# GIT

# Graphs

A. Ada, K. Sutner
Carnegie Mellon University

Spring 2018



## Boruvka's Problem



> Suppose you need to provide electricity to a number of households. For financial reasons, only some of the houses can be connected by a wire, and the cost of building these connections varies.
> What is the least cost associated with a network that connects all the households?

This problem was first tackled by Otakar Boruvka in 1926, in a proposal to construct an efficient electricity network for Bohemia.

Needless to say, he was way ahead of his time.

## MSTs 

To model Boruvka's problem we can use a connected ugraph $G = \langle V, E \rangle$ whose vertices represent the locations and whose edges represent the potential links. Moreover, we attach a cost to each edge, a map cost : $E \to \mathbb{R}_+$ .

We want to construct a spanning tree $T = \langle V, T \rangle$ (slight abuse of notation, but very elegant) that minimizes

$$\text{cost}(T) = \sum_{e \in T} \text{cost}(e)$$

## There Are Lots . . . 

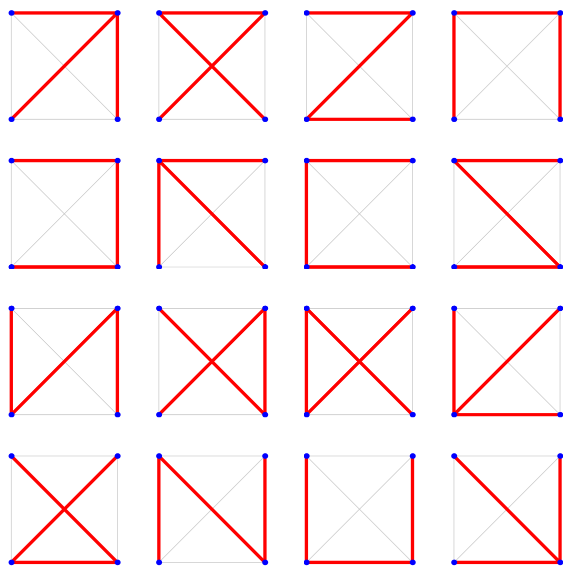
In general, the number of spanning trees of a graph is large.

**Theorem (Cayley)**

*The complete graph $K_n$ has $n^{n-2}$ spanning trees.*

**Theorem**

*The complete bipartite graph $K_{n,m}$ has $n^{m-1}m^{n-1}$ spanning trees.*



## Basic Strategy 

Since there are many potential trees, we cannot do anything resembling a brute force search.

Instead, do the biologically natural thing:

> Grow the tree in stages.

We start with an empty tree, or a single node tree or some such. Then we add edges until the MST emerges. So the real question is:

> How should we choose the next edge?

A fair guess would be to always pick a cheap edge.

## Cheap Edges 

### Proposition

*Let $e$ be a minimal cost edge. Then there is a MST containing $e$.*

*Proof.* Can produce a new MST be swapping edges:

$$T' = T + e - e'$$

where $e'$ is an edge on the cycle introduced by adding $e$. □

This swapping trick may seem trivial, but it is actually the foundation for an important topic in combinatorics: matroids.

## Growing a Forest 

A spanning forest is a collection of vertex-disjoint trees $T_i = \langle V_i, T_i \rangle$ such that $\bigcup V_i = V$.

Here is the key observation regarding spanning forests. The proof is almost exactly the same as for the last proposition.

### Lemma (Extension Lemma)

*Let $e$ be a minimal cost edge not introducing a cycle in a given forest. Then there is a spanning tree containing $e$ that has cost minimal in the class of all spanning trees containing the forest.*

## Greedy Wins 

This opens the door for greedy algorithms: keep adding cheap edges till the tree is complete. Initially we are dealing with a trivial tree/forest, so the class of extensions consists of all spanning trees. Hence we are dealing with a bonified MST.

Think of this as an edge-coloring game: initially all edges are white. We will color edges blue (added to tree) or red (permanently barred from the tree) while maintaining the following invariant:

> There is a MST containing all blue edges, but none of the red edges.

In other words, we have not made a mistake yet.

Good enough for CS.

## Prim's Algorithm 1957 

Sometimes called the *nearest neighbor algorithm*.

Works by choosing an arbitrary vertex $r$ as a root, and the growing a tree $T$ (non-spanning as yet) from there. Keep extending tree by single minimal cost edges until tree is spanning.

```
initialize tree T to r
while( T is not spanning )
       select cheapest edge e extending T
       add e to T
```

Blue edges: the ones chosen to extend $T$.
Red edges: the ones that would introduce cycles in $T$.

## Running Time 

Data structures: Need easy access to the next cheapest edge. Use a priority queue for the vertex complement of $T$, where the key is distance information: minimal distance to $T$.

Note that this looks very similar to Dijkstra's shortest path algorithm. Unsurprisingly, Dijkstra's also discovered Prim's algorithm, but in 1959.

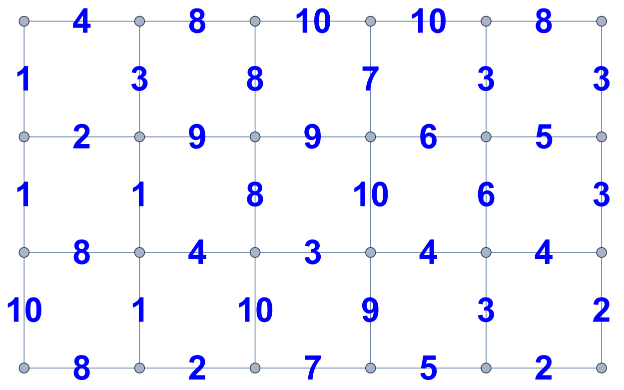
### Theorem

*Using a standard priority queue, the running time of Prim's algorithm is* $O(m \lg n)$.

Can be improved by Fibonacci heaps to $O(m + n \lg n)$.

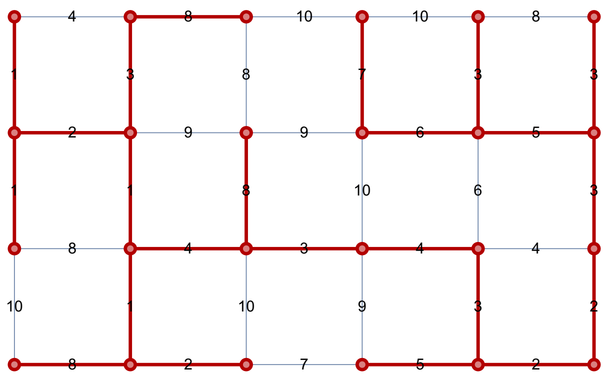
## Grid






## Grid MST






## Kruskal's Algorithm 1956



Works by starting with a trivial spanning forest consisting of $n$ one-point trees. Keep extending forest by adding a minimal cost edge that connects two trees in the forest.

More precisely, do the following:

```
sort edges by cost
initialize forest F to V

foreach edge e in E do            // in order of cost
    if( e creates no cycle )
        add e to F, merge two trees
```

Blue edges: the ones chosen to extend the forest.
Red edges: the ones that would introduce cycles in $T$.

## Implementation



Correctness follows from the Extension Lemma.

How about efficiency?

We have to sort the list of edges according to their weights and keep them in an array which takes $O(m \lg n)$ steps. Then we traverse the array.

The question now is: how hard is it to check if an edge $e$ connects two separate trees (or introduces a cycle in one tree).

This problem can be handled essentially in time linear in the number of queries and merges using a so-called Union/Find data structure. So we have

### Theorem

*The running time of Kruskal's algorithm is $O(m \lg n)$.*

## Demo



Greedy MST Demo

## And Boruvka?



The idea behind Boruvka's algorithm is this:

- Initialize a trivial spanning forest $F$.
- Determine a minimal cost protruding edge for each tree in $F$.
- Add these edges to $F$, with caution.
- Repeat.

The reason this is interesting is because it parallelizes nicely: we can search for the minimal cost protruding edges in parallel for each tree in the forest.

- Minimum Spanning Trees

2 Matchings

- Tutte Matrix

- Planarity

## Notation



Let $G$ be a simple ugraph.

- $\Gamma(x) = \{ y \mid \{x, y\} \in E \}$ denotes the (open) neighborhood of $x \in V$. Similarly write $\Gamma(X)$ for $X \subseteq V$.
- Similarly write $\Gamma_+(x) = \Gamma \cup \{x\}$ for the closed neighborhood.

## Perfect Matchings



Let $G = \langle V, E \rangle$ be a ugraph.

### Definition

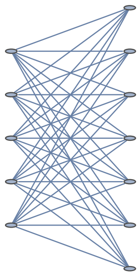
A matching for $G$ is a set $M \subseteq E$ such that every node in the subgraph $\langle V, M \rangle$ has degree at most $1$.

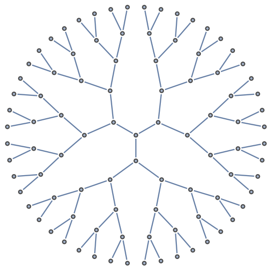
A perfect matching for $G$ is a set $M \subseteq E$ such that every node in the subgraph $\langle V, M \rangle$ has degree exactly $1$.

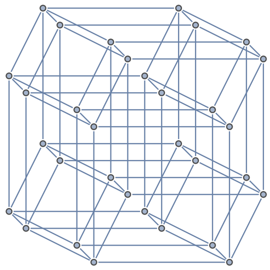
We focus on the case where $G$ is bipartite: there is a partition of the vertex set $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, so that all edges go from $V_1$ to $V_2$.

It is convenient to write $G[V_1, V_2]$ to indicate the partition of the vertex set ($V_1$ is "left", $V_2$ is "right").

## Example 







Note that a graph is bipartite iff we can 2-color its vertices.

## Resource Allocation 

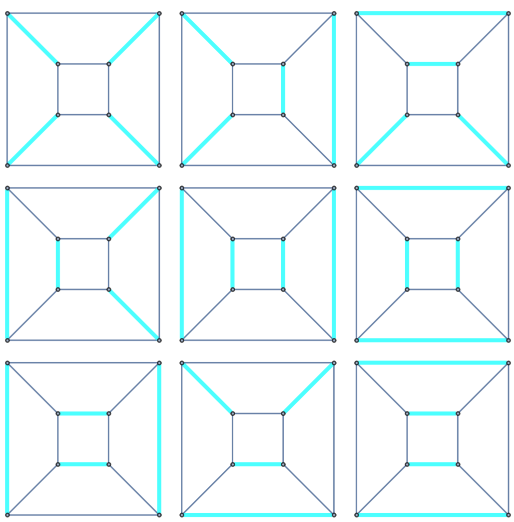
Think of $V_1$ as a collection of resources, and of $V_2$ as a collection of tasks. We would like to allocate one resource to each task, ideally exhausting all resources and handling all tasks.

Of course, this can only work if $|V_1| = |V_2|$: the graph must have even cardinality and be split in the middle.

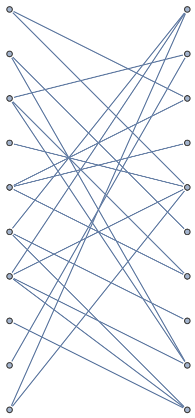
Note that a matching in a bipartite graph is essentially a partial bijection $V_1 \longleftrightarrow V_2$.

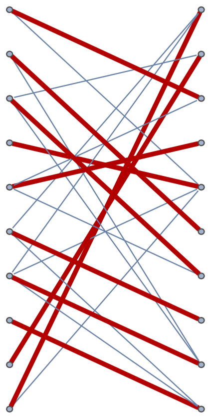
A perfect matching produces an actual bijection.

## All Cubical Perfect Matchings 



## A Random Perfect Matching



## Bipartite Graphs


### Proposition

*A graph is bipartite iff it has no odd-length cycles.*

*Proof.* $\Rightarrow$ is obvious, for $\Leftarrow$ we may safely assume that the graph is connected.

Pick an anchor point $v$ in $G$ and color it blue.

Then color the neighbors of $v$ red, the neighbors of these neighbors blue, and so on.

There will never be a clash: otherwise we would have an odd-length cycle.

$\square$

## Hall's Theorem


Since a perfect matching in a bipartite graph $G[V_1, V_2]$ is a bijection, we must have

$$|U| \leq |\Gamma(U)| \qquad (*)$$

for every $U \subseteq V_1$.

### Theorem (Hall's Theorem 1935)

*A bipartite graph $G[V_1, V_2]$ has a perfect matching iff condition $(*)$ holds for all $U \subseteq V_1$.*

## Proof I 

Assume that for all $U \subsetneq V_1$ we have the stronger condition

$$|U| < |\Gamma(U)|$$

Pick an edge $e = \{u, v\}$ and let $G' = G - u, v$.

Then $(*)$ still holds for $G'$ and by IH $G'$ has a perfect matching $M'$.

Add $e$ to $M'$ to get a perfect matching $M$ for $G$.

## Proof II 

Assume that for some $U \subsetneq V_1$ we are in the critical case

$$|U| = |\Gamma(U)|$$

Let $G'$ be the subgraph $G[U, \Gamma(U)]$ and $G''$ the subgraph $G[\overline{U}, \overline{\Gamma(U)}]$.

A moment's thought shows that both $G'$ and $G''$ satisfy $(*)$.

By IH we have two perfect matchings $M'$ and $M''$ which can be combined to a perfect matching $M$ for $G$.

$\square$

**Exercise**

*Think for a moment and draw some pictures.*

## Standard Application 

Suppose you split a deck of cards into 13 piles of size 4 each. Then one can pick one card from each pile to get one card from each rank.

To see why, consider $G[[13], [13]]$ where the vertices on the left represent the 13 piles and the vertices on the right represent the 13 ranks. Place an edge if the pile contains a card of that rank.

Each vertex has degree 4, so if we pick a set $U$ of piles on the left we have

$$|\Gamma(U)| \geq (\#\text{edges in neighborhood})/4 = 4|U|/4 = |U|$$

**Exercise**

*There is a slight bug in the proof. Exterminate it.*

## Algorithm?



Note that the proof of Hall's theorem is perfectly constructive: it shows how to build $M$ from smaller matchings on subgraphs.

Alas, it's exponential: we have to check the condition on arbitrary subsets $U \subsetneq V_1$.

That's better than doing a brute-force search over subsets of $E$, but not by much.

> **Real Question:** Is there a fast algorithm to find a perfect matching (or refute its existence)?

## Augmenting Paths



Suppose we have a matching $M$ in $G[V_1, V_2]$.

An alternating path is a path whose edges alternate between $M$ and $\overline{M}$.

An augmenting path is an alternating path whose source and target are unmatched.


A simple trick: swap the edges along the path in and out of $M$. This increases the size of the matching by 1.

So we can go on until we run out of augmenting paths.

## Petersen-König-Berge Lemma



### Lemma

*Suppose we have a matching $M$ in $G[V_1, V_2]$. Then there is a larger matching iff $M$ has an augmenting path.*

*Proof.*

First consider two arbitrary matchings $M_1$ and $M_2$. Let $E' \subseteq E$ be their symmetric difference.

Then the connected components of subgraph $G[V_1, V_2; E']$ are

- isolated points
- paths
- even length cycles

To see why note that all vertices in the subgraph have degree at most 2.

But then $|M_1| < |M_2|$ implies that at least one component must be a path.

Moreover, that path must be augmenting for $M_1$. □

Note that we can find an augmenting path by a modified version of BFS.

So the total running time is $O(nm) = O(n^3)$. There are better algorithms, but they are considerably more complicated.

### Exercise

*Implement the matching algorithm for bipartite graphs.*

## General Graphs



One would suspect that a similar algorithm should also work for general graphs, but there are several technical problems to deal with.

J. Edmonds
Paths, Trees and Flowers
Canad. J. Math. 17 (1965), 449-467.

This paper is particularly important, since it was one of the first to introduce the idea that polynomial time is a good model for feasible computation.

Of course, Gödel thought about this 10 years earlier.

## Tutte Matrix



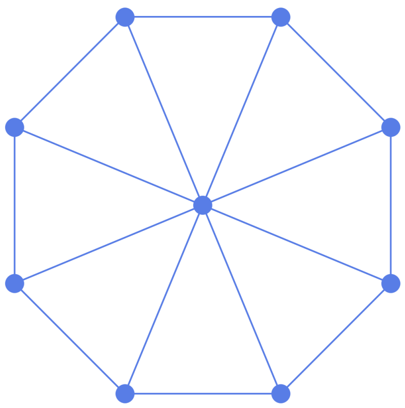
Suppose $G = \langle [n], E \rangle$ is a ugraph. Define its Tutte matrix by

$$T(i,j) = \begin{cases} x_{ij} & \text{if } ij \in E \text{ and } i < j, \\ -x_{ji} & \text{if } ij \in E \text{ and } i > j, \\ 0 & \text{otherwise.} \end{cases}$$

The determinant of this matrix is a polynomial with up to $n^2$ variables $x_{ij}$ and can be computed in polynomial time.

## 9 Wheel





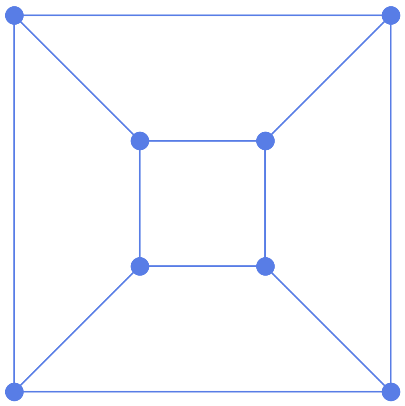
## Wheel Matrix



$$\begin{pmatrix}
0 & x_{1,2} & 0 & 0 & 0 & 0 & 0 & x_{1,8} & x_{1,9} \\
-x_{1,2} & 0 & x_{2,3} & 0 & 0 & 0 & 0 & 0 & x_{2,9} \\
0 & -x_{2,3} & 0 & x_{3,4} & 0 & 0 & 0 & 0 & x_{3,9} \\
0 & 0 & -x_{3,4} & 0 & x_{4,5} & 0 & 0 & 0 & x_{4,9} \\
0 & 0 & 0 & -x_{4,5} & 0 & x_{5,6} & 0 & 0 & x_{5,9} \\
0 & 0 & 0 & 0 & -x_{5,6} & 0 & x_{6,7} & 0 & x_{6,9} \\
0 & 0 & 0 & 0 & 0 & -x_{6,7} & 0 & x_{7,8} & x_{7,9} \\
-x_{1,8} & 0 & 0 & 0 & 0 & 0 & -x_{7,8} & 0 & x_{8,9} \\
-x_{1,9} & -x_{2,9} & -x_{3,9} & -x_{4,9} & -x_{5,9} & -x_{6,9} & -x_{7,9} & -x_{8,9} & 0
\end{pmatrix}$$

This matrix has determinant 0.

## Cube

## Cube Matrix


$$\begin{pmatrix}
0 & x_{1,2} & x_{1,3} & 0 & x_{1,5} & 0 & 0 & 0 \\
-x_{1,2} & 0 & 0 & x_{2,4} & 0 & x_{2,6} & 0 & 0 \\
-x_{1,3} & 0 & 0 & x_{3,4} & 0 & 0 & x_{3,7} & 0 \\
0 & -x_{2,4} & -x_{3,4} & 0 & 0 & 0 & 0 & x_{4,8} \\
-x_{1,5} & 0 & 0 & 0 & 0 & x_{5,6} & x_{5,7} & 0 \\
0 & -x_{2,6} & 0 & 0 & -x_{5,6} & 0 & 0 & x_{6,8} \\
0 & 0 & -x_{3,7} & 0 & -x_{5,7} & 0 & 0 & x_{7,8} \\
0 & 0 & 0 & -x_{4,8} & 0 & -x_{6,8} & -x_{7,8} & 0
\end{pmatrix}$$

This matrix has determinant

$$(x_{1,5}(x_{2,4}x_{3,7}x_{6,8} + x_{2,6}(-x_{3,7}x_{4,8} + x_{3,4}x_{7,8})) + x_{1,2}(x_{3,7}x_{4,8}x_{5,6}+ x_{3,4}(x_{5,7}x_{6,8}-x_{5,6}x_{7,8}))+x_{1,3}(x_{2,6}x_{4,8}x_{5,7}+x_{2,4}(-x_{5,7}x_{6,8}+x_{5,6}x_{7,8})))^2$$

## Tutte's Theorem


### Theorem (Tutte 1947)

*$G$ has a perfect matching iff its Tutte matrix has non-zero determinant.*

Note that these matrices are size $n \times n$ for a graph on $n$ points. Also, the entries are symbolic, so computing the determinant is a little tricky.

**Full Disclosure:** The real reason this is important is that there is a fast probabilistic zero check for multivariate polynomials (see Schwartz-Zippel Lemma).

## Proof Sketch



The determinant has the form

$$|T| = \sum_{\pi \in \mathfrak{S}_n} \pm\,\mathsf{sign}(\pi)\, T_{1\pi(1)} T_{2\pi(2)} \ldots T_{n\pi(n)}$$

where $\mathfrak{S}_n$ is the symmetric group on $n$ points and sign the usual sign function ($-1$ raised to the number of inversions in the permutation).

If there is no perfect matching, then all the product terms are 0: they all involve at least one non-edge.

On the other hand, if the graph has a perfect matching, it must have the form

$$M = \{\, \{u_i, v_i\} \mid i \in [n/2] \,\}$$

Now define $\pi(u_i) = v_i$ and $\pi(v_i) = u_i$: then $\pi$ is a permutation consisting only of 2-cycles.

But then the determinant of $T$ cannot be identically 0, since the corresponding monomial in the sum cannot be canceled out: for another permutation to produce the same term (up to sign), it would need to be composed of the same 2-cycles.

□

## How Many?



The number of connected simple graphs with perfect matchings, on $2n$ nodes:

$$1, 5, 95, 10297, 11546911, \ldots$$

These numbers are not particularly interesting, but the OEIS is:

`http://oeis.org/A218463`

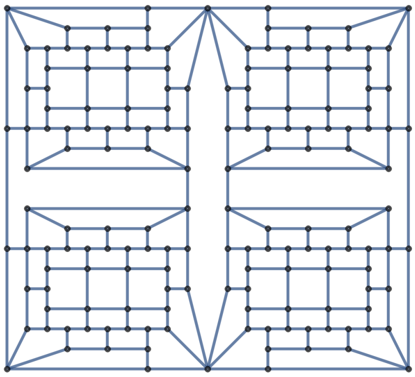
- Minimum Spanning Trees
- Matchings
- Tutte Matrix

4. Planarity

## Planar Graphs



Informally, a ugraph is planar if it can be drawn in the plane so that no edges cross.



## No Good



This "definition" is a disaster: it requires higher-order concepts from geometry.

Say $G = \langle V, E \rangle$ is our ugraph. For every edge $e \in E$ we want a (finite, non-self-intersecting) curve segment

$$\ell_e : [0,1] \longrightarrow \mathbb{R}^2$$

so that these segment overlap only at the endpoints, and only if the corresponding edges share vertices.

Remember, we are slum-dwellers, we don't understand the reals, much less planar curves.

## Simplification 

### Theorem (I. Fáry 1948)

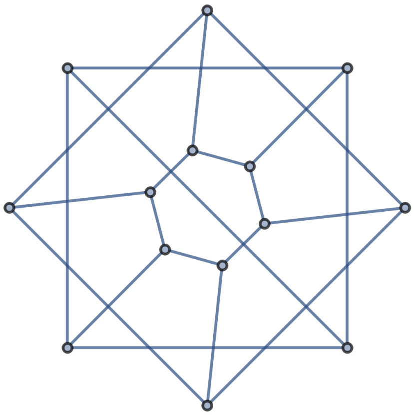
*Every planar graph admits an embedding using only straight line segments.*

Line segments are rather simple objects, planarity thus comes down to just a few linear equations over the reals.

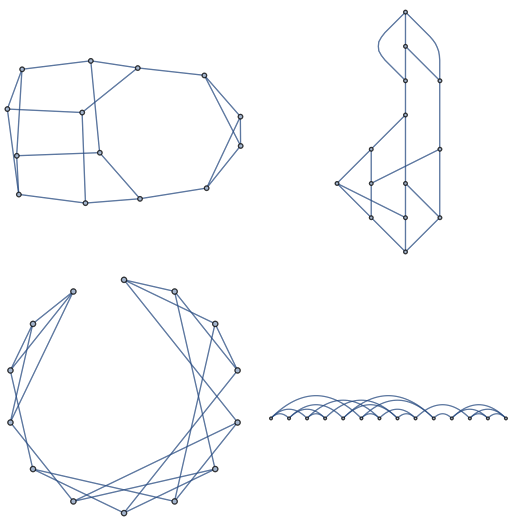
One can even insist to place the vertices on the integer grid, so there is no problem with complicated endpoints.

Still, it is far from clear how to check whether a given graph is planar.

## A Cubic Graph 

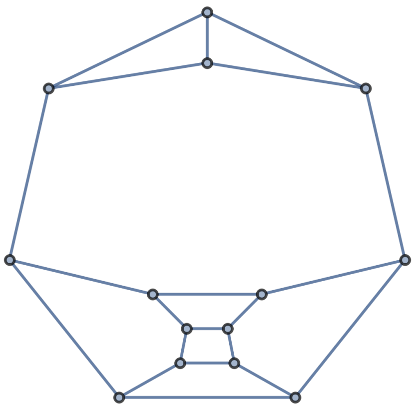


## Various Embeddings 



## A Planar Embedding





## But How?



Just to be clear, this graph would be given by, say, an edgelist:

$$1:2, 1:3, 1:4, 2:3, 2:4, 3:5, 4:6, 5:7, 5:8, 6:9, 6:10, 7:9,$$
$$7:11, 8:10, 8:12, 9:13, 10:14, 11:12, 11:13, 12:14, 13:14$$

One can do a little weeding out based on the following result:

### Proposition

*Let $G$ be finite, planar and connected, $v/e$ the number of vertices/edges, respectively, and $v \geq 3$. Then $e \leq 3v - 6$, and the average degree is less than $6$.*

## Proof Sketch



Let $f$ be the number of faces of $G$ (including the infinite, outer face).

Recall Euler's famous formula:

$$v - e + f = 2$$

Generically, every edges touches 2 faces, and every face touches at least three edges (but beware of degenerate cases). Hence $3f \leq 2e$ and our claim follows.

So planar graphs are quite sparse, but that's nowhere near enough.

## Linear Planarity Testing 

The following result is a small miracle, and took quite a bit of time to assemble from weaker results.

### Theorem (Hopcroft, Tarjan 1974)

*One can check in linear time whether a graph is planar (and construct an embedding if the answer is yes).*

The idea is to start with a partial embedding, and extend it gradually to a total one.

## Minors 

We can generalize the notion of a subgraph as follows.

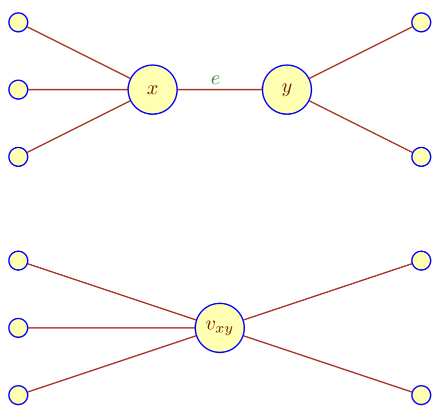
### Definition

A graph $H$ is a minor of $G$ if it can be obtained from $G$ by a sequence of

| | |
|---|---|
| vertex removals | Remove an isolated vertex. |
| edge removals | Remove an edge. |
| edge contractions | Remove an edge $xy$, introduce a new vertex $v$ and connect it to the neighbors of $x$, $y$ (kill multiple edges). |

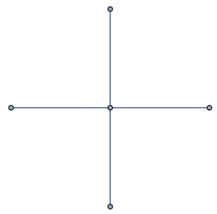
This is not the most elegant description, but easy to understand.

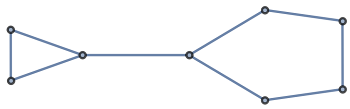
## Edge Contraction 



## Star and Target

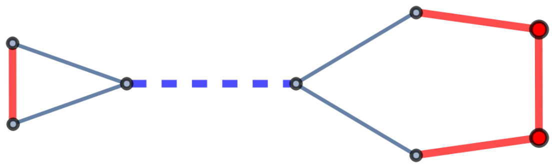






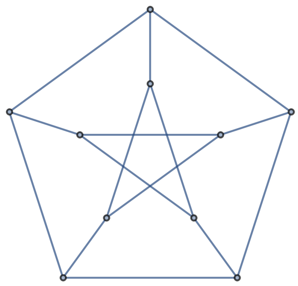
## Operations

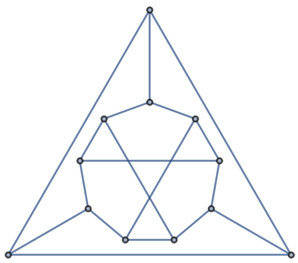




## Petersen-Tietze







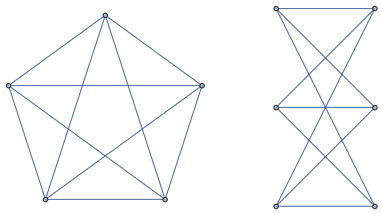
Checking by hand whether a graph is minor of another is quite tedious.
Try it for the Petersen and Tietze graphs.

# A Weird Planarity Test



Theorem (Kuratowski 1930, Wagner 1935)

*A graph is planar iff it does not contain a $K_5$ or a $K_{3,3}$ as a minor.*



Robertson and Seymour showed that for fixed $H$ one can check whether $H$ is a minor of $G$ in cubic time.