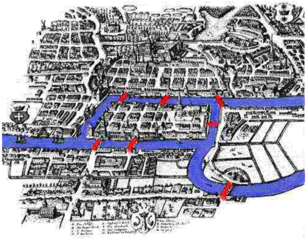
Graph Theory and Applications

Paul Van Dooren
Université catholique de Louvain
Louvain-la-Neuve, Belgium

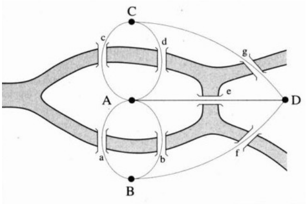
Dublin, August 2009

Inspired from the course notes of V. Blondel and L. Wolsey (UCL)

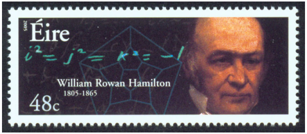
Graph theory started with Euler who was asked to find a nice path across the seven Köningsberg bridges



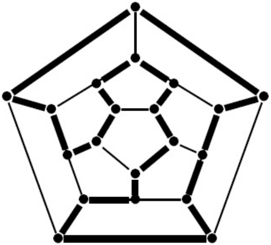
The (Eulerian) path should cross over each of the seven bridges exactly once



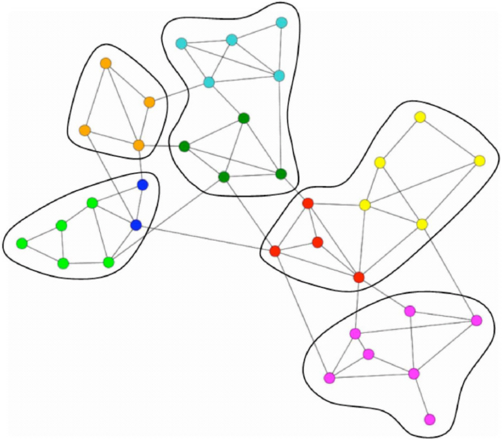
Another early bird was Sir William Rowan Hamilton (1805-1865)



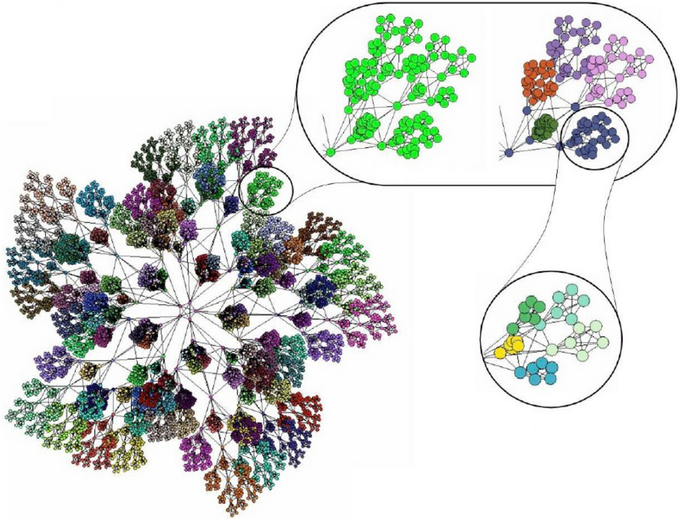
In 1859 he developed a toy based on finding a path visiting all cities in a graph exactly once and sold it to a toy maker in Dublin. It never was a big success.



But now graph theory is used for finding communities in networks



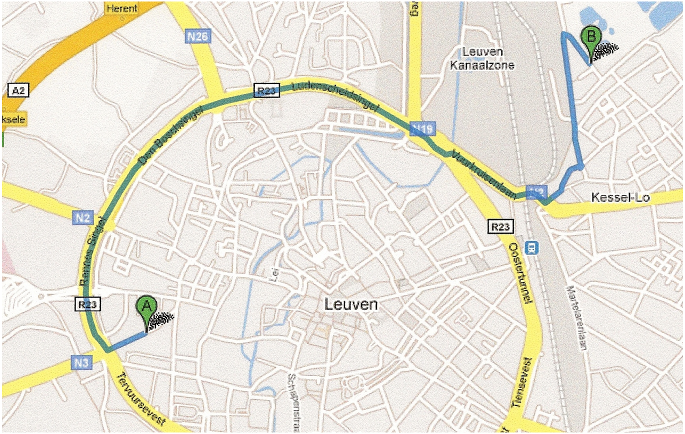
where we want to detect hierarchies of substructures and their sizes can become quite big ...



It is also used for ranking (ordering) hyperlinks

or by your GPS to find the shortest path home ...

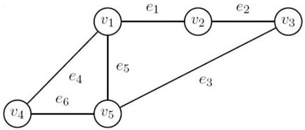
or by your GPS to find the shortest path home ...



What we will cover in this course

- Basic theory about graphs
  - Connectivity
  - Paths
  - Trees
  - Networks and flows
  - Eulerian and Hamiltonian graphs
  - Coloring problems
  - Complexity issues
- A number of applications (in large graphs)
  - Large scale problems in graphs
  - Similarity of nodes in large graphs
  - Telephony problems and graphs
  - Ranking in large graphs
  - Clustering of large graphs

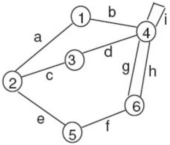
A graph $G = (V, E)$ is a pair of vertices (or nodes) V and a set of edges E, assumed finite i.e. $|V| = n$ and $|E| = m$.

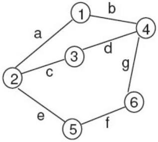


Here $V(G) = \{v_1, v_2, \ldots, v_5\}$ and $E(G) = \{e_1, e_2, \ldots, e_6\}$.

An edge $e_k = (v_i, v_j)$ is incident with the vertices v_i and v_j.

A simple graph has no self-loops or multiple edges like below





Some properties

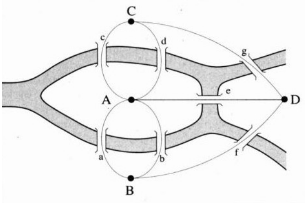
The degree $d(v)$ of a vertex V is its number of incident edges

A self-loop counts for 2 in the degree function.

An isolated vertex has degree 0.

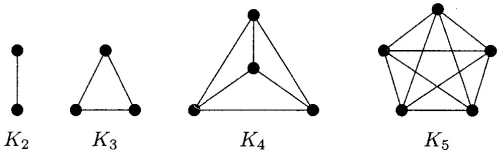
Proposition The sum of the degrees of a graph $G = (V, E)$ equals $2|E| = 2m$ (trivial)

Corollary The number of vertices of odd degree is even (trivial)

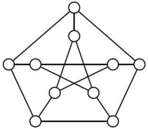


Special graphs

A complete graph K_n is a simple graph with all $B(n, 2) := \frac{n(n-1)}{2}$ possible edges, like the matrices below for $n = 2, 3, 4, 5$.

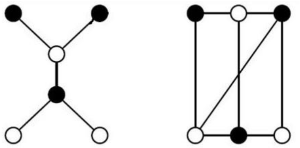


A *k*-regular graph is a simple graph with vertices of equal degree k

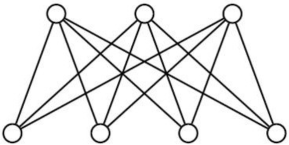


Corollary The complete graph K_n is $(n-1)$-regular

A bipartite graph is one where $V = V_1 \cup V_2$ such that there are no edges between V_1 and V_2 (the black and white nodes below)



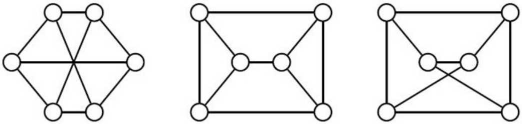
A complete bipartite graph is one where all edges between V_1 and V_2 are present (i.e. $|E| = |V_1|.|V_2|$). It is noted as K_{n_1,n_2}.



When is complete bipartite graph regular ?

When is G bipartite ?

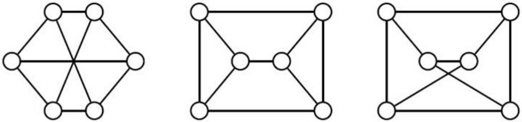
Which graph is bipartite ?



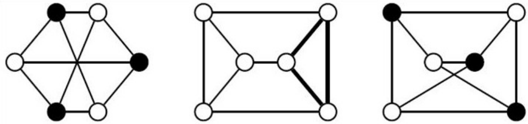
It suffices to find 2 colors that separate the edges as below

When is G bipartite ?

Which graph is bipartite ?



It suffices to find 2 colors that separate the edges as below



The second example is not bipartite because it has a triangle

(to be continued)

Walking in a graph

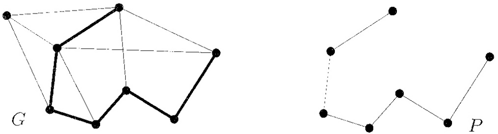
A walk of length k from node v_0 to node v_k is a non-empty graph $P = (V, E)$ of the form

$$V = \{v_0, v_1, \ldots, v_k\} \quad E = \{(v_0, v_1), \ldots, (v_{k-1}, v_k)\}$$

where edge j connects nodes $j - 1$ and j (i.e. $|V| = |E| + 1$).

A trail is a walk with all different edges.

A path is a walk with all different nodes (and hence edges).



A walk or trail is closed when $v_0 = v_k$.

A cycle is a walk with different nodes except for $v_0 = v_k$.

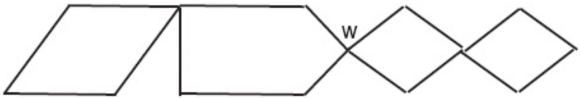
Try to prove the following wo (useful) lemmas

Proposition A walk from u to $v \neq u$ contains a path from u to v

Hint : eliminate subcycles

Proposition A closed walk of odd length contains a cycle of odd length

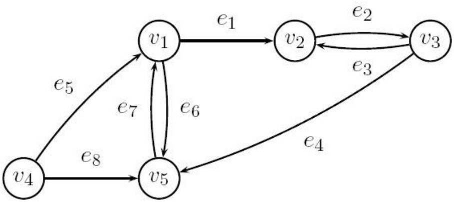
Hint : decompose recursively into distinct subgraphs and use induction



Question Is this only for simple graphs ?

Directed graphs

In a directed graph or digraph, each edge has a direction.



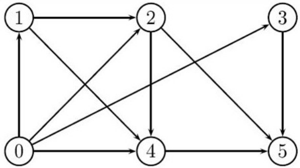
For $e = (v_s, v_t)$, v_s is the source node and v_t is the terminal node.

Each node v has an in-degree $d_{in}(v)$ and an out-degree $d_{out}(v)$.

A graph is balanced if $d_{in}(v) = d_{out}(v)$ for all nodes.

Topological order

Let us now try to order the nodes in a digraph.



Define a bijection $f_{ord} : V \to \{1, 2, \ldots, n\}$, then $f_{ord}(\cdot)$ is a topological order for the graph $G = (V, E)$ iff

$$f_{ord}(i) < f_{ord}(j), \quad \forall (i, j) \in E$$

This is apparently possible for the above graph.
It is easy to see that such a graph should have no cycles.

But is this also sufficient ?

An acyclic graph is a graph without cycles.

Proposition
Every acyclic graph contains at least one node with zero in-degree

Proof By contradiction.

Assume $d_{in}(v) > 0$ for all nodes, then each node i has a predecessor $p(i)$ such that $(v_{p(i)}, v_i) \in E$.

Start from an arbitrary v_0 to form a list of predecessors as below

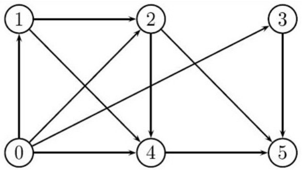
$$\begin{tikzcd} v_0 \arrow[r, bend left] & v_1 = p(v_0) \arrow[r, bend left] & v_2 = p(v_1) \end{tikzcd}$$

Since $|V|$ is bounded, one must eventually return to a node that was already visited; hence there is a cycle.

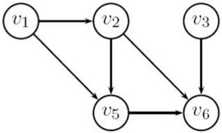
Let us use this to find a topological order

Algorithm FindTopOrd(G)
$t := 0; G^0 := G;$
while $\exists v \in G^t : d_{in}(v) = 0$ **do**
$G^{t+1} := G^t / \{v\}; order(v) := t + 1; t := t + 1;$
end while
if $t = n$ **then** G is acyclic;
else if $t < n$ **then** G has a cycle; **end if**
end if

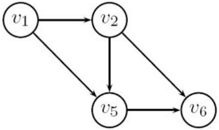
Let us verify this algorithm on the above example.



The only node of in-degree 0 is v_4. So for $t = 1$ we have



After removing v_4 there are two nodes of in-degree 0, v_1 and v_3. If we pick v_3 then we have for $t = 2$

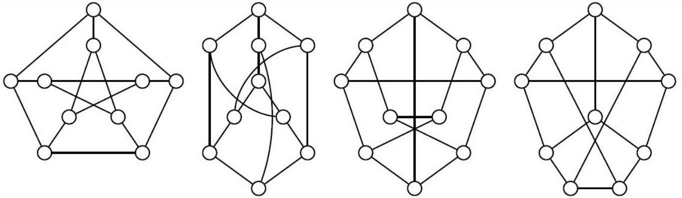


Further reductions yield the final order $\{v_4, v_3, v_1, v_2, v_5, v_6\}$.

What is the complexity of this algorithm ?

Isomorphic graphs

Two graphs G_1 and G_2 are isomorphic iff there is a bijection between their respective nodes which make each edge of G_1 correspond to exactly one edge of G_2, and vice versa.



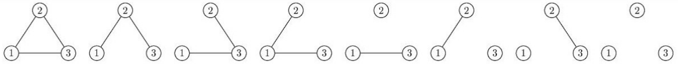
One must find a label numbering that makes the graphs identical

This problem is still believed to be NP hard

Counting graphs

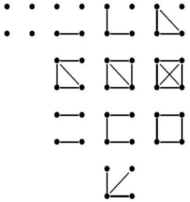
How many different simple graphs are there with n nodes ?

A graph with n nodes can have $B(n,2) := n(n-1)/2$ different edges and each of them can be present or not.



Hence there can be at most $2^{n(n-1)/2}$ graphs with n nodes.
For $n = 3$ only 4 of the graphs are different
(omitting the isomorphic ones)

With $n = 4$ one finds eventually 11 different graphs after collapsing the isomorphic ones



Let there be T_n non-isomorphic (simple) graphs with n nodes. Then

$$L_n := \frac{2^{n(n-1)/2}}{n!} \leq T_n \leq 2^{n(n-1)/2}$$

Exercise Explain the lower bound

Taking logarithms and using $n! < n^n$ yields the bounds

$$B(n,2) - n \log n \leq \log T_n \leq B(n,2)$$

which gives an idea of the growth of T_n

n	2	3	4	5	6	7	8
T_n	2	4	11	34	156	1044	12346
$\lceil L_n \rceil$	2	2	3	9	46	417	6658

Bipartite revisited

Let us look again at bipartite graphs

Proposition A graph is bipartite iff it has no cycles of odd length

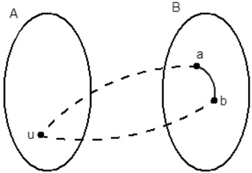
Necessity Trivial : color the nodes of the cycle black and white.

Sufficiency Pick $u \in V$ and let $f(v)$ be the length of a shortest path from u to v (∞ if there is no such path)

$$A = \{v \in V | f(v) = odd\} \quad B = \{v \in V | f(v) = even\}$$

Then A and B form a partition of the nodes of V connected to u.

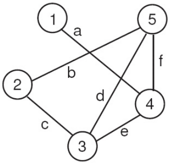
One then needs to show that there can be no links between any two nodes of A or any two nodes of B. If this would be the case, one could construct a cycle of odd length. Repeat on each subgraph.

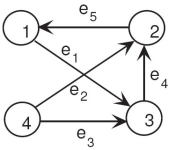


Representing graphs

A graph $G = (V, E)$ is often represented by its adjacency matrix.

It is an $n \times n$ matrix A with $A(i,j) = 1$ iff $(i,j) \in E$. For the graphs



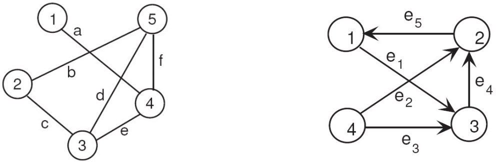


the adjacency matrices are

$$A_1 = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \quad A_2 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

A graph can also be represented by its $n \times m$ incidence matrix T.

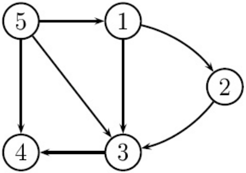
For an undirected graph $T(i,k) = T(j,k) = 1$ iff $e_k = (v_i, v_j)$.
For a directed graph $T(i,k) = -1$; $T(j,k) = 1$ iff $e_k = (v_i, v_j)$.
For the graphs



the incidence matrices are

$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \quad T_2 = \begin{bmatrix} -1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & -1 \\ 1 & 0 & 1 & -1 & 0 \\ 0 & -1 & -1 & 0 & 0 \end{bmatrix}$$

One can also use a sparse matrix representation of A and T. This is in fact nothing but a list of edges, organized e.g. by nodes.



$$\begin{aligned} V(1) &= \{2,3\} \\ V(2) &= \{3\} \\ V(3) &= \{4\} \\ V(4) &= \emptyset \\ V(5) &= \{1,3,4\} \end{aligned}$$

Notice that the size of the representation of a graph is thus linear in the number of edges in the graph (i.e. in $m = |E|$).

To be more precise, one should count the number of bits needed to represent all entries :

$$L = (n + m) \log n$$

since one needs $\log n$ bits to represent the vertex pointers.

Counting degrees

Let $\mathbf{1}$ be the vector of all ones, then $d_{in} = A^T\mathbf{1}$ and $d_{out} = A\mathbf{1}$ are the vectors of in-degrees and out-degrees of the nodes of A and $d_{out} = d_{in} = d$ for undirected graphs.

How should we then take self-loops into account ?
In an adjacency matrix of an undirected graph $A(i,i) = 2$
In an adjacency matrix of a directed graph $A(i,i) = 1$

For an undirected graph, we have $d = T\mathbf{1}$.
For a directed graph one can define T_t and T_s as the matrices containing the terminal and source nodes : $T = T_t - T_s$ with

$$T_t := \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, T_s := \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

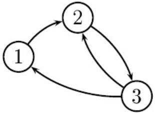
Then also we have $d_{in} = T_t\mathbf{1}$ and $d_{out} = T_s\mathbf{1}$.

Powers of A

Proposition $(A^k)_{ij}$ is the number of walks of length k from i to j

Proof Trivial for k=1; by induction for larger k.
The element (i, j) of $A^{k+1} = A^k \cdot A$ is the sum of the walks of length k to nodes that are linked to node j via the adjacency matrix A.

One verifies this in the following little example



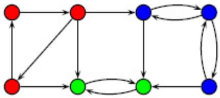
$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}, \quad A^2 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Corollary In a simple undirected graph one has the identities
$tr(A) = 0, tr(A^2)/2 = |E|$ and
$tr(A^3)/6$ equals the number of triangles in G.

Connected components

In a directed graph $G = (V, E)$, u and v are strongly connected if there exists a walk from u to v and from v to u.

This is an equivalence relation and hence leads to equivalence classes, which are called th connected components of the graph G.



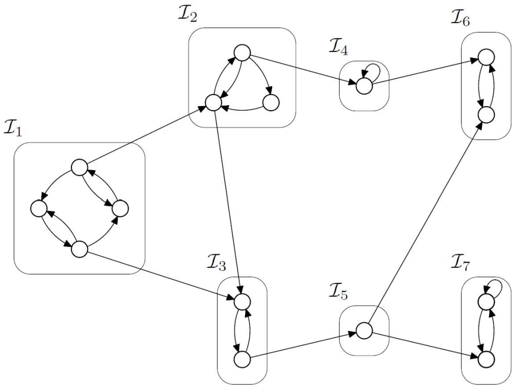
The graph reduced to its connected components is acyclic (why ?)

This shows up in many applications, e.g. in the dictionary graph.

The connected components are the groups of words that use each other in their definition (see later).

After the reduction one has an acyclic graph, which can be ordered topologically.

What do you obtain then ? Class orderings

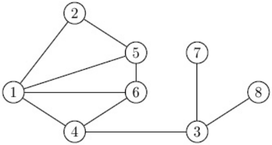


An initial class has $d_{in}(c) = 0$. A final class has $d_{out}(c) = 0$. The other ones are intermediate.

Verify (strong) connectivity of a graph based on its adjacency list

Idea : start from node s, explore the graph, mark what you visit

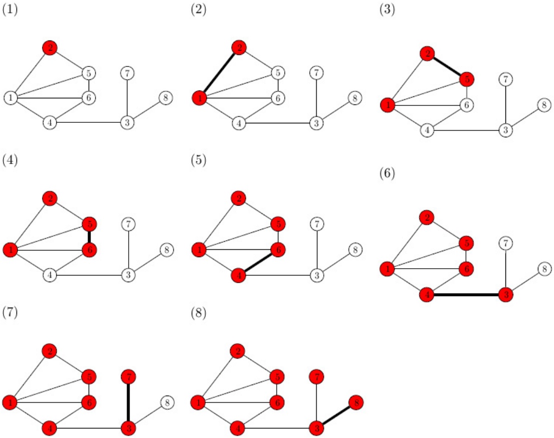
$V(1) = \{2, 4, 5, 6\}$
$V(2) = \{1, 5\}$
$V(3) = \{4, 7, 8\}$
$V(4) = \{1, 3, 6\}$
$V(5) = \{1, 2, 6\}$
$V(6) = \{1, 4, 5\}$
$V(7) = \{3\}$
$V(8) = \{3\}$



Algorithm GenericSearch(G,s)
mark(s); $L := \{s\}$
while $L \neq \emptyset$ **do**
    choose $u \in L$;
    if $\exists(u, v)$ such that v is unmarked **then**
        mark(v); $L := L \cup \{v\}$;
    else
        $L := L \backslash \{u\}$;
    end if
end while

Below we marked the chosen nodes and the discovered nodes

L	*mark*
$\{2\}$	2
$\{2, 1\}$	1
$\{2, 1, 5\}$	5
$\{2, 1, 5, 6\}$	6
$\{1, 5, 6\}$	
$\{1, 5, 6, 4\}$	4
$\{5, 6, 4\}$	
$\{5, 4\}$	
$\{5, 4, 3\}$	3
$\{5, 3\}$	
$\{5, 3, 7\}$	7
$\{5, 3\}$	
$\{3\}$	
$\{3, 8\}$	8
$\{3\}$	
$\{\}$	



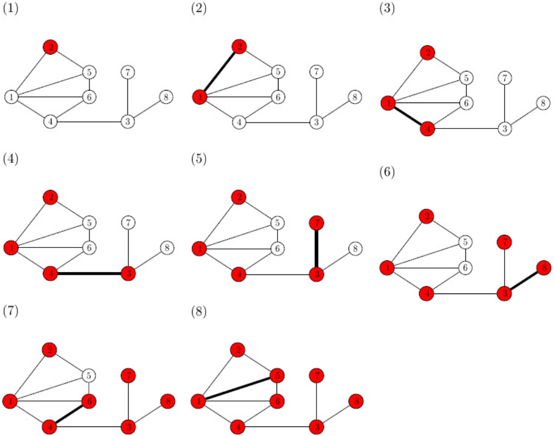
This algorithm has $2n$ steps : each node is added once and removed once. Its complexity is therefore linear in n.

Because of the choices, this algorithm allows for different versions
Let us use a LIFO list for L (Last In First Out) and choose for u the last element added to L. This is a depth first search (DFS).

Algorithm DeptFirstSearch(G,s)
mark(s); $L := \{s\}$;
while $L \neq \emptyset$ **do**
    $u := last(L)$
    if $\exists(u, v)$ such that v is unmarked **then**
        choose (u, v) with v of smallest index;
        mark(v); $L := L \cup \{v\}$;
    else
        $L := L \backslash \{u\}$
    end if
end while

Below we marked the chosen nodes and the discovered nodes

L	*mark*
$\{2\}$	2
$\{2, 1\}$	1
$\{2, 1, 4\}$	4
$\{2, 1, 4, 3\}$	3
$\{2, 1, 4, 3, 7\}$	7
$\{2, 1, 4, 3\}$	
$\{2, 1, 4, 3, 8\}$	8
$\{2, 1, 4, 3\}$	
$\{2, 1, 4\}$	
$\{2, 1, 4, 6\}$	6
$\{2, 1, 4, 6, 5\}$	5
$\{2, 1, 4, 6\}$	
$\{2, 1, 4\}$	
$\{2, 1\}$	
$\{2\}$	
$\{\}$	



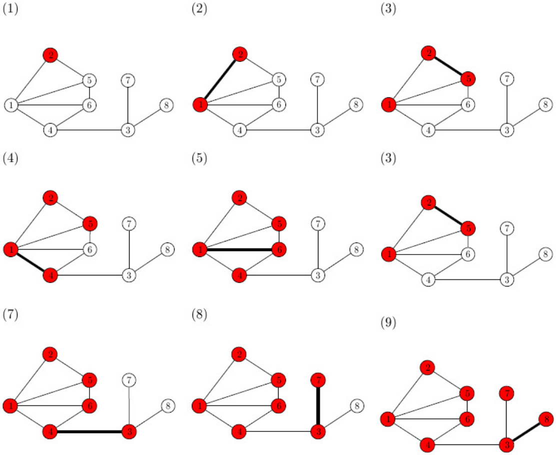
This algorithm builds longer paths than the generic one (depth first).

We now use a FIFO list for L (First In First Out) and choose for u the first element added to L. This is a breadth first search (BFS).

Algorithm BreadthFirstSearch(G,s)
mark(s); $L := \{s\}$;
while $L \neq \emptyset$ **do**
  $u := \mathit{first}(L)$
  if $\exists(u, v)$ such that v is unmarked **then**
    choose (u, v) with v of smallest index;
    mark(v); $L := L \cup \{v\}$;
  else
    $L := L \backslash \{u\}$
  end if
end while

Below we marked the chosen nodes and the discovered nodes

L	*mark*
{2}	2
{2, 1}	1
{2, 1, 5}	5
{1, 5}	
{1, 5, 4}	4
{1, 5, 4, 6}	6
{5, 4, 6}	
{4, 6}	
{4, 6, 3}	3
{6, 3}	
{3}	
{3, 7}	
{3, 7, 8}	8
{7, 8}	
{8}	
{}	

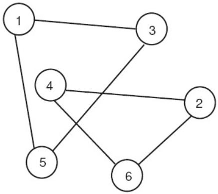


This algorithm builds a wider tree (breadth first).

Testing connectivity

The exploration algorithm finds the set of all nodes that can be reached by a path from a given node $u \in V$.

If the graph is undirected, each node in that set can follow a path back to u. They thus form the connected component $C(u)$ of u.



To find all connected components, repeat this exploration on a node of $V \backslash C(u)$, etc.

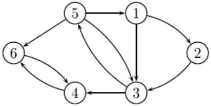
Testing strong connectivity

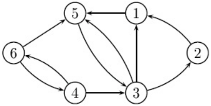
Proposition Let $G = (V, E)$ be a digraph and let $u \in V$.
If $\forall v \in V$ there exists a path from u to v and a path from v to u, then G is strongly connected.

The exploration algorithm finds the set of all nodes that can be reached by a path from a given node $u \in V$.

How can one find the nodes from which u can be reached ?

Construct for that the inverse graph by reversing all arrows





Show that the adjacency matrix of this graph is just A^T.

Proposition Let $G = (V, E)$ be a digraph and let $u \in V$.
Let $R_+(u)$ be the nodes that can be reached from u
and let $R_-(u)$ be the nodes that can reach u,
then the strongly connected component of u is
$C(u) = R_+(u) \cap R_-(u)$

The exploration algorithm applied to the inverse graph, starting from u finds the set $R_-(u)$

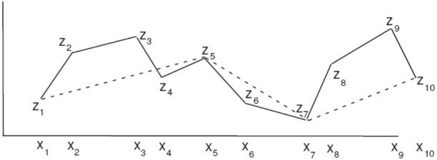
Here $R_+(v_6) = \{4, 6\}$ while $R_-(v_6) = V$ hence $C(v_6) = \{4, 6\}$

Find the other connected components.

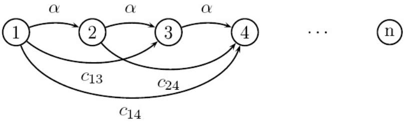
Shortest path problems

Find the shortest total length of a path between two nodes of a directed graph with lengths associated with each edge.

E.g. Find the best piecewise linear approximation of a function

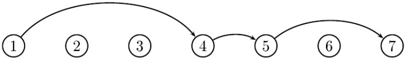


A cost $c_{ij} = \alpha + \beta \sum_{k=i}^{j} (f(x_k) - g(x_k))^2$ is associated with each linear section. This amounts to finding the shortest path in



Other example : Find the best production policy for a plant with a monthly demand d_i, a launching cost f_i, a storage cost h_i and a unit price p_i, for each period $i = 1, \dots, n$.

In the path below, we are e.g. producing in stages 1, 4 and 5.

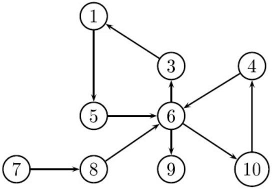


A cost is associated with each section. For the path (1,4) it is e.g. $c_{14} = f_1 + p_1(d_1 + d_2 + d_3) + h_1(d_2 + d_3) + h_2(d_3)$ which is the fixed cost + the production cost in periods 1, 2 and 3 + storage costs at the end of periods 1 and 2.

The minimization of the total cost amounts to a shortest path problem in a graph combining paths as above.

Proposition If there is a shortest walk from s to t, there is also a shortest path from s to t

Proof
Assume the walk is not a path; hence there is a recurring node. Eliminate the cycle between the first and last occurrence of this node. Repeat this procedure.



In the above graph the path $(7, 8, 6, 3, 1, 5, 6, 10, 4, 6, 9)$ has a cycle $(6, 3, 1, 5, 6, 10, 4, 6)$. After its elimination we have a path $(7, 8, 6, 9)$.

Corollary If G does not contain cycles of negative length, the resulting path is one of lower cost.

Proof Trivial

Dijkstra's algorithm

This method is for a digraph G that has positive edge lengths.

For undirected graphs one can duplicate each edge as follows

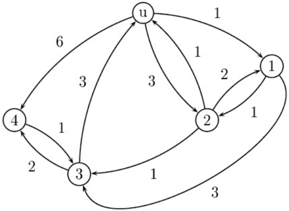
Below, $V^+(u)$ denotes the set of children of u.

Algorithm Dijkstra(G,u)
$S := \{u\}; d(u) := 0; d(v) := c(u,v) \ \ \forall v \neq u;$
while $S \neq V$ **do**
  choose $v' \notin S : d(v') \leq d(v) \ \ \forall v \notin S;$
  $S := S \cup \{v'\};$
  for each $v \in V^+(v')$ **do**
    $d(v) = \min\{d(v), d(v') + c(v', v)\}$
  end for
end while

Idea : Update a set S for which we know all shortest paths from u

Let us see the behavior of this algorithm on an example.

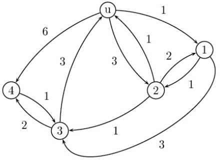
The table below indicates the steps and the distances computed for each node



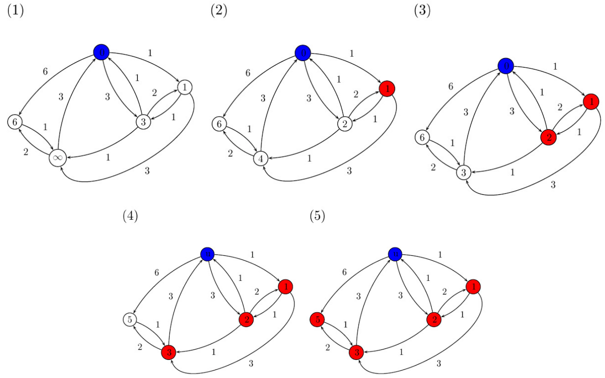
Iter	S	$d(u)$	$d(1)$	$d(2)$	$d(3)$	$d(4)$
0	$\{u\}$	0	1	3	∞	6
1	$\{u, 1\}$	0	1	2	4	6
2	$\{u, 1, 2\}$	0	1	2	3	6
3	$\{u, 1, 2, 3\}$	0	1	2	3	5
4	$\{u, 1, 2, 3, 4\}$	0	1	2	3	5

We indicate in more detail the exploration of the graph

S	$d(u)$	$d(1)$	$d(2)$	$d(3)$	$d(4)$
$\{u\}$	0	1	3	∞	6
$\{u,1\}$	0	1	2	4	6
$\{u,1,2\}$	0	1	2	3	6
$\{u,1,2,3\}$	0	1	2	3	5
$\{u,1,2,3,4\}$	0	1	2	3	5

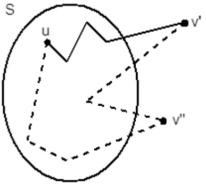


Below, the node u is blue and the explored nodes are red



Proposition Dijkstra's algorithm finds in $O(n^2)$ time the shortest path from u to all other nodes of V.

Proof By induction on the size of S, we show that
1. $\forall v \in S, d(v)$ is the length of the shortest path from u to v
2. $\forall v \in S^+$ (children of nodes of S), $d(v)$ is the length of the shortest path from u to v not passing exclusively via nodes of S



Trivial for $S = \{u\}, d(u) = 0, d(v) = c(u, v)$.

Let $v' \notin S : d(v') = \min_{v \notin S} d(v)$ then the shortest path to v' must lie completely in S. If not, $\exists v''$ outside S at a shorter distance.

We can update $S := S \cup \{v'\}$ and compute the shortest path from u to children of v' as $d(v) = \min\{d(v), d(v') + c(v', v))\}$. This gives the length of the shortest path to all $v \in S^+$.

The other distances are unknown as yet and hence set to ∞.

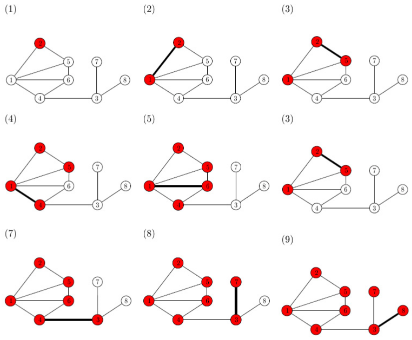
Variants

For a graph with edge lengths 1 it suffices to do a BFSearch and to keep track of the path lengths by incrementing them with 1 during the exploration phase. This is thus an $\mathcal{O}(m)$ time algorithm.

Algorithm ShortestPathBFS(G,v)
mark(v); $S := \{v\}$; $d(v) = 0$;
while not $S = \emptyset$ **do**
    $v :=$ *first*(S)
    if $\exists(v, v')$ such that v' is unmarked **then**
        choose (v, v') with v' of smallest index;
        mark(v'); $S := S \cup \{v'\}$; $d(v') = d(v) + 1$;
    else
        $S := S \backslash \{v\}$
    end if
end while

Proposition All nodes at distance exactly k are correctly identified before proceeding further.

Proof For $k = 0$ this is trivial (S is the original node u).
Induction step : suppose the statement is correct up to k.
After all nodes at distance k have been found, one finds nodes that are at a distance larger than k but since they are all neighboring nodes, they must be at distance exactly $k + 1$.



For an acyclic graph, one can just compute the topological order in $O(m)$ time (see earlier).

To solve the shortest path problem one then uses the algorithm

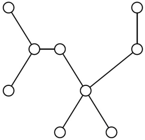
Algorithm ShortestPathAcyclic(G,v)
$d(1) = 0; d(i) := \infty$ for $i = 2, \dots, n$;
for $i = 1 : n-1$ **do**
    for $j \in V^+(i)$ **do**
        $d(j) := min_j\{d(j), d(i) + c(i,j)\}$;
    end for
end for

What is the complexity of this second step ?

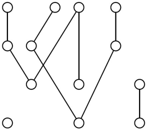
One can also see the shortest path problem as a flow problem or as a linear programming problem.
This leads to other algorithms like the Bellman-Ford Algorithm.

Trees and forests

A tree is an acyclic and connected graph



A forest is an acyclic graph (and hence a union of trees)



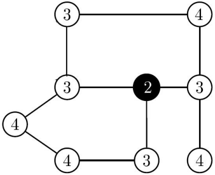
Proposition For a graph $G = (V, E)$ of order $n = |V|$, the following are equivalent

1. G is connected and has $n - 1$ edges
2. G is acyclic and has $n - 1$ edges
3. G is connected and acyclic
4. $\forall u, v \in V$ there is one and only one path from u to v
5. G is acyclic and adding an edge creates one and only one cycle
6. G is connected and removing an arbitrary edge disconnects it

Proofs ?

The following definitions are especially relevant for trees.

The eccentricity $\varepsilon(u) = \max_{v \in V} d(u, v)$ of a node is the maximum distance to any node $v \in V$. The eccentricity of each node is indicated in the graph below



The radius $rad(G) = \min_{u \in V} \varepsilon(u)$ of a graph G is the minimal eccentricity of all nodes in V

The diameter $diam(G) = \max_{u \in V} \varepsilon(u)$ of a graph G is the maximal eccentricity of all nodes in V. It is also the maximal distance between any two nodes in V

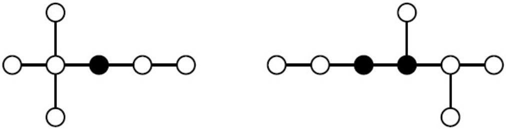
The center of a graph G is the set of nodes in V of minimal eccentricity (the black node)

A leaf of a tree T is a node of degree 1

Proposition Let T be a tree and let T' be the tree obtained by removing all its leafs, then $\varepsilon(T') = \varepsilon(T) - 1$ for all nodes of T'.

Proof ?

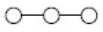
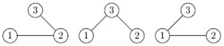
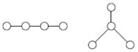
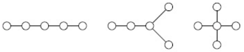
Proposition The center of a tree is a single node or a pair of adjacent nodes.



Proof By induction using the previous proposition.
Show that the center does not change.

Counting trees

How many different (labeled) trees are there with n nodes ?
The following table gives the count for small n

1			$\to 1$
2			$\to 1$
3			$\to 3$
4			$\to 16$
5			$\to 125$

The following theorem of Cayley gives the exact formula.

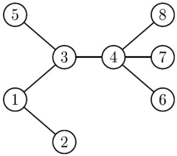
Proposition

The number of distinct labeled trees of order n equals n^{n-2}

We construct a bijection of T_n with a sequence via the algorithm

Algorithm PrüferSequence(T)
$s := (); t := ();$
while $|E| > 1$ **do**
  choose the leaf of smallest index i;
  $T := T \backslash \{i\}; s := (s, i); t := (t, neighbour(i));$
end while

On the graph below, it yields the table next to it



i	s_i	t_i
1	2	1
2	1	3
3	5	3
4	3	4
5	6	4
6	7	4

One shows that the graph can be reconstructed from the sequence t_i which are $n-2$ numbers from $\{1, \ldots, n\}$ and there are exactly n^{n-2} such sequences.

Spanning tree

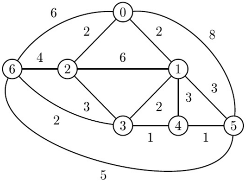
Remove from a connected graph as many edges as possible while remainig connected; this should yield a tree with $n-1$ edges.

This is the minimal spanning tree problem solved by the following algorithm, of time complexity $\mathcal{O}(m \log m)$

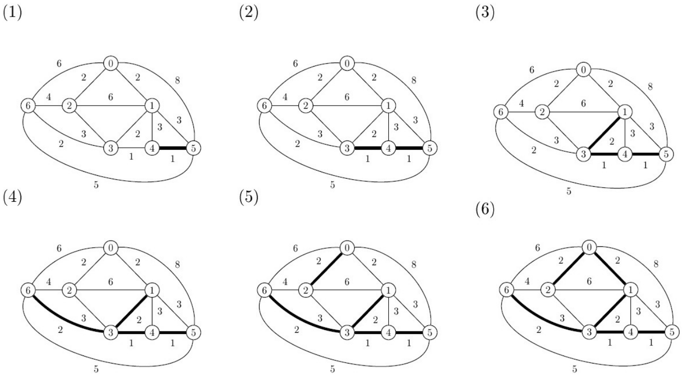
Algorithm KruskalMST(G)
$E_{ord} := sort(E)$; $E' := \emptyset$; $E_{rest} := E_{ord}$;
while $|E'| > n-1$ **do**
  $\alpha := first(E_{rest})$; $E_{rest} := E_{rest} \backslash \{\alpha\}$;
  if $(V, E' \cup \{\alpha\})$ is acyclic **then**
    $E' := E' \cup \{\alpha\}$;
  end if
end while

The sorting is done efficiently in $\mathcal{O}(m \log m)$ time as well.

Let us look at an example



The different steps of the algorithm are



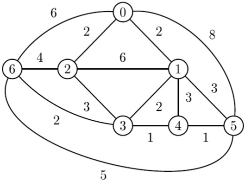
This constructs a tree which is a subgraph with $n - 1$ edges.

Now we look at an alternative algorithm of time complexity $O((m+n)\log n)$

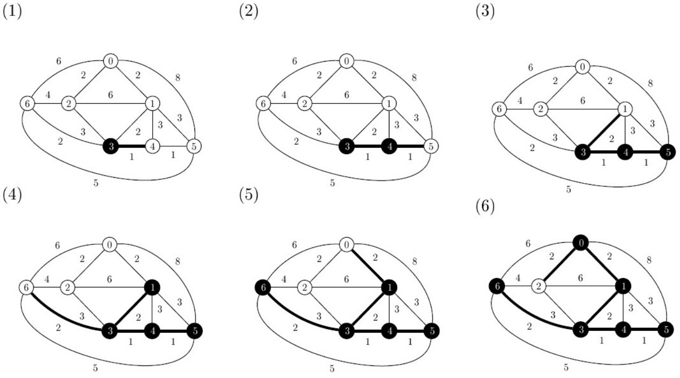
The idea is to pick a random node and then grow a minimal tree from there

Algorithm PrimMST(G)
Choose $u \in V$; $V' := \{u\}$; $E' := \emptyset$;
for $i = 1 : n-1$ **do**
  $E'' :=$ edges linking V to V';
  choose $e = (u, v) \in E''$ of minimal weight and such that $(V' \cup \{v\}, E'\{e\}$ is acyclic;
  $V' := V' \cup \{v\}$; $E' := E' \cup \{e\}$;
end for

Let us look at the same example



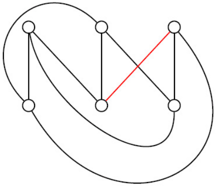
The different steps of the algorithm are

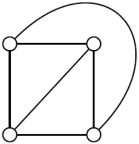


The graph (V, E') is a minimal spanning tree with $n-1$ edges

Planar graphs

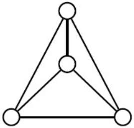
When drawing connected graphs one is naturally lead to the question of crossing edges. One says that a graph is planar if it can be drawn (or represented) without crossing edges



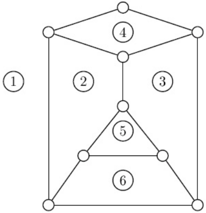


The above graphs represent $K_{3,3}$ (not planar) and K_4 (planar)

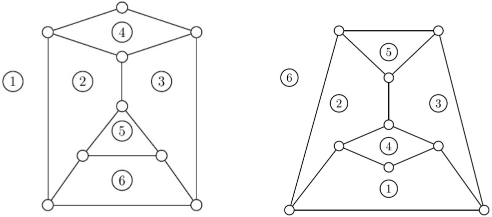
Proposition (Fary, 1948)
Every planar graph can be represented in the plane using straight edges only



For such graphs, one can now define faces. These are the regions encircled by edges that form a cycle. One has to identify also an exterior face as shown in this figure with 6 faces

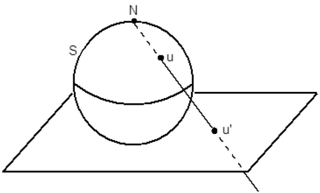


Proposition A planar representation of a graph can be transformed to another one where any face becomes the exterior face (a proof comes later)

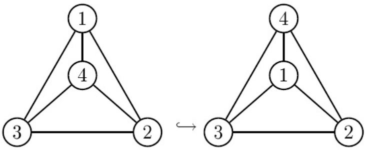


Proposition A graph can be represented in a plane if and only if it can be represented on a sphere (immersion)

Proof
Use a stereographic projection



Every face of the plane is mapped to a sector on the sphere. No point on the sphere can therefore belong to two different sectors. The external face is mapped to a sector containing the north pole.

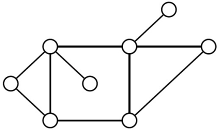


For the external face result, notice that by rotating the sphere, one can move any point (and hence sector) to the north pole

Characterisation

Proposition (Euler formula) Let G be planar, and let $n(G)$ be its number of vertices, $e(G)$ its number of edges, and $f(G)$ its number of faces. Then $f = e - n + 2$.

In the example shown here
$n = 8$; $e = 10$; $f = 4$



Proof Use induction on the number of faces f.
For $f = 1$ there are no cycles and hence the connected graph is a three, for which we know $e = n - 1$ and hence $f = e - n + 2$.
For $f \geq 2$, remove an edge (u, v) between two faces to construct $G' := G \backslash (u, v)$. Then $f(G') = f(G) - 1$; $e(G') = e(G) - 1$ and $n(G') = n(G)$. Use the result for smaller f to prove it for f.

Some exercices

Proposition Let G be planar with $f > 1$, then $3f \leq 2e$

Proposition Let G be planar with $f > 1$ and G have no triangles, then $2f \leq e$

Proposition Let G be a planar (connected) graph.
If $n \geq 3$ then $e \leq 2n - 6$

Proposition Let G be a planar (connected) graph.
If G has no triangles or is bipartite, then $e \leq 2n - 4$

These help to prove the following lemma

Proposition K_5 and $K_{3,3}$ are not planar.

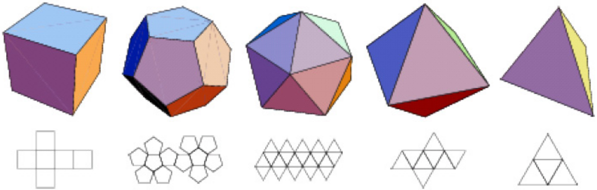
Corollary The average degree of the vertices of a planar connected graph G is smaller than $6 - \frac{12}{n}$

Corollary In a planar (connected) graph there always exists a vertex such that $d(v) \leq 5$

Corollary A planar graph can be colored with 6 colors (see later)

Proposition (Platonic solid) There are only 5 regular polyhedra

These so-called Platonic solids are shown below

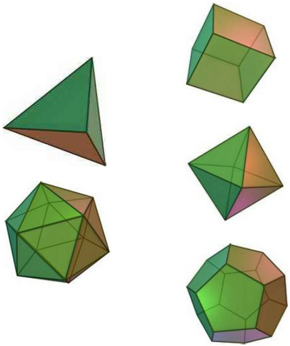


The Platonic solids are characterized by three equations $nk = 2e$, $fl = 2e$ for k, l integers, and $n + f = e + 2$

Explain why

It then follows that $2e/k + 2e/l - e = 2$ hence $2/k + 2/l > 1$ or $(k-2)(l-2) < 4$. The integer solutions are given by

Name	k	l	e	n	f
Tetraeder	3	3	6	4	4
Cube	3	4	12	8	6
Dodecaeder	3	5	30	20	12
Octaeder	4	3	12	6	8
Icosaeder	5	3	30	12	20

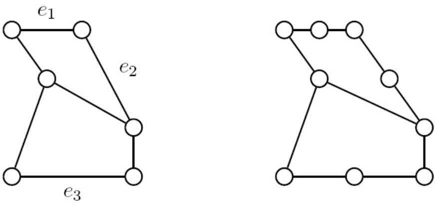


Test for planar graphs

We first need to introduce subdivisions and subgraphs.

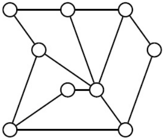
Let us expand a graph $G = (V, E)$ by a subdivision of one of its edges $e = (u, v) \in E$. We put a new node w on e and replace it by two new edges $e_1 = (u, w)$ and $e_2 = (w, v)$. The new graph is thus given by $G' = (V \cup \{w\}, E \cup \{e_1, e_2\} \backslash \{e\})$.

Two graphs are said to be homeomorphic to each other iff one can be derived from the other via a sequence of subdivisions.



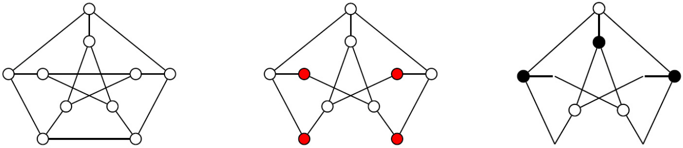
Corollary Homeomorphism is an equivalence relation.

A graph $G' = (V', E')$ is a subgraph of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$ (edges must disappear along with nodes)



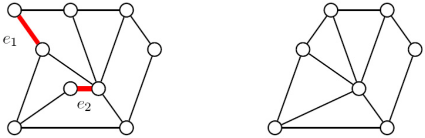
Proposition (Kuratowsky, 1930) A graph is planar iff it does not contain a subgraph homeomorphic to $K_{3,3}$ or K_5.

Example : the Petersen graph (subgraph + homeomorphism)



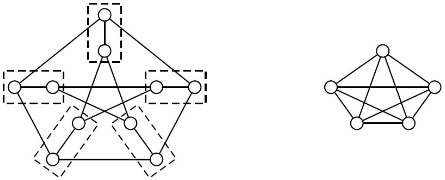
Minors

Let $e = (u, v)$ be an edge of a graph $G = (V, E)$. A contraction of the edge e consists of eliminating e and merging the nodes u and v into a new node w. The new graph G' is thus $G' = (V \backslash \{u, v\} \cup \{w\}, E \backslash \{e\})$



Proposition (Wagner, 1937) A graph is planar iff it does not have $K_{3,3}$ or K_5 as a minor.

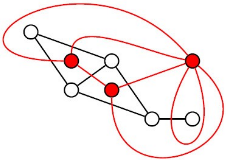
Example :
the Petersen
graph again



Proposition (Robertson-Seymour)
For a graph G, determining if a given graph H is a minor of H, can be solved in polynomial time (with respect to $n(G)$ and $m(G)$).

A dual graph G^* of a planar graph is obtained as follows
1. G^* has a vertex in each face of G
2. G^* has an edge between two vertices if G has an edge between the corresponding faces

This is again a planar graph but it might be a multigraph (with more than one edge betwee two vertices)

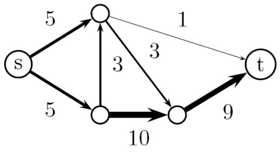


Exercise Show that Euler's formula is preserved

Exercise Show that $G = (G^*)^*$

Networks and flows

A network is a directed graph $N = (V, E)$ with a source node s (with $d_{out}(s) > 0$) and a terminal node t (with $d_{in}(t) > 0$). Moreover each edge has a strictly positive capacity $c(e) > 0$.

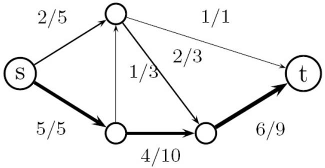


A flow $f : V^2 \to \mathbb{R}^+$ is associated with each edge $e = (u, v)$ s.t.
1. for each edge $e \in E$ we have $0 \leq f(e) \leq c(e)$
2. for each intermediate node $v \in V \backslash \{s, t\}$ the in- and out-flow at that node $\sum_{u \in V^-(v)} f(u, v) = \sum_{u \in V^+(v)} f(v, u)$ match

The total flow F of the network is then what leaves s or reaches t

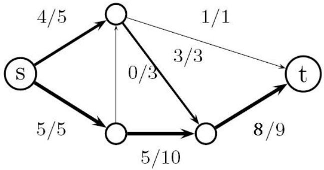
$$F(N) := \sum_{u \in V} f(s, u) - \sum_{u \in V} f(u, s) = \sum_{u \in V} f(u, t) - \sum_{u \in V} f(t, u)$$

Here is an example of a flow



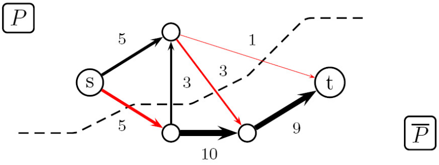
It has a value of $F(N) = 7$ and the conservation law is verified inside.

But the flow is not maximal, while the next one is ($F(N) = 9$) as we will show later. Notice that one edge is not being used ($f = 0$)



Cut of a network

A cut of a network is a partition of the vertex set $V = P \cup \overline{P}$ into two disjoint sets P (containing s) and $\overline{P}$ (containing t)



The capacity of a cut is the sum of the capacities of the edges (u, v) between P and $\overline{P}$

$$\kappa(P, \overline{P}) = \sum_{u \in P; v \in \overline{P}} c(u, v)$$

which in the above example equals $5 + 3 + 3 + 1 = 9$.

We now derive important properties of this capacity.

Proposition Let $(P, \overline{P})$ be any cut of a network $N = (V, E)$ then the associated flow is given by

$$F(N) = \sum_{u \in P; v \in \overline{P}} f(u, v) - \sum_{u \in P; v \in \overline{P}} f(v, u)$$

Proof First show that $F(N) = \sum_{u \in P} (\sum_v f(u, v) - \sum_v f(v, u))$ by summing all contributions in P and using conservation.

For all $v \in P$ the term between brackets is zero (conservation).

Hence we only need to keep the edges across the partition.

Corollary A flow is bounded by the capacity of any cut $F(N) \leq \kappa(P, \overline{P})$

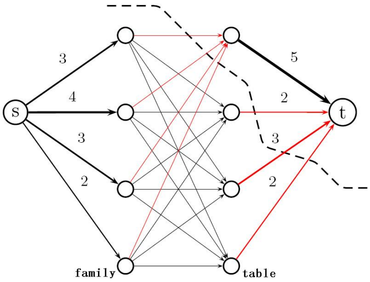
A minimal cut (with minimal capacity) also bounds $F(N)$

(we will construct one and will see it is in fact equal to $F(N)$)

Applications

The dining problem

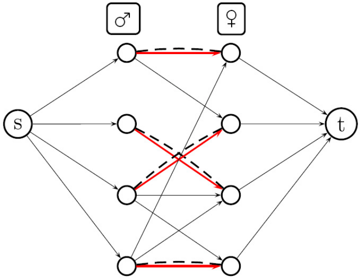
Can we seat 4 families with number of members (3,4,3,2) at 4 tables with number of seats (5,2,3,2) so that no two members of a same family sit at the same table ?



The central edges are the table assignments (a capacity of 1).
The cut shown has a capacity 11 which upper bounds $F(N)$.
We can therefore not seat all 12 members of the four families.

The marriage problem

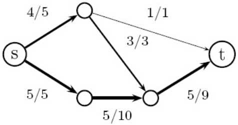
One wants to find a maximimum number of couplings between men and women where each couple has expressed whether or not this coupling was acceptable (central edges that exist or not)

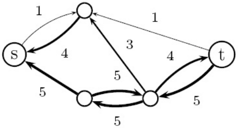


One wants to find a maximum number of disjoint paths in this directed graph. All the capacities of the existing edges are 1.

Given a network $N(V, E)$ and a flow f then its residual network N_f is a network with the same nodes V but with new capacities

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \textit{if } (u,v) \in E; \\ f(v,u) & \textit{if } (v,u) \in E \\ 0 & \textit{otherwise}. \end{cases}$$





An augmenting path is a directed path $v_0, \ldots, v_k$ from $S = v_0$ to $t = v_k$ for which
$\Delta_i = c(v_i, v_{i+1}) - f(v_i, v_{i+1}) > 0 \; \forall (v_i, v_{i+1}) \in E$ or
$\Delta_i = c(v_i, v_{i+1}) - f(v_{i+1}, v_i) > 0 \; \forall (v_{i+1}, v_i) \in E$
This path is not optimal since the original flow can be increased.

Max-flow Min-cut

Proposition
The flow is optimal if there exists no augmentation path from s to t

Proof Construct a cut $(P, \overline{P})$ where $u \in P$ if there is an augmentation path from s to u and $u \in \overline{P}$ otherwise.
Show that $(P, \overline{P})$ is a valid cut for which $F(N) = \kappa(P, \overline{P})$.

Proposition In a network N the following are equivalent
1. A flow is optimal
2. The residual graph does not contain an augmenting path
3. $F(N) = \kappa(P, \overline{P})$ for some cut $(P, \overline{P})$

The value of the optimal flow thus equals $F(N) = \min \kappa(P, \overline{P})$

Proof Left to the reader (combine earlier results)

This becomes an LP problem in the flows x_{ij} on the edges (i, j)
max $\left(\sum_{i:(s,i)} x_{si} = \sum_{i:(i,s)} x_{is}\right)$ subject to $\sum_i x_{ij} = \sum_i x_{ji}$ and $0 \leq x_{ij} \leq c_{ij}$

The Ford-Fulkerson algorithm (1956) calculates this optimal flow using augmentation paths.

Algorithm MaxFlowFF(N,s,t)
$f(u,v) := 0 \; \forall (u,v) \in E$;
while N_f contains a path from s to t **do**
  choose an augmentation path A_p from s to t
  $\Delta := \min_{(u,v) \in A_p} \Delta_i$
  Augment the flow by Δ along A_p
  Update N_f
end while

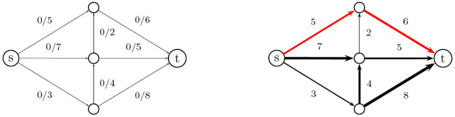
Finding a path in the residual graph can be implemented with a BFS or DFS exploration as shown below

At each step we show the graph (left) and the residual graph (right)

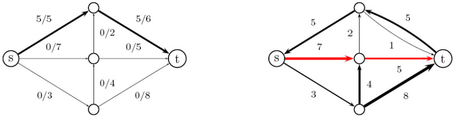
Augmentation paths are in red. In 5 steps we find $F(N) = 14$

Flows

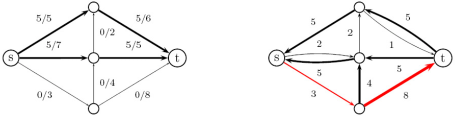
(1)


s
t
0/5
0/6
0/2
0/7
0/5
0/4
0/3
0/8
5
6
2
7
5
4
3
8

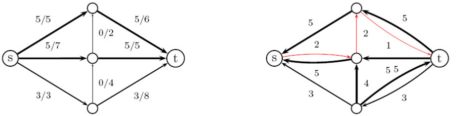
(2)


s
t
5/5
5/6
0/2
0/7
0/5
0/4
0/3
0/8
5
5
2
7
1
5
4
3
8

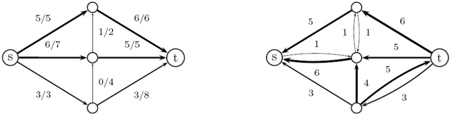
(3)


s
t
5/5
5/6
0/2
5/7
5/5
0/4
0/3
0/8
5
5
2
2
1
5
5
4
3
8

(4)

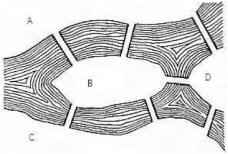

s
t
5/5
5/6
0/2
5/7
5/5
0/4
3/3
3/8
5
5
2
2
1
5 5
5
4
3
3

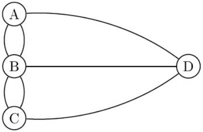
(5)


s
t
5/5
6/6
1/2
6/7
5/5
0/4
3/3
3/8
5
6
1
1
1
5
6
5
4
3
3

Eulerian tour (1756)

An Eulerian cycle (path) is a subgraph $G_e = (V, E_e)$ of $G = (V, E)$ which passes exactly once through each edge of G.
G must thus be connected and all vertices V are visited (perhaps more than once). One then says that G is Eulerian

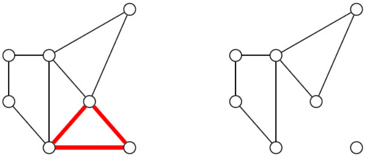




Proposition A graph G has an Eulerian cycle iff it is connected and has no vertices of odd degree
A graph G has an Eulerian path (i.e. not closed) iff it is connected and has 2 or no vertices of odd degree

This would prove that the above graph is not Eulerian.

Proof (of the first part regarding cycles)
Necessity Since G is Eulerian there is a cycle visiting all nodes. Each time we visit $v \in V$, we leave it again, hence $d(v)$ is even.
Sufficiency For a single isolated node, it is trivial. For $|V| > 1$ there must be a cycle ϕ in the graph. Consider the subgraph H with the same nodes but with the edges of ϕ removed.

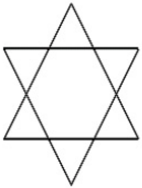


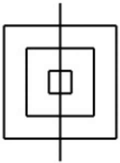
Each of its components H_i satisfy the even degree condition and again have an Eulerian cycle ϕ_i. By recurrence we then reduce G to its isolated vertices.

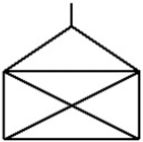
To reconstruct the Eulerian cycle, start from a basic cycle ϕ. Each time a node of another cycle ϕ_i is encountered, substitute that cycle to the node (and do this recursively).

Proof (of the second part regarding paths) Left as an exercise

The path problem says if you can draw a graph without lifting your pen. Apply this to the following examples.







Proposition A directed graph $G = V, E_d$ has an Eulerian tour G_e iff it is connected and balanced, i.e. all its nodes have $d_{in}(v) = d_{out}(v)$.

Proof Left as an exercise

The following algorithm of Fleury (1883) reconstructs a cycle C if it exists. E' is the set of edges already visited by the algorithm.

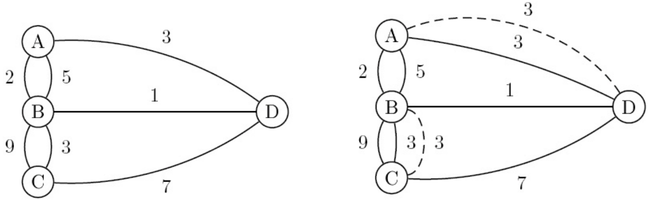
Algorithm FindEulerianCycle(G)
Choose $v_0 \in V$; $E' := \emptyset$; $C := \langle\rangle$;
for $i = 1 : m$ **do**
  choose $e = (v_{i-1}, v_i)$ s.t. $G' = (V, E \backslash E')$ has 1 conn. comp.;
  $E' := E' \cup \{e\}$; $C := \langle C, e\rangle$; $v_i := v_{i-1}$;
end for

Exercise
Propose a modification addressing the Eulerian Path Problem

But what if the graph is not Eulerian ? Can we find a mininimum cost modification of the problem ?

Chinese postman (1962)

We consider a minimum cost modification of the Eulerian cycle problem. A chinese postman needs to find a tour passing along all edges of a graph and minimize the length of the path.

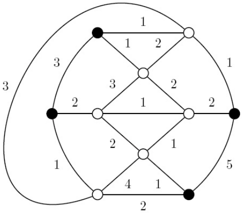


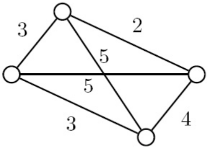
The edges have a cost and we need to make the graph Eulerian

Exercise

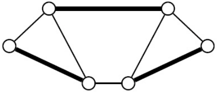
1. Give a simple lower bound. 2. When can this bound be met ?
3. Is there another solution (or a better one) ?

Solution : find all odd degree vertices and find the shortest paths between them





Now find a perfect matching of the nodes in this graph.
A perfect matching in a graph is a set of disjoint edges of a graph to which all vertices are incident.

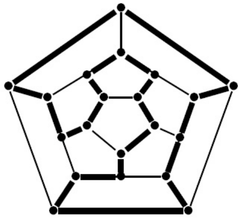


This can be solved in $0(n^3)$ time with the Hungarian algorithm.

Hamiltonian cycle (1859)

Was a game sold by Hamilton in 1859 to a toy maker in Dublin.

A Hamiltonian cycle is a cyclic subgraph $G_h = (V, E_h)$ of $G = (V, E)$ which passes exactly once through all nodes



It is a so-called hard problem and there is no general condition for its existence (in contrast with the Eulerian path problem).

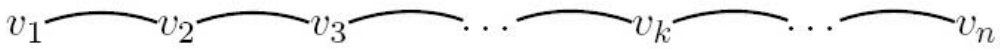
It exists for Platonic solids and complete graphs,
but not for the Petersen graph

Proposition (Dirac, 1951) A graph G with $n \geq 3$ nodes and $d(v) \geq n/2$, $\forall v \in V$, is Hamiltonian

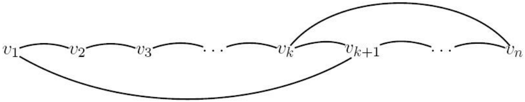
Proof

G is connected, otherwise its smallest component would have all edges with $d(v) < n/2$

Then consider a longest path $v_1 v_2 ... v_n$ (with maybe $n < |V|$)



Because $d(v_1), d(v_n) \geq n/2$, it must also be covered by a cycle (because all the neigbors of v_1 and v_n are on that path)

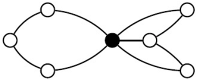


Because of connectedness $n = |V|$ and it is a Hamiltonian cycle.

Exercise Construct a graph with $d(v) < n/2$ and yet has a Hamiltonian cycle

Proposition
If $G = (V, E)$ has a Hamiltonian cycle, then $G - V'$ has at most $|V'|$ connected components for any subset of vertices $V' \subset V$.
Proof Let H be a Hamiltonian subgraph of G, then $H - V'$ has less than $|V'|$ connected components. But $G - V'$ has the same vertices as $H - V'$ and it has additional edges.

Exercise Does this graph have a Hamiltonian cycle ?



Exercise Prove that a complete bipartite graph $K_{m,n}$ is Hamiltonian iff $m = n$

Traveling Salemen Problem

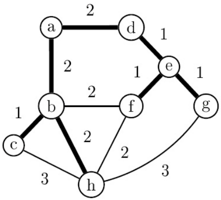
A traveling salesman is supposed to visit a number of cities (nodes in a graph) and minimize the travel time (or total length)

This is NP-hard but can often be solved approximately in reasonable time. Consider a distance graphs with triangle inequality $d(u, v) + d(v, w) \geq d(u, w)\ \forall u, v, w \in V$

Construct a minimal weight spanning tree T and visit the nodes using BFS.

For this example we would have a cycle (a,b,c,b,h,b,a,d,e,f,e,g,e,a)

Notice that all edges are visited twice.



The optimal path P^* satisfies the inequalities $cost(T) < cost(P^*) \leq 2.cost(T)$

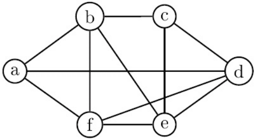
Exercise Explain why

Test for planar graph

There is a simple way to test if a Hamiltonian graph is planar

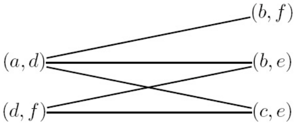
1. Draw G with the Hamiltonian graph H at the outside

The following graph is already drawn with $H = (a, b, c, d, e, f, a)$ outside



2. Define K as the graph whose nodes are the edges $e_1, \ldots, e_r$ not in H and with an edge between e_i and e_j if they cross in G.

The following graph has the vertices $(a, d), (b, f), b, e), c, e), (d, f)$ and five edges, corresponding to the crossings in G

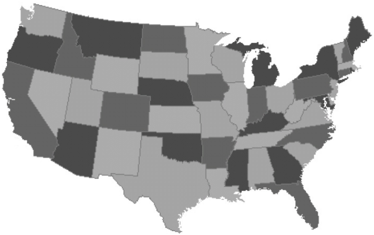


Then G is planar iff K is bipartite

Exercise Explain why

Four color problem

In 1852 it was conjectured that a country map (like the USA map) could always be colored with only four colors. There is an underlying assumption for point borders.



This was proven in 1976 by K. Appel and W. Haken but their proof used a computer search over 1200 so-called critical cases.

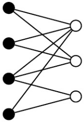
Exercise What property does the underlying graph have ?

Coloring nodes

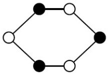
A k-coloring of a graph $G = (V, E)$ is a mapping $f : V \rightarrow 1, \ldots, k$ such that $f(v_i) \neq f(v_j)$ if $(v_i, v_j) \in E$.
The chromatic number of a graph is the smallest number k for which there exists a k-coloring.

Some examples of known chromatic numbers are :

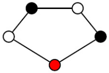
Bipartite graph
$\chi(G) = 2$



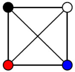
Even cycle
$\chi(G) = 2$



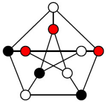
Odd cycle
$\chi(G) = 3$



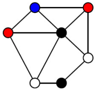
Clique
$\chi(K_n) = n$



Petersen Graph
$\chi(G) = 3$



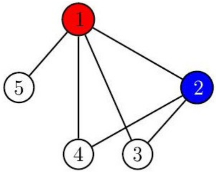
Planar graph
$\chi(G) = 4$



The coloring problem for general graphs is NP-complete but such problems often lead to more interesting applications

Exam scheduling problem

Student \ Course	1	2	3	4	5
1	1	1	1		
2	1				1
3	1		1		
4	1	1		1	
5					1



The table on the left gives the exams each student takes
The chromatic number $\chi(G)$ of the corresponding graph gives the minimum numbers of time slots for the exams

Exercise Can you formulate such a slot problem with students choosing out of k pre-set programs ?

Bounds

Proposition

Let G be connected and $m = |E|$, then $\chi(G) \leq \frac{1}{2} + \sqrt{2m + \frac{1}{4}}$

Proof Let $C = \{C_1, \ldots, C_k\}$ be the partition of V according to colors. There is at least one edge between two colors, which implies $m > B(k, 2)$ and hence $k^2 - k - 2m \leq 0$.

Proposition

Let $\Delta(G) = \max\{d(v) | v \in V\}$, then $\chi(G) \leq \Delta(G) + 1$ (trivial)

Proposition (Brooks, 1941)

$\chi(G) \leq \Delta(G)$ for any graph different from K_n or an odd cycle

Proposition

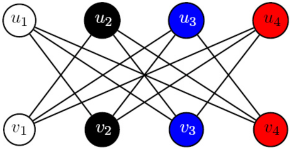
$\chi(G) \leq 1 + \max_i\{\min(d_i, i - 1)\}$ when ordering $d_1 \geq \ldots \geq d_n$.

Proof Order the nodes like the d_i's and use the greedy algorithm

Greedy algorithm

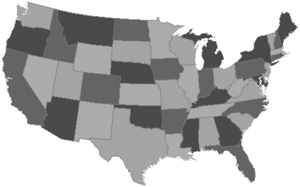
Algorithm GreedyColor(G)
$L := sort(V); c := sort(colors)$
for $v \in V$ **do**
  choose smallest c_i not used by colored neigbors
end for

On a bipartite graph this greedy algorithm is optimal when numbering the nodes per part but it can be bad for other numberings, such as $\{u_1, v_1, u_2, v_2, u_3, v_3, u_4, v_4\}$



Exercise
Does each graph have a good numbering for the greedy algorithm

Let us come back to the map coloring problem



and try to prove the following (simpler) result

Exercise Every planar graph can be colored with 6 colors

Show that $e \leq 3n - 6$

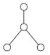
Show then that for planar graphs $average(d(v)) \leq 6 - 12/n$

Finally prove that there exists a v such that $d(v) \leq 5$

Now use induction to prove the proposition (remove nodes)

Chromatic polynomial (Birkhoff-Lewis 1918)

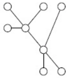
The chromatic polynomial of a graph $p_G(k)$ indicates how many different ways a graph can be colored with k colors. E.g.



k	1	2	3	k
$p_G(k)$	0	2	24	$k(k-1)^3$

k	1	2	3	k
$p_G(k)$	0	0	6	$k(k-1)(k-2)$

k	1	2	k
$p_G(k)$	1	16	k^4



k	1	2	k
$p_G(k)$	0	2	$k(k-1)^{n-1}$

Exercise Prove the above formulas

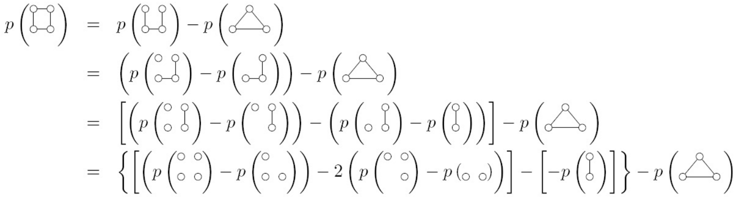
Notice that $\chi(G) = \min\{p(G)(k) > 0\}$. Does this help ?

There is a powerful induction theorem using the simpler graphs $G - (u, v)$ (remove an edge) and $G \circ (u, v)$ (contract an edge)

Proposition If $(u, v) \in E$ then $p_G(k) = p_{G-(u,v)}(k) - p_{G\circ(u,v)}(k)$

Proof u and v have different colors in G and the same in $G \circ (u, v)$

This can be used to compute the chromatic polynomial of more complex networks



$$
\begin{aligned}
&= (k^4 - k^3) - 2(k^3 - k^2) + k(k-1) - k(k-1)(k-2) \\
&= k^4 - 4k^3 + 6k^2 - 3k \\
&= k(k-1)(k^2 - 3k + 3)
\end{aligned}
$$

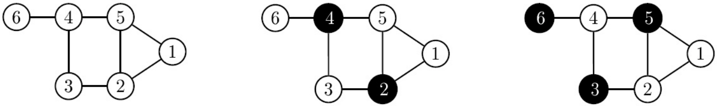
but the problem remains combinatorial and thus hard

Exercise Derive this quicker using the result for a tree

Stable sets

An independent or stable set S in a graph $G = (V, E)$ is a subgraph of G without any edges, i.e. $\forall u, v \in S : (u, v) \in E$

The two sets of black nodes are stable sets of the left graph



Such sets can clearly be colored with only one color, which proves

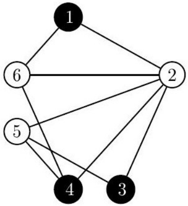
Proposition If a graph is k-colorable then V can be partitioned as k stable sets

The independence number $\alpha(G)$ is the size of the largest possible stable set.

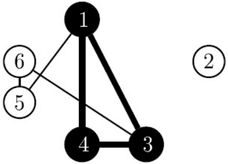
Proposition One has $\chi(G) \cdot \alpha(G) \geq n$ (trivial)

The following example requires finding a maximal stable set. Find the maximum number of projects one can realize when the table indicates which students are needed for each project.

Student \ Project	1	2	3	4	5	6
1	1					
2		1		1		1
3	1	1				1
4		1	1		1	
5				1	1	
6			1			
7			1			



Notice that it is equivalent to finding a maximal clique (or complete subgraph) in the complementary graph $G_c = (V, E_c)$, where E_c is the complement of E



Algorithm complexity

We distinguish problems from algorithms used to solve them. There is also the issue of time complexity and space complexity.

The function $C_A(s)$ of an algorithm is the number of time steps needed to solve a problem of size s with that algorithm.
A problem is called polynomial if there exists an algorithm with $C_A(s) = \mathcal{O}(p(n))$ for some polynomial $p(\cdot)$, meaning

$$\exists n_0 : C_A(s) \leq p(n) \quad \forall n \geq n_0.$$

The relative times needed to solve problems of different complexity

$C_A(n)$ \ n	10	20	30	40	50	60
n	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s	0,00006 s
n^2	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0025 s	0,0036 s
n^3	0,001 s	0,008 s	0,027 s	0,064 s	0,125 s	0,216 s
n^5	0,1 s	3,2 s	24,3 s	1,7 min	5,2 min	13,0 min
2^n	0,001 s	1,0 s	17,9 min	12,7 days	35,7 years	366 cents
3^n	0,059 s	58 min	6,5 years	3855 cents	2.10^8 cents	$1,3.10^{13}$ cents

This shows the importance of having a polynomial problem

Better is to look at the size of the problems one can solve when the machines speed up 100 or 1000 times

$C_A(n)$	size	100 times	1000 times
n	N_1	$100N_1$	$1000N_1$
n^2	N_2	$10N_2$	$31,6N_2$
n^3	N_3	$4,64N_3$	$10N_3$
n^5	N_4	$2,5N_4$	$3,98N_4$
2^n	N_5	$N_5+6,64$	$N_5+9,97$
3^n	N_6	$N_6+4,19$	$N_6+6,29$

Here are a number of polynomial time problems

Finding the shortest path between 2 vertices
Testing if a graph is planar
Testing if a graph is Eulerian
Finding a spanning tree
Solving the perfect marriage problem

Here are a number of problems that are not polynomial

Finding the chromatic number of a graph
Finding a Hamiltonian cycle in a graph
Finding the largest stable set in a graph
Solving the travelling salesman problem
Testing if two graphs are isomorphic (not known)

Comparing problems

A problem Y is reducible (in polynomial time) to a problem X if X is at least as difficult to solve as Y, denoted as $X \geq_p Y$. Then

$X \geq_p Y$ and $X \in \mathcal{P}$ implies $Y \in \mathcal{P}$
$X \geq_p Y$ and $Y \notin \mathcal{P}$ implies $X \notin \mathcal{P}$

Define the problem [$LongestPath(u, v, w, N)$] of finding a path of length $\geq$ any N from u to v in a graph with integer weights w

Proposition [$HamiltonianCycle$] $\leq_p$ [$LongestPath(u, v, w, N)$]

Proof Choose unit weights w. Pick an edge $e = (u, v)$.
If there is a longest path of length $N = n - 1$ in $G' = G \backslash e$, then G is Hamiltonian. Try out all $m < n^2/2$ edges.

Since we know that the Hamiltonian cycle problem in not in $\mathcal{P}$ the longest path problem is also not in $\mathcal{P}$.

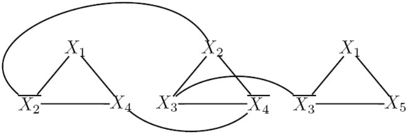
A Boolean clause is a disjunction of Boolean terms $X_i \in \{0, 1\}$ and their negation $\overline{X}_i \in \{0, 1\}$, e.g. $X_1 \vee \overline{X}_2 \vee X_4 \vee \overline{X}_7$ is a 4-term.

Define the problem [SAT] as checking if a set of Boolean clauses can be simultaneously satisfied ([3SAT] involves only 3-terms). E.g. $\{\overline{X}_2 \vee X_2, \overline{X}_2 \vee X_3 \vee \overline{X}_4, \overline{X}_1 \vee X_4\}$ can be satisfied by choosing $X_1 = 1, X_2 = 0, X_3 = 1, X_4 = 0$.

Proposition $[SAT] \leq_p [3SAT]$ and $[3SAT] \leq_p [StableSet]$

Proof We do not prove the first part involving only 3-terms.

Construct a triangle for each 3-term and then connect the negations across triangles



For a stable set, I can choose only one node in each triangle. Then there is a stable set of size $n/3$ iff [3SAT] is satisfiable.

$\mathcal{NP}$ and $\mathcal{NP}$-complete

A problem is Non-deterministic Polynomial ($\mathcal{NP}$) if the validity of a solution can be checked in polynomial time.

Checking if a given cycle is Hamiltonian can be solved in polynomial time, but finding it is difficult.

In $\mathcal{P}$ the problem can be solved in polynomial time,
in $\mathcal{NP}$ a solution can be checked in polynomial time.

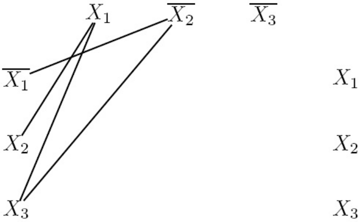
It is still an open question of $\mathcal{P} = \mathcal{NP}$ (Cray prize = 1 million dollar)

A problem X is $\mathcal{NP}$-complete if $X \in \mathcal{NP}$ and $\forall Y \in \mathcal{NP}$, $Y \leq_p X$.

Corollary If one $\mathcal{NP}$-complete problem is in $\mathcal{P}$ then $\mathcal{P} = \mathcal{NP}$

Corollary If one $\mathcal{NP}$-complete problem is not in $\mathcal{P}$ then $\mathcal{P} \neq \mathcal{NP}$

[3SAT] is known to be $\mathcal{NP}$-complete.
We now prove that also the [CLIQUE] problem is $\mathcal{NP}$-complete
The [CLIQUE] problem is checking if there exists a clique (complete subgraph) of size k in a graph $G = (V, E)$
Proof Consider $\{X_1 \vee \overline{X}_2 \vee \overline{X}_3, \overline{X}_1 \vee X_2 \vee X_3, X_1 \vee X_2 \vee X_3\}$.
Construct a graph with the terms of each clause as nodes.
Then connect all pairs of variable except their negation (partially done below)



If this graph contains a clique of size 3, the clause is satisfiable.

Some useful literature

J.A. Bondy and U.S.R. Murty, *Graph Theory with Applications*, (2nd Edition), North Holland, 1976.

Reinhard Diestel, *Graph Theory*, Graduate Texts in Mathematics, Vol. 173, Springer Verlag, Berlin, 1991.

Douglas West, *Introduction to Graph Theory*, (2nd Edition), Prentice Hall, 2000.

B. Bollobas, *Modern Graph Theory*, Springer-Verlag.

Fan Cheung and Linyuan Lu, *Complex Graphs and Networks*, Regional Conference Series in Mathematics, Vol. 107, AMS, 2004

Dieter Jungnickel, *Graphs, Networks and Algorithms*, Algorithms and Computation in Mathematics, Vol. 5, Springer Verlag, Berlin, 2005.