Algebraic Graph Algorithms
Part I - Introduction

Piotr Sankowski

University of Warsaw

2nd PCC October 17-23, 2008

Outline

- Part I - Introduction
- Part II - Dynamic Transitive Closure
- Part III - Finding Matchings
- Part IV - Weighted Problems

Outline - Part I

- Algebraic algorithms - idea
- Simple example - perfect matchings
- Dynamic algebraic algorithms
  - determinant and inverse
  - matrix rank and characteristic polynomial
- Dynamic graph algorithms
  - transitive closure
  - distances in graphs
  - vertex connectivity and matchings
- Static graph algorithms
  - matchings in graphs
  - distances in graphs

Matrix Multiplication

$$C = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \times \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,n} \end{bmatrix}$$

Naive algorithm

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j} = a_{i,1} b_{1,j} + a_{i,2} b_{2,j} + \ldots + a_{i,n} b_{n,j}.$$

requires n operations to compute each element of C.
This gives $\sim n^3$ operations in total.

Strassen's Algorithm

$$\begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \times \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix}$$

$$Q_1 = (a_{1,1} + a_{2,2})(b_{1,1} + b_{2,2})$$

$$Q_2 = (a_{2,1} + a_{2,2})b_{1,1}$$

$$Q_3 = a_{1,1}(b_{1,2} - b_{2,2})$$

$$Q_4 = a_{2,2}(-b_{1,1} + b_{2,1})$$

$$Q_5 = (a_{1,1} + a_{1,2})b_{2,2}$$

$$Q_6 = (-a_{1,1} + a_{2,1})(b_{1,1} + b_{1,2})$$

$$Q_7 = (a_{1,2} - a_{2,2})(b_{2,1} + b_{2,2})$$

$$c_{1,1} = Q_1 + Q_4 - Q_5 + Q_7$$

$$c_{2,1} = Q_2 + Q_4$$

$$c_{1,2} = Q_3 + Q_5$$

$$c_{2,2} = Q_1 + Q_3 - Q_2 + Q_6$$

The matrix C can be computed with use of 7 multiplications instead 8!

Strassen Algorithm

After dividing the matrix in blocks we get

$$\begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix} \times \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

we have to do 2×2 matrix multiplication on blocks.

Using this recursive multiplication, we need

$$\sim n^{\log_2 7} = n^{2.81},$$

operations to multiply $n \times n$ matrices.

Fast Matrix Multiplication

The matrix multiplication exponent is denoted by ω.

The $n \times n$ by $n \times n$ multiplication requires

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix} \times \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,n} \end{bmatrix}$$

$\sim n^{\omega}$ operations.

The best known bound is $\omega < 2.376$ — *Coppersmith, Winograd '90.*

Fast Matrix Multiplication

In $O(n^\omega)$ time for a $n \times n$ matrix we can:

- compute the determinant,
- compute the characteristic polynomial,
- compute the inverse matrix,
- solve the system of linear equations,
- solve the system of linear equations over polynomials.

We will use the above algorithms to solve graph problems.

Algebraic Algorithms

The determinant of the $n \times n$ matrix A is given as:

$$\det(A) = \sum_{p \in \Pi_n} \sigma(p) \prod_{i=0}^{n} a_{i,p_i}.$$

Is it possible to encode the graph problem in the matrix A in such a way that the element of the sum correspond to the solution of the problem?

(Permutations are powerful.)

By testing if the determinant is non-zero we will know if the problem has a solution.

Dynamic Problems

We want to solve given problem for a data structure that can be changed, e.g., we add and remove edges from the graph.

Can the algebraic methods be used in such a case?

YES

- if we can show dynamic algorithms for algebraic problems,
- if we can show appropriate reductions.

Outline

- Introduction
- **Simple example - perfect matchings**
- Dynamic algebraic algorithms
  - determinant and inverse
  - matrix rank and characteristic polynomial
- Dynamic graph algorithms
  - transitive closure
  - distances in graphs
  - vertex connectivity and matchings
- Static graph algorithms
  - matchings in graphs
  - distances in graphs

Example: Matchings

A *matching* in the graph $G = (V, E)$ is a subset of edges $M \subseteq E$ such, that no two edges in M share a common endpoint.

A *perfect* matching is a matching of size $|V|/2$.

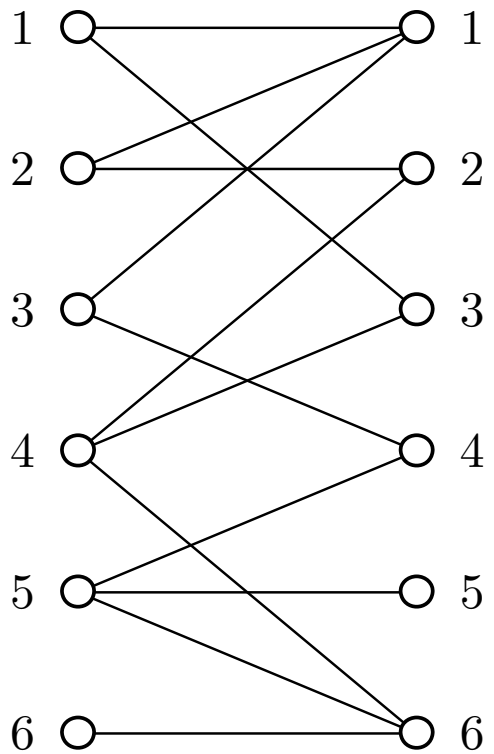
We want to:

- test if a graph contains a perfect matching,
- find any perfect matching in a graph,
- *find the maximum matching in the graph*.

Symbolic Adjacency Matrix

A symbolic adjacency matrix of the bipartite graph:

G



$\Longrightarrow$

$$\tilde{A}(G) = \begin{pmatrix} x_{11} & 0 & x_{13} & 0 & 0 & 0 \\ x_{21} & x_{22} & 0 & 0 & 0 & 0 \\ x_{31} & 0 & 0 & x_{34} & 0 & 0 \\ 0 & x_{42} & x_{43} & 0 & 0 & x_{46} \\ 0 & 0 & 0 & x_{54} & x_{55} & x_{56} \\ 0 & 0 & 0 & 0 & 0 & x_{66} \end{pmatrix}$$

Symbolic Adjacency Matrix

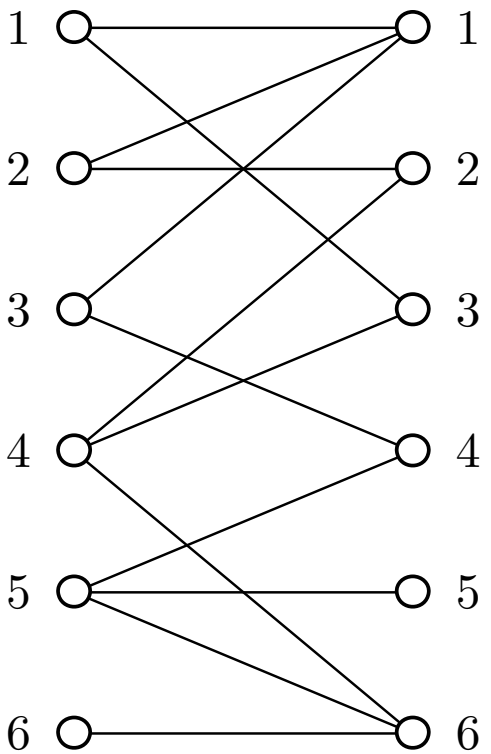
$$\det \begin{pmatrix} x_{11} & 0 & x_{13} & 0 & 0 & 0 \\ x_{21} & x_{22} & 0 & 0 & 0 & 0 \\ x_{31} & 0 & 0 & x_{34} & 0 & 0 \\ 0 & x_{42} & x_{43} & 0 & 0 & x_{46} \\ 0 & 0 & 0 & x_{54} & x_{55} & x_{56} \\ 0 & 0 & 0 & 0 & 0 & x_{66} \end{pmatrix} =$$

$$= -x_{13}x_{21}x_{34}x_{42}x_{55}x_{66} - x_{11}x_{22}x_{34}x_{43}x_{55}x_{66}.$$

Symbolic Adjacency Matrix

A symbolic adjacency matrix of the bipartite graph:

G



$$\Longrightarrow \quad \det(\tilde{A}(G)) =$$

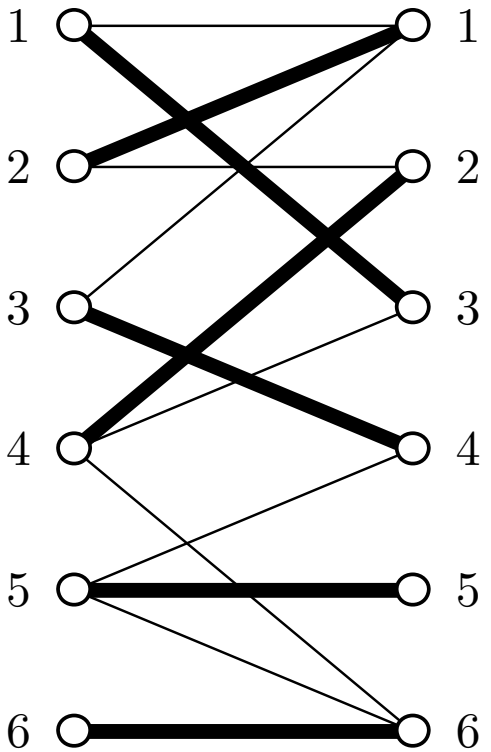
$$-x_{13}x_{21}x_{34}x_{42}x_{55}x_{66}$$

$$-x_{11}x_{22}x_{34}x_{43}x_{55}x_{66}.$$

Symbolic Adjacency Matrix

A symbolic adjacency matrix of the bipartite graph:

G

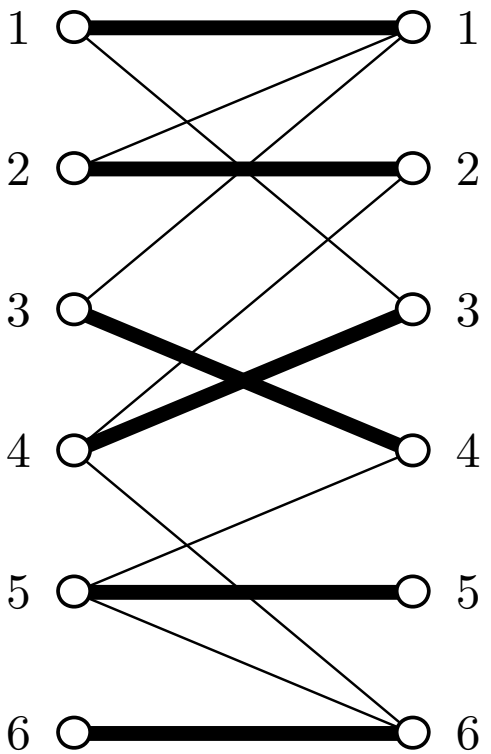


$$\Longrightarrow \quad \det(\tilde{A}(G)) = -x_{13}x_{21}x_{34}x_{42}x_{55}x_{66} - x_{11}x_{22}x_{34}x_{43}x_{55}x_{66}.$$

Symbolic Adjacency Matrix

A symbolic adjacency matrix of the bipartite graph:

G



$$\Longrightarrow \quad \det(\tilde{A}(G)) =$$

$$-x_{13}x_{21}x_{34}x_{42}x_{55}x_{66}$$

$$-x_{11}x_{22}x_{34}x_{43}x_{55}x_{66}.$$

Symbolic Adjacency Matrix

The determinant is given as:

$$\det(A) = \sum_{p \in \Pi_n} \sigma(p) \prod_{i=1}^{n} a_{i,p_i}.$$

p assigns different vertex p_i to each vertex i.

The elements of the sum correspond to perfect matchings in the graph.

The determinant is non-zero iff the graph has a perfect matching.

Lovászs Idea

The polynomial $\det(\tilde{A}(G))$ can have exponentially many terms. Can we efficiently test whether it is non-zero?

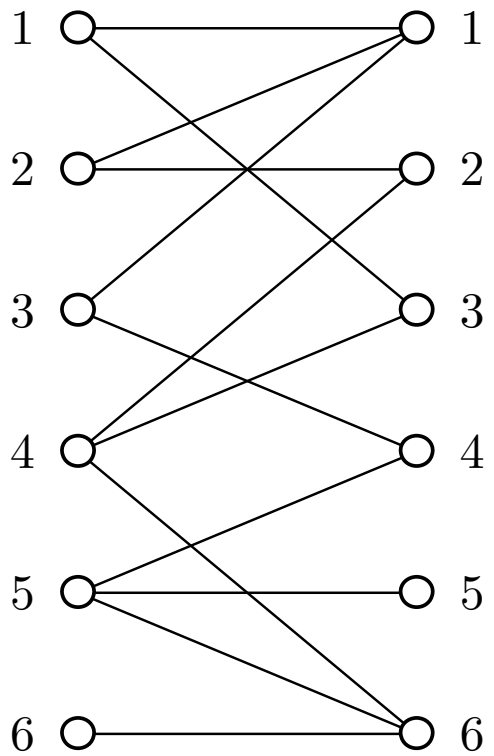
Substitute random numbers into variables in $\tilde{A}(G)$ and compute the determinant of the resulting matrix $A(G)$ — *random adjacency matrix*.

With high probability $\det A(G) \neq 0$ iff $\det \tilde{A}(G) \neq 0$, because „polynomials do not have many zeros".

Random Adjacency Matrix

There is a perfect matchings.

G



$\Longrightarrow$

$$A(G)$$

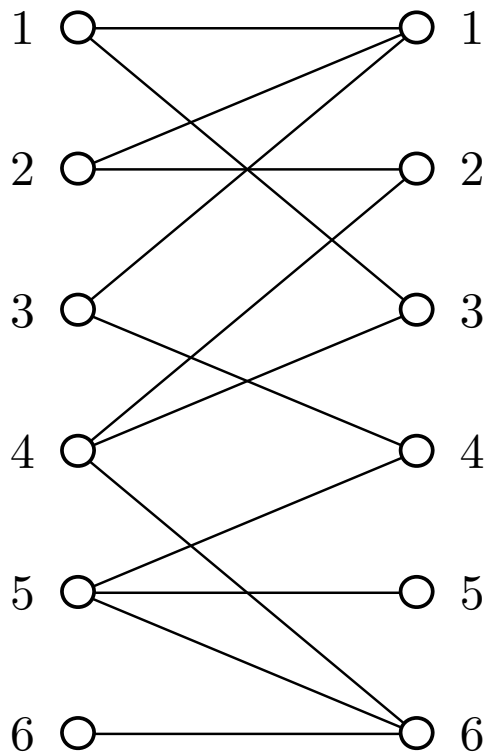
$$\begin{pmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\det(A(G)) = 2$$

Random Adjacency Matrix

There is a perfect matchings.

G



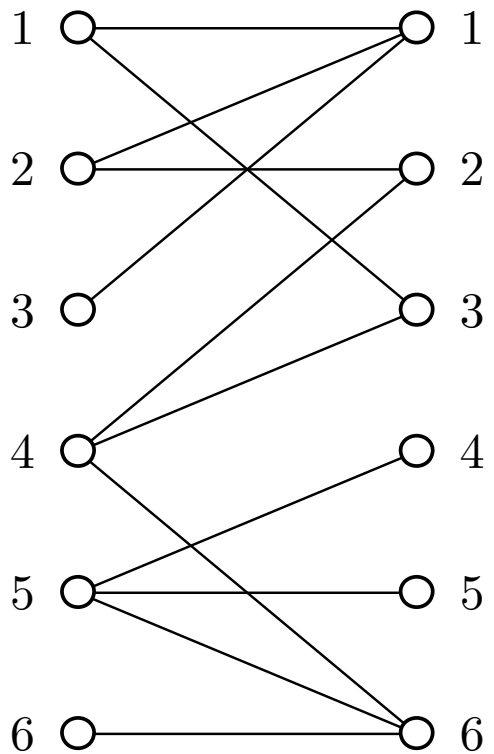
$$\Longrightarrow \quad A(G) = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\det(A(G)) = 0$$

Random Adjacency Matrix

There is no perfect matchings.

G



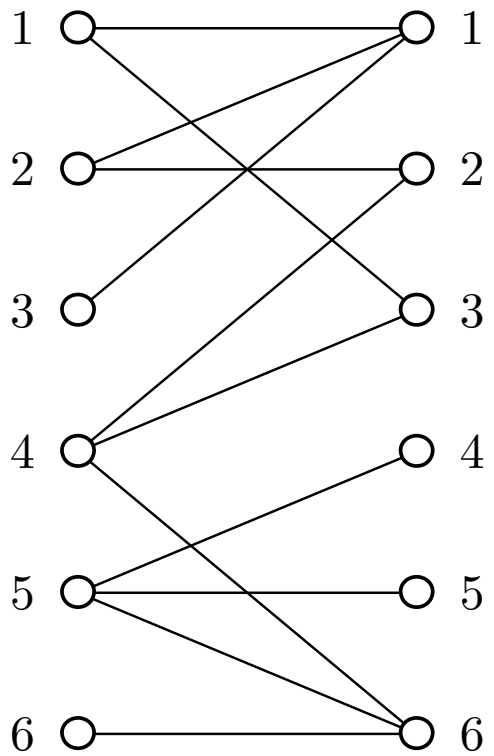
$$\Longrightarrow \quad A(G) = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\det(A(G)) = 0$$

Random Adjacency Matrix

There is no perfect matchings.

G



$\Longrightarrow$

$$A(G) = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\det(A(G)) = 0$$

Zippel-Schwartz Lemma

Lemma 1 (Zippel, Schwartz) *Let $f(x_1, \ldots, x_k)$ be a degree n polynomial over the field F. Polynomial f has no more than $\frac{n}{|F|}|F|^k$ zeros.*

Let $F = \mathcal{Z}_p$ for some prime number $p = \Theta(n^{1+c})$, then the operations in Z_p can be performed in constant time.

The probability of a *false zero* – we get zero value for a non-zero polynomial – equals $O(\frac{1}{n^c})$.

Outline

- Introduction
- Simple example - perfect matchings
- **Dynamic algebraic algorithms**
  - **determinant and inverse**
  - matrix rank and characteristic polynomial
- Dynamic graph algorithms
  - transitive closure
  - distances in graphs
  - vertex connectivity and matchings
- Static graph algorithms
  - matchings in graphs
  - distances in graphs

Dynamic Functions

Let $f : \mathcal{R}^n \to \mathcal{R}^m$ be an n argument function returning m results.

A dynamic algorithm for f supports following operations:

- **initialization($x_1, \ldots, x_n$):** set the input vector to $(x_1, \ldots, x_n)$,
- **update(k,x'_k):** change the k-th input to x'_k,
- **query(k):** return the k-th result.

We will consider the problems of dynamically computing the determinant, the inverse matrix and the matrix rank.

Dynamic Matrix Functions

We are given the matrix:

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \qquad \det(A) = -2$$

Dynamic Matrix Functions

We are given the matrix:

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \qquad \det(A) = -2$$

After the change:

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \qquad \det(A) = -4$$

Dynamic Matrix Functions

We are given the matrix:

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \qquad \det(A) = -2$$

After the change:

$$A = \begin{bmatrix} 2 & 1 & 2 \\ 2 & 2 & 2 \\ 1 & 2 & 2 \end{bmatrix} \qquad \det(A) = 2$$

Dynamic Matrix Inverse

Theorem 2 (Sherman and Morrison '49)
The problem of dynamically computing:

- *the determinant,*
- *the inverse matrix,*

for non-singular column updates can be solved with the following costs:

- **initialization:** $O(n^\omega)$ *time,*
- **update:** $O(n^2)$ *time,*
- **query:** $O(1)$ *time.*

Dynamic Matrix Functions

We are given the matrix:

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \qquad \det(A) = -2$$

Dynamic Matrix Functions

We are given the matrix:

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \qquad \det(A) = -2$$

After the change:

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 2 & 2 \\ 0 & 2 & 2 \end{bmatrix} \qquad \det(A) = 0$$

Dynamic Matrix Functions

We are given the matrix:

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix} \qquad \det(A) = -2$$

After the change:

$$A = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 2 & 2 \\ 0 & 2 & 2 \end{bmatrix} \qquad \det(A) = 0$$

Algorithm returns FALIURE.

Dynamic Matrix Inverse

Theorem 3 (Sankowski '04)
The problem of dynamically computing:

- *the determinant,*
- *the inverse matrix,*

for non-singular element updates can be solved with the following costs:

- **initialization:** $O(n^{\omega})$ *time,*
- **update:** $O(n^{1.575})$ *time,*
- **query:** $O(n^{0.575})$ *time.*

Dynamic Matrix Inverse

Theorem 4 (Sankowski '04)
The problem of dynamically computing:

- *the determinant,*
- *the inverse matrix,*

for non-singular element updates can be solved with the following costs:

- **initialization:** $O(n^{\omega})$ *time,*
- **update:** $O(n^{1.495})$ *time,*
- **query:** $O(n^{1.495})$ *time.*

Outline

- Introduction
- Simple example - perfect matchings
- **Dynamic algebraic algorithms**
  - determinant and inverse
  - **matrix rank and characteristic polynomial**
- Dynamic graph algorithms
  - transitive closure
  - distances in graphs
  - vertex connectivity and matchings
- Static graph algorithms
  - matchings in graphs
  - distances in graphs

Dynamic Matrix Rank

Theorem 5 (Sankowski '07)

Dynamic Matrix Inverse
Update $O(n^\alpha)$ operations
Query $O(n^\alpha)$ operations
assumes non-singularity

Dynamic Matrix Rank
Update $O(n^\alpha)$ time
Query $O(1)$ time
randomized

Dynamic Characteristic Poly.

Theorem 6 (Frandsen and Sankowski '08)
There exists a dynamic algorithm for computing the characteristic polynomial of a generic matrix supporting **column updates** *in* $\tilde{O}(n^2)$ *worst case time and* **queries** *in* $O(1)$ *time.*

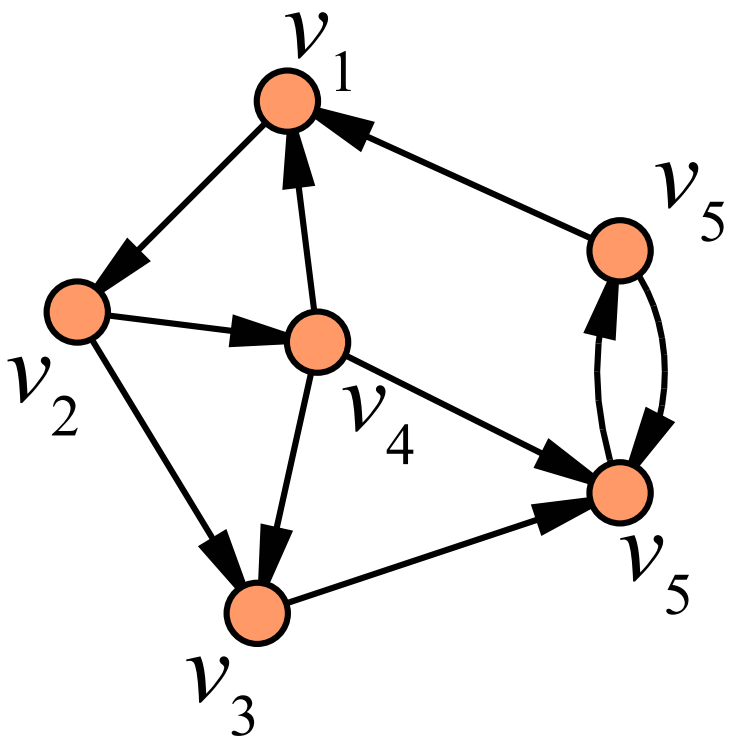
- Can work with non-generic matrices, but requires more time.
- Can be used to approximate eigenvalues of the matrix.

Outline

- Introduction
- Simple example - perfect matchings
- Dynamic algebraic algorithms
  - determinant and inverse
  - matrix rank and characteristic polynomial
- **Dynamic graph algorithms**
  - **transitive closure**
  - distances in graphs
  - vertex connectivity and matchings
- Static graph algorithms
  - matchings in graphs
  - distances in graphs

Dynamic Transitive Closure

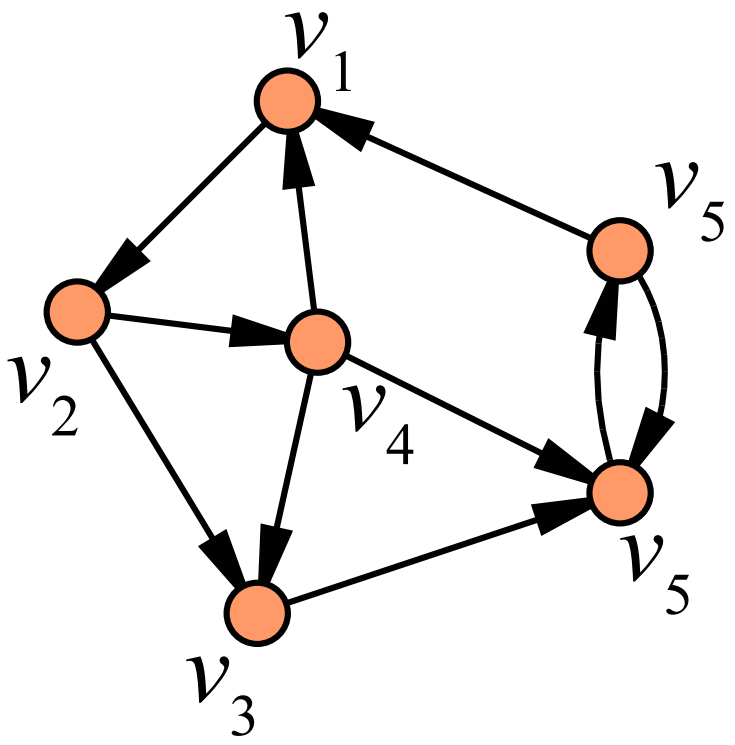
For a given graph:



Is there a path from v_1 to v_4?

Dynamic Transitive Closure

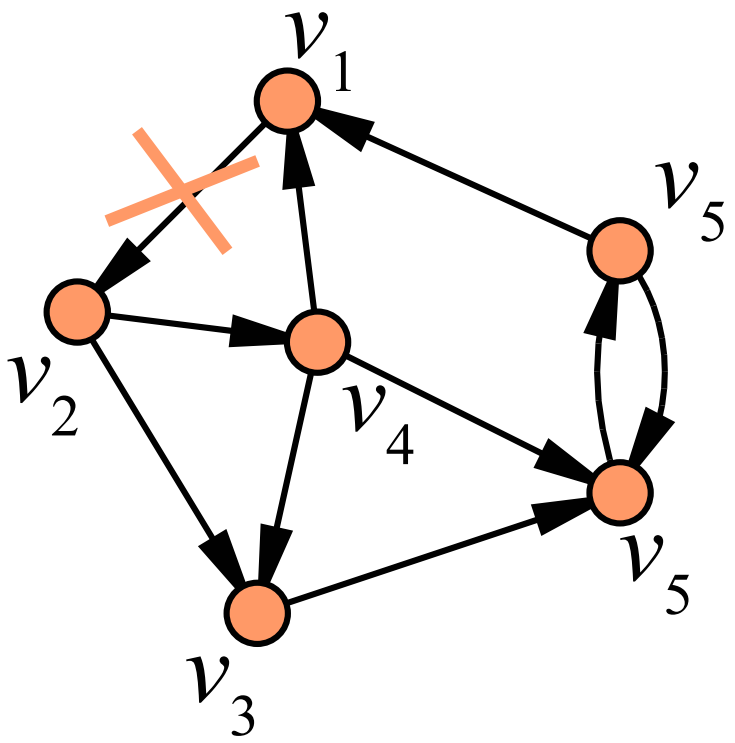
For a given graph:



Is there a path from v_1 to v_4? YES

Dynamic Transitive Closure

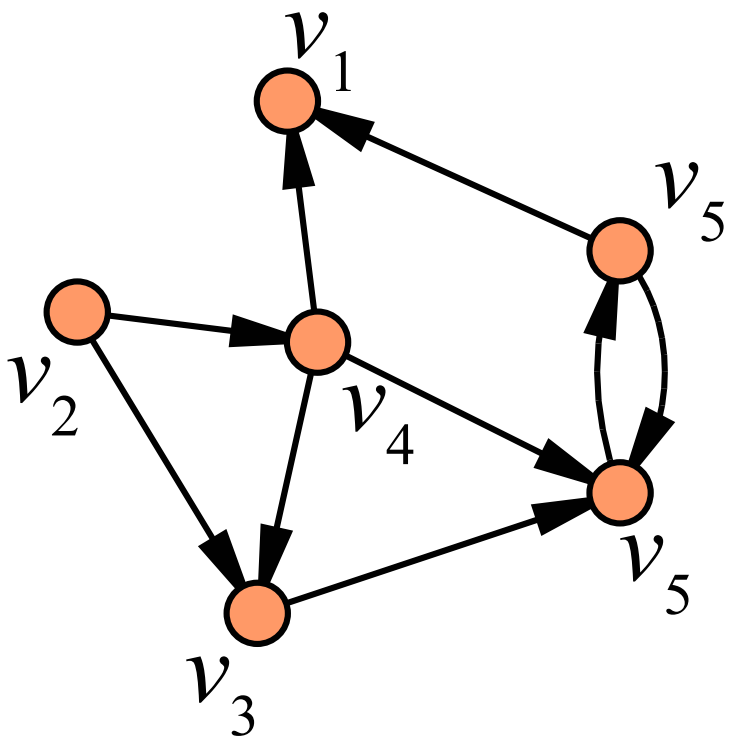
For a given graph:



Is there a path from v_1 to v_4?

Dynamic Transitive Closure

For a given graph:



Is there a path from v_1 to v_4? NO

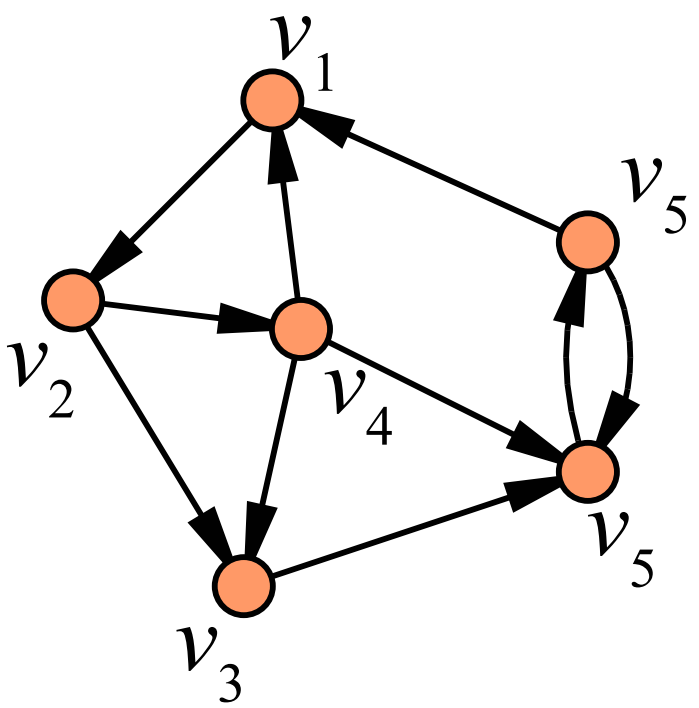
Dynamic Transitive Closure

	Update	**Query**
Henzinger and King '95	$\tilde{O}(nm^{0.58})$	$\Theta(n/\log n)$
King and Sagert '99	$O(n^{2.26})$	$O(1)$
King '99	$O(n^2 \log n)$	$O(1)$
Demetrescu and Italiano '00	$O(n^2)$	$O(1)$
Roditty and Zwick '02	$O(m\sqrt{n})$	$O(\sqrt{n})$
Roditty and Zwick '04	$O(m + n \log n)$	$O(n)$
Sankowski '04 (worst-case)	$O(n^2)$	$O(1)$
Sankowski '04 (worst-case)	$O(n^{1.575})$	$O(n^{0.575})$
Sankowski '04 (worst-case)	$O(n^{1.495})$	$O(n^{1.495})$

Symbolic Adjacency Matrix

Symbolic adjacency matrix of the graph:

G



$$\Longrightarrow \quad \tilde{A}(G) = \begin{pmatrix} 1 & x_{1,2} & 0 & 0 & 0 & 0 \\ 0 & 1 & x_{2,3} & x_{2,4} & 0 & 0 \\ 0 & 0 & 1 & 0 & x_{3,5} & 0 \\ x_{4,1} & 0 & x_{4,3} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & x_{5,6} \\ x_{6,1} & 0 & 0 & 0 & x_{6,5} & 1 \end{pmatrix}$$

Symbolic Adjacency Matrix

Let us compute $\mathrm{adj}(A)_{1,3} = (-1)^{1+3} \det(A^{3,1})$.

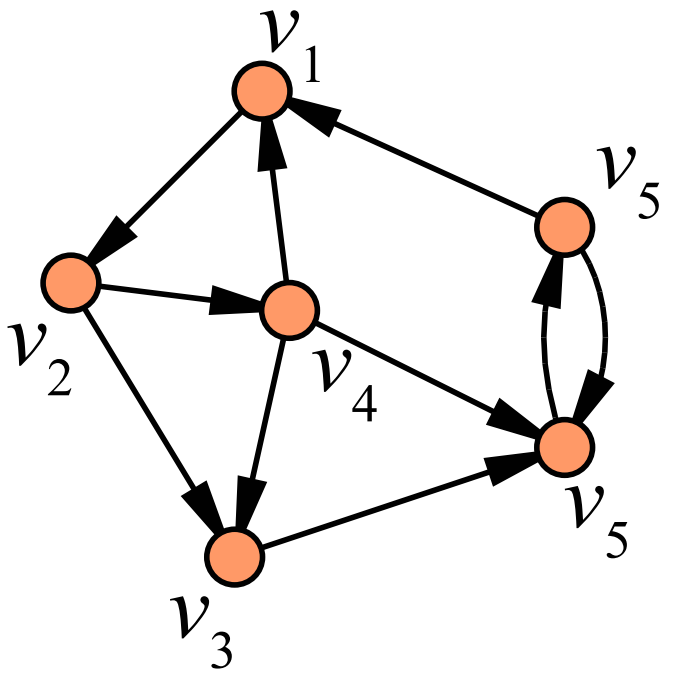
$$A$$

$$\left(\begin{array}{c|cccc|c} 1 & x_{1,2} & 0 & 0 & 0 & 0 \\ 0 & 1 & x_{2,3} & x_{2,4} & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & x_{3,5} & 0 \\ \hline x_{4,1} & 0 & x_{4,3} & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & x_{5,6} \\ x_{6,1} & 0 & 0 & 0 & x_{6,5} & 1 \end{array}\right) \Longrightarrow$$

$$A^{3,1}$$

$$\begin{pmatrix} x_{1,2} & 0 & 0 & 0 & 0 \\ 1 & x_{2,3} & x_{2,4} & 0 & 0 \\ 0 & x_{4,3} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_{5,6} \\ 0 & 0 & 0 & x_{6,5} & 1 \end{pmatrix}$$

Symbolic Adjacency Matrix



$$\det \begin{pmatrix} x_{1,2} & 0 & 0 & 0 & 0 \\ 1 & x_{2,3} & x_{2,4} & 0 & 0 \\ 0 & x_{4,3} & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_{5,6} \\ 0 & 0 & 0 & x_{6,5} & 1 \end{pmatrix} =$$

$$= x_{1,2}x_{2,3} - x_{1,2}x_{2,4}x_{4,3} +$$

$$-x_{1,2}x_{2,3}x_{5,6}x_{6,5} + x_{1,2}x_{2,4}x_{4,3}x_{5,6}x_{6,5}.$$

The monomials of the determinant correspond to paths from v_1 to v_3 in G.

Dynamic Transitive Closure

Theorem 7 (Sankowski '04) *Let $\tilde{A}$ be a symbolic adjacency matrix of G, substitute random numbers into variables in obtaining the matrix A:*

- *there is a path from i to j in G iff $(I - A)^{-1}_{ij}$ is non-zero (with high probability).*

This allows us to compute the transitive closure by inverting the matrix once — can be easily used in the dynamic case.

Transitive Closure

Theorem 8 (Sankowski '04)

Dynamic matrix inverse

Update in $O(n^\alpha)$ time

Query in $O(n^\beta)$ time

can assume nonsingularity

Dynamic transitive closure

Update in $O(n^\alpha)$ time

Query in $O(n^\beta)$ time

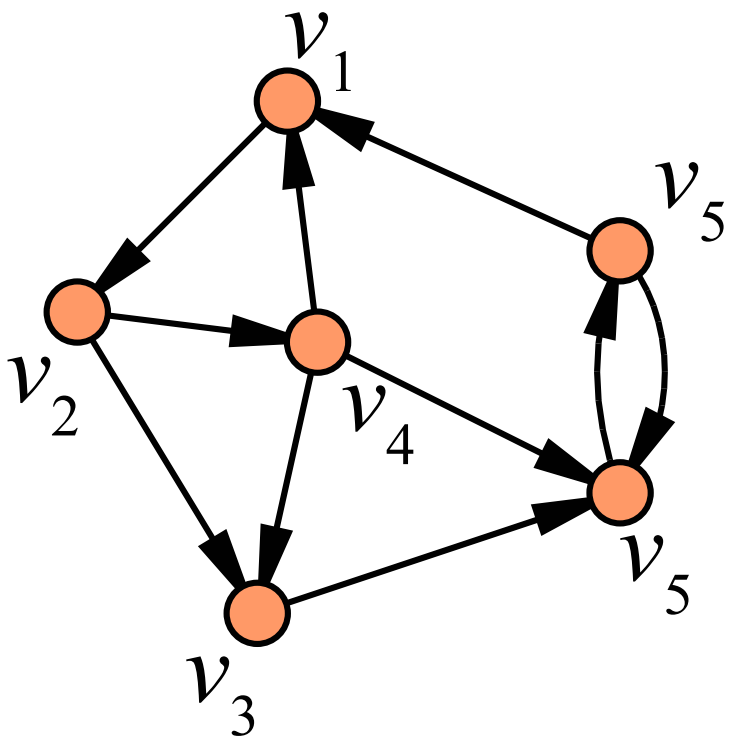
randomized with one sided error

Outline

- Introduction
- Simple example - perfect matchings
- Dynamic algebraic algorithms
  - determinant and inverse
  - matrix rank and characteristic polynomial
- **Dynamic graph algorithms**
  - transitive closure
  - **distances in graphs**
  - vertex connectivity and matchings
- Static graph algorithms
  - matchings in graphs
  - distances in graphs

Dynamic Distances

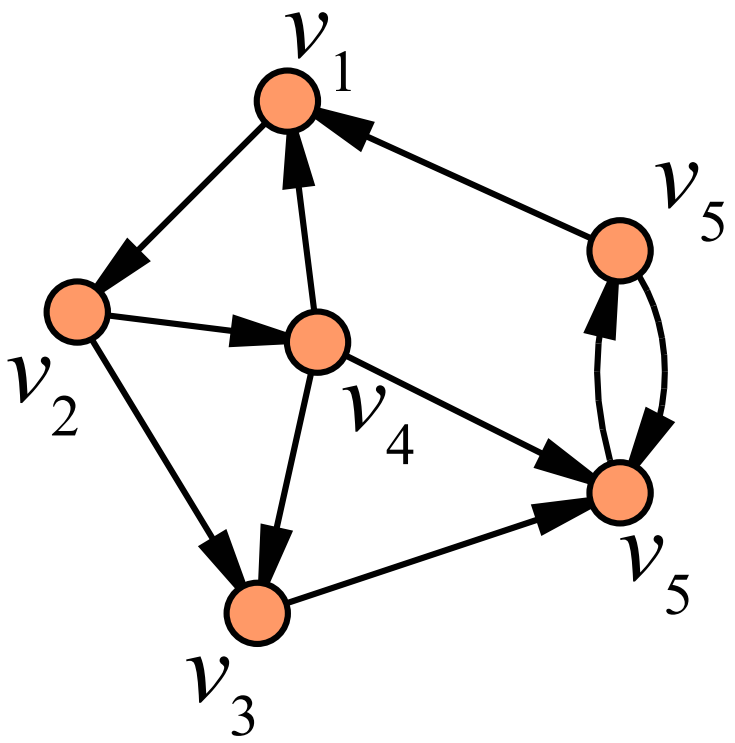
For a given graph:



What is the distance from v_1 to v_3?

Dynamic Distances

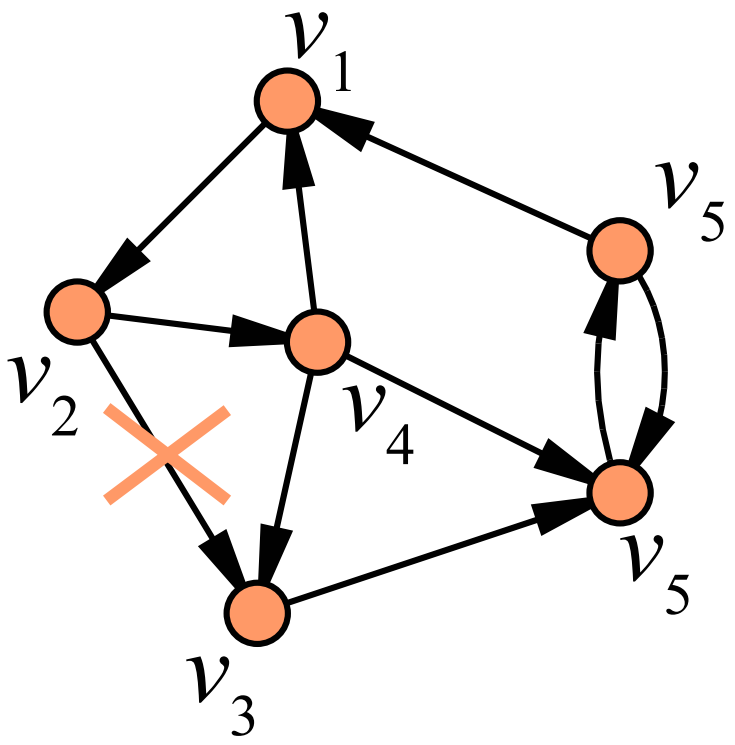
For a given graph:



What is the distance from v_1 to v_3? 2

Dynamic Distances

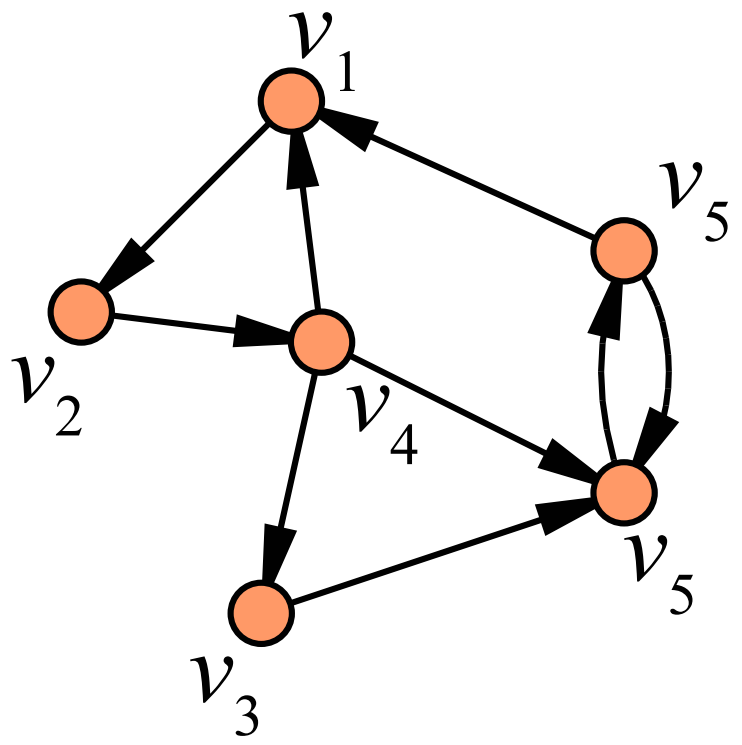
For a given graph:



What is the distance from v_1 to v_3?

Dynamic Distances

For a given graph:



What is the distance from v_1 to v_3? 3

Dynamic Distances

	Update	**Query**
Henzinger et al. '97 planar graphs	$O(n^{\frac{4}{3}}\log(nC))$	$O(n^{\frac{4}{3}}\log(nC))$
Fakcharoemphol and Rao '02 planar graphs	$O(n^{\frac{4}{5}}\log^{\frac{13}{5}}n)$	$O(n^{\frac{4}{5}}\log^{\frac{13}{5}}n)$
King '99	$O(n^{2.5}\sqrt{C\log n})$	$O(1)$
Demetrescu and Italiano '00	$O(n^{2.5}\sqrt{S\log^3 n})$	$O(1)$
Demetrescu and Italiano '03	$\tilde{O}(n^2)$	$O(1)$
Thorup '05	$\tilde{O}(n^{2.75})$ *(worst case)*	$O(1)$
Sankowski '05	$O(n^{1.932})$ *(worst case)*	$O(n^{1.288})$

Dynamic Distances

Extending the technique from transitive closure and using the path decomposition technique of Knuth-Greene, we get:

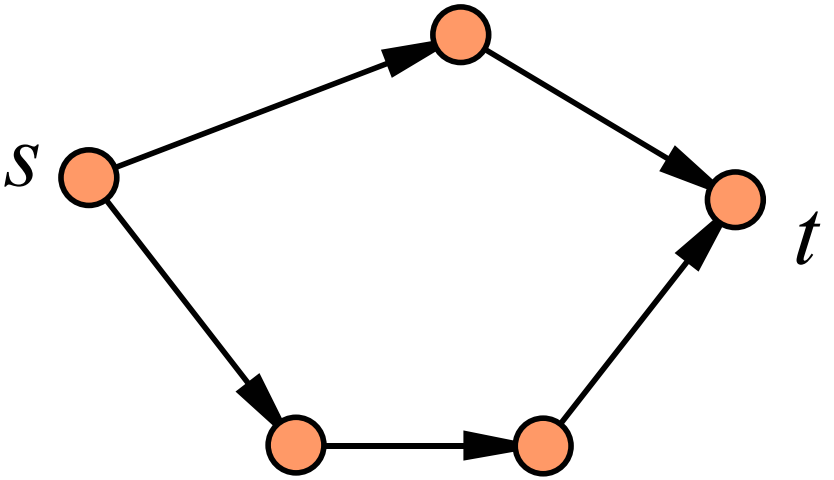
Theorem 9 (Sankowski '05) *There exists a randomized dynamic algorithm for computing the distances in a directed graph with unit edge weights in supporting* **updates** *in* $O(n^{1.932})$ *time and* **queries** *in* $O(n^{1.288})$ *time.*

Outline

- Introduction
- Simple example - perfect matchings
- Dynamic algebraic algorithms
  - determinant and inverse
  - matrix rank and characteristic polynomial
- **Dynamic graph algorithms**
  - transitive closure
  - distances in graphs
  - **vertex connectivity and matchings**
- Static graph algorithms
  - matchings in graphs
  - distances in graphs

Dynamic Vertex Connectivity

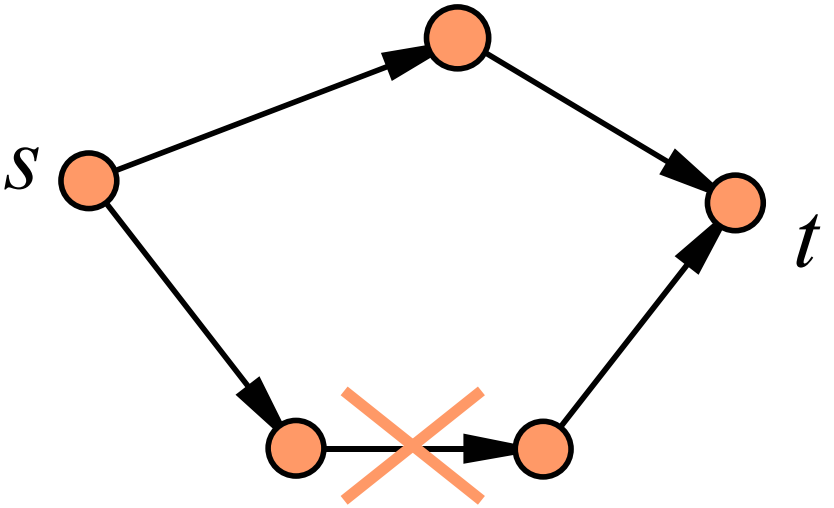
For the dynamic graph:



we want to compute the number of vertex disjoint paths from s to t? 2

Dynamic Vertex Connectivity

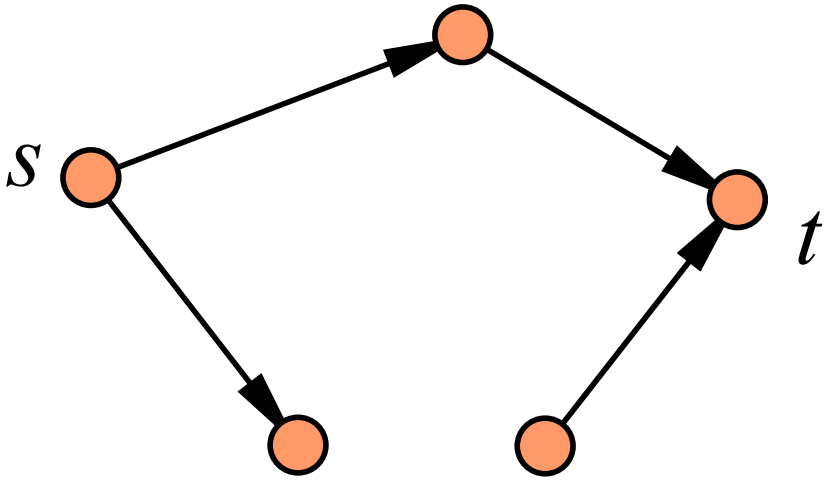
For the dynamic graph:



we want to compute the number of vertex disjoint paths from s to t?

Dynamic Vertex Connectivity

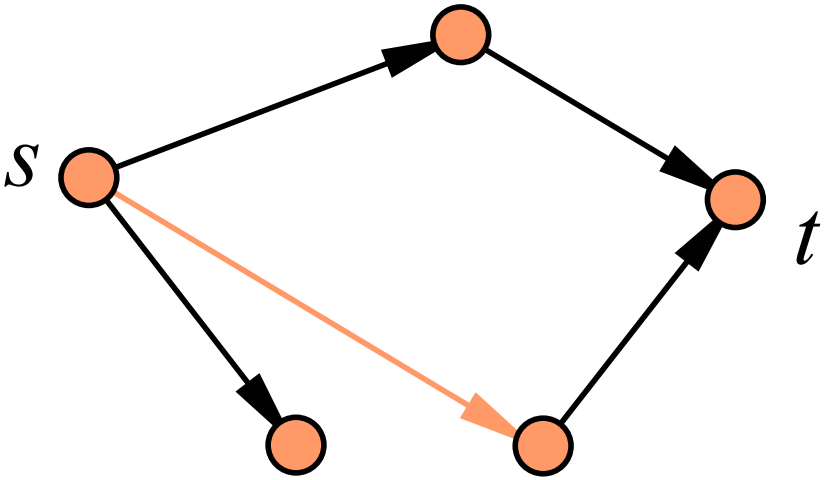
For the dynamic graph:



we want to compute the number of vertex disjoint paths from s to t? 1

Dynamic Vertex Connectivity

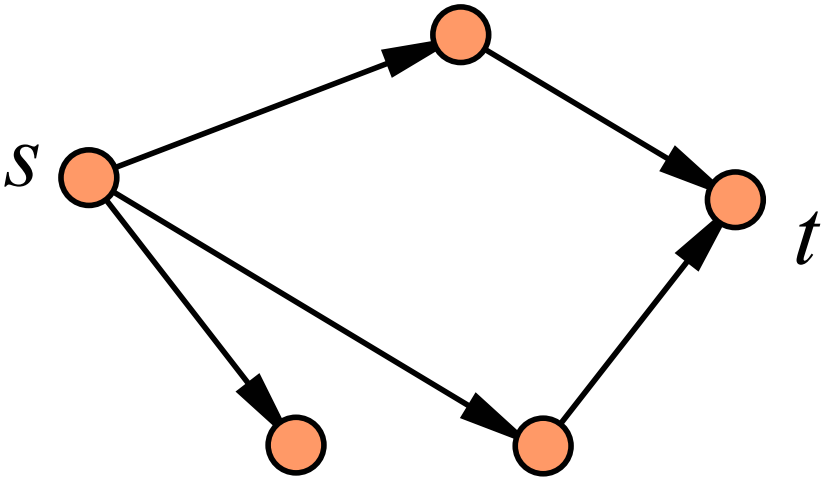
For the dynamic graph:



we want to compute the number of vertex disjoint paths from s to t?

Dynamic Vertex Connectivity

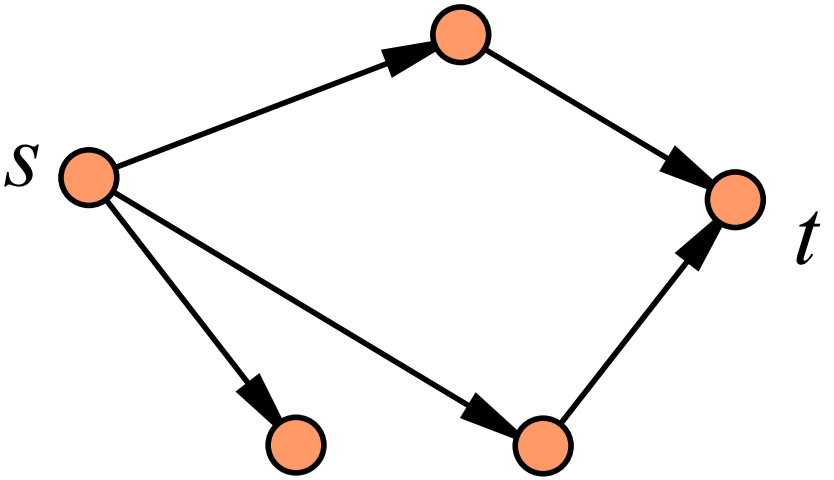
For the dynamic graph:



we want to compute the number of vertex disjoint paths from s to t? 2

Dynamic Vertex Connectivity

For the dynamic graph:



we want to compute the number of vertex disjoint paths from s to t? 2

The s and t vertices can be common for these paths.

Dynamic Vertex Connectivity

Theorem 10 (Sankowski '07)

Dynamic matrix rank		Dynamic s,t-paths
Update in $O(n^\alpha)$ time	$\Rightarrow$	Update in $O(n^\alpha)$ time
Query in $O(1)$ time		Query in $O(1)$ time

Theorem 11 (Sankowski '07) *There exists an algorithm for dynamically testing if the graph is k vertex connected that supports edge updates in $\tilde{O}(n^{1.575} + nk^2)$ time and queries in $O(1)$ time. The algorithm is randomized.*

Dynamic Results

Dynamic k-vertex connectivity:

- undirected connectivity and 2-vertex connectivity in $\tilde{O}(1)$ time by Holm *et al.* '00.
- undirected 3-vertex connectivity and 4-vertex connectivity in $\tilde{O}(n)$ time by Eppstein *et al.* '97.
- directed 1-vertex connectivity, i.e., strong connectivity in $O(n^{1.575})$ time using the transitive closure algorithm by Sankowski '04.

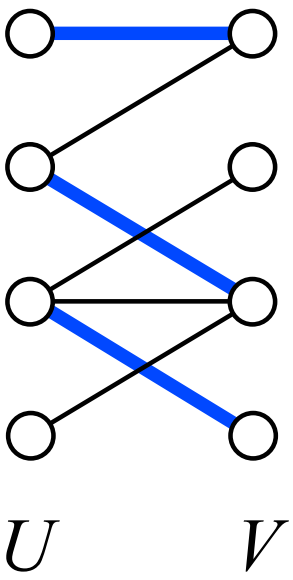
Static Results

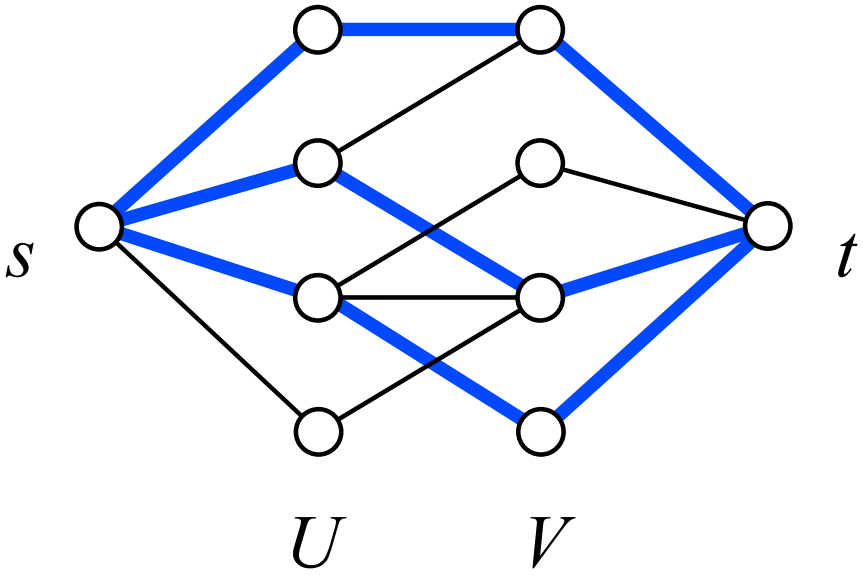
Static k-vertex connectivity:

Complexity	Author
$\tilde{O}(n^\omega + nk^\omega)$	*(undirected)* Linial, Lovász and Wigderson '88
$\tilde{O}(n^\omega + nk^\omega)$	Cheriyan and Reif '92
$O((k^{5/2} + n)m)$	Gabow '00
$O((k + n^{1/4})n^{3/4}m)$	Gabow '00
$O((k^{5/2} + n)kn)$	*(undirected)* Gabow '00
$O((k + n^{1/4})kn^{7/4})$	*(undirected)* Gabow '00

Our result – $\tilde{O}(n^{1.575} + nk^2)$ time per update.

Dynamic Maximum Matchings





Theorem 12 (Sankowski '07)

Dynamic matrix rank $\Rightarrow$ Dynamic max. matchings

Update in $O(n^\alpha)$ time $\Rightarrow$ Update in $O(n^\alpha)$ time

Query in $O(1)$ time $\Rightarrow$ Query in $O(1)$ time

Outline

- Introduction
- Simple example - perfect matchings
- Dynamic algebraic algorithms
  - determinant and inverse
  - matrix rank and characteristic polynomial
- Dynamic graph algorithms
  - transitive closure
  - distances in graphs
  - vertex connectivity and matchings
- **Static graph algorithms**
  - **matchings in graphs**
  - distances in graphs

Matchings

Theorem 13 (Lovász) *Let A be a random adjacency matrix of the graph G, then:*

- *G contains a perfect matching iff* $\det(A) \neq 0$ *(with high probability).*

This allows us to test if the graph contains a perfect matching by computing the determinant once — can be used in the dynamic case!

Can we use this idea to find the perfect matching?

Matchings

The edge e is *allowed* if it is contained in some perfect matching.

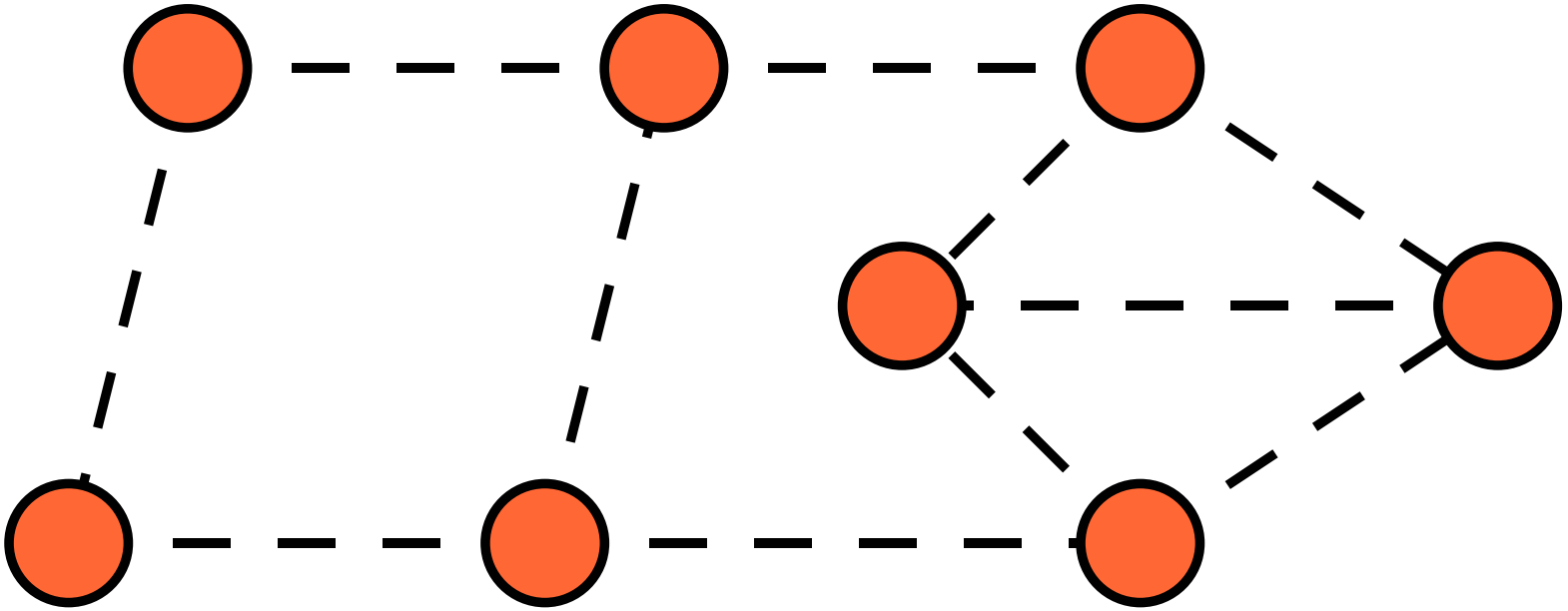
Theorem 14 (Rabin and Vazrirani) *Let A be a random adjacency matrix of the graph G, then:*

- *the edge uv is allowed iff $A^{-1}_{u,v} \neq 0$ (with high probability).*

Thus we can determine the allowed edges using the inverse matrix.

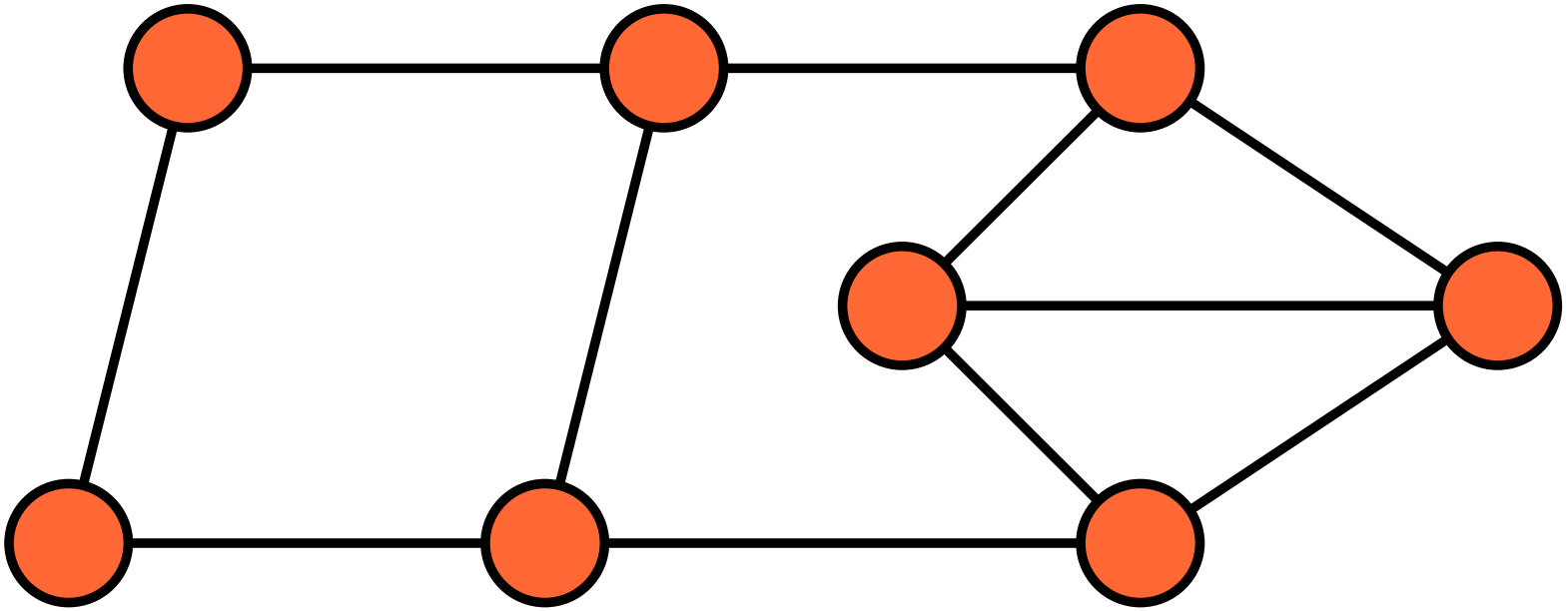
Matchings

For a dynamic graph we can determine allowed edges and build a matching step by step:



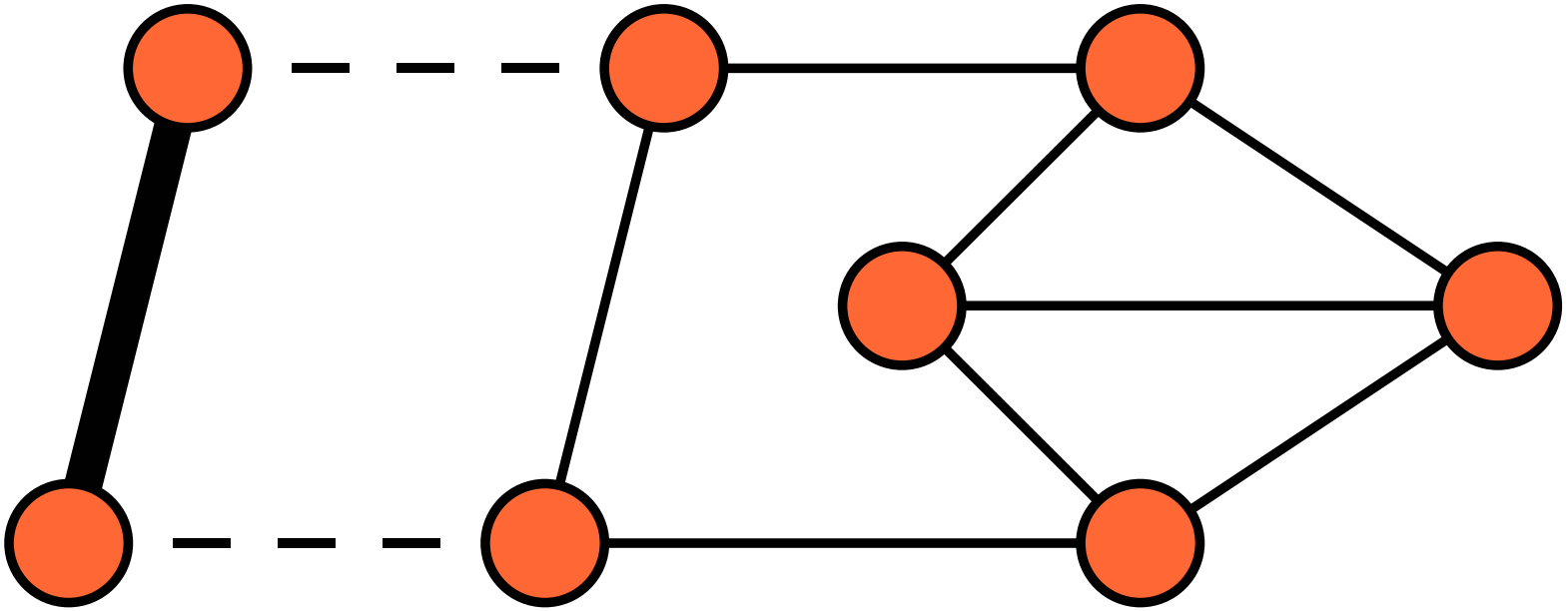
Matchings

For a dynamic graph we can determine allowed edges and build a matching step by step:



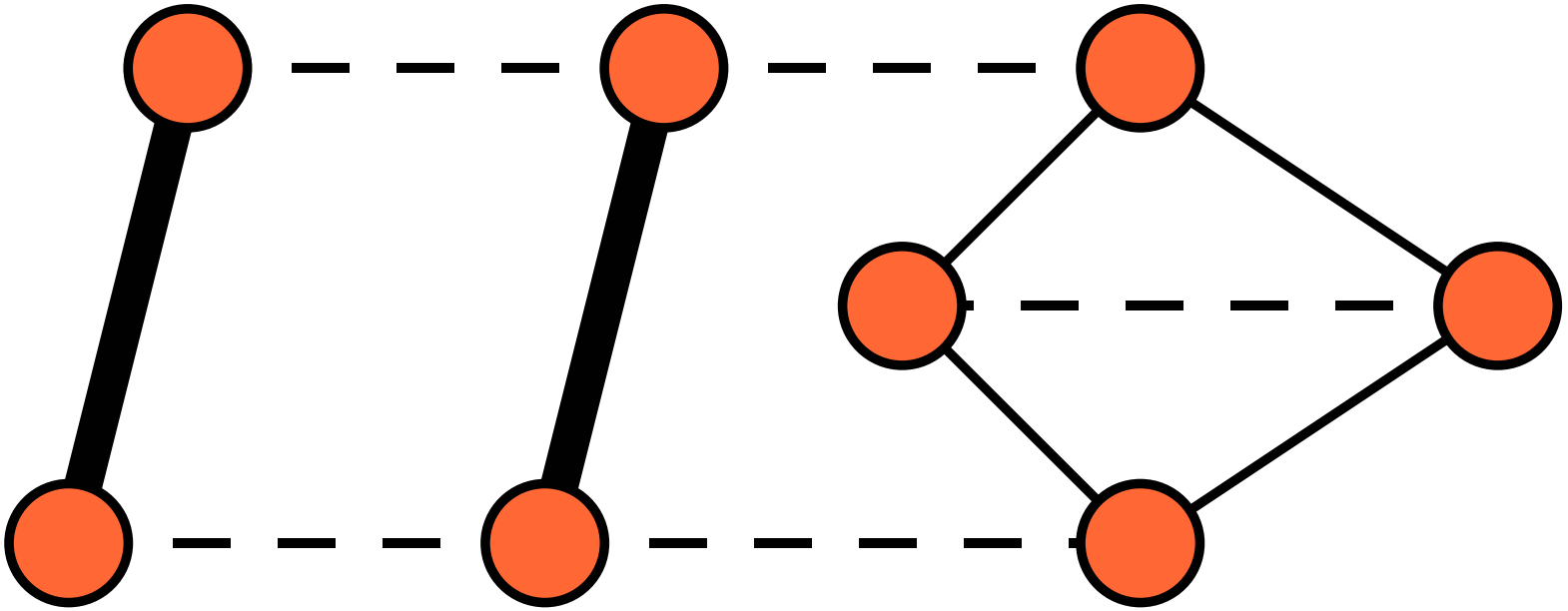
Matchings

For a dynamic graph we can determine allowed edges and build a matching step by step:



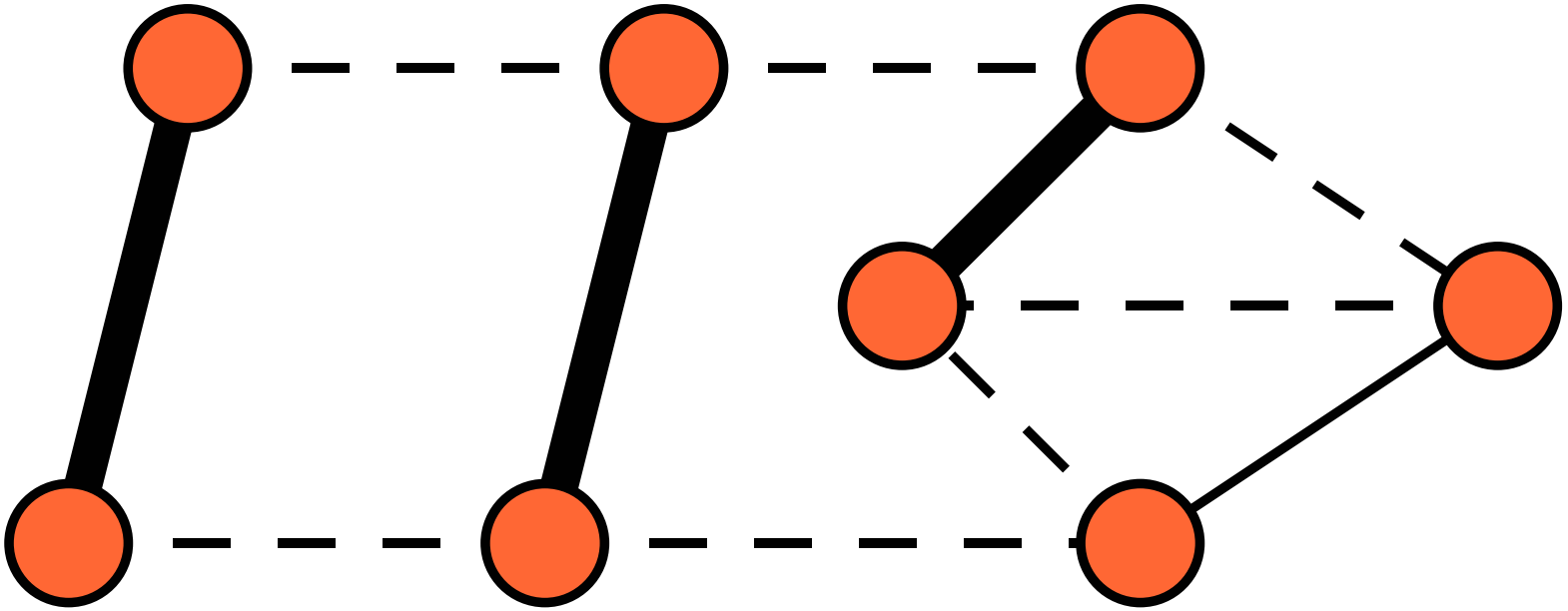
Matchings

For a dynamic graph we can determine allowed edges and build a matching step by step:



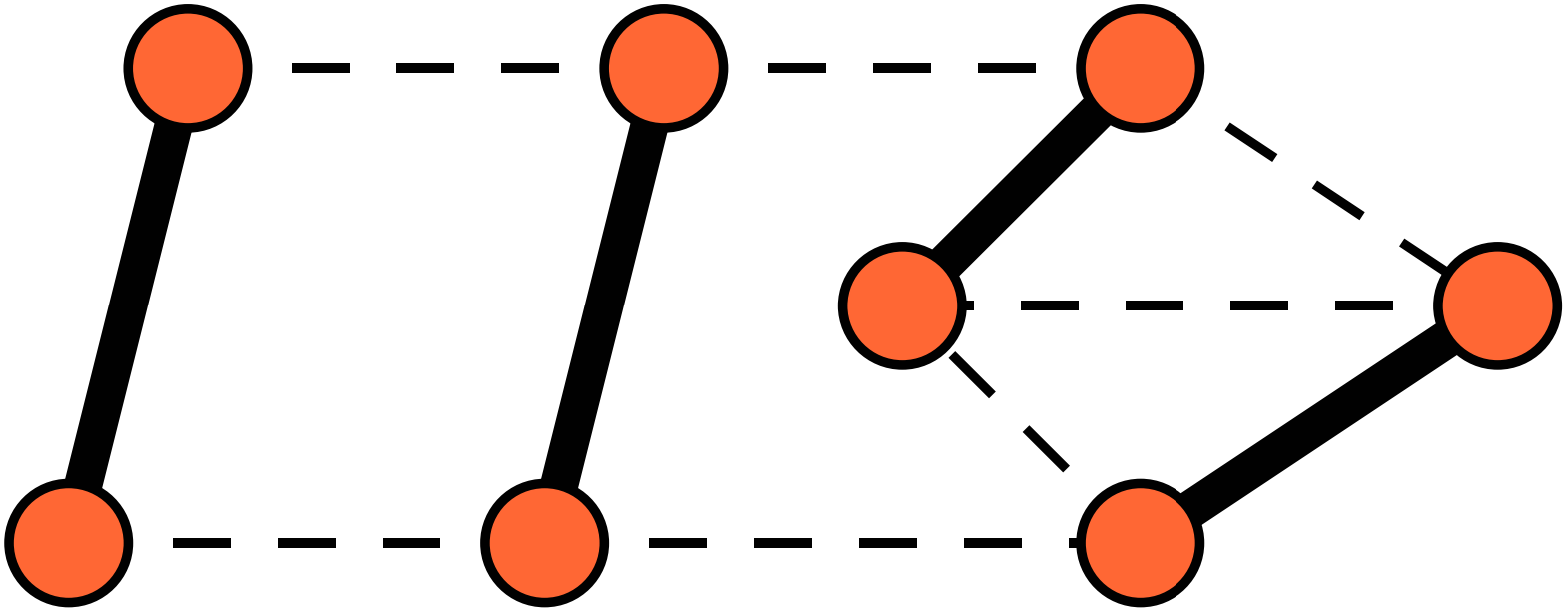
Matchings

For a dynamic graph we can determine allowed edges and build a matching step by step:



Matchings

For a dynamic graph we can determine allowed edges and build a matching step by step:



Matchings

Using dynamic matrix inverse we can find allowed edges and remove other edges — we get an $O(n^{2.575})$ time algorithm for finding perfect matchings in graphs.

Using Gaussian Elimination that works in $O(n^{\omega})$ time we get:

Theorem 15 (Sankowski and Mucha '04) *There exist a randomized algorithm finding perfect (and maximum) matchings in graph in $O(n^{\omega})$ time.*

Previous Results

Previous known results:

- $O(\sqrt{n}m)$ — for bipartite graphs (Hopcroft, Karp '73),
- $O(\sqrt{n}m)$ — for general graphs (Micali, Vazirani '80).

For dense graphs the algorithms work in $O(n^{2.5})$ time.

Planar Graphs

The results *(Mucha & Sankowski '04)*:

- simple $O(n^{\frac{3}{2}})$ time algorithm,
- $O(n^{\frac{\omega}{2}}) = O(n^{1.19})$ time algorithm.

Previous results:

- $\tilde{O}(n^{\frac{4}{2}})$ for perfect matchings in bipartite graphs — Klein et al. '94,
- $O(n^{\frac{3}{2}})$ for general planar graphs — Lipton, Tarjan '77 or Micali, Vazirani '80.

Weighted Matchings

We are given the graph $G = (V, E)$ and the function $w : E \to \{0, \ldots, W\}$ that ascribes weights to edges.

The *weight* of the matching M is the sum of the weights of the edges in M.

We want to find a perfect matching with maximum weight.

Storjohann's Algorithm

Theorem 16 (Storjohann '03)
Let $A \in K[x]^{n \times n}$ be a polynomial matrix of degree d and $b \in K[x]^{n \times 1}$ be a polynomial vector of the same degree, then

- *rational system solution $A^{-1}b$,*
- *determinant* $\det(A)$,

can be computed in $O(n^\omega d)$ operations in K, with high probability.

Weighted Matchings

The weighted problem in bipartite graphs can be reduced to unweighed one in $O(Wn^\omega)$ time.

Theorem 17 (Sankowski '06) *There exist a randomized algorithm finding maximum weight perfect matchings in bipartite graphs in $O(Wn^\omega)$ time.*

Previous results:

- $O(nm)$ — Edmonds and Karp '72,
- $O(\sqrt{n}m\log(nW))$ — Gabow and Tarjan '89.

For dense graphs it is $O(n^3)$ or $O(n^{2.5}\log(nW))$.

Outline

- Introduction
- Simple example - perfect matchings
- Dynamic algebraic algorithms
  - determinant and inverse
  - matrix rank and characteristic polynomial
- Dynamic graph algorithms
  - transitive closure
  - distances in graphs
  - vertex connectivity and matchings
- **Static graph algorithms**
  - matchings in graphs
  - **distances in graphs**

Distances in Graphs

We are given the graph $G = (V, E)$ and the weight function $w : E \to \{-W, \ldots, W\}$ that ascribes weights to the edges.

The *length* of a path p is the sum of the weights of the edges on p.

The *distance* from v to w is the length of the shortest path from v to w.

For the given vertex v we want to compute the distances to all other vertices or find a negative length cycle.

Distances in Graphs

We can compute the distances by solving a linear system of equalities over polynomials.

Theorem 18 (Sankowski '05) *There exist a randomized algorithm for computing the distances from a given vertex in $O(Wn^{\omega})$ time.*

As well Yuster and Zwick '05

Previous results:

- $O(nm)$ — arbitrary weights (Belman and Ford) ,
- $O(\sqrt{n}m \log(W))$ — integer weights (Goldberg),

For dense graphs we get $O(n^3)$ or $O(n^{2.5} \log(W))$.

Conclusions

The algebraic techniques can be used to construct the asymptotically fastest algorithms for:

- dynamic transitive closure,
- dynamic distances in graphs,
- dynamic vertex connectivity,
- dynamic maximum matchings,
- maximum matchings in graphs,
- maximum weighted matchings in bipartite graphs,
- distances in graphs.

i.e., the algebraic approach is useful.

Open Problems

Some open problems:

- Can one close the complexity gap between dynamic shortest paths and transitive closure?
- Can one find maximum weighted matchings in general graphs in matrix multiplication time?
- Can one dynamically compute the characteristic polynomial/eigenvalues in subquadratic time?