Matching

Algorithms and Networks

Universiteit Utrecht

This lecture

- Matching: problem statement and applications
- Bipartite matching
- Matching in arbitrary undirected graphs: Edmonds algorithm
- Perfect matchings in regular bipartite graphs
  - Schrijvers algorithm
  - Edge coloring and classroom scheduling application
- Diversion: generalized tic-tac-toe

1

Problem and applications

Universiteit Utrecht

Matching

- Set of edges M $\subseteq$ E such that no vertex is endpoint of more than one edge.
- Maximal matching
  - No $e \notin$ E with M $\cup$ $\{e\}$ also a matching
- Maximum matching
  - Matching with $|M|$ as large as possible
- Perfect matching
  - $|M| = n/2$: each vertex endpoint of edge in M.

Cost versions

- Each edge has cost; look for perfect matching with minimum cost
- Also polynomial time solvable, but harder

Problems

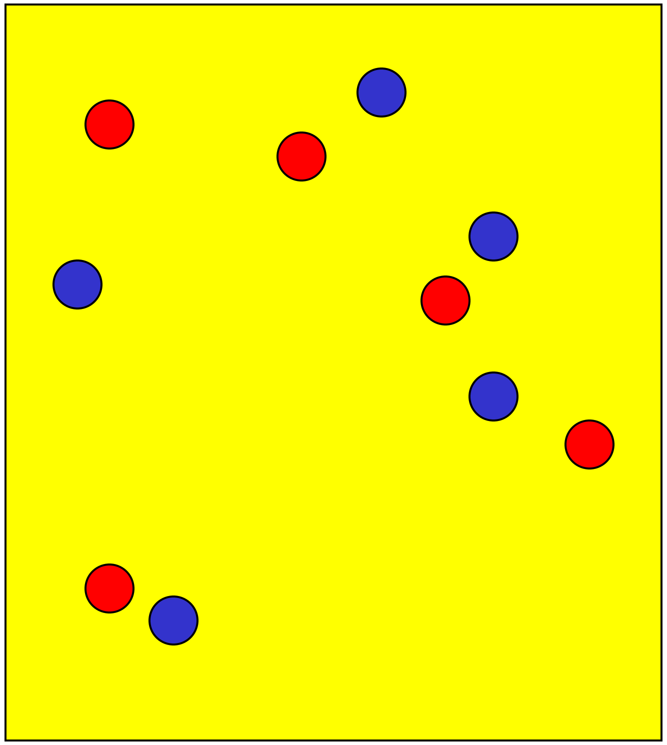
- Given graph G, find
  - Maximal matching: easy (greedy algorithm)
  - Maximum matching
    - Polynomial time; not easy.
    - Important easier case: bipartite graphs
  - Perfect matching
    - Special case of maximum matching
    - A theorem for regular bipartite graphs and Schrijver's algorithm

Applications

- Personnel assignment
  - Tasks and competences
- Classroom assignment
- Scheduling
- Opponents selection for sport competitions

Application: matching moving objects

- Moving objects, seen at two successive time moments
- Which object came from where?



2

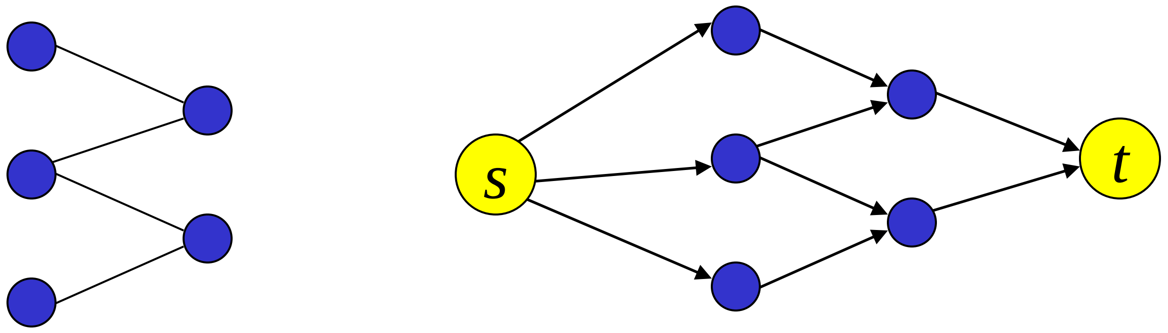
Bipartite matching

Universiteit Utrecht

Bipartite graphs: using maximum flow algorithms

- Finding maximum matching in bipartite graphs:
  - Model as flow problem, and solve it: make sure algorithm finds integral flow.

Capacities 1

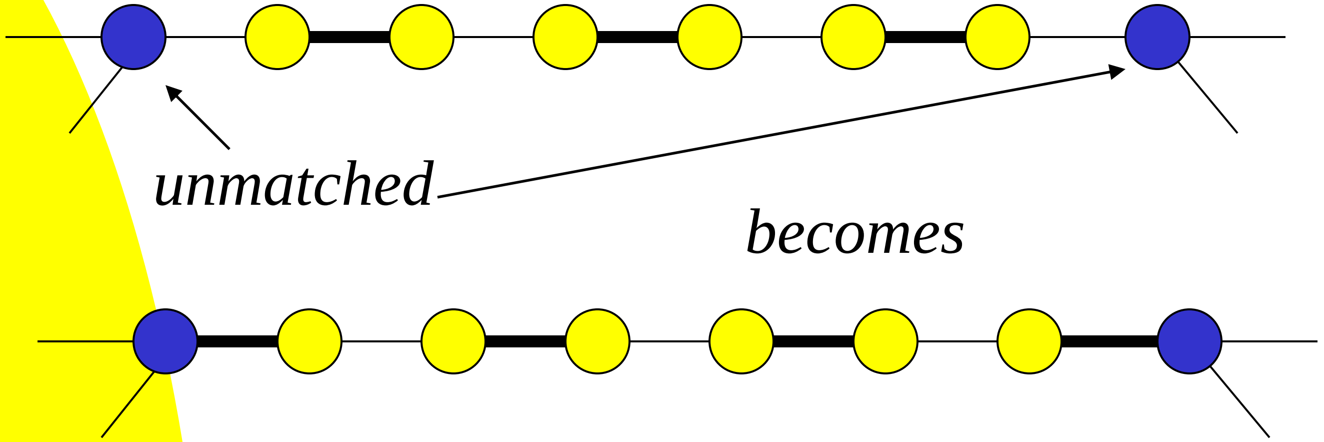


Technique works for variants too

- Minimum cost perfect matching in bipartite graphs
  - Model as mincost flow problem
- b-matchings in bipartite graphs
  - Function b: V $\rightarrow$ **N**.
  - Look for set of edges M, with each *v* endpoint of exactly b(*v*) edges in M.

Steps by Ford-Fulkerson on the bipartite graph

- M-augmenting path:



in a flow augmentation step

3

Edmonds algorithm:
matching in (possibly non-bipartite) undirected graphs

Universiteit Utrecht

A theorem that also works when the graph is not bipartite

Theorem. Let M be a matching in graph G. M is a maximum matching, if and only if there is no M-augmenting path.

– If there is an M-augmenting path, then M is not a maximum matching.
– Suppose M is not a maximum matching. Let N be a larger matching. Look at N*M = N $\cup$ M – N $\cap$ M.
  - Every node in N*M has degree 0, 1, 2: collection of paths and cycles. All cycles alternatingly have edge from N and from M.
  - There must be a path in N*M with more edges from N than from M: this is an augmenting path.

Algorithm of Edmonds

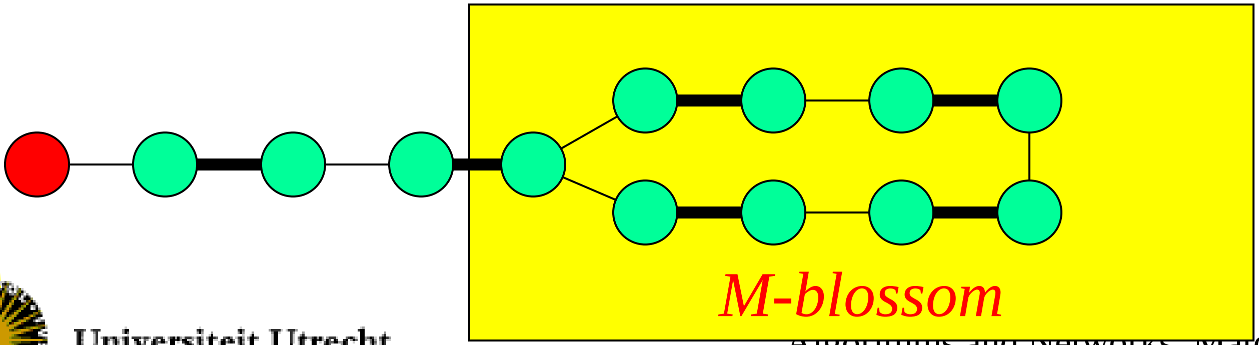
- Finds maximum matching in a graph in polynomial time

Jack Edmonds

Jack Edmonds

Definitions

- M-alternating walk:
  - (Possibly not simple) path with edges alternating in M, and not M.
- M-flower
  - M-alternating walk that starts in an unmatched vertex, and ends as:



Finding an M-augmenting path or an M-flower – I

- Let X be the set of unmatched vertices.
- Let Y be the set of vertices with an edge not in M to a vertex in X.
- Build digraph D = (V,A) with
  - A = { (u,v) | there is an x with $\{u,x\} \in$ E-M and $\{x,v\} \in$ M}.
- Find a shortest walk P from a vertex in X to a vertex in Y of length at least 1. (BFS in D.)
- Take P’: P, followed by an edge to X.
- P’ is M-alternating walk between two unmatched vertices.

Finding M-augmenting path or M-flower – II

Two cases:

- P’ is a simple path: it is an M-augmenting path
- P’ is not simple. Look to start of P’ until the first time a vertex is visited for the second time.
  - This is an M-flower:
    - Cycle-part of walk cannot be of even size, as it then can be removed and we have a shorter walk in D.

Algorithmic idea

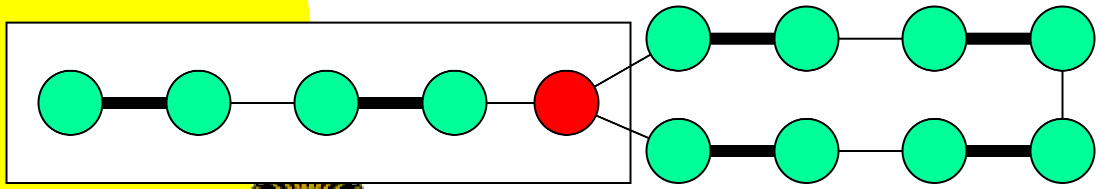
- Start with some matching M, and find either M-augmenting path or M-blossom.
- If we find an M-augmenting path:
  - Augment M, and obtain matching of one larger size; repeat.
- If we find an M-blossom, we *shrink* it, and obtain an equivalent smaller problem; recurs.

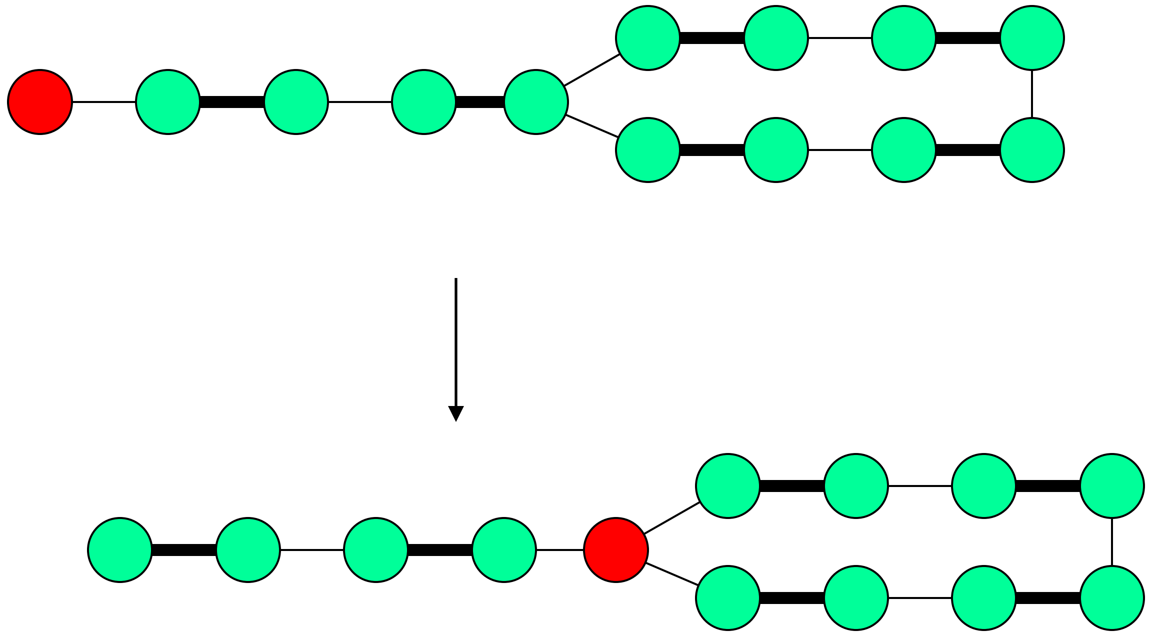
Shrinking M-blossoms

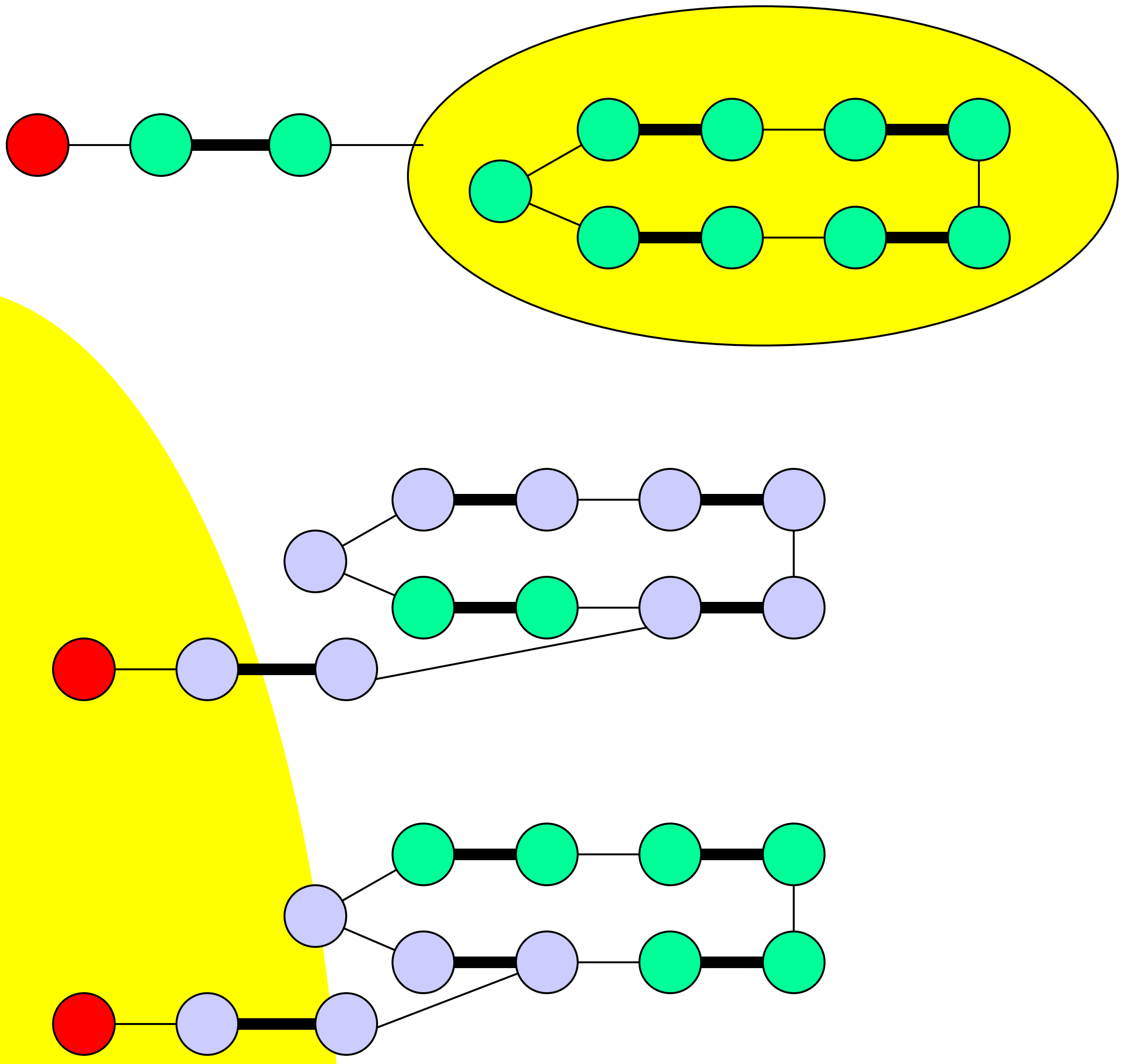
- Let B be a set of vertices in G.
- **G/B** is the graph, obtained from G by contracting B to a single vertex.
  - M/B: those edges in M that are not entirely on B.

Theorem

- **Theorem**: Let B be an M-blossom. Then M is a maximum size matching in G, if and only if M/B is a maximum size matching in G/B.
  - Suppose M/B is not max matching in G/B. Let P be M/B-augmenting path in G/B.
    - P does not traverse the vertex representing B: P also M-augmenting path in G: M not max matching in G.
    - P traverses B: case analysis helps to construct M-augmenting path in G.
  - Suppose M not max matching in G. Change M, such that vertex on M-blossom not endpoint of M.







Proof (continued)

- Take M-augmenting path P in G.
- If P does not intersect B then P also M/B-augmenting path, M/B not maximum matching.
- Otherwise, assume P does not start in B, otherwise reverse P.
  - Now, use start of P to get M/B augmenting path.

Subroutine

- *Given*: Graph G, matching M
- *Question*: Find M-augmenting path if it exists.
  - Let X be the vertices not endpoint of edge in M.
  - Build D, and test if there is an M-alternating walk P from X to X of positive length. (Using Y, etc.)
  - If no such walk exists: M is maximum matching.
  - If P is a path: output P.
  - If P is not a path:
    - Find M-blossom B on P.
    - Shrink B, and recourse on G/B and M/B.
    - If G/B has no M/B augmenting path, then M is maximum matching.
    - Otherwise, expand M/B-augmenting path to an M-augmenting path.

Edmonds algorithm

- A maximum matching can be found in O(n^2m) time.
  - Start with empty (or any) matching, and repeat improving it with M-augmenting paths until this stops.
  - O(n) iterations. Recursion depth is O(n); work per recursive call O(m).
- A perfect matching in a graph can be found in O(n^2m) time, if it exists.

Improvements

- Better analysis and data structures gives O(n^3) algorithm.
- Faster is possible: O($n^{1/2} m$) time.
- Minimum cost matchings with more complicated structural ideas.

4

Matching in regular bipartite graphs

Regular bipartite graphs

- Regular = all vertices have the same degree
- Say *d* is the degree of all vertices
- Theorem (proof follows): each regular bipartite graph has a perfect matching
- Schrijver’s algorithm: finds such a perfect matching quickly
- Coming: a nice application for scheduling classrooms and lessons

A simple non-constructive proof of a well known theorem

Theorem. Each regular bipartite graph has a perfect matching.

Proof:

- Construct flow model of G. Set flow of edges from *s*, or to *t* to 1, and other edges flow to 1/*d*.
- This flow has value *n*/2, which is optimal.
- Ford-Fulkerson will find flow of value *n*/2; which corresponds to perfect matching.

Perfect matchings in regular bipartite graphs

- Schrijver’s algorithm to find one:
  - Each edge *e* has a weight $w(e)$.
  - Initially all weights are 1.
  - Let G_w denote the graph formed by the edges of positive weight.
  - While G_w has a circuit
    - Take such a circuit C (which must have even length).
    - Split C into two matchings M and N, with $w(M) \geq w(N)$.
    - Increase the weight of each edge in M by 1.
    - Decrease the weight of each edge in N by 1.

On the algorithm

- Let each vertex have degree *d*.
- Invariant: the sum of the weights of the incident edges of a vertex is *d*.
- At termination: no circuit in G_w, and by the invariant, it follows G_w must be a perfect matching.

Time to find circuits

- Finding circuits:
  - Keep a path P with edges of weight between 1 and $d - 1$
  - Let v be last vertex on P.
  - v must have edge not on P with weight between 1 and $d - 1$, say $\{v,x\}$.
  - If x on P: we have a circuit.
    - Apply step on circuit.
    - Remove circuit from P, and work with smaller path.
  - Otherwise, add $\{v,x\}$ to P, and repeat
- O(|C|) per circuit, plus $O(n+m)$ additional overhead.

Time analysis

- Look at the sum over all edges of $w(e)^2$.
- Each improvement over a cycle C increases this sum by at least $|C|$.
- Initially m, never larger than nd^2.
- So, total time $O(nd^2) = O(dm)$.

Sum of squares increases by |C|

$$\sum_{e\in M}\left(w(e)+1\right)^2+\sum_{e\in N}\left(w(e)-1\right)^2=$$

$$\sum_{e\in M}\left(w(e)\right)^2+\sum_{e\in N}\left(w(e)\right)^2+2\sum_{e\in M}w(e)-2\sum_{e\in N}w(e)+|M\cup N|\geq$$

$$\sum_{e\in M}\left(w(e)\right)^2+\sum_{e\in N}\left(w(e)\right)^2+|M\cup N|$$

5

An application of matching in regular bipartite graphs:

Edge coloring and classroom schedules

Universiteit Utrecht

Edge coloring and classroom schedules

- Teachers
- Class
- Some teachers should teach some classes but:
  - No teacher more than one class at a time
  - No class more than one lesson at the time
  - How many hours needed???

	Jansen	Petersen	Klaassen
1b	X		
2a			X
2b		X	

	Jansen	Petersen	Klaassen
1b	X	1-2	2-3
2a	1-2	2-3	X
2b	2-3	X	1-2

Edge coloring model

- Take bipartite graph with vertices for teachers and for classes
- Look for a coloring of the edges such that no vertex has two incident edges with the same color.
- What is the minimum number of colors needed?
  - Lower bound: maximum degree. (Interpretation!)
  - We can attain the lower bound with help of matchings!!

A theorem

- Let G be a bipartite graph with maximum degree *d*. Then G has an edge coloring with *d* colors.
  - Step 1: Make G regular by adding vertices and edges.
  - Step 2: Repeatedly find a matching and remove it.

Making G regular

- Suppose G has vertex sets V_1 and V_2 with edges only between V_1 and V_2. (Usually called: *color classes*).
- If $|V_1| > |V_2|$ then add $|V_1| - |V_2|$ isolated vertices to $|V_2|$.
- If $|V_2| > |V_1|$ then add $|V_2| - |V_1|$ isolated vertices to $|V_1|$.
- While not every vertex in $V_1 \cup V_2$ has degree d:
  - Find a vertex v in V_1 of degree less than d ← *Must exist*
  - Find a vertex w in V_2 of degree less than d ←
  - Add the edge $\{v,w\}$ to the graph.

Edge coloring a regular graph

- Say G’ is regular of degree *d*.
- For *i* = 1 to *d* do
  - Find a perfect matching M in G’.
  - Give all edges in M color *i*.
  - Remove all edges in M from G’. (Note that G’ stays regular!)

Final step

- Take the edge coloring c of G'. Color G in the same way: G is subgraph of G.
- Time: carrying out *d* times a perfect matching algorithm in a regular graph:
  - O(nd^3) if we use Schrijver's algorithm.
  - Can be done faster by other algorithms.

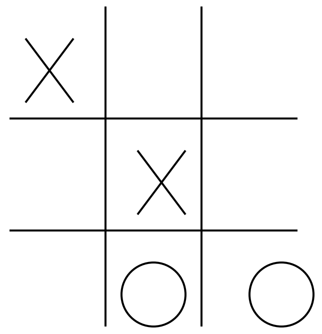
6

Diversion: multidimensional tic-tac-toe

Universiteit Utrecht

Trivial drawing strategies in multidimensional tic-tac-toe

- Tic-tac-toe
- Generalizations
  - More dimensions
  - Larger board size
- Who has a winning strategy?
  - Either first player has winning strategy, or second player has drawing strategy



Trivial drawing strategy

- If lines are long enough: pairing of squares such that each line has a pair
- If player 1 plays in a pair, then player 2 plays to other square in pair

v	i	a	a	f
j	b	h	u	b
c	i		g	c
d	u	h	d	f
j	e	e	g	v

Trivial drawing strategies and generalized matchings

- Bipartite graph: line-vertices and square-vertices; edge when square is part of line
- Look for set of edges M, such that:
  - Each line-vertex is incident to two edges in M
  - Each square-vertex is incident to at most one edge in M
- There exists such a set of edges M, if and only if there is a trivial drawing strategy (of the described type).

Consequences

- Testing if trivial drawing strategy exists and finding one if so can be done efficiently (flow algorithm).
- *n* by *n* by …by *n* tic-tac-toe (*d*-dimensional) has a trivial drawing strategy if *n* is at least $2*3^d-1$
  - A square belongs to at most 3^d-1 lines.
  - Results from matching theory (see next dia) can be used

Technicalities

- Take a graph with two vertices per line, and one vertex per square, with an edge if the square belongs to the line
- Max degree is max(n, $2*3^d-1$)
- This graph has a matching that matches all line vertices – gives the desired strategy
  - Proof: add edges and vertices to make a regular bipartite graph of degree n and take a perfect matching in this graph

7

Conclusions

Universiteit Utrecht

Conclusion

- Many applications of matching! Often bipartite…
- Algorithms for finding matchings:
  - Bipartite: flow models
    Bipartite, regular: Schrijver
  - General: with M-augmenting paths and blossom-shrinking
- Minimum cost matching can also be solved in polynomial time: more complex algorithm
  - Min cost matching on bipartite graphs is solved using min cost flow