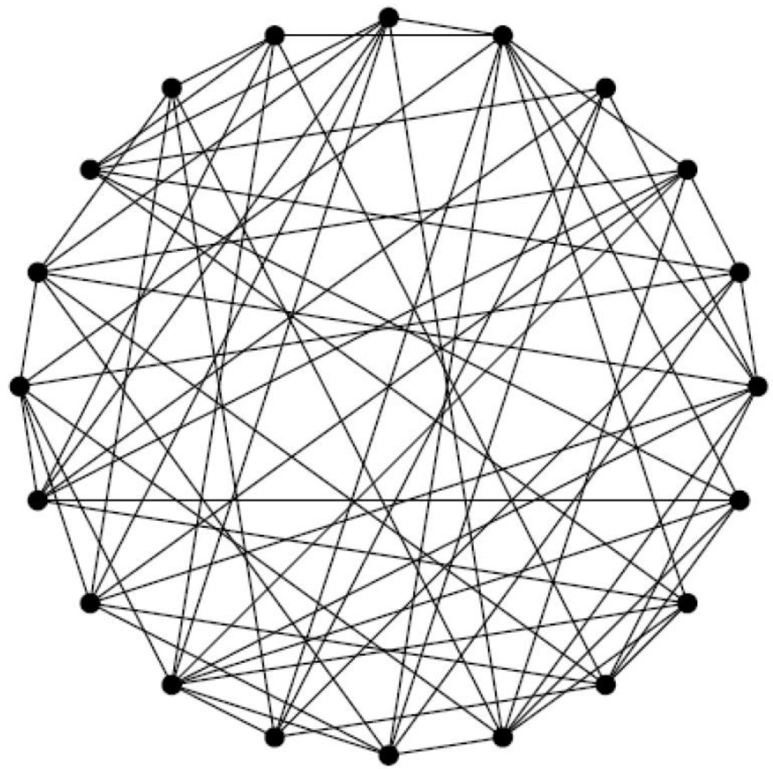
Using Linear Algebra in Algorithms Example 1: Perfect Matching

Abbas Mehrabian

a place of mind
THE UNIVERSITY OF BRITISH COLUMBIA

Acknowledgement: some of the following slides were prepared by Nick Harvey and are downloaded from http://www.cs.ubc.ca/~nickhar/Publications/PublicationsSelected.html

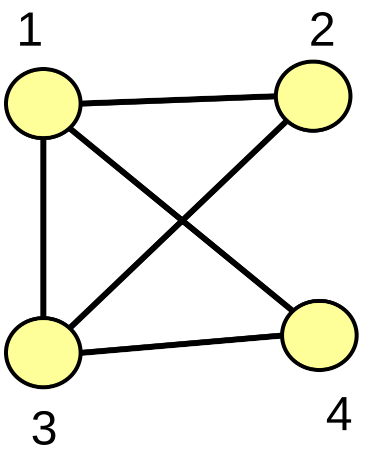
Toy problem: finding a triangle



- Running time: $\binom{n}{3} = O(n^3)$ … can we do better?

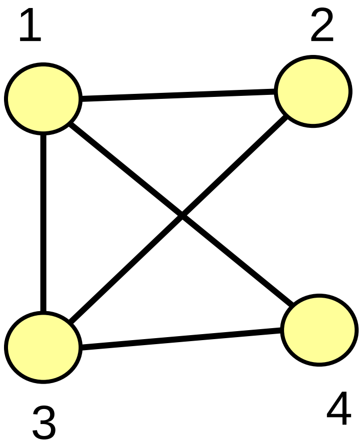
Adjacency matrix

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$



Adjacency matrix and its square

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

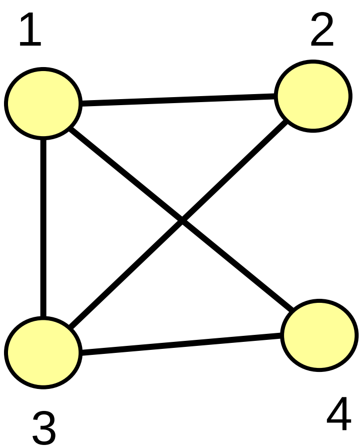


$$B = A^2 = \begin{bmatrix} 3 & 1 & 2 & 1 \\ 1 & 2 & 1 & 2 \\ 2 & 1 & 3 & 1 \\ 1 & 2 & 1 & 2 \end{bmatrix}$$

$$B_{i,j} = \sum_{k=1}^{n} A_{i,k} \times A_{k,j} = number\ of\ common\ neighbours\ of\ i\ and\ j$$

Adjacency matrix and its square

$$A = \begin{bmatrix} 0 & \boxed{1} & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$



$$B = A^2 = \begin{bmatrix} 3 & \boxed{1} & 2 & 1 \\ 1 & 2 & 1 & 2 \\ 2 & 1 & 3 & 1 \\ 1 & 2 & 1 & 2 \end{bmatrix}$$

So, vertices 1 and 2 are in a triangle!

$$B_{i,j} = \sum_{k=1}^{n} A_{i,k} \times A_{k,j} = number\ of\ common\ neighbours\ of\ i\ and\ j$$

How fast can we find A^2?

- Naïve multiplication: $O(n^3)$
- Strassen algorithm'69: $O(n^{2.81})$ for multiplying two nxn matrices

  (Very nice algorithm… read wikipedia or CLRS http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap31.htm)
- Coppersmith-Winograd'90: $O(n^{2.38})$

  (Best known exponent is Le Gall'14: 2.3728639)

How fast can we find A^2?

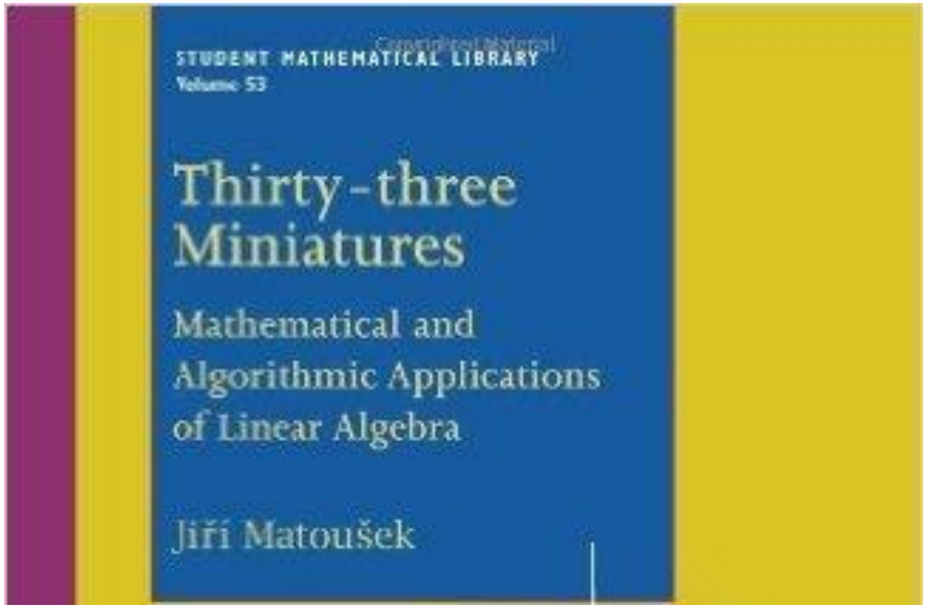
- Naïve multiplication: $O(n^3)$
- Strassen algorithm'69: $O(n^{2.81})$ for multiplying two nxn matrices

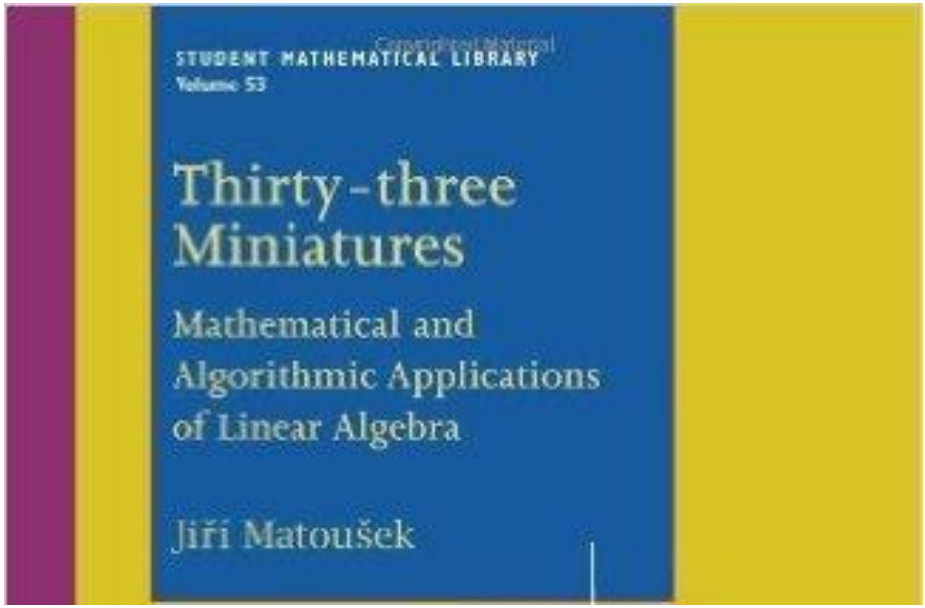
  (Very nice algorithm… read wikipedia or CLRS http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap31.htm)
- Coppersmith-Winograd'90: $O(n^{2.38})$

  (Best known exponent is Le Gall'14: 2.3728639)
- Gives a running time of

$$O(n^{2.38}) + n^2 + n = O(n^{2.38})$$

for finding a triangle in an n-vertex graph.


STUDENT MATHEMATICAL LIBRARY
Volume 53
Thirty-three Miniatures
Mathematical and Algorithmic Applications of Linear Algebra
Jiří Matoušek



چهار نمونه از کاربردهای جبرخطی در دیگر شاخه‌های ریاضیات

عباس محرابیان
دانشگاه بریتیش کلمبیا
abbas.mehrabian@gmail.com

۲۷ مهر ۱۳۹۴

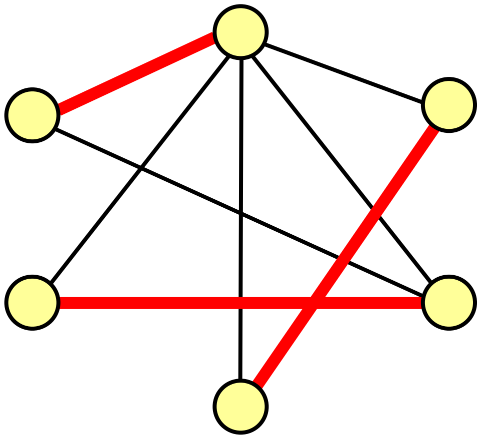
mathematicsjournal@gmail.com
www.sharifmathjournal.ir

مجله ریاضی شریف

http://www.math.uwaterloo.ca/~amehrabi/Articles/linearalgebramethods.pdf

Perfect Matching

- Perfect matching is **M**$\subseteq$E such that each vertex incident with *exactly* one edge in **M**
- Problem: find a perfect matching in a given graph!



Matching History

Dense Graphs
$m=n^2$

Edmonds	1965	$O(n^4)$
Even-Kariv	1975	$O(n^{2.5})$
Micali-Vazirani	1980-1990	$O(n^{2.5})$
Rabin-Vazirani	1989	$O(n^{3.38})$
Mucha-Sankowski	2004	$O(n^3)$
Mucha-Sankowski	2004	$O(n^{2.38})$
Harvey	2006	$O(n^{2.38})$

Algebraic algorithms are probabilistic: probability of correctness > 99%

Generic Matching Algorithm

If G has no perfect matching, halt

For each edge *e*

    If *e* is contained in a perfect matching

        Add *e* to solution

        Delete endpoints of e

Generic Matching Algorithm

If G has no perfect matching, halt
For each edge *e*
    If *e* is contained in a perfect matching
        Add *e* to solution
        Delete endpoints of e

Key step

- How can we test this?
- Randomization and linear algebra play key role

Outline

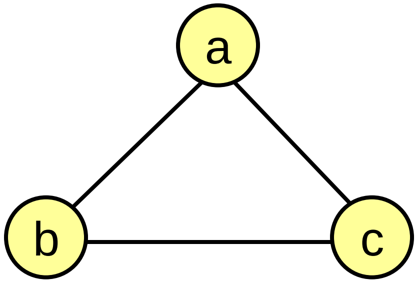
- Implementing Generic Algorithm
  - **$O(n^{4.38})$ algorithm (4.38 = 2 + 2.38)**

  - **$O(n^{3.38})$ algorithm**
    Rabin-Vazirani'89
  - **$O(n^{3})$ algorithm**
    Micha-Sankowski'04
  - **$O(n^{2.38})$ algorithm**
    Harvey'06

Matching & Tutte Matrix

- Let G=(V,E) be a graph
- Define variable x_{uv} for each edge uv
- Define a skew-symmetric matrix T s.t.

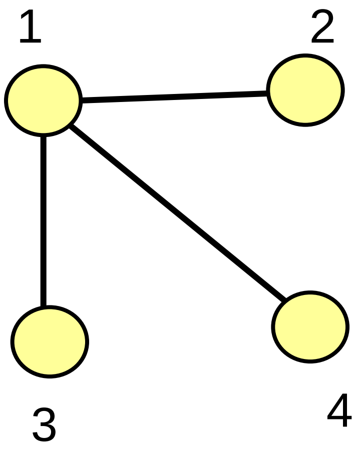
$$T_{u,v} = \begin{cases} \pm\, x_{\{u,v\}} & \text{if } \{u,v\} \in E \\ 0 & \text{otherwise} \end{cases}$$



$$\begin{vmatrix} 0 & -x_{\{a,b\}} & -x_{\{a,c\}} \\ x_{\{a,b\}} & 0 & -x_{\{b,c\}} \\ x_{\{a,c\}} & x_{\{b,c\}} & 0 \end{vmatrix}$$

Properties of Tutte Matrix

Lemma [Tutte'47]: G has a perfect matching if and only if det(T)≠0.



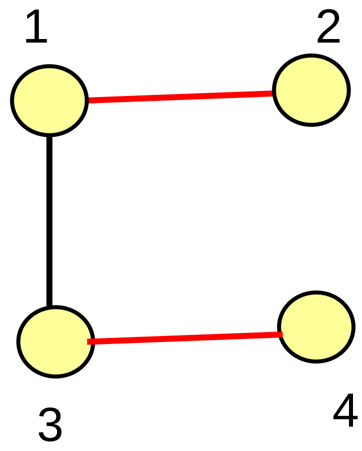
This graph has no perfect matching

$$det\begin{bmatrix} 0 & -x_{1,2} & -x_{1,3} & -x_{1,4} \\ x_{1,2} & 0 & 0 & 0 \\ x_{1,3} & 0 & 0 & 0 \\ x_{1,4} & 0 & 0 & 0 \end{bmatrix} \equiv 0$$

Properties of Tutte Matrix

Lemma [Tutte'47]: G has a perfect matching if and only if det(T)≠0.

This graph has a perfect matching



$$det\begin{bmatrix} 0 & -x_{1,2} & -x_{1,3} & 0 \\ x_{1,2} & 0 & 0 & 0 \\ x_{1,3} & 0 & 0 & -x_{3,4} \\ 0 & 0 & x_{3,4} & 0 \end{bmatrix} = x_{1,2}^2 x_{3,4}^2 \neq 0$$

Properties of Tutte Matrix

Lemma [Tutte'47]: G has a perfect matching if and only if $\det(T) \neq 0$.

Computing det(T) very slow: Contains variables, and can have exponential number of terms.

Lemma [Lovász'79]: This result holds with probability 99% if we randomly choose values for $x_{\{u,v\}}$'s.

Computing determinant of an nxn matrix of numbers can be done in time $O(n^{2.38})$

$O(n^{2.38})$ algorithm for *deciding* a perfect matching

$O(n^{4.38})$ algorithm for *building* PM

Choose random values for variables and compute det (T) (Takes $O(n^{2.38})$ time)

If det(T) = 0 halt

For each edge uv

Let U be Tutte matrix of G – {u, v}

If det(U) $\neq$ 0 (Edge uv is contained in a PM)

Add uv to matching

Delete vertices u and v

Outline

- Implementing Generic Algorithm
  - **$O(n^{4.38})$ algorithm**
  - **$O(n^{3.38})$ algorithm Rabin-Vazirani'89**
  - **$O(n^{3})$ algorithm Micha-Sankowski'04**
  - **$O(n^{2.38})$ algorithm Harvey'06**

Properties of Tutte Matrix

Lemma [Tutte'47]: G has a perfect matching if and only if $\det(T) \neq 0$.

Lemma [Rabin-Vazirani'89]: G-{u,v} has a perfect matching if and only if $(T^{-1})_{u,v} \neq 0$.

Properties of Tutte Matrix

Lemma [Tutte'47]: G has a perfect matching if and only if $\det(T) \neq 0$.

Lemma [Rabin-Vazirani'89]: G-{u,v} has a perfect matching if and only if $(T^{-1})_{u,v} \neq 0$.

Proof:

$$T = \left[\begin{array}{cc|c} 0 & -x_{1,2} & \multirow{2}{*}{U} \\ x_{1,2} & 0 & \\ \hline \multicolumn{2}{c|}{V} & W \end{array}\right]$$

G-{1,2} has a perfect matching $\Leftrightarrow$

$\det(W) \neq 0$

Properties of Tutte Matrix

Lemma [Tutte'47]: G has a perfect matching if and only if det(T) $\neq$ 0.

Lemma [Rabin-Vazirani'89]: G-{u,v} has a perfect matching if and only if $(T^{-1})_{u,v} \neq 0$.

Proof:

$$T = \left[\begin{array}{cc|c} 0 & -x_{1,2} & U \\ x_{1,2} & 0 & \\ \hline \multicolumn{2}{c|}{V} & W \end{array}\right] \qquad T^{-1} = \left[\begin{array}{cc|c} 0 & -a & \tilde{U} \\ a & 0 & \\ \hline \multicolumn{2}{c|}{\tilde{V}} & \tilde{W} \end{array}\right]$$

G-{1,2} has a perfect matching $\Leftrightarrow$

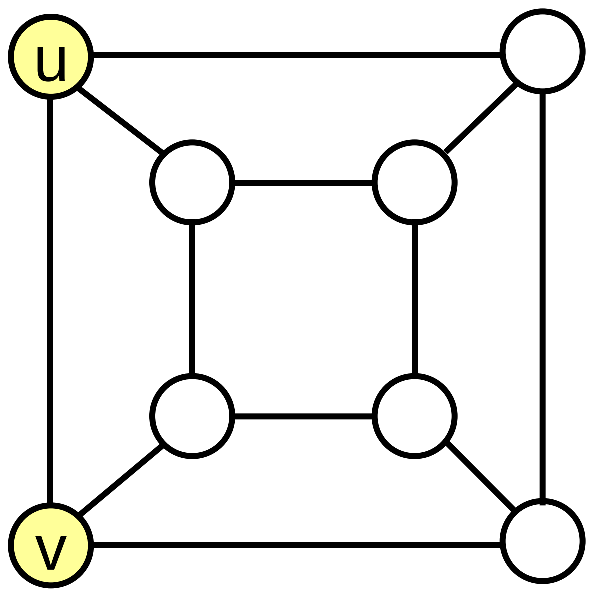
det(W) $\neq$ 0 $\Leftrightarrow$ $det(T) \times det \begin{bmatrix} 0 & -a \\ a & 0 \end{bmatrix} \neq 0 \Leftrightarrow a \neq 0$

Properties of Tutte Matrix

Lemma [Tutte'47]: G has a perfect matching if and only if $\det(T) \neq 0$.

Lemma [Rabin-Vazirani'89]: G-{u,v} has a perfect matching if and only if $(T^{-1})_{u,v} \neq 0$.

$(T^{-1})_{u,v} \neq 0$

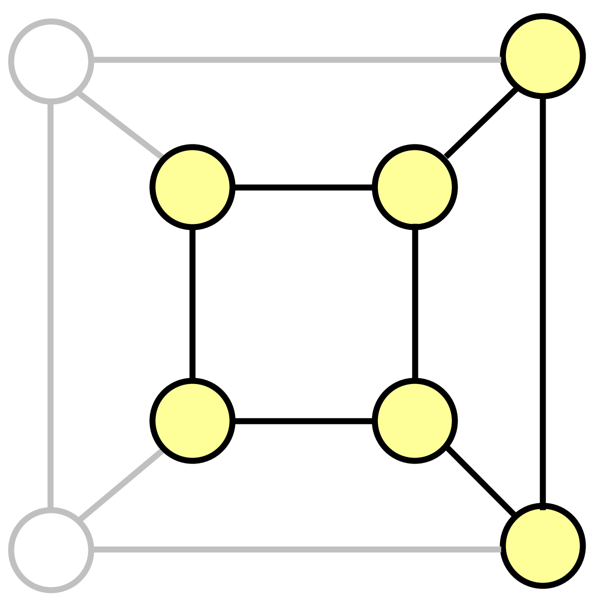


Properties of Tutte Matrix

Lemma [Tutte'47]: G has a perfect matching if and only if $\det(T) \neq 0$.

Lemma [Rabin-Vazirani'89]: G-{u,v} has a perfect matching if and only if $(T^{-1})_{u,v} \neq 0$.

$(T^{-1})_{u,v} \neq 0$



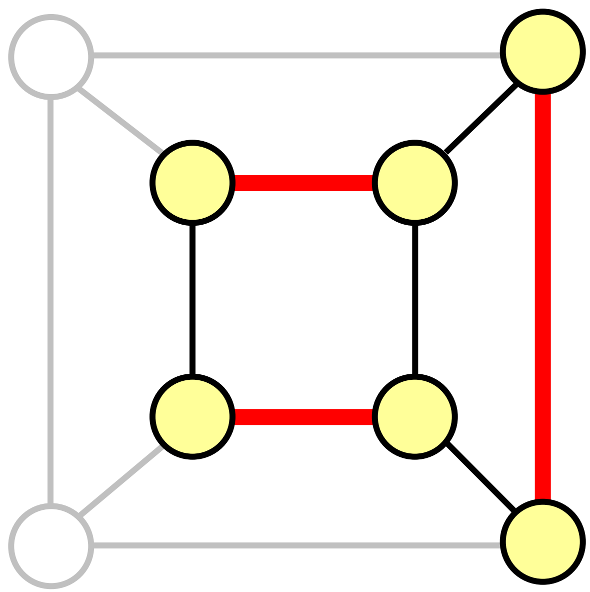
G-{u,v} has perfect matching

Properties of Tutte Matrix

Lemma [Tutte'47]: G has a perfect matching if and only if $\det(T) \neq 0$.

Lemma [Rabin-Vazirani'89]: G-{u,v} has a perfect matching if and only if $(T^{-1})_{u,v} \neq 0$.

$(T^{-1})_{u,v} \neq 0$



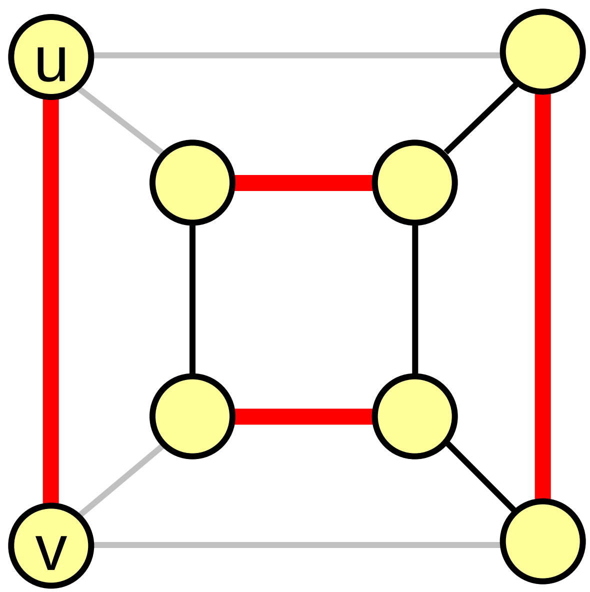
G-{u,v} has perfect matching

Properties of Tutte Matrix

Lemma [Tutte'47]: G has a perfect matching if and only if $\det(T) \neq 0$.

Lemma [Rabin-Vazirani'89]: G-{u,v} has a perfect matching if and only if $(T^{-1})_{u,v} \neq 0$.

$(T^{-1})_{u,v} \neq 0$



G-{u,v} has perfect matching

$O(n^{3.38})$ algorithm

Rabin-Vazirani '89

Choose random values for variables and compute det (T)

If det(T) = 0 halt

Repeat n/2 times

  Compute T^{-1} (Takes $O(n^{2.38})$ time)

  Find an edge uv with $T^{-1}_{u,v} \neq 0$

  Add uv to matching (Takes $O(n^2)$ time)

  Delete u and v from G

$O(n^{3.38})$ algorithm

Rabin-Vazirani '89

Choose random values for variables and compute det (T)
If det(T) = 0 halt
Repeat n/2 times
    Compute T^{-1} (Takes $O(n^{2.38})$ time)
    Find an edge uv with $T^{-1}_{u,v} \neq 0$
    Add uv to matching (Takes $O(n^2)$ time)
    Delete u and v from G

- Total running time: $O(n^{3.38})$
- Improve: Recompute T^{-1} quickly in each iteration?

Outline

- Implementing Generic Algorithm
  - **$O(n^{4.38})$ algorithm**
  - **$O(n^{3.38})$ algorithm**
    Rabin-Vazirani'89
  - **$O(n^{3})$ algorithm**
    Micha-Sankowski'04
  - **$O(n^{2.38})$ algorithm**
    Harvey'06

Quick updating lemma

- **Lemma.** After deleting two vertices, inverse of the new Tutte matrix can be found in time $O(n^2)$

Quick updating lemma

- **Lemma.** After deleting two vertices, inverse of the new Tutte matrix can be found in time $O(n^2)$
- **Proof.**

$$T = \left[\begin{array}{cc|c} 0 & -x_{1,2} & \\ x_{1,2} & 0 & U \\ \hline \multicolumn{2}{c|}{V} & W \end{array}\right] \qquad T^{-1} = \left[\begin{array}{cc|c} 0 & -a & \\ a & 0 & \tilde{U} \\ \hline \multicolumn{2}{c|}{\tilde{V}} & \tilde{W} \end{array}\right]$$

Quick updating lemma

- **Lemma.** After deleting two vertices, inverse of the new Tutte matrix can be found in time O(n²)
- **Proof.**

$$T = \left[\begin{array}{cc|c} 0 & -x_{1,2} & \multirow{2}{*}{U} \\ x_{1,2} & 0 & \\ \hline \multicolumn{2}{c|}{V} & W \end{array}\right] \qquad T^{-1} = \left[\begin{array}{cc|c} 0 & -a & \multirow{2}{*}{$\tilde{U}$} \\ a & 0 & \\ \hline \multicolumn{2}{c|}{\tilde{V}} & \tilde{W} \end{array}\right]$$

Sherman-Morrison-Woodbury Formula:

$$W^{-1} = \tilde{W} - \tilde{V} \begin{bmatrix} 0 & -a \\ a & 0 \end{bmatrix}^{-1} \tilde{U} = \tilde{W} - \frac{1}{a}\left(\tilde{V}_2 \tilde{U}_1 - \tilde{V}_1 \tilde{U}_2\right)$$

$O(n^3)$ algorithm

Micha-Sankowski'04

Choose random values for variables and compute det (T)

If det(T) = 0 halt

Compute T^{-1} (Takes $O(n^{2.38})$ time)

Repeat n/2 times

Find an edge uv with $T^{-1}_{u,v} \neq 0$

Add uv to matching (Takes $O(n^2)$ time)

Delete u and v from G

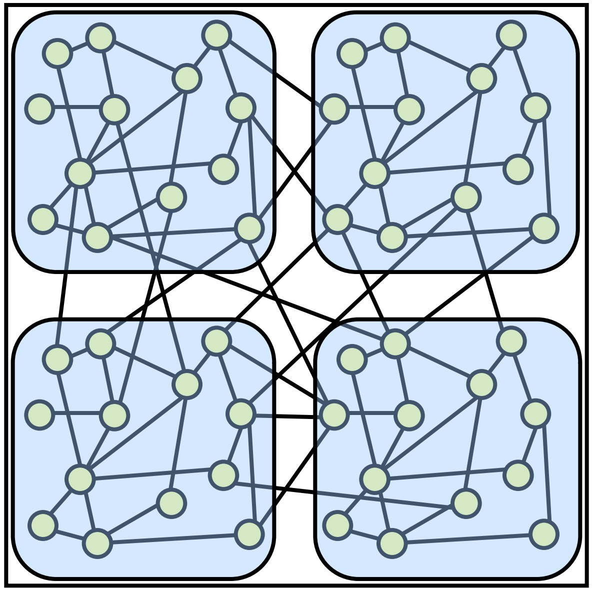
Update T^{-1} (Takes $O(n^2)$ time)

- Total runtime: $O(n^3)$

Outline

- Implementing Generic Algorithm
  - **$O(n^{4.38})$ algorithm**
  - **$O(n^{3.38})$ algorithm**
    Rabin-Vazirani'89
  - **$O(n^{3})$ algorithm**
    Micha-Sankowski'04
  - **$O(n^{2.38})$ algorithm**
    Harvey'06

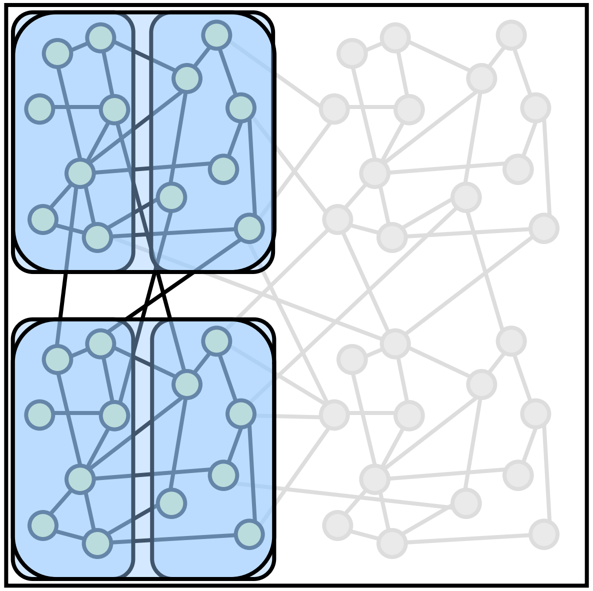
New Recursive Approach



(Here c=4 parts)

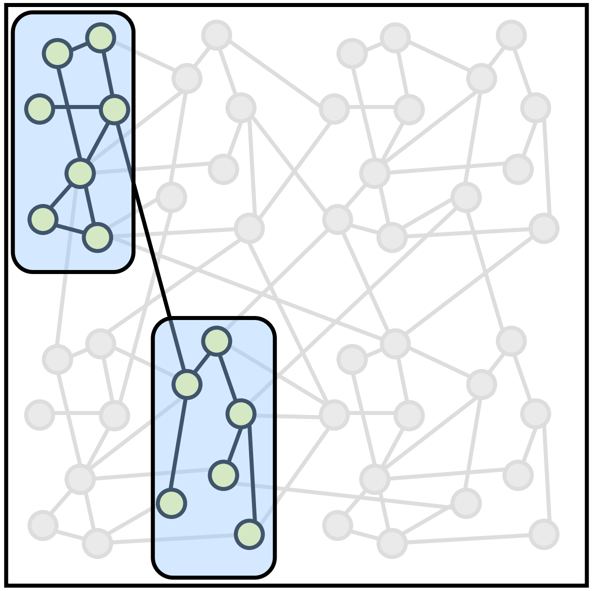
- Partition into *c* parts $\{V_1,\ldots,V_c\}$ (arbitrarily)
- **For each pair** of parts $\{V_a,V_b\}$ (arbitrary order)
  - Recurse on $G[V_a \cup V_b]$

New Recursive Approach



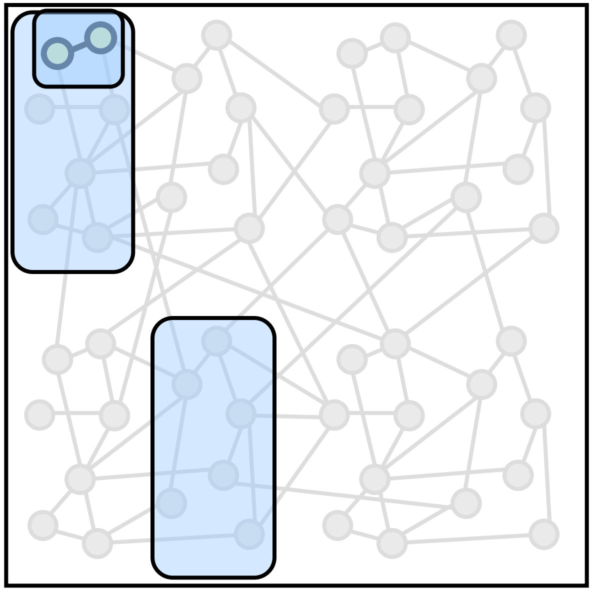
- Partition into *c* parts $\{V_1,\ldots,V_c\}$ (arbitrarily)
- For each pair of parts $\{V_a,V_b\}$ (arbitrary order)
  - Recurse on $G[V_a \cup V_b]$

New Recursive Approach



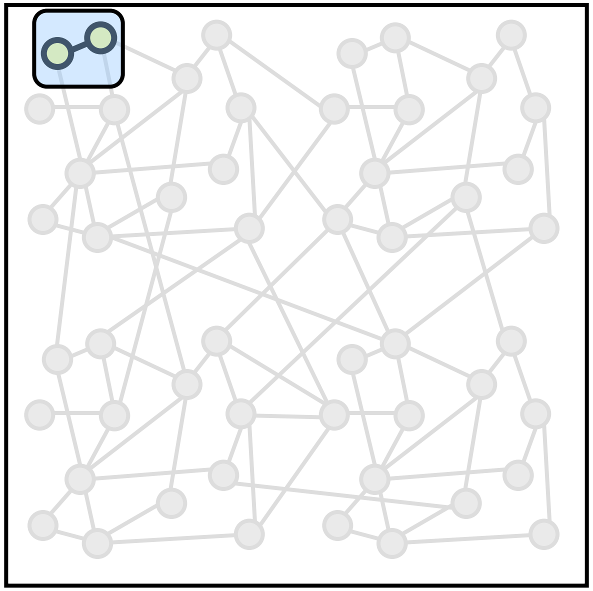
- Partition into c parts $\{V_1,\ldots,V_c\}$ (arbitrarily)
- For each pair of parts $\{V_a,V_b\}$ (arbitrary order)
  - Recurse on $G[V_a \cup V_b]$

New Recursive Approach



- Partition into c parts $\{V_1,\ldots,V_c\}$ (arbitrarily)
- For each pair of parts $\{V_a,V_b\}$ (arbitrary order)
  - Recurse on $G[V_a \cup V_b]$
- Base case: 2 vertices

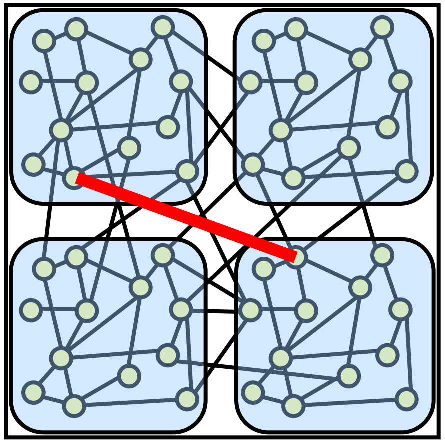
New Recursive Approach



- Partition into *c* parts $\{V_1,\ldots,V_c\}$ (arbitrarily)
- For each pair of parts $\{V_a,V_b\}$ (arbitrary order)
  - Recurse on $G[V_a \cup V_b]$
- Base case: 2 vertices $\{u,v\}$
  - If $T^{-1}_{u,v} \neq 0$, add $\{u,v\}$ to matching, update T^{-1}

Recursion F.A.Q.

- Why not just recurse on G[V_a]?
  - Edges between parts would be missed
  - **Claim**: Our recursion examines every pair of vertices $\Rightarrow$ examines every edge
- Why does algorithm work?
  - It implements Rabin-Vazirani Algorithm!
- Isn't this horribly slow?
  - No: we'll see recurrence next



- Partition into *c* parts $\{V_1,\ldots,V_c\}$ (arbitrarily)
- For each pair of parts $\{V_a,V_b\}$ (arbitrary order)
  - Recurse on $G[V_a \cup V_b]$
- Base case: 2 vertices {u,v}
  - If $T^{-1}_{u,v} \neq 0$, add {u,v} to matching, update T^{-1}

Final Matching Algorithm

If base case with 2 vertices {u,v}

    If $T^{-1}_{u,v} \neq 0$, add {u,v} to matching

Else

    Partition into *c* parts $\{V_1,\ldots,V_c\}$

    For each pair $\{V_a,V_b\}$

        Recurse on $G[V_a \cup V_b]$

        Apply updates to **current** subproblem

s = size of subproblem

$$R(s) = \binom{c}{2} \cdot R\left(\frac{2}{c}s\right) + O\left(\binom{c}{2} \cdot s^{\omega}\right)$$

Final Matching Algorithm

If base case with 2 vertices {u,v}

  If $T^{-1}_{u,v} \neq 0$, add {u,v} to matching

Else

  Partition into *c* parts $\{V_1,\ldots,V_c\}$

  For each pair $\{V_a,V_b\}$

    Recurse on $G[V_a \cup V_b]$

    Apply updates to **current** subproblem

s = size of subproblem

$$R(s) = \binom{c}{2} \cdot R\left(\frac{2}{c}s\right) + O\left(\binom{c}{2} \cdot s^{\omega}\right)$$

Final Matching Algorithm

If base case with 2 vertices {u,v}

  If $T^{-1}_{u,v} \neq 0$, add {u,v} to matching

Else

  Partition into c parts $\{V_1,\ldots,V_c\}$

  For each pair $\{V_a,V_b\}$

    Recurse on $G[V_a \cup V_b]$

    Apply updates to **current** subproblem

Assume: $O(s^\omega)$ time

s = size of subproblem

$$R(s) = \binom{c}{2} \cdot R\left(\frac{2}{c}s\right) + O\left(\binom{c}{2} \cdot s^{\omega}\right)$$

Time Analysis

$$R(s) = \binom{c}{2} \cdot R\left(\frac{2}{c} s\right) + O\left(\binom{c}{2} \cdot s^{\omega}\right)$$

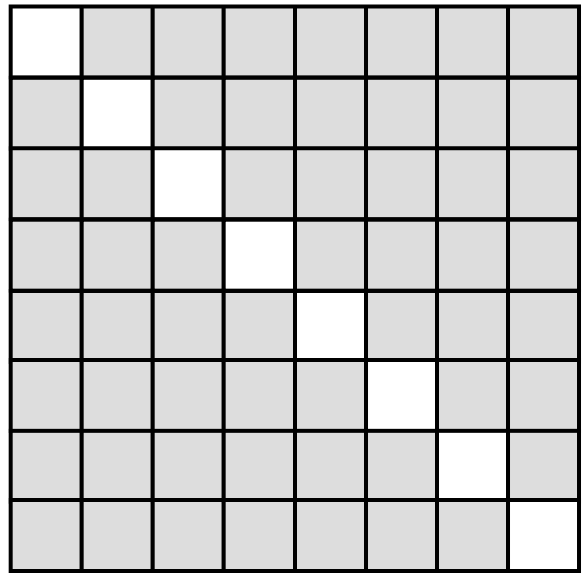
- Basic Divide-and-Conquer
  - If $\log_{c/2} \binom{c}{2} < \omega$ then $R(n) = O(n^{\omega})$

- Since $\log_{c/2} \binom{c}{2} < 2 + \dfrac{1}{\log c - 1}$,

  just choose *c* large enough!

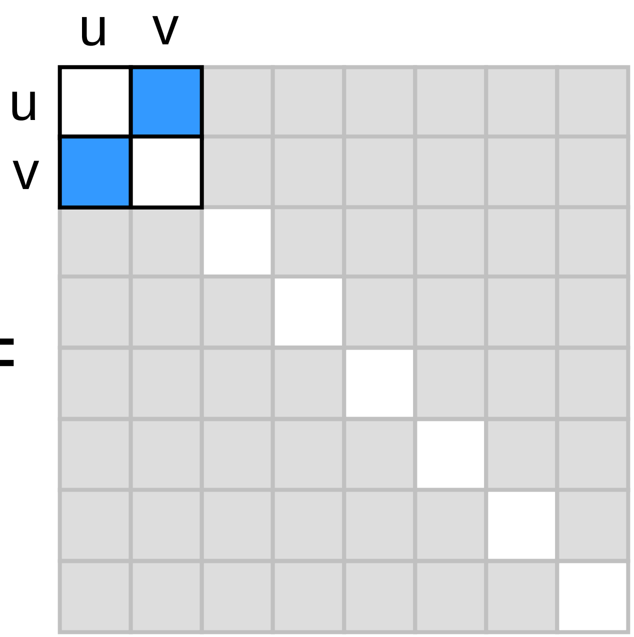
c = 13 is large enough if ω = 2.38

Handling Updates

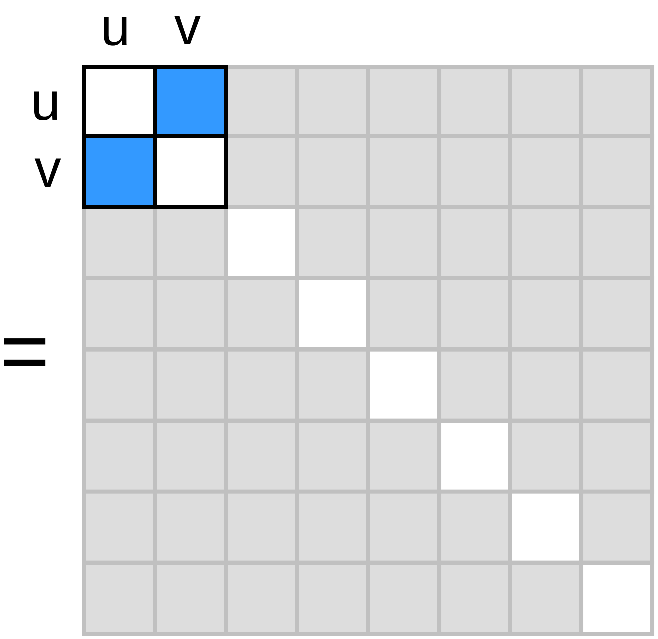
T = 

Handling Updates

$T =$

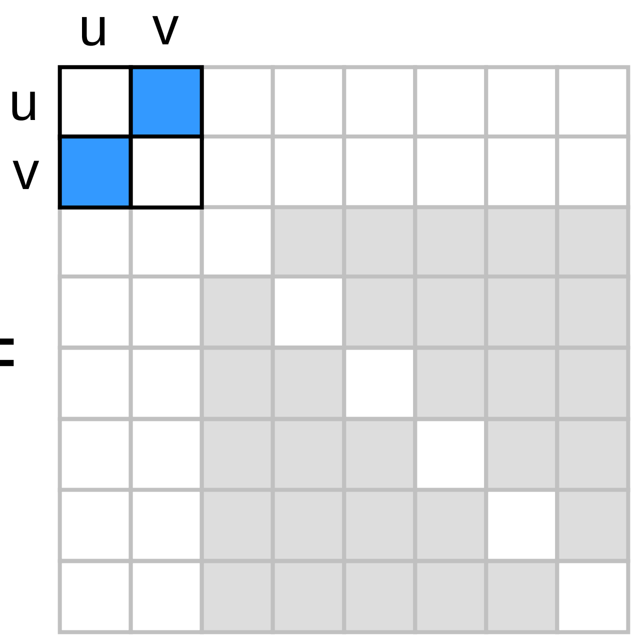


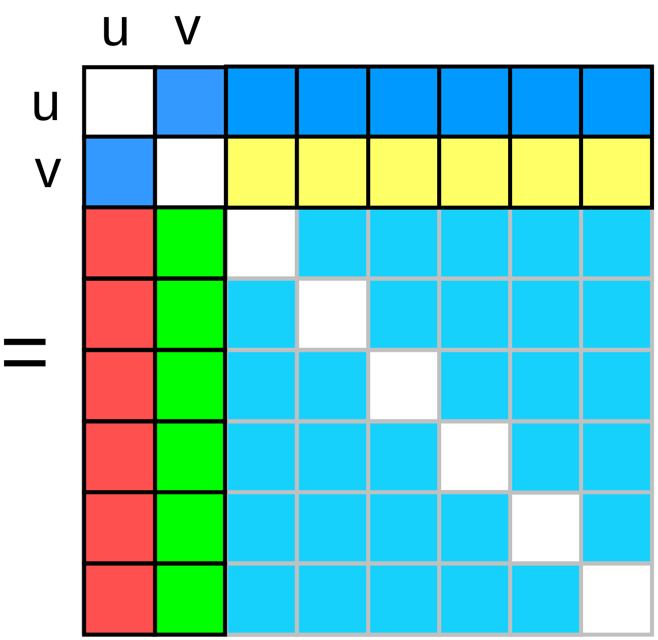
$T^{-1} =$



- Delete vertices u and v

Handling Updates (Naively)

$T =$ 

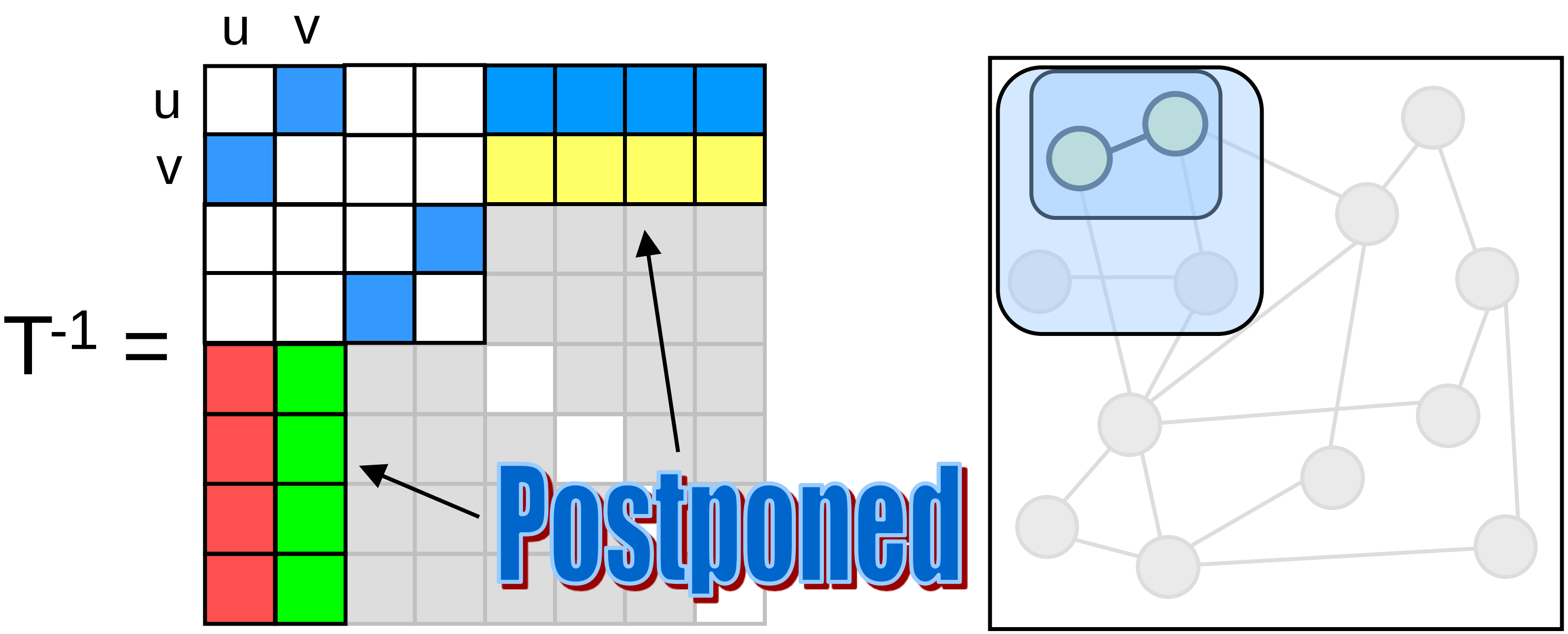
$T^{-1} =$ 

- Delete vertices u and v $\Rightarrow$ clear rows / columns
- Causes rank-1 updates to T^{-1}
- Algorithm still takes $\Omega(n^3)$ time

Matching Outline

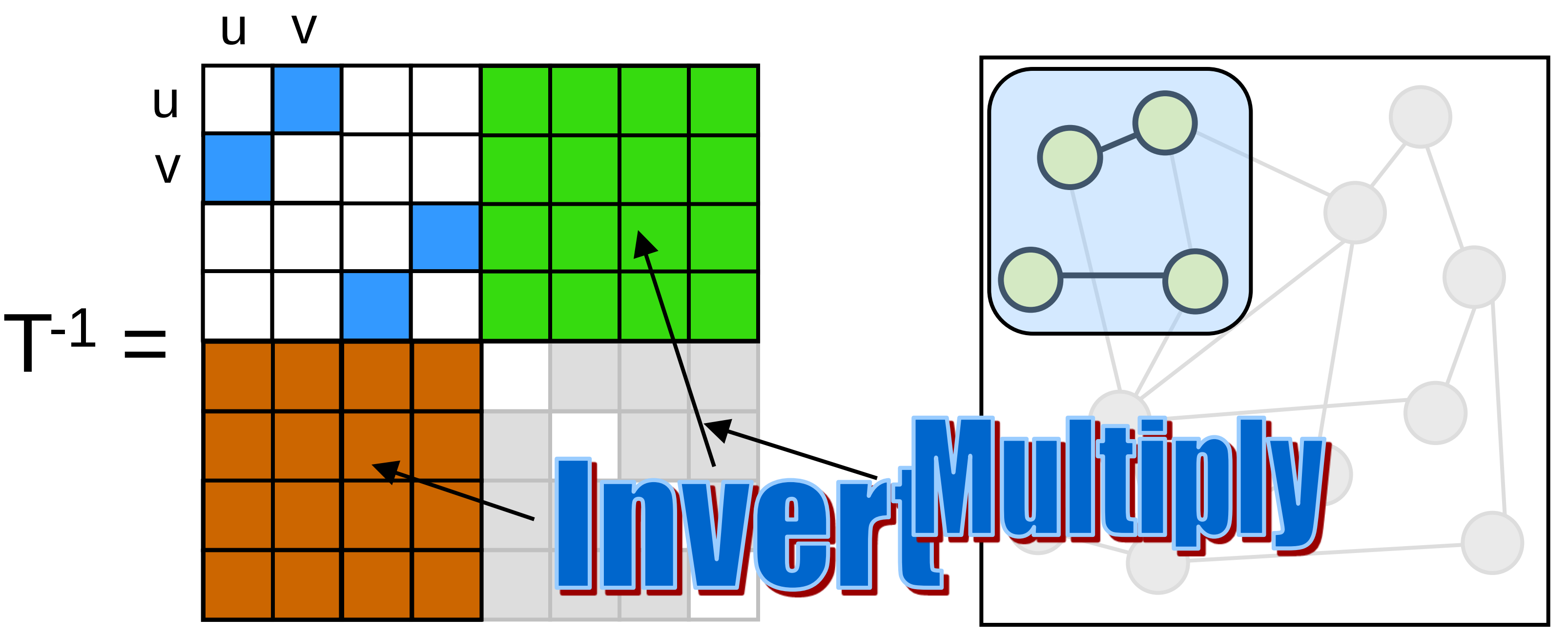
– Tutte Matrix & Properties
– Rabin-Vazirani Algorithm
– Rank-1 Updates
– Rabin-Vazirani with Rank-1 Updates
– Our Recursive Algorithm (overview)
– **Our Recursive Algorithm (fast updates)**

Just-in-time Updates



- Don’t update entire matrix!
- Just update parent in recursion tree
- Updates outside of parent are postponed

Postponed Updates



- Accumulate batches of updates
- **Claim**: New updates can be applied with matrix multiplication and inversion

Final Matching Algorithm

If base case with 2 vertices {u,v}

    If $T^{-1}_{u,v} \neq 0$, add {u,v} to matching

Else

    Partition into *c* parts $\{V_1,\ldots,V_c\}$

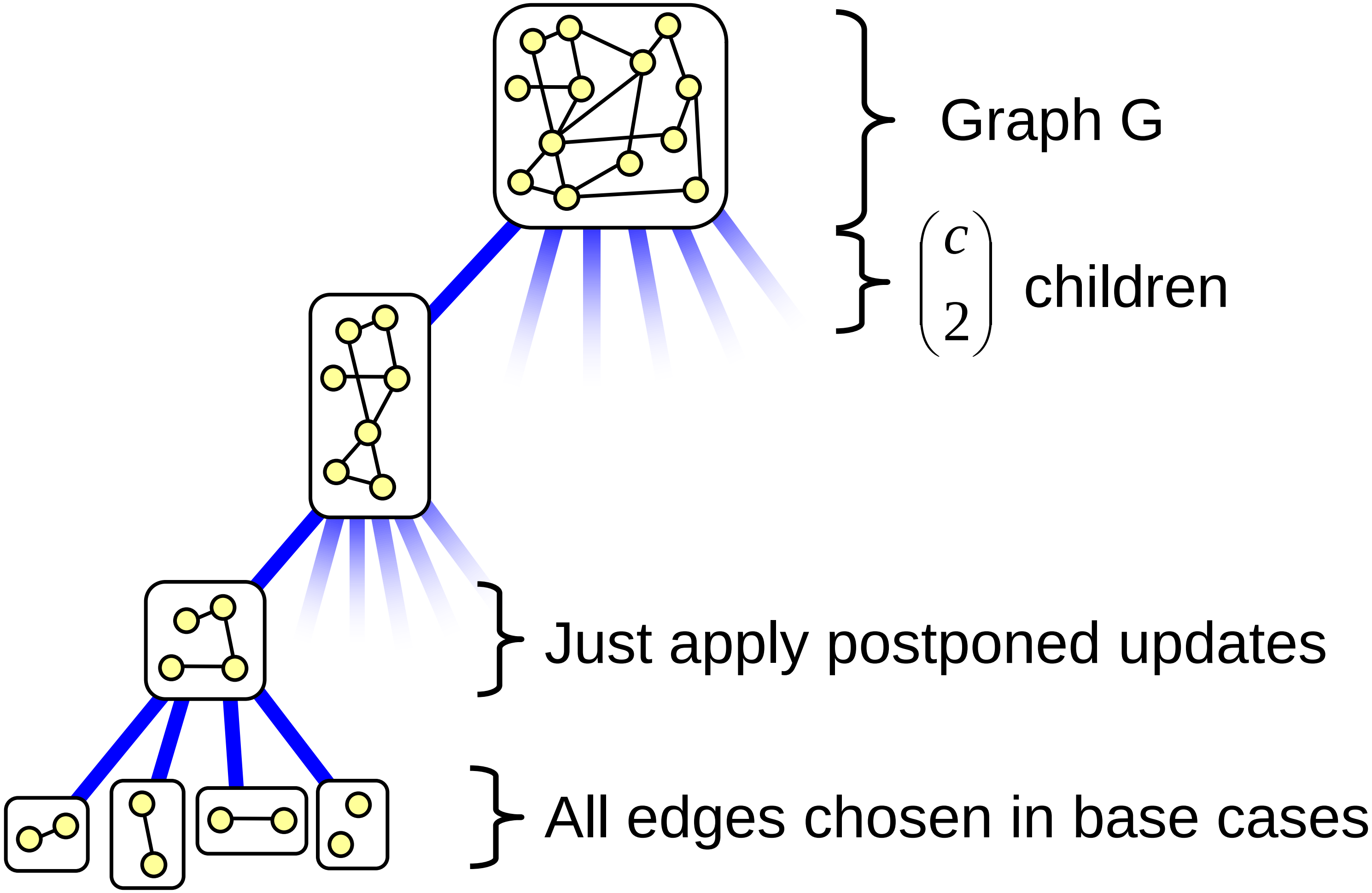
    For each pair $\{V_a,V_b\}$

        Recurse on $G[V_a \cup V_b]$

        Apply updates to **current** subproblem

Invariant: Before / after child subproblem, parent's submatrix is completely updated

$\Rightarrow$ in every base case, $T^{-1}_{u,v}$ is up-to-date!



Only take edges that **can be extended to a perfect matching** in the whole graph.
This decision is possible because invariant ensures that $T^{-1}_{u,v}$ is up-to-date.

Matching Summary

- **Theorem.** We can compute a perfect matching in $O(n^{2.38})$ time, with correctness probability > 99%
- Algorithm uses only simple randomization, linear algebra and divide-and-conquer
- Easy to implement
  (200 lines of MATLAB code)
- Extensions for: (by existing techniques)
  - Maximum matching
  - Las Vegas

Conclusion

- Aim: highlight the power of linear algebra in algorithm design … read!
- Another example tomorrow:
  maximum flow problem
- Easy-to-read reference for this talk:
  Algebraic algorithms for matching, Ivan, Virza, and Yuen
  https://madars.org/projects/6854/AlgMatching.pdf

Conclusion

- Aim: highlight the power of linear algebra in algorithm design … read!
- Another example tomorrow: maximum flow problem
- Easy-to-read reference for this talk:
  Algebraic algorithms for matching, Ivan, Virza, and Yuen
  https://madars.org/projects/6854/AlgMatching.pdf
- Nick Harvey's paper is also readable:
  http://www.cs.ubc.ca/~nickhar/Publications/AlgebraicMatching/AlgebraicMatching.pdf

Thanks for listening