
  - Place recommendation
  - Privacy risks

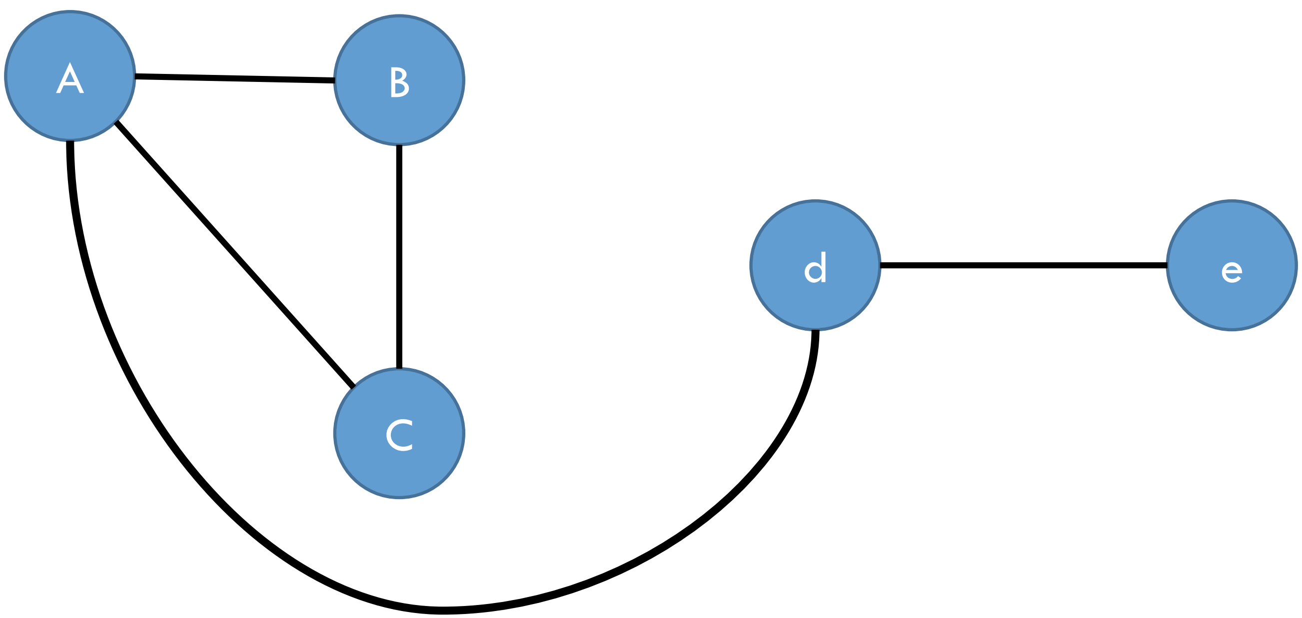
APPLICATIONS – COMMUNITY DETECTION

- Edges: common interests
- Applications:
  - Collaborative filtering based recommendation system
  - Friend recommendation



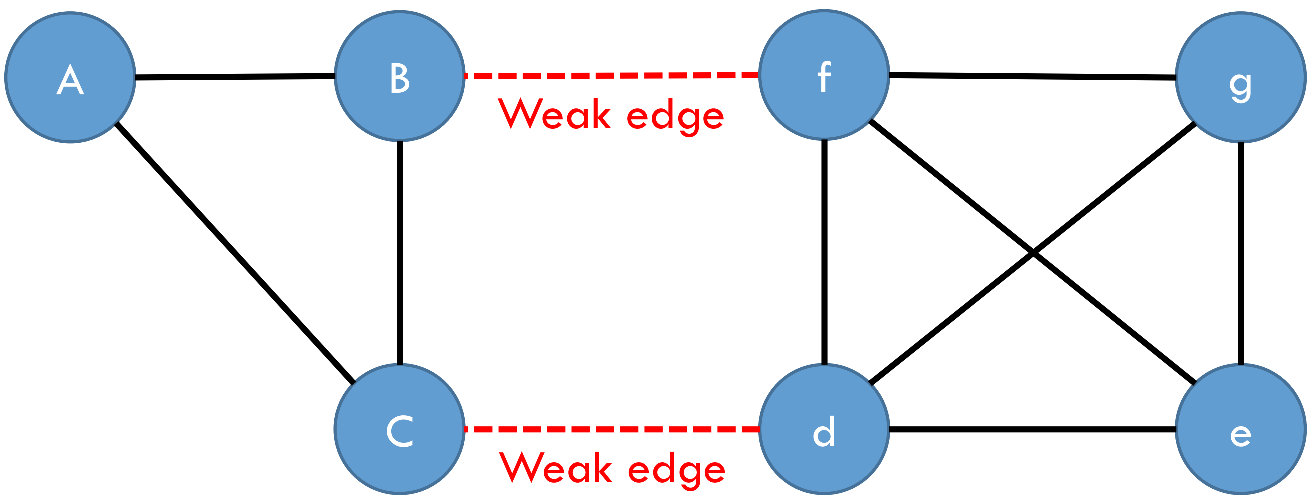
APPLICATIONS – COMMUNITY DETECTION

- Edges: common interests
- Applications:
  - Collaborative filtering based recommendation system
  - Friend recommendation



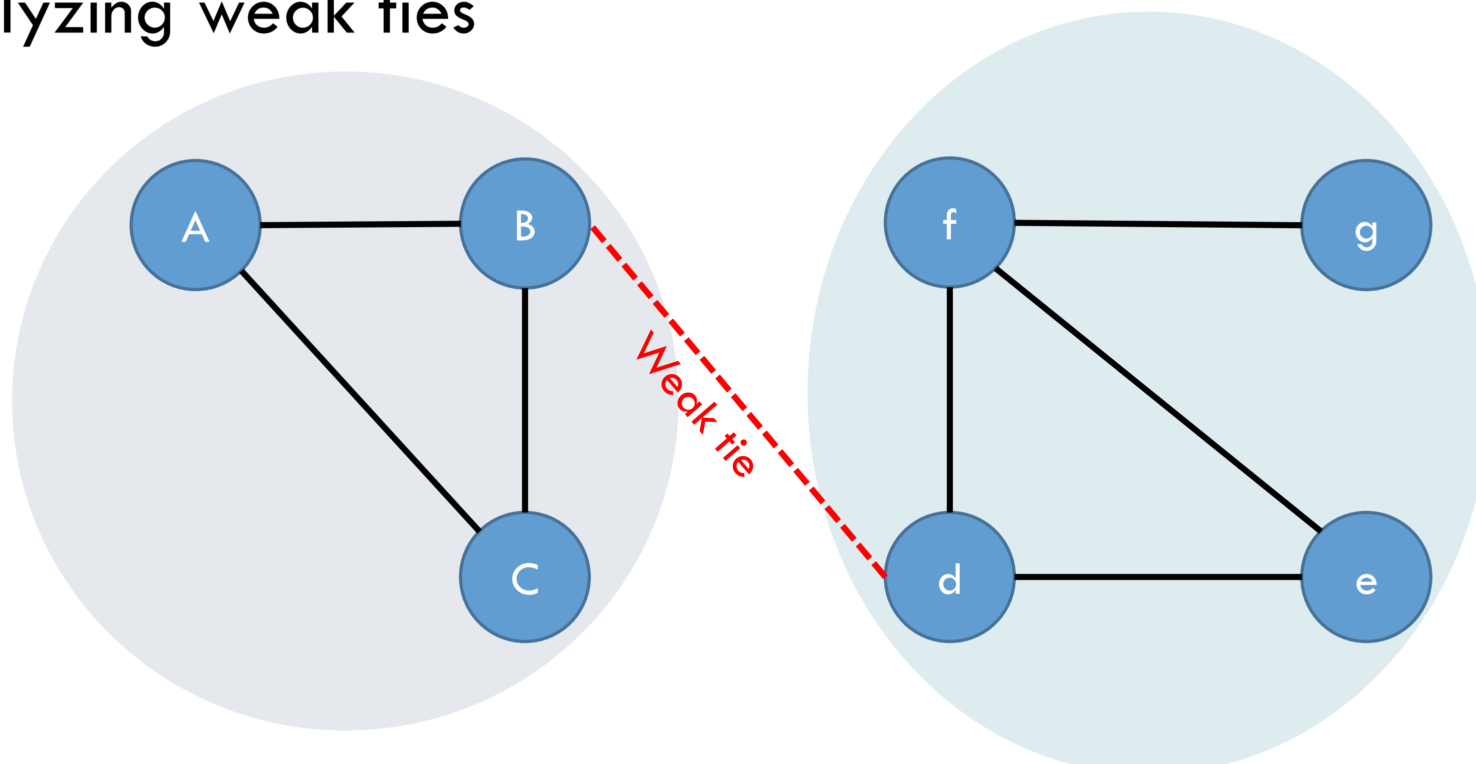
APPLICATIONS – WEAKNESS ON NETWORK

- Find vulnerable connections on a network
  - Weak edges
- Example:
  - Vulnerability on Sensor Network
    - Each node has limited range
    - Finding sink node



APPLICATIONS – WEAK TIES

- Weak ties in social media
  - (Granovetter 1973)
- Analyzing weak ties



Community A

Community B

CONCLUSION

- The min-cut problem has many variations (directed, undirected, weighted, multiway cut) and many applications.
- Min-cut can be solved using max-flow based techniques.
- Karger introduced an algorithm that solves it directly.
  - Because only few edges cross the min-cut, they are unlikely to be contracted.
- Karger and Stein improved this algorithm to become
  - faster than max-flow based algorithms (but only on dense graphs) and
  - parallelizable.
- The algorithm is easier to implement, but it is also a Monte Carlo algorithm.

CONCLUSION

Algorithm	Runtime	Success	Implementation
Brute Force	$O(2^n \cdot m)$	1	easy
Max-flow based	$\tilde{O}(nm)$	1	hard
Karger's	$O(n^4 \log n) = \tilde{O}(n^4)$	$1 - O(1/n^c)$	easy
Karger+Stein	$O(n^2 \log^3 n) = \tilde{O}(n^2)$	$1 - O(1/n)$	still easy

- The minimum cut problem can be solved in RNC using n^2 processors.

REFERENCES

1. Karger, David R. "Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm." *SODA*. Vol. 93. 1993.
2. Karger, David R., and Clifford Stein. "A new approach to the minimum cut problem." *Journal of the ACM (JACM)* 43.4 (1996): 601-640.
3. Arora, Sanjeev. “Lecture 2: Karger’s Min Cut Alogirhtm”. Princeton University F’13 COS 521: Advanced Algorith Design. https://www.cs.princeton.edu/courses/archive/fall13/cos521/lecnotes/lec2final.pdf
4. Roughgarden, Tim. “Algorithms: Design and Analysis, Part 1”. Coursera Lecture. https://www.coursera.org/course/algo
5. Zhang, Jinxue, et al. "Your actions tell where you are: Uncovering Twitter users in a metropolitan area." *Communications and Network Security (CNS), 2015 IEEE Conference on*. IEEE, 2015.
6. Granovetter, Mark S. "The strength of weak ties." *American journal of sociology* (1973): 1360-1380.