Introduction to Graph Theory

Definitions, Traversal, Analysis and Examples

Contents

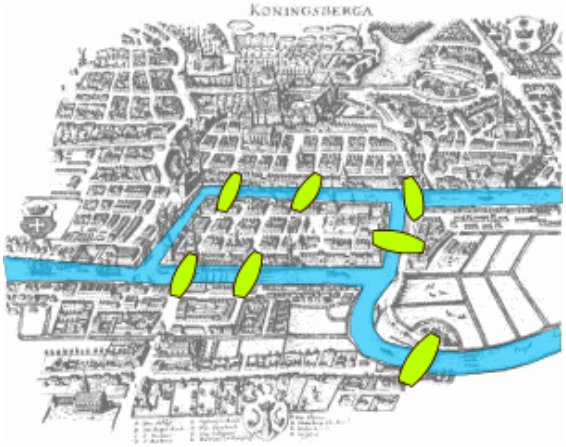
Chapter 1

Introduction

1.1 Seven Bridges of Königsberg

This article is about an abstract problem. For the historical group of bridges in the city once known as Königsberg, and those of them that still exist, see § Present state of the bridges.

The **Seven Bridges of Königsberg** is a historically notable problem in mathematics. Its negative resolution by Leonhard Euler in 1736 laid the foundations of graph theory and prefigured the idea of topology.[1]



Map of Königsberg in Euler's time showing the actual layout of the seven bridges, highlighting the river Pregel and the bridges

The city of Königsberg in Prussia (now Kaliningrad, Russia) was set on both sides of the Pregel River, and included two large islands which were connected to each other, or to the two mainland portions of the city, by seven bridges. The problem was to devise a walk through the city that would cross each of those bridges once and only once.

By way of specifying the logical task unambiguously, solutions involving either

1. reaching an island or mainland bank other than via one of the bridges, or

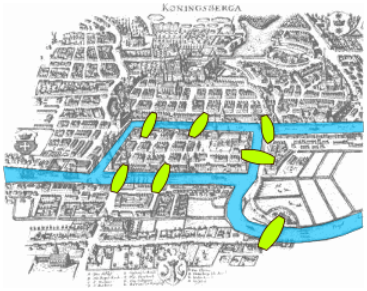
2. accessing any bridge without crossing to its other end

are explicitly unacceptable.

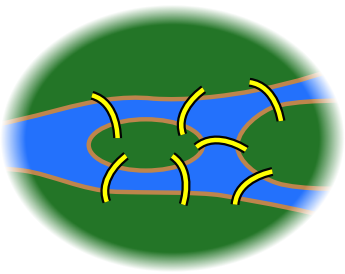
Euler proved that the problem has no solution. The difficulty he faced was the development of a suitable technique of analysis, and of subsequent tests that established this assertion with mathematical rigor.

1.1.1 Euler's analysis

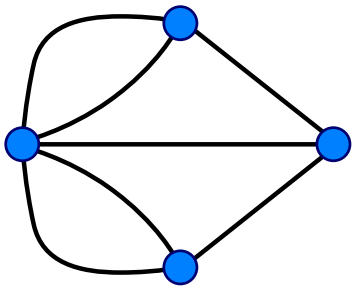
First, Euler pointed out that the choice of route inside each land mass is irrelevant. The only important feature of a route is the sequence of bridges crossed. This allowed him to reformulate the problem in abstract terms (laying the foundations of graph theory), eliminating all features except the list of land masses and the bridges connecting them. In modern terms, one replaces each land mass with an abstract "vertex" or node, and each bridge with an abstract connection, an "edge", which only serves to record which pair of vertices (land masses) is connected by that bridge. The resulting mathematical structure is called a graph.



→



→



Since only the connection information is relevant, the shape of pictorial representations of a graph may be distorted in any way, without changing the graph itself. Only the existence (or absence) of an edge between each pair of nodes is significant. For example, it does not matter whether the edges drawn are straight or curved, or whether one node is to the left or right of another.

Next, Euler observed that (except at the endpoints of the walk), whenever one enters a vertex by a bridge, one leaves the vertex by a bridge. In other words, during any walk in the graph, the number of times one enters a non-terminal vertex equals the number of times one leaves it. Now, if every bridge has been traversed exactly once, it follows that, for each land mass (except for the ones chosen for the start and finish), the number of bridges touching that land mass must be *even* (half of them, in the particular traversal, will be traversed "toward" the landmass; the other half, "away" from it). However, all four of the land masses in the original problem are touched by an *odd* number of bridges (one is touched by 5 bridges, and each of the other three is touched by 3). Since, at most, two land masses can serve as the endpoints of a walk, the proposition of a walk traversing each bridge once leads to a contradiction.

In modern language, Euler shows that the possibility of a walk through a graph, traversing each edge exactly once, depends on the degrees of the nodes. The degree of a node is the number of edges touching it. Euler's argument shows that a necessary condition for the walk of the desired form is that the graph be connected and have exactly zero or two nodes of odd degree. This condition turns out also to be sufficient—a result stated by Euler and later proven by Carl Hierholzer. Such a walk is now called an *Eulerian path* or *Euler walk* in his honor. Further, if there are nodes of odd degree, then any Eulerian path will start at one of them and end at the other. Since the graph corresponding to historical Königsberg has four nodes of odd degree, it cannot have an Eulerian path.

An alternative form of the problem asks for a path that traverses all bridges and also has the same starting and ending point. Such a walk is called an *Eulerian circuit* or an *Euler tour*. Such a circuit exists if, and only if, the graph is connected, and there are no nodes of odd degree at all. All Eulerian circuits are also Eulerian paths, but not all Eulerian paths are Eulerian circuits.

Euler's work was presented to the St. Petersburg Academy on 26 August 1735, and published as *Solutio problematis ad geometriam situs pertinentis* (The solution of a problem relating to the geometry of position) in the journal *Commentarii academiae scientiarum Petropolitanae* in 1741.[2] It is available in English in The World of Mathematics.

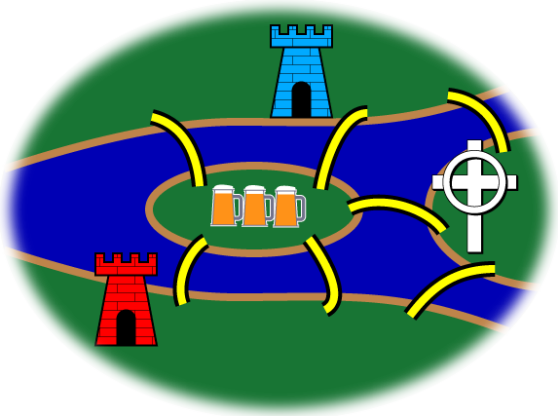
1.1.2 Significance in the history of mathematics

In the history of mathematics, Euler's solution of the Königsberg bridge problem is considered to be the first theorem of graph theory and the first true proof in the theory of networks,[3] a subject now generally regarded as a branch of combinatorics. Combinatorial problems of other types had been considered since antiquity.

In addition, Euler's recognition that the key information was the number of bridges and the list of their endpoints (rather than their exact positions) presaged the development of topology. The difference between the actual layout and the graph schematic is a good example of the idea that topology is not concerned with the rigid shape of objects.

1.1.3 Variations

The classic statement of the problem, given above, uses **unidentified** nodes—that is, they are all alike except for the way in which they are connected. There is a variation in which the nodes are **identified**—each node is given a unique name or color.



A variant with red and blue castles, a church and an inn.

The northern bank of the river is occupied by the *Schloß*, or castle, of the Blue Prince; the southern by that of the Red Prince. The east bank is home to the Bishop's *Kirche*, or church; and on the small island in the center is a *Gasthaus*, or inn.

It is understood that the problems to follow should be taken in order, and begin with a statement of the original problem:

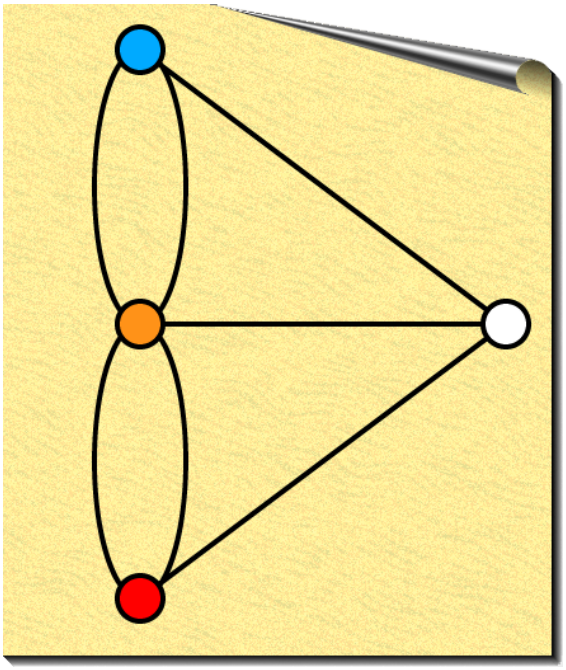
It being customary among the townsmen, after some hours in the *Gasthaus*, to attempt to **walk the bridges**, many have returned for more refreshment claiming success. However, none have been able to repeat the feat by the light of day.

Bridge 8: The Blue Prince, having analyzed the town's bridge system by means of graph theory, concludes that the bridges cannot be walked. He contrives a stealthy plan to build an eighth bridge so that he can begin in the evening at his *Schloß*, walk the bridges, and end at the *Gasthaus* to brag of his victory. Of course, he wants the Red Prince to be unable to duplicate the feat from the Red Castle. *Where does the Blue Prince build the eighth bridge?*

Bridge 9: The Red Prince, infuriated by his brother's Gordian solution to the problem, wants to build a ninth bridge, enabling *him* to begin at his *Schloß*, walk the bridges, and end at the *Gasthaus* to rub dirt in his brother's face. As an extra bit of revenge, his brother should then no longer be able to walk the bridges starting at his *Schloß* and ending at the *Gasthaus* as before. *Where does the Red Prince build the ninth bridge?*

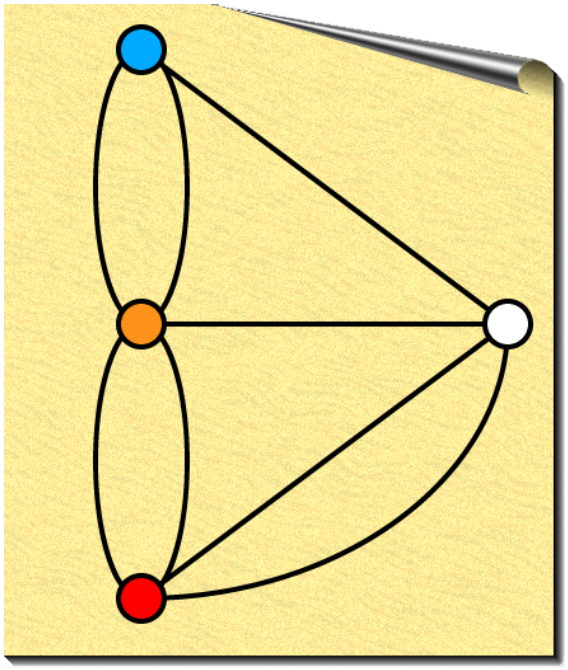
Bridge 10: The Bishop has watched this furious bridge-building with dismay. It upsets the town's *Weltanschauung* and, worse, contributes to excessive drunkenness. He wants to build a tenth bridge that allows *all* the inhabitants to walk the bridges and return to their own beds. *Where does the Bishop build the tenth bridge?*

Solutions



The colored graph

Reduce the city, as before, to a graph. Color each node.



The 8th edge

As in the classic problem, no Euler walk is possible; coloring does not affect this. All four nodes have an odd number of edges.

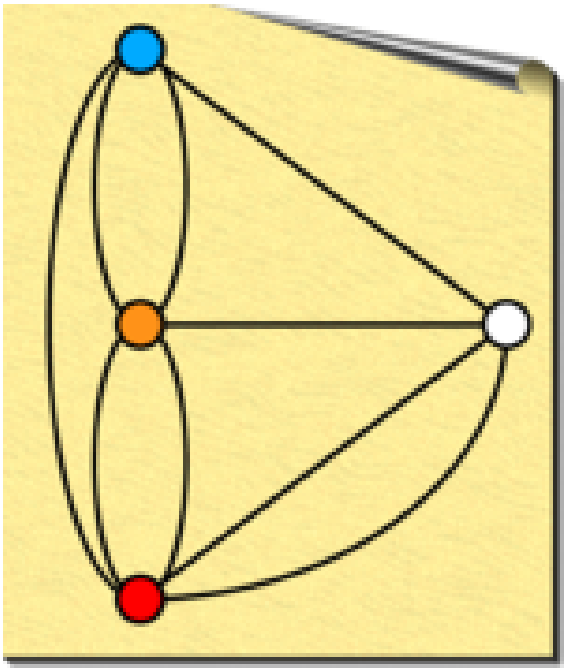
Bridge 8: Euler walks are possible if exactly zero or two nodes have an odd number of edges. If we have 2 nodes with an odd number of edges, the walk must begin at one such node and end at the other. Since there are only 4 nodes in the puzzle, the solution is simple. The walk desired must begin at the blue node and end at the orange node. Thus, a new edge is drawn between the other two nodes. Since they each formerly had an odd number of edges, they must now have an even number of edges, fulfilling all conditions. This is a change in parity from an odd to even degree.

Bridge 9: The 9th bridge is easy once the 8th is solved. The desire is to enable the red castle and forbid the blue castle as a starting point; the orange node remains the end of the walk and the white node is unaffected. To change the parity of both red and blue nodes, draw a new edge between them.

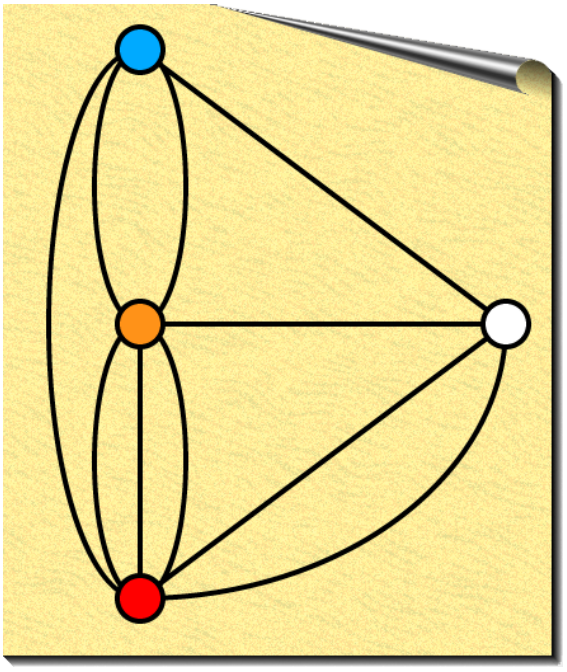
Bridge 10: The 10th bridge takes us in a slightly different direction. The Bishop wishes every citizen to return to his starting point. This is an Euler circuit and requires that all nodes be of even degree. After the solution of the 9th bridge, the red and the orange nodes have odd degree, so their parity must be changed by adding a new edge between them.

1.1.4 Present state of the bridges

Two of the seven original bridges did not survive the bombing of Königsberg in World War II. Two others were



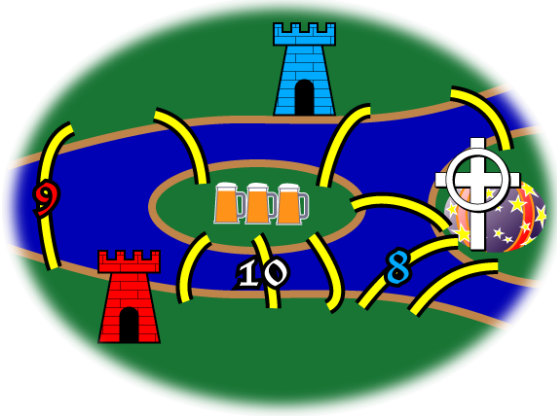
The 9th edge



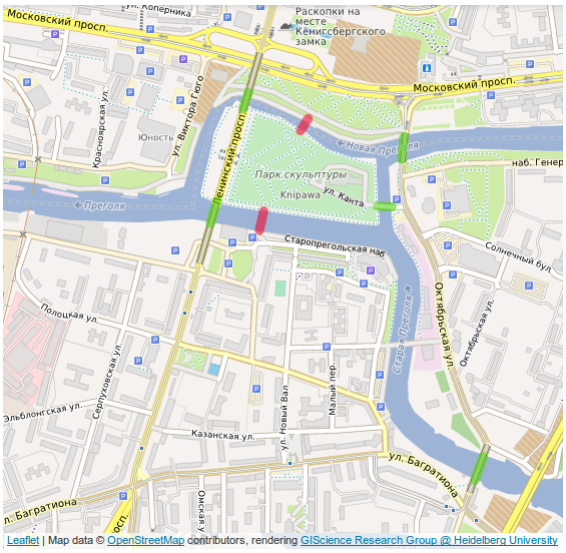
The 10th edge

later demolished and replaced by a modern highway. The three other bridges remain, although only two of them are from Euler's time (one was rebuilt in 1935).[4] Thus, as of 2000, there are five bridges in Kaliningrad that were a part of the Euler's problem.

In terms of graph theory, two of the nodes now have degree 2, and the other two have degree 3. Therefore, an Eulerian path is now possible, but it must begin on one island and end on the other.[5]



8th, 9th, and 10th bridges

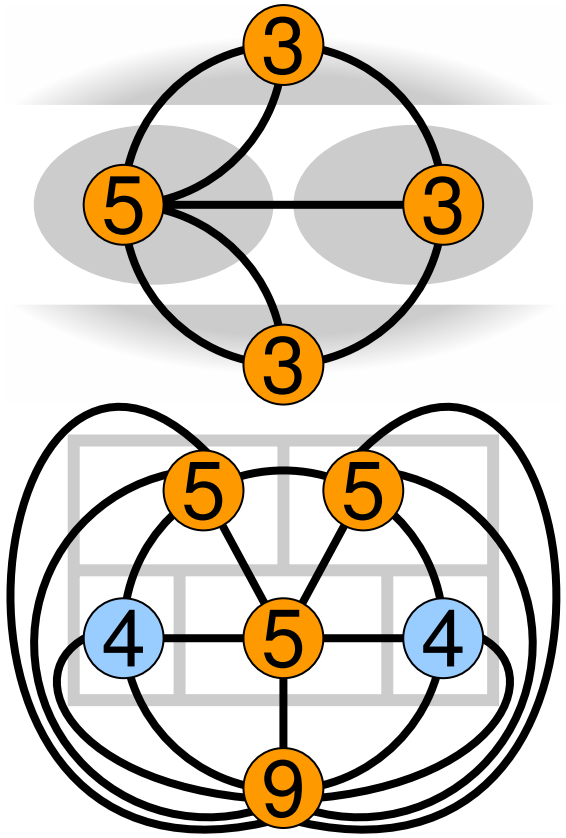


Modern map of Kaliningrad. Locations of the remaining bridges are highlighted in green, while those destroyed are highlighted in red.

Canterbury University in Christchurch, New Zealand, has incorporated a model of the bridges into a grass area between the old Physical Sciences Library and the Erskine Building, housing the Departments of Mathematics, Statistics and Computer Science.[6] The rivers are replaced with short bushes and the central island sports a stone tōrō. Rochester Institute of Technology has incorporated the puzzle into the pavement in front of the Gene Polisseni Center, an ice hockey arena that opened in 2014.[7]

1.1.5 See also

- Eulerian path
- Five room puzzle
- Glossary of graph theory
- Hamiltonian path



Comparison of the graphs of the Seven bridges of Konigsberg (top) and Five-room puzzles (bottom). The numbers denote the number of edges connected to each node. Nodes with an odd number of edges are shaded orange.

- Icosian game
- Water, gas, and electricity

1.1.6 References

[1] See Shields, Rob (December 2012). 'Cultural Topology: The Seven Bridges of Königsburg 1736' in Theory Culture and Society 29. pp.43-57 and in versions online for a discussion of the social significance of Euler's engagement with this popular problem and its significance as an example of (proto-)topological understanding applied to everyday life.

[2] The Euler Archive, commentary on publication, and original text, in Latin.

[3] Newman, M. E. J. *The structure and function of complex networks* (PDF). Department of Physics, University of Michigan.

[4] Taylor, Peter (December 2000). "What *Ever* Happened to Those Bridges?". Australian Mathematics Trust. Archived from the original on 19 March 2012. Retrieved 11 November 2006.

[5] Stallmann, Matthias (July 2006). "The 7/5 Bridges of Koenigsberg/Kaliningrad". Retrieved 11 November 2006.

[6] "About – Mathematics and Statistics – University of Canterbury". *math.canterbury.ac.nz*. Retrieved 4 November 2010.

[7] https://twitter.com/ritwhky/status/501529429185945600

1.1.7 External links

- Kaliningrad and the Konigsberg Bridge Problem at Convergence
- Euler's original publication (in Latin)
- The Bridges of Königsberg
- How the bridges of Königsberg help to understand the brain
- Euler's Königsberg's Bridges Problem at Math Dept. Contra Costa College
- Pregel – A Google graphing tool named after this problem

Coordinates: 54°42′12″N 20°30′56″E / 54.70333°N 20.51556°E

1.2 Glossary of graph theory

This is a **glossary of graph theory terms**. Graph theory is the study of graphs, systems of nodes or vertices connected in pairs by edges.

Contents :

- M
- N
- O
- P
- Q
- R
- S
- T
- U
- V
- W
- X
- Y
- Z
- See also
- References

1.2.1 !$@

[] $G[S]$ is the *induced subgraph* of a graph G for vertex subset S.. prime symbol

′ The prime symbol is often used to modify notation for graph invariants so that it applies to the line graph instead of the given graph. For instance, $\alpha(G)$ is the independence number of a graph; $\alpha'(G)$ is the matching number of the graph, which equals the independence number of its line graph. Similarly, $\chi(G)$ is the chromatic number of a graph; $\chi'(G)$ is the chromatic index of the graph, which equals the chromatic number of its line graph.

1.2.2 A

achromatic The achromatic number of a graph is the maximum number of colors in a complete coloring.[1]

acyclic 1. A graph is acyclic if it has no cycles. An acyclic undirected graph is the same thing as a forest. Acyclic directed graphs are more often called directed acyclic graphs.[2]

2. An acyclic coloring of an undirected graph is a proper coloring in which every two color classes induce a forest.[3]

adjacency matrix The adjacency matrix of a graph is a matrix whose rows and columns are both indexed by vertices of the graph, with a one in the cell for row i and column j when vertices i and j are adjacent, and a zero otherwise.[4]

adjacent The relation between two vertices that are both endpoints of the same edge.[2]

α For a graph G, $\alpha(G)$ (using the Greek letter alpha) is its independence number (see *independent*), and $\alpha'(G)$ is its matching number (see *matching*).

alternating In a graph with a matching, an alternating path is a path whose edges alternate between matched and unmatched edges. An alternating cycle is, similarly, a cycle whose edges alternate between matched and unmatched edges. An augmenting path is an alternating path that starts and ends at unsaturated vertices. A larger matching can be found as the symmetric difference of the matching and the augmenting path; a matching is maximum if and only if it has no augmenting path.

anti-edge Synonym for *non-edge*, a pair of non-adjacent vertices.

anti-triangle A three-vertex independent set, the complement of a triangle.

apex 1. An apex graph is a graph in which one vertex can be removed, leaving a planar subgraph. The removed vertex is called the apex. A k-apex graph is a graph that can be made planar by the removal of k vertices.

2. Synonym for universal vertex, a vertex adjacent to all other vertices.

arborescence Synonym for a rooted and directed tree; see *tree*.

arc See *edge*.

arrow An ordered pair of *vertices*, such as an *edge* in a *directed graph*. An arrow (x, y) has a *tail* x, a *head* y, and a *direction* from x to y; y is said to be the *direct successor* to x and x the *direct predecessor* to y. The arrow (y, x) is the *inverted arrow* of the arrow (x, y).

articulation point A *vertex* in a *connected graph* whose removal would *disconnect* the graph.

-ary A k-ary tree is a rooted tree in which every internal vertex has no more than k children. A 1-ary tree is just a path. A 2-ary tree is also called a binary tree, although that term more properly refers to 2-ary trees in which the children of each node are distinguished as being left or right children (with at most one of each type). A k-ary tree is said to be complete if every internal vertex has exactly k children.

augmenting A special type of alternating path; see *alternating*.

automorphism A graph automorphism is a symmetry of a graph, an isomorphism from the graph to itself.

1.2.3 B

bag One of the sets of vertices in a *tree decomposition*.

balanced A bipartite or multipartite graph is balanced if each two subsets of its vertex partition have sizes within one of each other.

bandwidth The bandwidth of a graph G is the minimum, over all orderings of vertices of G, of the length of the longest edge (the number of steps in the ordering between its two endpoints). It is also one less than the size of the maximum clique in a proper interval completion of G, chosen to minimize the clique size.

biclique Synonym for complete bipartite graph or complete bipartite subgraph; see *complete*.

biconnected Synonym for 2-vertex-connected. See *connected*; for biconnected components, see *component*.

bipartite A bipartite graph is a graph with no odd cycles; equivalently, it is a graph that may be properly colored with two colors. Bipartite graphs are often written $G = (U,V,E)$ where U and V are the subsets of vertices of each color. However, unless the graph is connected, it may not have a unique 2-coloring.

biregular A biregular graph is one in which there are only two different vertex degrees.

block 1. A block or biconnected component is a maximal subgraph in which every two vertices or edges belong to a simple cycle. It may be a 2-vertex-connected subgraph, a bridge edge, or an isolated vertex. In a connected graph, the collection of blocks and the articulation points separating them form the vertices of a tree, the block-cut tree, whose edges connect blocks to the articulation points within those blocks. The block graph of a graph G is another graph whose vertices are the blocks of G, with an edge connecting two vertices when the corresponding blocks share an articulation point; that is, it is the intersection graph of the blocks of G.

2. A block graph (also called a clique tree, and sometimes erroneously called a Husimi tree) is a graph all of whose blocks are complete. The block graph of any graph is a block graph, and every block graph may be constructed as the block graph of a graph.

bond A *minimal cut-set*: a set of edges whose removal disconnects the graph, for which no proper subset has the same property.

book 1. A book, book graph, or triangular book is a complete tripartite graph $K_{1,1,n}$; a collection of n triangles joined at a shared edge.

2. Another type of graph, also called a book, or a quadrilateral book, is a collection of 4-cycles joined at a shared edge; the Cartesian product of a star with an edge.

3. A book embedding is an embedding of a graph onto a topological book, a space formed by joining a collection of half-planes along a shared line. Usually, the vertices of the embedding are required to be on the line, which is called the spine of the embedding, and the edges of the embedding are required to lie within a single half-plane, one of the pages of the book.

bramble A collection of mutually touching connected subgraphs, where two subgraphs touch if they share a vertex or each includes one endpoint of an edge. The order of a bramble is the smallest size of a set of vertices that has a nonempty intersection with all of the subgraphs. The treewidth of a graph is the maximum order of any of its brambles.

branch-decomposition A branch-decomposition of G is a hierarchical clustering of the edges of G, represented by an unrooted binary tree with its leaves labeled by the edges of G. The width of a branch-decomposition is the maximum, over edges e of this binary tree, of the number of shared vertices between the subgraphs determined by the edges of G in the two subtrees separated by e. The branch-width of G is the minimum width of any branch-decomposition of G.

branchwidth See *branch-decomposition*.

bridge 1. A bridge, isthmus, or cut edge is an edge whose removal would disconnect the graph. A bridgeless graph is one that has no bridges; equivalently, a 2-edge-connected graph.

2. Especially in the context of planarity testing, a bridge of a cycle is a maximal subgraph that is disjoint from the cycle and in which each two edges belong to a path that is internally disjoint from the cycle. A chord is a one-edge bridge. A peripheral cycle is a cycle with at most one bridge; it must be a face in any planar embedding of its graph.

3. Bridged graph, a graph in which every cycle of four or more vertices has a shortcut, a pair of vertices closer in the graph than they are in the cycle.

bridgeless A bridgeless graph is a graph that has no bridge edges; that is, a 2-edge-connected graph.

butterfly 1. The butterfly graph has five vertices and six edges; it is formed by two triangles that share a vertex.

2. The butterfly network is a graph used as a network architecture in distributed computing, closely related to the cube-connected cycles.

1.2.4 C

C *Cn* is an n-vertex cycle graph; see *cycle*.

cactus A cactus graph, cactus tree, cactus, or Husimi tree is a connected graph in which each edge belongs to at most one cycle. Its blocks are cycles or single edges. If, in addition, each vertex belongs to at most two blocks, then it is called a Christmas cactus.

cage A cage is a regular graph with the smallest possible order for its girth.

canonical

canonization A canonical form of a graph is an invariant such that two graphs have equal invariants if and only if they are isomorphic. Canonical forms may also be called canonical invariants or complete invariants, and are sometimes defined only for the graphs within a particular family of graphs. Graph canonization is the process of computing a canonical form.

card A graph formed from a given graph by deleting one vertex, especially in the context of the reconstruction conjecture. See also *deck*, the multiset of all cards of a graph.

carving width Carving width is a notion of graph width analogous to branchwidth, but using hierarchical clusterings of vertices instead of hierarchical clusterings of edges.

caterpillar A caterpillar tree or caterpillar is a tree in which the internal nodes induce a path.

center The center of a graph is the set of vertices of minimum *eccentricity*.

chain 1. Synonym for *walk*.

2. When applying methods from algebraic topology to graphs, an element of a chain complex, namely a set of vertices or a set of edges.

Cheeger constant See *expansion*.

χ $\chi(G)$ (using the Greek letter chi) is the chromatic number of G and $\chi'(G)$ is its chromatic index; see *chromatic* and *coloring*.

child In a rooted tree, a child of a vertex v is a neighbor of v along an outgoing edge, one that is directed away from the root.

chord

chordal 1. A chord of a cycle is an edge that does not belong to the cycle, for which both endpoints belong to the cycle.

2. A chordal graph is a graph in which every cycle of four or more vertices has a chord, so the only induced cycles are triangles.

3. A strongly chordal graph is a chordal graph in which every cycle of length six or more has an odd chord.

4. A chordal bipartite graph is not chordal (unless it is a forest); it is a bipartite graph in which every cycle of six or more vertices has a chord, so the only induced cycles are 4-cycles.

5. A chord of a circle is a line segment connecting two points on the circle; the intersection graph of a collection of chords is called a circle graph.

chromatic Having to do with coloring; see *color*. Chromatic graph theory is the theory of graph coloring. The chromatic number $\chi(G)$ is the minimum number of colors needed in a proper coloring of G. $\chi'(G)$ is the chromatic index of G, the minimum number of colors needed in a proper edge coloring of G.

choosable

choosability A graph is k-choosable if it has a list coloring whenever each vertex has a list of k available colors. The choosability of the graph is the smallest k for which it is k-choosable.

circle A circle graph is the intersection graph of chords of a circle.

circuit A circuit may refer to a simple cycle, a trail (a closed tour without repeated edges), or an element of the cycle space (an Eulerian spanning subgraph). The circuit rank of a graph is the dimension of its cycle space.

circumference The circumference of a graph is the length of its longest simple cycle. The graph is Hamiltonian if and only if its circumference equals its order.

class 1. A class of graphs or family of graphs is a (usually infinite) collection of graphs, often defined as the graphs having some specific property. The word "class" is used rather than "set" because, unless special restrictions are made (such as restricting the vertices to be drawn from a particular set, and defining edges to be sets of two vertices) classes of graphs are usually not sets when formalized using set theory.

2. A color class of a colored graph is the set of vertices or edges having one particular color.

3. In the context of Vizing's theorem, on edge coloring simple graphs, a graph is said to be of class one if its chromatic index equals its maximum degree, and class two if its chromatic index equals one plus the degree. According to Vizing's theorem, all simple graphs are either of class one or class two.

claw A claw is a tree with one internal vertex and three leaves, or equivalently the complete bipartite graph $K_{1,3}$. A claw-free graph is a graph that does not have an induced subgraph that is a claw.

clique A clique is usually a complete subgraph, but some sources define it as a maximal complete subgraph, one that is not part of any larger complete subgraph. A k-clique is a clique of order k. The clique number $\kappa(G)$ of a graph G is the order of its largest clique. A clique graph is an intersection graph of maximal cliques. See also *biclique*, a complete bipartite subgraph.

clique tree A synonym for a *block graph*.

clique-width The clique-width of a graph G is the minimum number of distinct labels needed to construct G by operations that create a labeled vertex, form the disjoint union of two labeled graphs, add an edge connecting all pairs of vertices with given labels, or relabel all vertices with a given label. The graphs of clique-width at most 2 are exactly the cographs.

closed 1. A closed neighborhood is one that includes its central vertex; see *neighbourhood*.

2. A closed walk is one that starts and ends at the same vertex; see *walk*.

3. A graph is transitively closed if it equals its own transitive closure; see *transitive*.

4. A graph property is closed under some operation on graphs if, whenever the argument or arguments to the operation have the property, then so does the result. For instance, hereditary properties are closed under induced subgraphs; monotone properties are closed under subgraphs; and minor-closed properties are closed under minors.

closure 1. For the transitive closure of a directed graph, see *transitive*.

2. A closure of a directed graph is a set of vertices that have no outgoing edges to vertices outside the closure. For instance, a sink is a one-vertex closure. The closure problem is the problem of finding a closure of minimum or maximum weight.

co- This prefix has various meanings usually involving complement graphs. For instance, a cograph is a graph produced by operations that include complementation; a cocoloring is a coloring in which each vertex induces either an independent set (as in proper coloring) or a clique (as in a coloring of the complement).

color

coloring 1. A graph coloring is a labeling of the vertices of a graph by elements from a given set of colors, or equivalently a partition of the vertices into subsets, called "color classes", each of which is associated with one of the colors.

2. Some authors use "coloring", without qualification, to mean a proper coloring, one that assigns different colors to the endpoints of each edge. In graph coloring, the goal is to find a proper coloring that uses as few colors as possible; for instance, bipartite graphs are the graphs that have colorings with only two colors, and the four color theorem states that every planar graph can be colored with at most four colors. A graph is said to be k-colored if it has been (properly) colored with k colors, and k-colorable or k-chromatic if this is possible.

3. Many variations of coloring have been studied, including edge coloring (coloring edges so that no two edges with the same endpoint share a color), list coloring (proper coloring with each vertex restricted to a subset of the available colors), acyclic coloring (every 2-colored subgraph is acyclic), co-coloring (every color class induces an independent set or a clique), complete coloring (every two color classes share an edge), and total coloring (both edges and vertices are colored).

4. The coloring number of a graph is one plus the degeneracy. It is so called because applying a greedy coloring algorithm to a degeneracy ordering of the graph uses at most this many colors.

comparability An undirected graph is a comparability graph if its vertices are the elements of a partially ordered set and two vertices are adjacent when they are comparable in the partial order. Equivalently, a comparability graph is a graph that has a transitive orientation. Many other classes of graphs can be defined as the comparability graphs of special types of partial order.

complement The complement graph $\bar{G}$ of a simple graph G is another graph on the same vertex set as G, with an edge for each two vertices that are not adjacent in G.

complete 1. A complete graph is one in which every two vertices are adjacent: all edges that could exist are present. A complete graph with n vertices is often denoted K_n. A complete bipartite graph is one in which every two vertices on opposite sides of the partition of vertices are adjacent. A complete bipartite graph with a vertices on one side of

the partition and b vertices on the other side is often denoted Ka,b. The same terminology and notation has also been extended to complete multipartite graphs, graphs in which the vertices are divided into more than two subsets and every pair of vertices in different subsets are adjacent; if the numbers of vertices in the subsets are a, b, c, ... then this graph is denoted $Ka, b, c, ...$

2. A completion of a given graph is a supergraph that has some desired property. For instance, a chordal completion is a supergraph that is a chordal graph.

3. A complete matching is a synonym for a perfect matching; see *matching*.

4. A complete coloring is a proper coloring in which each pairs of colors is used for the endpoints of at least one edge. Every coloring with a minimum number of colors is complete, but there may exist complete colorings with larger numbers of colors. The achromatic number of a graph is the maximum number of colors in a complete coloring.

5. A complete invariant of a graph is a synonym for a canonical form, an invariant that has different values for non-isomorphic graphs.

component A connected component of a graph is a maximal connected subgraph. The term is also used for maximal subgraphs or subsets of a graph's vertices that have some higher order of connectivity, including biconnected components, triconnected components, and strongly connected components.

condensation The condensation of a directed graph G is a directed acyclic graph with one vertex for each strongly connected component of G, and an edge connecting pairs of components that contain the two endpoints of at least one edge in G.

cone A graph that contains a universal vertex.

connect Cause to be *connected*.

connected A connected graph is one in which each pair of vertices forms the endpoints of a path. Higher forms of connectivity include strong connectivity in directed graphs (for each two vertices there are paths from one to the other in both directions), k-vertex-connected graphs (removing fewer than k vertices cannot disconnect the graph), and k-edge-connected graphs (removing fewer than k edges cannot disconnect the graph).

converse The converse graph is a synonym for the transpose graph; see *transpose*.

core 1. A k-core is the induced subgraph formed by removing all vertices of degree less than k, and all vertices whose degree becomes less than k after earlier removals. See *degeneracy*.

2. A core is a graph G such that every graph homomorphism from G to itself is an isomorphism.

3. The core of a graph G is a minimal graph H such that there exist homomorphisms from G to H and vice versa. H is unique up to isomorphism. It can be represented as an induced subgraph of G, and is a core in the sense that all of its self-homomorphisms are isomorphisms.

4. In the theory of graph matchings, the core of a graph is an aspect of its Dulmage–Mendelsohn decomposition, formed as the union of all maximum matchings.

cotree 1. The complement of a spanning tree.

2. A rooted tree structure used to describe a cograph, in which each cograph vertex is a leaf of the tree, each internal node of the tree is labeled with 0 or 1, and two cograph vertices are adjacent if and only if their lowest common ancestor in the tree is labeled 1.

cover A vertex cover is a set of vertices incident to every edge in a graph. An edge cover is a set of edges incident to every vertex in a graph.

critical A critical graph for a given property is a graph that has the property but such that every subgraph formed by deleting a single vertex does not have the property. For instance, a factor-critical graph is one that has a perfect matching (a 1-factor) for every vertex deletion, but (because it has an odd number of vertices) has no perfect matching itself. Compare *hypo-*, used for graphs which do not have a property but for which every one-vertex deletion does.

cube

cubic 1. Cube graph, the eight-vertex graph of the vertices and edges of a cube.

2. Hypercube graph, a higher-dimensional generalization of the cube graph.

3. Folded cube graph, formed from a hypercube by adding a matching connecting opposite vertices.

4. Halved cube graph, the half-square of a hypercube graph.

5. Partial cube, a distance-preserving subgraph of a hypercube.

6. The cube of a graph G is the graph power G^3.

7. Cubic graph, another name for a 3-regular graph, one in which each vertex has three incident edges.

8. Cube-connected cycles, a cubic graph formed by replacing each vertex of a hypercube by a cycle.

cut

cut-set A cut is a partition of the vertices of a graph into two subsets. An edge is said to span the cut if it has endpoints in both subsets, and a cut-set is the set of edges that span a cut. Thus, the removal of the cut-set from a connected graph disconnects it.

cut point See *articulation point*.

cut space The cut space of a graph is a GF(2)-vector space having the *cut-sets of the graph as its elements and symmetric difference of sets as its vector addition operation.*

cycle A cycle may either refer to a closed walk (also called a tour) or more specifically to a closed walk without repeated vertices or edges (a simple cycle). In either case, the choice of starting vertex is usually considered unimportant: cyclic permutations of the walk produce the same cycle. Important special cases of cycles include Hamiltonian cycles, induced cycles, peripheral cycles, and the shortest cycle, which defines the girth of a graph. A k-cycle is a cycle of length k; for instance a 2-cycle is a digon and a 3-cycle is a triangle. A cycle graph is a graph that is itself a simple cycle; a cycle graph with n vertices is commonly denoted C_n. The cycle space is a vector space generated by simple cycles in a graph.

1.2.5 D

DAG Abbreviation for directed acyclic graph, a directed graph without any directed cycles.

deck The multiset of graphs formed from a single graph G by deleting a single vertex in all possible ways, especially in the context of the reconstruction conjecture. An edge-deck is formed in the same way by deleting a single edge in all possible ways. The graphs in a deck are also called *cards*. See also *critical* (graphs that have a property that is not held by any card) and *hypo-* (graphs that do not have a property that is held by all cards).

decomposition See *tree decomposition*, *path decomposition*, or *branch-decomposition*.

degenerate

degeneracy A k-degenerate graph is an undirected graph in which every induced subgraph has minimum degree at most k. The degeneracy of a graph is the smallest k for which it is k-degenerate. A degeneracy ordering is an ordering of the vertices such that each vertex has minimum degree in the induced subgraph of it and all later vertices; in a degeneracy ordering of a k-degenerate graph, every vertex has at most k later neighbours. Degeneracy is also known as the k-core number, width, and linkage, and one plus the degeneracy is also called the coloring number or Szekeres–Wilf number. k-degenerate graphs have also been called k-inductive graphs.

degree 1. The degree of a vertex in a graph is its number of incident edges.[2] The degree of a graph G (or its maximum degree) is the maximum of the degrees of its vertices, often denoted $\Delta(G)$; the minimum degree of G is the minimum of its vertex degrees, often denoted $\delta(G)$. Degree is sometimes called *valency*; the degree of v in G may be denoted $dG(v)$, $d(G)$, or $\deg(v)$. The total degree is the sum of the degrees of all vertices; by the handshaking lemma it is an even number. The degree sequence is the collection of degrees of all vertices, in sorted order from largest to smallest. In a directed graph, one may distinguish the in-degree (number of incoming edges) and out-degree (number of outgoing edges).[2]

2. The homomorphism degree of a graph is a synonym for its *Hadwiger number*, the order of the largest clique minor.

Δ, δ $\Delta(G)$ (using the Greek letter delta) is the maximum degree of a vertex in G, and $\delta(G)$ is the minimum degree; see *degree*.

diameter The diameter of a connected graph is the maximum length of a shortest path. That is, it is the maximum of the distances between pairs of vertices in the graph. If the graph has weights on its edges, then its weighted diameter measures path length by the sum of the edge weights along a path, while the unweighted diameter measures path length by the number of edges. For disconnected graphs, definitions vary: the diameter may be defined as infinite, or as the largest diameter of a connected component, or it may be undefined.

diamond The diamond graph is an undirected graph with four vertices and five edges.

diconnected *strongly* connected.

digon A digon is a simple cycle of length two in a directed graph or a multigraph. Digons cannot occur in simple undirected graphs, as forming a closed walk by repeating the same edge twice does not produce a simple cycle.

digraph Synonym for directed graph.[2]

dipath See *directed path*.

direct predecessor The starting endpoint of a directed edge to the given vertex.

direct successor The final endpoint of a directed edge that starts at the given vertex.

directed A directed graph is one in which the edges have a distinguished direction, from one vertex to another.[2] In a mixed graph, a directed edge is again one that has a distinguished direction; directed edges may also be called arcs or arrows.

directed arc See *arrow*.

directed edge See *arrow*.

directed line See *arrow*.

directed path A *path* in which all the *edges have the same* direction. *If a directed path leads from* vertex x *to vertex* y, x *is a* predecessor *of* y, y *is a* successor *of* x, *and* y *is said to be* reachable *from* x.

direction 1. The asymmetric relation between two *adjacent vertices* in a *graph*, represented as an *arrow*.

2. The asymmetric relation between two vertices in a *directed path*.

disconnect Cause to be *disconnected*.

disconnected Not *connected*.

disjoint 1. Two subgraphs are edge disjoint if they share no edges, and vertex disjoint if they share no vertices.

2. The disjoint union of two or more graphs is a graph whose vertex and edge sets are the disjoint unions of the corresponding sets.

distance The distance between any two vertices in a graph is the length of the shortest path having the two vertices as its endpoints.

domatic A domatic partition of a graph is a partition of the vertices into dominating sets. The domatic number of the graph is the maximum number of dominating sets in such a partition.

dominating A dominating set is a set of vertices that includes or is adjacent to every vertex in the graph; not to be confused with a vertex cover, a vertex set that is incident to all edges in the graph. Important special types of dominating sets include independent dominating sets (dominating sets that are also independent sets) and connected dominating sets (dominating sets that induced connected subgraphs). A single-vertex dominating set may also be called a universal vertex. The domination number of a graph is the number of vertices in the smallest dominating set.

1.2.6 E

E $E(G)$ is the edge set of G; see *edge set*.

ear An ear of a graph is a path whose endpoints may coincide but in which otherwise there are no repetitions of vertices or edges.

ear decomposition An ear decomposition is a partition of the edges of a graph into a sequence of ears, each of whose endpoints (after the first one) belong to a previous ear and each of whose interior points do not belong to any previous ear. An open ear is a simple path (an ear that does not repeat any vertices), and an open ear decomposition is an ear decomposition in which each ear after the first is open; a graph has an open ear decomposition if and only if it is biconnected. An ear is odd if it has an odd number of edges, and an odd ear decomposition is an ear decomposition in which each ear is odd; a graph has an odd ear decomposition if and only if it is factor-critical.

eccentricity The eccentricity of a vertex is the farthest distance from it to any other vertex.

edge An edge is (together with vertices) one of the two basic units out of which graphs are constructed. Each edge has two (or in hypergraphs, more) vertices to which it is attached, called its endpoints. Edges may be directed or undirected; undirected edges are also called lines and directed edges are also called arcs or arrows. In an undirected simple graph, an edge may be represented as the set of its vertices, and in a directed simple graph it may be represented as an ordered pair of its vertices. An edge that connects vertices x and y is sometimes written xy.

edge cut A set of *edges whose removal* disconnects *the* graph. *A one-edge cut is called a* bridge, isthmus, *or* cut edge.

edge set The set of edges of a given graph G, sometimes denoted by $E(G)$.

edgeless graph The edgeless graph or totally disconnected graph on a given set of vertices is the graph that has no edges. It is sometimes called the empty graph, but this term can also refer to a graph with no vertices.

embedding A graph embedding is a topological representation of a graph as a subset of a topological space with each vertex represented as a point, each edge represented as a curve having the endpoints of the edge as endpoints of the curve, and no other intersections between vertices or edges. A planar graph is a graph that has such an embedding onto the Euclidean plane, and a toroidal graph is a graph that has such an embedding onto a torus. The genus of a graph is the minimum possible genus of a two-dimensional manifold onto which it can be embedded.

empty graph 1. An edgeless graph on a set of vertices.

2. The graph with no vertices and no edges.

end An end of an infinite graph is an equivalence class of rays, where two rays are equivalent if there is a third ray that includes infinitely many vertices from both of them.

endpoint One of the two vertices incident to a given edge, or one of the two vertices at the start and end of a walk or path.

enumeration Graph enumeration is the problem of counting the graphs in a given class of graphs, as a function of their order. More generally, enumeration problems can refer either to problems of counting a certain class of combinatorial objects (such as cliques, independent sets, colorings, or spanning trees), or of algorithmically listing all such objects.

Eulerian An Eulerian path is a walk that uses every edge of a graph exactly once. An Eulerian circuit (also called an Eulerian cycle or an Euler tour) is a closed walk that uses every edge exactly once. An Eulerian graph is a graph that has an Eulerian circuit. For an undirected graph, this means that the graph is connected and every vertex has even degree. For a directed graph, this means that the graph is strongly connected and every vertex has in-degree equal to the out-degree. In some cases, the connectivity requirement is loosened, and a graph meeting only the degree requirements is called Eulerian.

even Divisible by two; for instance, an even cycle is a cycle whose length is even.

expander An expander graph is a graph whose edge expansion, vertex expansion, or spectral expansion is bounded away from zero.

expansion 1. The edge expansion, isoperimetric number, or Cheeger constant of a graph G is the minimum ratio, over subsets S of at most half of the vertices of G, of the number of edges leaving S to the number of vertices in S.

2. The vertex expansion, vertex isoperimetric number, or magnification of a graph G is the minimum ratio, over subsets S of at most half of the vertices of G, of the number of vertices outside but adjacent to S to the number of vertices in S.

3. The unique neighbor expansion of a graph G is the minimum ratio, over subsets of at most half of the vertices of G, of the number of vertices outside S but adjacent to a unique vertex in S to the number of vertices in S.

4. The spectral expansion of a d-regular graph G is the spectral gap between the largest eigenvalue d of its adjacency matrix and the second-largest eigenvalue.

5. A family of graphs has bounded expansion if all its r-shallow minors have a ratio of edges to vertices bounded by a function of r, and polynomial expansion if the function of r is a polynomial.

1.2.7 F

factor A factor of a graph is a spanning subgraph: a subgraph that includes all of the vertices of the graph. The term is primarily used in the context of regular subgraphs: a k-factor is a factor that is k-regular. In particular, a 1-factor is the same thing as a perfect matching. A factor-critical graph is a graph for which deleting any one vertex produces a graph with a 1-factor.

face In a plane graph or graph embedding, a connected component of the subset of the plane or surface of the embedding that is disjoint from the graph. For an embedding in the plane, all but one face will be bounded; the one exceptional face that extends to infinity is called the outer face.

factorization A graph factorization is a partition of the edges of the graph into factors; a k-factorization is a partition into k-factors. For instance a 1-factorization is an edge coloring with the additional property that each vertex is incident to an edge of each color.

family A synonym for *class*.

finite A graph is finite if it has a finite number of vertices and a finite number of edges. Many sources assume that all graphs are finite without explicitly saying so. A graph is locally finite if each vertex has a finite number of incident edges. An infinite graph is a graph that is not finite: it has infinitely many vertices, infinitely many edges, or both.

first order The first order logic of graphs is a form of logic in which variables represent vertices of a graph, and there exists a binary predicate to test whether two vertices are adjacent. To be distinguished from second order logic, in which variables can also represent sets of vertices or edges.

-flap For a set of vertices X, an X-flap is a connected component of the induced subgraph formed by deleting X. The flap terminology is commonly used in the context of *havens*, functions that map small sets of vertices to their flaps. See also the *bridge* of a cycle, which is either a flap of the cycle vertices or a chord of the cycle.

forbidden A forbidden graph characterization is a characterization of a family of graphs as being the graphs that do not have certain other graphs as subgraphs, induced subgraphs, or minors. If H is one of the graphs that does not occur as a subgraph, induced subgraph, or minor, then H is said to be forbidden.

forest A forest is an undirected graph without cycles (a disjoint union of unrooted trees), or a directed graph formed as a disjoint union of rooted trees.

Frucht 1. Robert Frucht

2. The Frucht graph, one of the two smallest cubic graphs with no nontrivial symmetries.

3. Frucht's theorem that every finite group is the group of symmetries of a finite graph.

full Synonym for *induced*.

1.2.8 G

G A variable often used to denote a graph.

genus The genus of a graph is the minimum genus of a surface onto which it can be embedded; see *embedding*.

geodesic As a noun, a geodesic is a synonym for a shortest path. When used as an adjective, it means related to shortest paths or shortest path distances.

giant In the theory of random graphs, a giant component is a connected component that contains a constant fraction of the vertices of the graph. In standard models of random graphs, there is typically at most one giant component.

girth The girth of a graph is the length of its shortest cycle.

graph The fundamental object of study in graph theory, a system of vertices connected in pairs by edges. Often subdivided into directed graphs or undirected graphs according to whether the edges have an orientation or not. Mixed graphs include both types of edges.

greedy Produced by a greedy algorithm. For instance, a greedy coloring of a graph is a coloring produced by considering the vertices in some sequence and assigning each vertex the first available color.

Grötzsch 1. Herbert Grötzsch

2. The Grötzsch graph, the smallest triangle-free graph requiring four colors in any proper coloring.

3. Grötzsch's theorem that triangle-free planar graphs can always be colored with at most three colors.

Grundy number 1. The Grundy number of a graph is the maximum number of colors produced by a greedy coloring, with a badly-chosen vertex ordering.

1.2.9 H

H A variable often used to denote a graph, especially when another graph has already been denoted by G.

***H*-coloring** An H-coloring of a graph G (where H is also a graph) is a homomorphism from H to G.

***H*-free** A graph is H-free if it does not have an induced subgraph isomorphic to H, that is, if H is a forbidden induced subgraph. The H-free graphs are the family of all graphs (or, often, all finite graphs) that are H-free.[5] For instance the triangle-free graphs are the graphs that do not have a triangle graph as a subgraph. The property of being H-free is always hereditary. A graph is H-minor-free if it does not have a minor isomorphic to H.

Hadwiger 1. Hugo Hadwiger

2. The Hadwiger number of a graph is the order of the largest complete minor of the graph. It is also called the contraction clique number or the homomorphism degree.

3. The Hadwiger conjecture is the conjecture that the Hadwiger number is never less than the chromatic number.

Hamiltonian A Hamiltonian path or Hamiltonian cycle is a simple spanning path or simple spanning cycle: it covers all of the vertices in the graph exactly once. A graph is Hamiltonian if it contains a Hamiiltonian cycle, and traceable if it contains a Hamiltonian path.

haven A k-haven is a function that maps every set X of fewer than k vertices to one of its flaps, often satisfying additional consistency conditions. The order of a haven is the number k. Havens can be used to characterize the treewidth of finite graphs and the ends and Hadwiger numbers of infinite graphs.

hereditary A hereditary property of graphs is a property that is closed under induced subgraphs: if G has a hereditary property, then so must every induced subgraph of G. Compare *monotone* (closed under all subgraphs) or *minor-closed* (closed under minors).

hole A hole is an induced cycle of length four or more. An odd hole is a hole of odd length. An anti-hole is an induced subgraph of order four whose complement is a cycle; equivalently, it is a hole in the complement graph. This terminology is mainly used in the context of perfect graphs, which are characterized by the strong perfect graph theorem as being the graphs with no odd holes or odd anti-holes. The hole-free graphs are the same as the chordal graphs.

homomorphic equivalence Two graphs are homomorphically equivalent if there exist two homomorphisms, one from each graph to the other graph.

homomorphism 1. A graph homomorphism is a mapping from the vertex set of one graph to the vertex set of another graph that maps adjacent vertices to adjacent vertices. This type of mapping between graphs is the one that is most commonly used in category-theoretic approaches to graph theory. A proper graph coloring can equivalently be described as a homomorphism to a complete graph.

2. The homomorphism degree of a graph is a synonym for its *Hadwiger number*, the order of the largest clique minor.

hyperedge An edge in a hypergraph, having any number of endpoints, in contrast to the requirement that edges of graphs have exactly two endpoints.

hypercube A hypercube graph is a graph formed from the vertices and edges of a geometric hypercube.

hypergraph A hypergraph is a generalization of a graph in which each edge (called a hyperedge in this context) may have more than two endpoints.

hypo- This prefix, in combination with a graph property, indicates a graph that does not have the property but such that every subgraph formed by deleting a single vertex does have the property. For instance, a hypohamiltonian graph is one that does not have a Hamiltonian cycle, but for which every one-vertex deletion produces a Hamiltonian subgraph. Compare *critical*, used for graphs which have a property but for which every one-vertex deletion does not.[6]

1.2.10 I

in-degree The number of incoming edges in a directed graph; see *degree*.

incidence An incidence in a graph is a vertex-edge pair such that the vertex is an endpoint of the edge.

incidence matrix The incidence matrix of a graph is a matrix whose rows are indexed by vertices of the graph, and whose columns are indexed by edges, with a one in the cell for row i and column j when vertex i and edge j are incident, and a zero otherwise.

incident The relation between an edge and one of its endpoints.[2]

incomparability An incomparability graph is the complement of a comparability graph; see *comparability*.

independent 1. An independent set is a set of vertices that induces an edgeless subgraph. It may also be called a stable set or a coclique. The independence number $\alpha(G)$ is the size of the maximum independent set.

2. In the graphic matroid of a graph, a subset of edges is independent if the corresponding subgraph is a tree or forest. In the bicircular matroid, a subset of edges is independent if the corresponding subgraph is a pseudoforest.

indifference An indifference graph is another name for a proper interval graph or unit interval graph; see *proper*.

induced An induced subgraph or full subgraph of a graph is a subgraph formed from a subset of vertices and from all of the edges that have both endpoints in the subset. Special cases include induced paths and induced cycles, induced subgraphs that are paths or cycles.

inductive Synonym for *degenerate*.

infinite An infinite graph is one that is not finite; see *finite*.

internal A vertex of a path or tree is internal if it is not a leaf; that is, if its degree is greater than one. Two paths are internally disjoint (some people call it *independent*) if they do not have any vertex in common, except the first and last ones.

intersection 1. The intersection of two graphs is their largest common subgraph, the graph formed by the vertices and edges that belong to both graphs.

2. An intersection graph is a graph whose vertices correspond to sets or geometric objects, with an edge between two vertices exactly when the corresponding two sets or objects have a nonempty intersection. Several classes of graphs may be defined as the intersection graphs of certain types of objects, for instance chordal graphs (intersection graphs of subtrees of a tree), circle graphs (intersection graphs of chords of a circle), interval graphs (intersection graphs of intervals of a line), line graphs (intersection graphs of the edges of a graph), and clique graphs (intersection graphs of the maximal cliques of a graph). Every graph is an intersection graph for some family of sets, and this family is called an intersection representation of the graph. The intersection number of a graph G is the minimum total number of elements in any intersection representation of G.

interval An interval graph is an intersection graph of intervals of a line.

interval thickness A synonym for *pathwidth*.

invariant A synonym of *pathwidth*.

inverted arrow An *arrow* with an opposite *direction* compared to another arrow. The arrow (y, x) is the inverted arrow of the arrow (x, y).

isolated An isolated vertex of a graph is a vertex whose degree is zero, that is, a vertex with no incident edges.[2]

isomorphic Two graphs are isomorphic if there is an isomorphism between them; see *isomorphism*.

isomorphism A graph isomorphism is a one-to-one incidence preserving correspondence of the vertices and edges of one graph to the vertices and edges of another graph. Two graphs related in this way are said to be isomorphic.

isoperimetric See *expansion*.

isthmus Synonym for *bridge*, in the sense of an edge whose removal disconnects the graph.

1.2.11 K

K For the notation for complete graphs, complete bipartite graphs, and complete multipartite graphs, see *complete*.

κ $\kappa(G)$ (using the Greek letter kappa) is the size of the maximum clique in G; see *clique*.

knot An inescapable section of a *directed graph*. See knot (mathematics) and knot theory.

1.2.12 L

L $L(G)$ is the line graph of G; see *line*.

label 1. Information associated with a vertex or edge of a graph. A labeled graph is a graph whose vertices or edges have labels. The terms *vertex-labeled* or *edge-labeled* may be used to specify which objects of a graph have labels. Graph labeling refers to several different problems of assigning labels to graphs subject to certain constraints. See also graph coloring, in which the labels are interpreted as colors.

2. In the context of graph enumeration, the vertices of a graph are said to be labeled if they are all distinguishable from each other. For instance, this can be made to be true by fixing a one-to-one correspondence between the vertices and the integers from 1 to the order of the graph. When vertices are labeled, graphs that are isomorphic to each other (but with different vertex orderings) are counted as separate objects. In contrast, when the vertices are unlabeled, graphs that are isomorphic to each other are not counted separately.

leaf 1. A leaf vertex or pendant vertex (especially in a tree) is a vertex whose degree is 1. A leaf edge or pendant edge is the edge connecting a leaf vertex to its single neighbour.

2. A leaf power of a tree is a graph whose vertices are the leaves of the tree and whose edges connect leaves whose distance in the tree is at most a given threshold.

length In an unweighted graph, the length of a cycle, path, or walk is the number of edges it uses. In a weighted graph, it may instead be the sum of the weights of the edges that it uses. Length is used to define the shortest path, girth (shortest cycle length), and longest path between two vertices in a graph.

line A synonym for an undirected edge. The line graph $L(G)$ of a graph G is a graph with a vertex for each edge of G and an edge for each pair of edge that share an endpoint in G.

linkage A synonym for *degeneracy*.

list 1. An adjacency list is a computer representation of graphs for use in graph algorithms.

2. List coloring is a variation of graph coloring in which each vertex has a list of available colors.

local A local property of a graph is a property that is determined only by the neighbourhoods of the vertices in the graph. For instance, a graph is locally finite if all of its neighborhoods are finite.

loop A loop or self-loop is an edge both of whose endpoints are the same vertex. It forms a cycle of length 1. These are not allowed in simple graphs.

1.2.13 M

magnification Synonym for vertex *expansion*.

matching A matching is a set of edges in which no two share any vertex. A vertex is matched or saturated if it is one of the endpoints of an edge in the matching. A perfect matching or complete matching is a matching that matches every vertex; it may also be called a 1-factor, and can only exist when the order is even. A near-perfect matching, in a graph with odd order, is one that saturates all but one vertex. A maximum matching is a matching that uses as many edges as possible; the matching number $\alpha'(G)$ of a graph G is the number of edges in a maximum matching. A maximal matching is a matching to which no additional edges can be added.

maximal 1. A subgraph of given graph G is maximal for a particular property if it has that property but no other supergraph of it that is also a subgraph of G also has the same property. That is, it is a maximal element of the subgraphs with the property. For instance, a maximal clique is a complete subgraph that cannot be expanded to a larger complete subgraph. The word “maximal” should be distinguished from “maximum": a maximum subgraph is always maximal, but not necessarily vice versa.

2. A simple graph with a given property is maximal for that property if it is not possible to add any more edges to it (keeping the vertex set unchanged) while preserving both the simplicity of the graph and the property. Thus, for instance, a maximal planar graph is a planar graph such that adding any more edges to it would create a non-planar graph.

maximum A subgraph of a given graph G is maximum for a particular property if it is the largest subgraph (by order or size) among all subgraphs with that property. For instance, a maximum clique is any of the largest cliques in a given graph.

median 1. A median of a triple of vertices, a vertex that belongs to shortest paths between all pairs of

vertices, especially in median graphs and modular graphs.

2. A median graph is a graph in which every three vertices have a unique median.

Meyniel 1. Henri Meyniel, French graph theorist.

2. A Meyniel graph is a graph in which every odd cycle of length five or more has at least two chords.

minimal A subgraph of given graph is minimal for a particular property if it has that property but no proper subgraph of it also has the same property. That is, it is a minimal element of the subgraphs with the property.

minimum cut A *cut* whose *cut-set* has minimum total weight, possibly restricted to cuts that separate a designated pair of vertices; they are characterized by the max-flow min-cut theorem.

minor A graph H is a minor of another graph G if H can be obtained by deleting edges or vertices from G and contracting edges in G. It is a shallow minor if it can be formed as a minor in such a way that the subgraphs of G that were contracted to form vertices of H all have small diameter. H is a topological minor of G if G has a subgraph that is a subdvision of H. A graph is H-minor-free if it does not have H as a minor. A family of graphs is minor-closed if it is closed under minors; the Robertson–Seymour theorem characterizes minor-closed families as having a finite set of *forbidden* minors.

mixed A mixed graph is a graph that may include both directed and undirected edges.

modular 1. Modular graph, a graph in which each triple of vertices has at least one median vertex that belongs to shortest paths between all pairs of the triple.

2. Modular decomposition, a decomposition of a graph into subgraphs within which all vertices connect to the rest of the graph in the same way.

3. Modularity of a graph clustering, the difference of the number of cross-cluster edges from its expected value.

monotone A monotone property of graphs is a property that is closed under subgraphs: if G has a hereditary property, then so must every subgraph of G. Compare *hereditary* (closed under induced subgraphs) or *minor-closed* (closed under minors).

Moore graph A Moore graph is a regular graph for which the Moore bound is met exactly. The Moore bound is an inequality relating the degree, diameter, and order of a graph, proved by Edward F. Moore. Every Moore graph is a cage.

multigraph A multigraph is a graph that allows multiple adjacencies (and, often, self-loops); a graph that is not required to be simple.

multiple adjacency A multiple adjacency or multiple edge is a set of more than one edge that all have the same endpoints (in the same direction, in the case of directed graphs). A graph with multiple edges is often called a multigraph.

multiplicity The multiplicity of an edge is the number of edges in a multiple adjacency. The multiplicity of a graph is the maximum multiplicity of any of its edges.

1.2.14 N

N 1. For the notation for open and closed neighborhoods, see *neighbourhood*.

2. A lower-case n is often used (especially in computer science) to denote the number of vertices in a given graph.

neighbor

neighbour A vertex that is adjacent to a given vertex.

neighborhood

neighbourhood The open neighbourhood (or neighborhood) of a vertex v is the subgraph induced by all vertices that are adjacent to v. The closed neighbourhood is defined in the same way but also includes v itself. The open neighborhood of v in G may be denoted $NG(v)$ or $N(v)$, and the closed neighborhood may be denoted $NG[v]$ or $N[v]$. When the openness or closedness of a neighborhood is not specified, it is assumed to be open.

network A graph in which attributes (e.g. names) are associated with the nodes and/or edges.

node A synonym for *vertex*.

non-edge A non-edge or anti-edge is a pair of vertices that are not adjacent; the edges of the complement graph.

null graph See *empty graph*.

1.2.15 O

odd 1. An odd cycle is a cycle whose length is odd. The odd girth of a non-bipartite graph is the length of its shortest odd cycle. An odd hole is a special case of an odd cycle: one that is induced and has four or more vertices.

2. An odd vertex is a vertex whose degree is odd. By the handshaking lemma every finite undirected graph has an even number of odd vertices.

3. An odd ear is a simple path or simple cycle with an odd number of edges, used in odd ear decompositions of factor-critical graphs; see *ear*.

4. An odd chord is an edge connecting two vertices that are an odd distance apart in an even cycle. Odd chords are used to define strongly chordal graphs.

5. An odd graph is a special case of a Kneser graph, having one vertex for each $(n-1)$-element subset of a $(2n-1)$-element set, and an edge connecting two subsets when their corresponding sets are disjoint.

open 1. See *neighbourhood*.

2. See *walk*.

order 1. The order of a graph G is the number of its vertices, $|V(G)|$. The variable n is often used for this quantity. See also *size*, the number of edges.

2. A type of logic of graphs; see *first order* and *second order*.

3. An order or ordering of a graph is an arrangement of its vertices into a sequence, especially in the context of topological ordering (an order of a directed acyclic graph in which every edge goes from an earlier vertex to a later vertex in the order) and degeneracy ordering (an order in which each vertex has minimum degree in the induced subgraph of it and all later vertices).

4. For the order of a haven or bramble, see *haven* and *bramble*.

orientation

oriented 1. An orientation of an undirected graph is an assignment of directions to its edges, making it into a directed graph. An oriented graph is one that has been assigned an orientation. So, for instance, a polytree is an oriented tree; it differs from a directed tree (an arborescence) in that there is no requirement of consistency in the directions of its edges. Other special types of orientation include tournaments, orientations of complete graphs; strong orientations, orientations that are strongly connected; acyclic orientations, orientations that are acyclic; Eulerian orientations, orientations that are Eulerian; and transitive orientations, orientations that are transitively closed.

2. Oriented graph, used by some authors as a synonym for a directed graph.

out-degree See *degree*.

outer See *face*.

outerplanar An outerplanar graph is a graph that can be embedded in the plane (without crossings) so that all vertices are on the outer face of the graph.

1.2.16 P

path A path may either be a walk (a sequence of vertices and edges, with both endpoints of an edge appearing adjacent to it in the sequence) or a simple path (a walk with no repetitions of vertices or edges), depending on the source. Important special cases include induced paths and shortest paths.

path decomposition A path decomposition of a graph G is a tree decomposition whose underlying tree is a path. Its width is defined in the same way as for tree decompositions, as one less than the size of the largest bag. The minimum width of any path decomposition of G is the pathwidth of G.

pathwidth The pathwidth of a graph G is the minimum width of a path decomposition of G. It may also be defined in terms of the clique number of an interval completion of G. It is always between the bandwidth and the treewidth of G. It is also known as interval thickness, vertex separation number, or node searching number.

pendant See *leaf*.

perfect 1. A perfect graph is a graph in which, in every induced subgraph, the chromatic number equals the clique number. The perfect graph theorem and strong perfect graph theorem are two theorems about perfect graphs, the former proving that their complements are also perfect and the latter proving that they are exactly the graphs with no odd holes or anti-holes.

2. A perfectly orderable graph is a graph whose vertices can be ordered in such a way that a greedy coloring algorithm with this ordering optimally colors every induced subgraph. The perfectly orderable graphs are a subclass of the perfect graphs.

3. A perfect matching is a matching that saturates every vertex; see *matching*.

4. A perfect 1-factorization is a partition of the edges of a graph into perfect matchings so that each two matchings form a Hamiltonian cycle.

peripheral 1. A peripheral cycle or non-separating cycle is a cycle with at most one bridge.

2. A peripheral vertex is a vertex whose *eccentricity* is maximum. In a tree, this must be a leaf.

Petersen 1. Julius Petersen (1839–1910), Danish graph theorist.

2. The Petersen graph, a 10-vertex 15-edge graph frequently used as a counterexample.

3. Petersen's theorem that every bridgeless cubic graph has a perfect matching.

planar A planar graph is a graph that has an embedding onto the Euclidean plane. A plane graph is a planar graph for which a particular embedding has already been fixed. A k-planar graph is one that can be drawn in the plane with at most k crossings per edge.

polytree A polytree is an oriented tree; equivalently, a directed acyclic graph whose underlying undirected graph is a tree.

power 1. A graph power G^k of a graph G is another graph on the same vertex set; two vertices are adjacent in G^k when they are at distance at most k in G. A leaf power is a closely related concept, derived from a power of a tree by taking the subgraph induced by the tree's leaves.

2. Power graph analysis is a method for analyzing complex networks by identifying cliques, bicliques, and stars within the network.

3. Power laws in the degree distributions of scale-free networks are a phenomenon in which the number of vertices of a given degree is proportional to a power of the degree.

predecessor A *vertex* coming before a given vertex in a *directed path*.

proper 1. A proper subgraph is a subgraph that removes at least one vertex or edge relative to the whole graph; for finite graphs, proper subgraphs are never isomorphic to the whole graph, but for infinite graphs they can be.

2. A proper coloring is an assignment of colors to the vertices of a graph (a coloring) that assigns different colors to the endpoints of each edge; see *color*.

3. A proper interval graph or proper circular arc graph is an intersection graph of a collection of intervals or circular arcs (respectively) such that no interval or arc contains another interval or arc. Proper interval graphs are also called unit interval graphs (because they can always be represented by unit intervals) or indifference graphs.

property A graph property is something that can be true of some graphs and false of others, and that depends only on the graph structure and not on incidental information such as labels. Graph properties may equivalently be described in terms of classes of graphs (the graphs that have a given property). More generally, a graph property may also be a function of graphs that is again independent of incidental information, such as the size, order, or degree sequence of a graph; this more general definition of a property is also called an invariant of the graph.

pseudoforest A pseudoforest is an undirected graph in which each connected component has at most one cycle, or a directed graph in which each vertex has at most one outgoing edge.

pseudograph A pseudograph is a graph or multigraph that allows self-loops.

1.2.17 Q

quasi-line graph A quasi-line graph or locally cobipartite graph is a graph in which the open neighborhood of every vertex can be partitioned into two cliques. These graphs are always claw-free and they include as a special case the line graphs. They are used in the structure theory of claw-free graphs.

quiver A quiver is a directed multigraph, as used in category theory. The edges of a quiver are called arrows.

1.2.18 R

radius The radius of a graph is the minimum *eccentricity* of any vertex.

Ramanujan A Ramanujan graph is a graph whose spectral expansion is as large as possible. That is, it is a d-regular graph, such that the second-largest eigenvalue of its adjacency matrix is at most $2\sqrt{d-1}$.

ray A ray, in an infinite graph, is an infinite simple path with exactly one endpoint. The ends of a graph are equivalence classes of rays.

reachability The ability to get from one *vertex* to another within a *graph*.

reachable Has an affirmative *reachability*. A *vertex* y is said to be reachable from a vertex x if there exists a *path* from x to y.

recognizable In the context of the reconstruction conjecture, a graph property is recognizable if its truth can be determined from the deck of the graph. Many graph properties are known to be recognizable. If the reconstruction conjecture is true, all graph properties are recognizable.

reconstruction The reconstruction conjecture states that each undirected graph G is uniquely determined by its *deck*, a multiset of graphs formed by removing one vertex from G in all possible ways. In this context, reconstruction is the formation of a graph from its deck.

regular A graph is d-regular when all of its vertices have degree d. A regular graph is a graph that is d-regular for some d.

reverse See *transpose*.

root 1. A designated vertex in a graph, particularly in directed trees and rooted graphs.

2. The inverse operation to a graph power: a kth root of a graph G is another graph on the same vertex set such that two vertices are adjacent in G if and only if they have distance at most k in the root.

1.2.19 S

second order The second order logic of graphs is a form of logic in which variables may represent vertices, edges, sets of vertices, and (sometimes) sets of edges. This logic includes predicates for testing whether a vertex and edge are incident, as well as whether a vertex or edge belongs to a set. To be distinguished from first order logic, in which variables can only represent vertices.

saturated See *matching*.

searching number Node searching number is a synonym for *pathwidth*.

self-loop Synonym for *loop*.

separating vertex See *articulation point*.

separation number Vertex separation number is a synonym for *pathwidth*.

simple 1. A simple graph is a graph with no loops and with no multiple adjacencies. That is, each edge connects two distinct endpoints and no two edges have the same endpoints. A simple edge is an edge that is not part of a multiple adjacency. In many cases, graphs are assumed to be simple unless specified otherwise.

2. A simple path or a simple cycle is a path or cycle that has no repeated vertices (and no repeated edges).

sink A sink, in a directed graph, is a vertex with no outgoing edges.

size The size of a graph G is the number of its edges, $|E(G)|$.[7] The variable m is often used for this quantity. See also *order*, the number of vertices.

small-world network A small-world network is a graph in which most nodes are not neighbors of one another, but most nodes can be reached from every other node by a small number of hops or steps. Specifically, a small-world network is defined to be a graph where the typical distance L between two randomly chosen nodes (the number of steps required) grows proportionally to the logarithm of the number of nodes N in the network [8]

source A source, in a directed graph, is a vertex with no incoming edges.

space In algebraic graph theory, several vector spaces over the binary field may be associated with a graph. Each has sets of edges or vertices for its vectors, and symmetric difference of sets as its vector sum operation. The edge space is the space of all sets of edges, and the vertex space is the space of all sets of vertices. The cut space is a subspace of the edge space that has the cut-sets of the graph as its elements. The cycle space has the Eulerian spanning subgraphs as its elements.

spanner A spanner is a (usually sparse) graph whose shortest path distances approximate those in a dense graph or other metric space. Variations include geometric spanners, graphs whose vertices are points in a geometric space; tree spanners, spanning trees of a graph whose distances approximate the graph distances, and graph spanners, sparse subgraphs of a dense graph whose distances approximate the original graph's distances. A greedy spanner is a graph spanner constructed by a greedy algorithm, generally one that considers all edges from shortest to longest and keeps the ones that are needed to preserve the distance approximation.

spanning A subgraph is spanning when it includes all of the vertices of the given graph. Important cases include spanning trees, spanning subgraphs that are trees, and perfect matchings, spanning subgraphs that are matchings. A spanning subgraph may also be called a factor, especially (but not only) when it is regular.

sparse A sparse graph is one that has few edges relative to its number of vertices. In some definitions the same property should also be true for all subgraphs of the given graph.

spectral

spectrum The spectrum of a graph is the collection of eigenvalues of its adjacency matrix. Spectral graph theory is the branch of graph theory that uses spectra to analyze graphs. See also spectral *expansion*.

split 1. A split graph is a graph whose vertices can be partitioned into a clique and an independent set. A related class of graphs, the double split graphs, are used in the proof of the strong perfect graph theorem.

2. A split of an arbitrary graph is a partition of its vertices into two nonempty subsets, such that the edges spanning this cut form a complete bipartite subgraph. The splits of a graph can be represented by a tree structure called its *split decomposition*. A split is called a strong split when it is not crossed by any other split. A split is called nontrivial when both of its sides have more than one vertex. A graph is called prime when it has no nontrivial splits.

square 1. The square of a graph G is the graph power G^2; in the other direction, G is the square root of G^2. The half-square of a bipartite graph is the subgraph of its square induced by one side of the bipartition.

2. A squaregraph is a planar graph that can be drawn so that all bounded faces are 4-cycles and all vertices of degree ≤ 3 belong to the outer face.

3. A square grid graph is a lattice graph defined from points in the plane with integer coordinates connected by unit-length edges.

stable A stable set is a synonym for an *independent set*.

star A star is a tree with one internal vertex; equivalently, it is a complete bipartite graph $K_{1,n}$ for some $n \geq 2$. The special case of a star with three leaves is called a claw.

strength The strength of a graph is the minimum ratio of the number of edges removed from the graph to components created, over all possible removals; it is analogous to toughness, based on vertex removals.

strong 1. For strong connectivity and strongly connected components of directed graphs, see *connected* and *component*. A strong orientation is an orientation that is strongly connected; see *orientation*.

2. For the strong perfect graph theorem, see *perfect*.

3. A strongly regular graph is a regular graph in which every two adjacent vertices have the same number of shared neighbours and every two non-adjacent vertices have the same number of shared neighbours.

4. A strongly chordal graph is a chordal graph in which every even cycle of length six or more has an odd chord.

5. A strongly perfect graph is a graph in which every induced subgraph has an independent set meeting all maximal cliques. The Meyniel graphs are also called "very strongly perfect graphs" because in them, every vertex belongs to such an independent set.

subforest A subgraph of a *forest*.

subgraph A subgraph of a graph G is another graph formed from a subset of the vertices and edges of G. The vertex subset must include all endpoints of the edge subset, but may also include additional vertices. A spanning subgraph is one that includes all vertices of the graph; an induced subgraph is one that includes all the edges whose endpoints belong to the vertex subset.

subtree A subtree is a connected subgraph of a tree. Sometimes, for rooted trees, subtrees are defined to be a special type of connected subgraph, formed by all vertices and edges reachable from a chosen vertex.

successor A *vertex* coming after a given vertex in a *directed path*.

superconcentrator A superconcentrator is a graph with two designated and equal-sized subsets of vertices I and O, such that for every two equal-sized subsets S of I and T O there exists a family of disjoint paths connecting every vertex in S to a vertex in T. Some sources require in addition that a superconcentrator be a directed acyclic graph, with I as its sources and O as its sinks.

supergraph A graph formed by adding vertices, edges, or both to a given graph. If H is a subgraph of G, then G is a supergraph of H.

1.2.20 T

theta 1. A theta graph is the union of three internally disjoint (simple) paths that have the same two distinct end vertices.[9]

2. The theta graph of a collection of points in the Euclidean plane is constructed by constructing a system of cones surrounding each point and adding one edge per cone, to the point whose projection onto a central ray of the cone is smallest.

3. The Lovász number or Lovász theta function of a graph is a graph invariant related to the clique number and chromatic number that can be computed in polynomial time by semidefinite programming.

topological 1. A topological graph is a representation of the vertices and edges of a graph by points and curves in the plane (not necessarily avoiding crossings).

2. Topological graph theory is the study of graph embeddings.

3. Topological sorting is the algorithmic problem of arranging a directed acyclic graph into a topological order, a vertex sequence such that each edge goes from an earlier vertex to a later vertex in the sequence.

totally disconnected Synonym for *edgeless*.

tour A closed trail, a *walk* that starts and ends at the same vertex and has no repeated edges. Euler tours are tours that use all of the graph edges; see *Eulerian*.

tournament A tournament is an orientation of a complete graph; that is, it is a directed graph such that every two vertices are connected by exactly one directed edge (going in only one of the two directions between the two vertices).

traceable A traceable graph is a graph that contains a Hamiltonian path.

trail A walk without repeated edges.

transitive Having to do with the transitive property. The transitive closure of a given directed graph is a graph on the same vertex set that has an edge from one vertex to another whenever the original graph has a path connecting the same two vertices. A transitive reduction of a graph is a minimal graph having the same transitive closure; directed acyclc graphs have a unique transitive reduction. A transitive orientation is an orientation of a graph that is its own transitive closure; it exists only for comparability graphs.

transpose The transpose graph of a given directed graph is a graph on the same vertices, with each edge reversed in direction. It may also be called the converse or reverse of the graph.

tree 1. A tree is an undirected graph that is both connected and acyclic, or a directed graph in which there exists a unique walk from one vertex (the root of the tree) to all remaining vertices.

2. A k-tree is a graph formed by gluing $(k + 1)$-cliques together on shared k-cliques. A tree in the ordinary sense is a 1-tree according to this definition.

tree decomposition A tree decomposition of a graph G is a tree whose nodes are labeled with sets of vertices of G; these sets are called bags. For each vertex v, the bags that contain v must induce a subtree of the tree, and for each edge uv there must exist a bag that contains both u and v. The width of a tree decomposition is one less than the maximum number of vertices in any of its bags; the treewidth of G is the minimum width of any tree decomposition of G.

treewidth The treewidth of a graph G is the minimum width of a tree decomposition of G. It can also be defined in terms of the clique number of a chordal completion of G, the order of a haven of G, or the order of a bramble of G.

triangle A cycle of length three in a graph. A triangle-free graph is an undirected graph that does not have any triangle subgraphs.

Turán 1. Pál Turán

2. A Turán graph is a balanced complete multipartite graph.

3. Turán's theorem states that Turán graphs have the maximum number of edges among all clique-free graphs of a given order.

4. Turán's brick factory problem asks for the minimum number of crossings in a drawing of a complete bipartite graph.

1.2.21 U

undirected An undirected graph is a graph in which the two endpoints of each edge are not distinguished from each other. See also *directed* and *mixed*. In a mixed graph, an undirected edge is again one in which the endpoints are not distinguished from each other.

uniform A hypergraph is k-uniform when all its edges have k endpoints, and uniform when it is k-uniform for some k. For instance, ordinary graphs are the same as 2-uniform hypergraphs.

universal 1. A universal graph is a graph that contains as subgraphs all graphs in a given family of graphs, or all graphs of a given size or order within a given family of graphs.

2. A universal vertex (also called an apex or dominating vertex) is a vertex that is adjacent to every other vertex in the graph. For instance, wheel graphs and connected threshold graphs always have a universal vertex.

3. In the logic of graphs, a vertex that is universally quantified in a formula may be called a universal vertex for that formula.

unweighted graph A *graph* whose *vertices* and *edges* *have not been assigned* weights; the opposite of a *weighted graph*.

1.2.22 V

V See *vertex set*.

valency Synonym for *degree*.

vertex A vertex (plural vertices) is (together with edges) one of the two basic units out of which graphs are constructed. Vertices of graphs are often considered to be atomic objects, with no internal structure.

vertex cut

separating set A set of *vertices* whose removal *disconnects* the *graph*. A one-vertex cut is called an *articulation point* or *cut vertex*.

vertex set The set of vertices of a given graph G, sometimes denoted by $V(G)$.

vertices See *vertex*.

Vizing 1. Vadim G. Vizing

2. Vizing's theorem that the chromatic index is at most one more than the maximum degree.

3. Vizing's conjecture on the domination number of Cartesian products of graphs.

1.2.23 W

W The letter W is used in notation for wheel graphs and windmill graphs. The notation is not standardized.

Wagner 1. Klaus Wagner

2. The Wagner graph, an eight-vertex Möbius ladder.

3. Wagner's theorem characterizing planar graphs by their forbidden minors.

4. Wagner's theorem characterizing the K_5-minor-free graphs.

walk A walk is an alternating sequence of vertices and edges, starting and ending at a vertex, in which each edge is adjacent in the sequence to its two endpoints. In a directed graph the ordering of the endpoints of each edge in the sequence must be consistent with the direction of the edge. Some sources call walks paths, while others reserve the term "path" for a simple path (a walk without repeated vertices or edges). Walks are also sometimes called *chains*.[10] A walk is open if its starts and ends at two different vertices, and closed if it starts and ends at the same vertex. A closed walk may also be called a cycle. Alternatively, the word "cycle" may be reserved for a simple closed walk (one without repeated vertices or edges except for the repetition of the starting and final vertex). A walk without repeated edges (but with vertex repetition allowed) may be called a trail and a closed trail may be called a tour. In the context of ear decomposition, a walk that can have the same starting and ending vertex but otherwise avoids any repeated vertices may be called an *ear*.

weight A numerical value, assigned as a label to a vertex or edge of a graph. The weight of a subgraph is the sum of the weights of the vertices or edges within that subgraph.

weighted graph A *graph* whose *vertices* or *edges have been assigned* weights; more specifically, a vertex-weighted graph has weights on its vertices and an edge-weighted graph has weights on its edges.

well-colored A well-colored graph is a graph all of whose greedy colorings use the same number of colors.

well-covered A well-covered graph is a graph all of whose maximal independent sets are the same size.

wheel A wheel graph is a graph formed by adding a universal vertex to a simple cycle.

width 1. A synonym for *degeneracy*.

2. For other graph invariants known as width, see *bandwidth*, *branchwidth*, *clique-width*, *pathwidth*, and *treewidth*.

3. The width of a tree decomposition or path decomposition is one less than the maximum size of one of its bags, and may be used to define treewidth and pathwidth.

windmill A windmill graph is the union of a collection of cliques, all of the same order as each other, with one shared vertex belonging to all the cliques and all other vertices and edges distinct.

1.2.24 See also

- List of graph theory topics
- Gallery of named graphs
- Graph algorithms
- Glossary of areas of mathematics

1.2.25 References

[1] Farber, M.; Hahn, G.; Hell, P.; Miller, D. J. (1986), "Concerning the achromatic number of graphs", *Journal of Combinatorial Theory, Series B*, **40** (1): 21–39, doi:10.1016/0095-8956(86)90062-6.

[2] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001), "B.4 Graphs", *Introduction to Algorithms* (2 ed.), MIT Press and McGraw-Hill, pp. 1080–1084.

[3] Grünbaum, B. (1973), "Acyclic colorings of planar graphs", *Israel Journal of Mathematics*, **14**: 390–408, doi:10.1007/BF02764716.

[4] Cormen et al. (2001), p. 529.

[5] Brandstädt, Andreas; Le, Van Bang; Spinrad, Jeremy (1999), "Chapter 7: Forbidden Subgraph", *Graph Classes: A Survey*, SIAM Monographs on Discrete Mathematics and Applications, pp. 105–121, ISBN 0-89871-432-X

[6] Mitchem, John (1969), "Hypo-properties in graphs", *The Many Facets of Graph Theory (Proc. Conf., Western Mich. Univ., Kalamazoo, Mich., 1968)*, Springer, pp. 223–230, doi:10.1007/BFb0060121, MR 0253932.

[7] Harris, John M. (2000). *Combinatorics and Graph Theory*. New York: Springer-Verlag. p. 5. ISBN 0-387-98736-3.

[8] http://www.nature.com/nature/journal/v393/n6684/full/393440a0.html

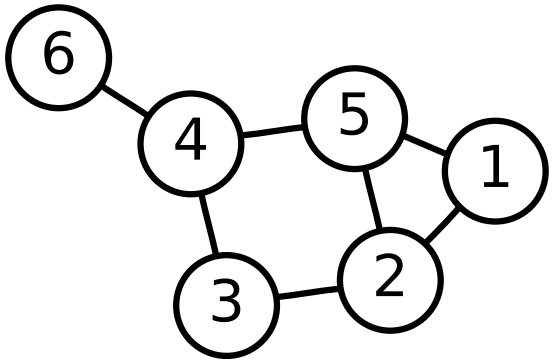
[9] Bondy, J. A. (1972), "The "graph theory" of the Greek alphabet", *Graph theory and applications (Proc. Conf., Western Michigan Univ., Kalamazoo, Mich., 1972; dedicated to the memory of J. W. T. Youngs)*, Lecture Notes in Mathematics, **303**, Springer, pp. 43–54, doi:10.1007/BFb0067356, MR 0335362

[10] Encyclopedia Britannica online

1.3 Graph theory

This article is about sets of vertices connected by edges. For graphs of mathematical functions, see Graph of a function. For other uses, see Graph (disambiguation).

In mathematics **graph theory** is the study of *graphs*,



A drawing of a graph

which are mathematical structures used to model pairwise relations between objects. A graph in this context is made up of *vertices*, *nodes*, or *points* which are connected by *edges*, *arcs*, or *lines*. A graph may be *undirected*, meaning that there is no distinction between the two vertices associated with each edge, or its edges may be *directed* from one vertex to another; see Graph (discrete mathematics) for more detailed definitions and for other variations in the types of graph that are commonly considered. Graphs are one of the prime objects of study in discrete mathematics.

Refer to the glossary of graph theory for basic definitions in graph theory.

1.3.1 Definitions

Definitions in graph theory vary. The following are some of the more basic ways of defining graphs and related mathematical structures.

Graph

In the most common sense of the term,[1] a **graph** is an ordered pair $G = (V, E)$ comprising a set V of *vertices* or *nodes* or *points* together with a set E of *edges* or *arcs* or *lines*, which are 2-element subsets of V (i.e. an edge is associated with two vertices, and that association takes the form of the unordered pair comprising those two vertices). To avoid ambiguity, this type of graph may be described precisely as undirected and simple.

Other senses of *graph* stem from different conceptions of the edge set. In one more generalized notion,[2] V is a set together with a relation of *incidence* that associates with each edge two vertices. In another generalized notion, E is a multiset of unordered pairs of (not necessarily distinct) vertices. Many authors call this type of object a multigraph or pseudograph.

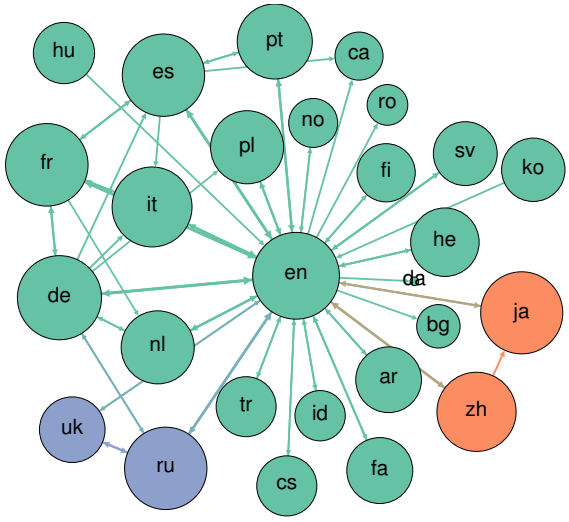
All of these variants and others are described more fully below.

The vertices belonging to an edge are called the *ends* or *end vertices* of the edge. A vertex may exist in a graph and not belong to an edge.

V and E are usually taken to be finite, and many of the well-known results are not true (or are rather different) for infinite graphs because many of the arguments fail in the infinite case. The *order* of a graph is $|V|$, its number of vertices. The *size* of a graph is $|E|$, its number of edges. The *degree* or *valency* of a vertex is the number of edges that connect to it, where an edge that connects a vertex to itself (a loop) is counted twice.

For an edge $\{x, y\}$, graph theorists usually use the somewhat shorter notation xy.

1.3.2 Applications



The network graph formed by Wikipedia editors (edges) contributing to different Wikipedia language versions (vertices) during one month in summer 2013[3]

Graphs can be used to model many types of relations and processes in physical, biological,[4] social and information systems. Many practical problems can be represented by graphs. Emphasizing their application to real-world systems, the term *network* is sometimes defined to mean a graph in which attributes (e.g. names) are associated with the nodes and/or edges.

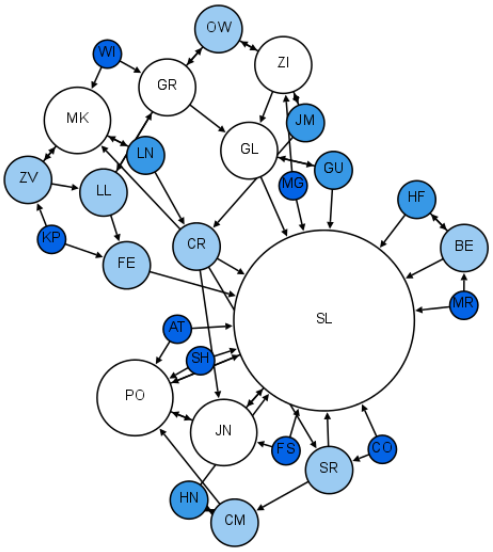
In computer science, graphs are used to represent networks of communication, data organization, computational devices, the flow of computation, etc. For instance, the link structure of a website can be represented by a directed graph, in which the vertices represent web pages and directed edges represent links from

one page to another. A similar approach can be taken to problems in social media,[5] travel, biology, computer chip design, and many other fields. The development of algorithms to handle graphs is therefore of major interest in computer science. The transformation of graphs is often formalized and represented by graph rewrite systems. Complementary to graph transformation systems focusing on rule-based in-memory manipulation of graphs are graph databases geared towards transaction-safe, persistent storing and querying of graph-structured data.

Graph-theoretic methods, in various forms, have proven particularly useful in linguistics, since natural language often lends itself well to discrete structure. Traditionally, syntax and compositional semantics follow tree-based structures, whose expressive power lies in the principle of compositionality, modeled in a hierarchical graph. More contemporary approaches such as head-driven phrase structure grammar model the syntax of natural language using typed feature structures, which are directed acyclic graphs. Within lexical semantics, especially as applied to computers, modeling word meaning is easier when a given word is understood in terms of related words; semantic networks are therefore important in computational linguistics. Still other methods in phonology (e.g. optimality theory, which uses lattice graphs) and morphology (e.g. finite-state morphology, using finite-state transducers) are common in the analysis of language as a graph. Indeed, the usefulness of this area of mathematics to linguistics has borne organizations such as TextGraphs, as well as various 'Net' projects, such as WordNet, VerbNet, and others.

Graph theory is also used to study molecules in chemistry and physics. In condensed matter physics, the three-dimensional structure of complicated simulated atomic structures can be studied quantitatively by gathering statistics on graph-theoretic properties related to the topology of the atoms. In chemistry a graph makes a natural model for a molecule, where vertices represent atoms and edges bonds. This approach is especially used in computer processing of molecular structures, ranging from chemical editors to database searching. In statistical physics, graphs can represent local connections between interacting parts of a system, as well as the dynamics of a physical process on such systems. Similarly, in computational neuroscience graphs can be used to represent functional connections between brain areas that interact to give rise to various cognitive processes, where the vertices represent different areas of the brain and the edges represent the connections between those areas. Graphs are also used to represent the micro-scale channels of porous media, in which the vertices represent the pores and the edges represent the smaller channels connecting the pores.

Graph theory is also widely used in sociology as a way, for example, to measure actors' prestige or to explore rumor spreading, notably through the use of social network analysis software. Under the umbrella of social networks are many different types of graphs.[7] Acquaintanceship and friendship graphs describe whether people know each other. Influence graphs model whether certain people can influence the behavior of others. Finally, collaboration graphs model whether two people work together in a particular way, such as acting in a movie together.



Graph theory in sociology: Moreno Sociogram (1953)[6]

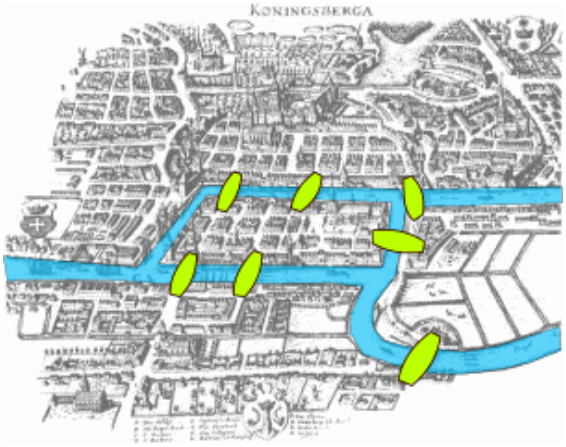
Likewise, graph theory is useful in biology and conservation efforts where a vertex can represent regions where certain species exist (or inhabit) and the edges represent migration paths, or movement between the regions. This information is important when looking at breeding patterns or tracking the spread of disease, parasites or how changes to the movement can affect other species.

In mathematics, graphs are useful in geometry and certain parts of topology such as knot theory. Algebraic graph theory has close links with group theory.

A graph structure can be extended by assigning a weight to each edge of the graph. Graphs with weights, or weighted graphs, are used to represent structures in which pairwise connections have some numerical values. For example, if a graph represents a road network, the weights could represent the length of each road.

1.3.3 History

The paper written by Leonhard Euler on the *Seven Bridges of Königsberg* and published in 1736 is regarded as the first paper in the history of graph theory.[8] This paper, as well as the one written by Vandermonde on the *knight problem,* carried on with the *analysis situs* initiated by Leibniz. Euler's formula relating the number of edges, vertices, and faces of a convex polyhedron was studied and generalized by Cauchy[9] and L'Huillier,[10] and represents the beginning of the branch of mathematics



The Königsberg Bridge problem

known as topology.

More than one century after Euler's paper on the bridges of Königsberg and while Listing was introducing the concept of topology, Cayley was led by an interest in particular analytical forms arising from differential calculus to study a particular class of graphs, the *trees*.[11] This study had many implications for theoretical chemistry. The techniques he used mainly concern the enumeration of graphs with particular properties. Enumerative graph theory then arose from the results of Cayley and the fundamental results published by Pólya between 1935 and 1937. These were generalized by De Bruijn in 1959. Cayley linked his results on trees with contemporary studies of chemical composition.[12] The fusion of ideas from mathematics with those from chemistry began what has become part of the standard terminology of graph theory.

In particular, the term "graph" was introduced by Sylvester in a paper published in 1878 in *Nature*, where he draws an analogy between "quantic invariants" and "covariants" of algebra and molecular diagrams:[13]

> "[…] Every invariant and co-variant thus becomes expressible by a *graph* precisely identical with a Kekuléan diagram or chemicograph. […] I give a rule for the geometrical multiplication of graphs, *i.e.* for constructing a *graph* to the product of in- or co-variants whose separate graphs are given. […]" (italics as in the original).

The first textbook on graph theory was written by Dénes Kőnig, and published in 1936.[14] Another book by Frank Harary, published in 1969, was "considered the world over to be the definitive textbook on the subject",[15] and enabled mathematicians, chemists, electrical engineers and social scientists to talk to each other. Harary donated all of the royalties to fund the Pólya Prize.[16]

One of the most famous and stimulating problems in graph theory is the four color problem: "Is it true that any map drawn in the plane may have its regions colored with four colors, in such a way that any two regions having a common border have different colors?" This problem was first posed by Francis Guthrie in 1852 and its first written record is in a letter of De Morgan addressed to Hamilton the same year. Many incorrect proofs have been proposed, including those by Cayley, Kempe, and others. The study and the generalization of this problem by Tait, Heawood, Ramsey and Hadwiger led to the study of the colorings of the graphs embedded on surfaces with arbitrary genus. Tait's reformulation generated a new class of problems, the *factorization problems*, particularly studied by Petersen and Kőnig. The works of Ramsey on colorations and more specially the results obtained by Turán in 1941 was at the origin of another branch of graph theory, *extremal graph theory*.

The four color problem remained unsolved for more than a century. In 1969 Heinrich Heesch published a method for solving the problem using computers.[17] A computer-aided proof produced in 1976 by Kenneth Appel and Wolfgang Haken makes fundamental use of the notion of "discharging" developed by Heesch.[18][19] The proof involved checking the properties of 1,936 configurations by computer, and was not fully accepted at the time due to its complexity. A simpler proof considering only 633 configurations was given twenty years later by Robertson, Seymour, Sanders and Thomas.[20]

The autonomous development of topology from 1860 and 1930 fertilized graph theory back through the works of Jordan, Kuratowski and Whitney. Another important factor of common development of graph theory and topology came from the use of the techniques of modern algebra. The first example of such a use comes from the work of the physicist Gustav Kirchhoff, who published in 1845 his Kirchhoff's circuit laws for calculating the voltage and current in electric circuits.

The introduction of probabilistic methods in graph theory, especially in the study of Erdős and Rényi of the asymptotic probability of graph connectivity, gave rise to yet another branch, known as *random graph theory*, which has been a fruitful source of graph-theoretic results.

1.3.4 Graph drawing

Main article: Graph drawing

Graphs are represented visually by drawing a dot or circle for every vertex, and drawing an arc between two vertices if they are connected by an edge. If the graph is directed, the direction is indicated by drawing an arrow.

A graph drawing should not be confused with the graph itself (the abstract, non-visual structure) as there are several ways to structure the graph drawing. All that matters is which vertices are connected to which others by how many edges and not the exact layout. In practice it is of-

ten difficult to decide if two drawings represent the same graph. Depending on the problem domain some layouts may be better suited and easier to understand than others.

The pioneering work of W. T. Tutte was very influential in the subject of graph drawing. Among other achievements, he introduced the use of linear algebraic methods to obtain graph drawings.

Graph drawing also can be said to encompass problems that deal with the crossing number and its various generalizations. The crossing number of a graph is the minimum number of intersections between edges that a drawing of the graph in the plane must contain. For a planar graph, the crossing number is zero by definition.

Drawings on surfaces other than the plane are also studied.

1.3.5 Graph-theoretic data structures

Main article: Graph (abstract data type)

There are different ways to store graphs in a computer system. The data structure used depends on both the graph structure and the algorithm used for manipulating the graph. Theoretically one can distinguish between list and matrix structures but in concrete applications the best structure is often a combination of both. List structures are often preferred for sparse graphs as they have smaller memory requirements. Matrix structures on the other hand provide faster access for some applications but can consume huge amounts of memory.

List structures include the incidence list, an array of pairs of vertices, and the adjacency list, which separately lists the neighbors of each vertex: Much like the incidence list, each vertex has a list of which vertices it is adjacent to.

Matrix structures include the incidence matrix, a matrix of 0's and 1's whose rows represent vertices and whose columns represent edges, and the adjacency matrix, in which both the rows and columns are indexed by vertices. In both cases a 1 indicates two adjacent objects and a 0 indicates two non-adjacent objects. The Laplacian matrix is a modified form of the adjacency matrix that incorporates information about the degrees of the vertices, and is useful in some calculations such as Kirchhoff's theorem on the number of spanning trees of a graph. The distance matrix, like the adjacency matrix, has both its rows and columns indexed by vertices, but rather than containing a 0 or a 1 in each cell it contains the length of a shortest path between two vertices.

1.3.6 Problems in graph theory

Enumeration

There is a large literature on graphical enumeration: the problem of counting graphs meeting specified conditions. Some of this work is found in Harary and Palmer (1973).

Subgraphs, induced subgraphs, and minors

A common problem, called the subgraph isomorphism problem, is finding a fixed graph as a subgraph in a given graph. One reason to be interested in such a question is that many graph properties are *hereditary* for subgraphs, which means that a graph has the property if and only if all subgraphs have it too. Unfortunately, finding maximal subgraphs of a certain kind is often an NP-complete problem. For example:

- Finding the largest complete subgraph is called the clique problem (NP-complete).

A similar problem is finding induced subgraphs in a given graph. Again, some important graph properties are hereditary with respect to induced subgraphs, which means that a graph has a property if and only if all induced subgraphs also have it. Finding maximal induced subgraphs of a certain kind is also often NP-complete. For example:

- Finding the largest edgeless induced subgraph or independent set is called the independent set problem (NP-complete).

Still another such problem, the minor containment problem, is to find a fixed graph as a minor of a given graph. A minor or subcontraction of a graph is any graph obtained by taking a subgraph and contracting some (or no) edges. Many graph properties are hereditary for minors, which means that a graph has a property if and only if all minors have it too. For example, Wagner's Theorem states:

- A graph is planar if it contains as a minor neither the complete bipartite graph $K_{3,3}$ (see the Three-cottage problem) nor the complete graph K_5.

A similar problem, the subdivision containment problem, is to find a fixed graph as a subdivision of a given graph. A subdivision or homeomorphism of a graph is any graph obtained by subdividing some (or no) edges. Subdivision containment is related to graph properties such as planarity. For example, Kuratowski's Theorem states:

- A graph is planar if it contains as a subdivision neither the complete bipartite graph $K_{3,3}$ nor the complete graph K_5.

Another problem in subdivision containment is Kelmans-Seymour conjecture:

- Every 5-vertex-connected graph that is not planar contains a subdivision of the 5-vertex complete graph K_5.

Another class of problems has to do with the extent to which various species and generalizations of graphs are determined by their *point-deleted subgraphs*. For example:

- The reconstruction conjecture

Graph coloring

Many problems have to do with various ways of coloring graphs, for example:

- Four-color theorem
- Strong perfect graph theorem
- Erdős–Faber–Lovász conjecture (unsolved)
- Total coloring conjecture, also called Behzad's conjecture (unsolved)
- List coloring conjecture (unsolved)
- Hadwiger conjecture (graph theory) (unsolved)

Subsumption and unification

Constraint modeling theories concern families of directed graphs related by a partial order. In these applications, graphs are ordered by specificity, meaning that more constrained graphs—which are more specific and thus contain a greater amount of information—are subsumed by those that are more general. Operations between graphs include evaluating the direction of a subsumption relationship between two graphs, if any, and computing graph unification. The unification of two argument graphs is defined as the most general graph (or the computation thereof) that is consistent with (i.e. contains all of the information in) the inputs, if such a graph exists; efficient unification algorithms are known.

For constraint frameworks which are strictly compositional, graph unification is the sufficient satisfiability and combination function. Well-known applications include automatic theorem proving and modeling the elaboration of linguistic structure.

Route problems

- Hamiltonian path problem
- Minimum spanning tree
- Route inspection problem (also called the "Chinese postman problem")
- Seven bridges of Königsberg
- Shortest path problem
- Steiner tree
- Three-cottage problem
- Traveling salesman problem (NP-hard)

Network flow

There are numerous problems arising especially from applications that have to do with various notions of flows in networks, for example:

- Max flow min cut theorem

Visibility problems

- Museum guard problem

Covering problems

Covering problems in graphs are specific instances of subgraph-finding problems, and they tend to be closely related to the clique problem or the independent set problem.

- Set cover problem
- Vertex cover problem

Decomposition problems

Decomposition, defined as partitioning the edge set of a graph (with as many vertices as necessary accompanying the edges of each part of the partition), has a wide variety of question. Often, it is required to decompose a graph into subgraphs isomorphic to a fixed graph; for instance, decomposing a complete graph into Hamiltonian cycles. Other problems specify a family of graphs into which a given graph should be decomposed, for instance, a family of cycles, or decomposing a complete graph Kn into $n - 1$ specified trees having, respectively, 1, 2, 3, …, $n - 1$ edges.

Some specific decomposition problems that have been studied include:

- Arboricity, a decomposition into as few forests as possible
- Cycle double cover, a decomposition into a collection of cycles covering each edge exactly twice
- Edge coloring, a decomposition into as few matchings as possible
- Graph factorization, a decomposition of a regular graph into regular subgraphs of given degrees

Graph classes

Many problems involve characterizing the members of various classes of graphs. Some examples of such questions are below:

- Enumerating the members of a class
- Characterizing a class in terms of forbidden substructures
- Ascertaining relationships among classes (e.g. does one property of graphs imply another)
- Finding efficient algorithms to decide membership in a class
- Finding representations for members of a class

1.3.7 See also

- Gallery of named graphs
- Glossary of graph theory
- List of graph theory topics
- List of unsolved problems in graph theory
- Publications in graph theory

Related topics

- Algebraic graph theory
- Citation graph
- Conceptual graph
- Data structure
- Disjoint-set data structure
- Dual-phase evolution
- Entitative graph
- Existential graph
- Graph algebra
- Graph automorphism
- Graph coloring
- Graph database
- Graph data structure
- Graph drawing
- Graph equation
- Graph rewriting
- Graph sandwich problem
- Graph property
- Intersection graph
- Logical graph
- Loop
- Network theory
- Null graph
- Pebble motion problems
- Percolation
- Perfect graph
- Quantum graph
- Random regular graphs
- Semantic networks
- Spectral graph theory
- Strongly regular graphs
- Symmetric graphs
- Transitive reduction
- Tree data structure

Algorithms

- Bellman–Ford algorithm
- Dijkstra's algorithm
- Ford–Fulkerson algorithm
- Kruskal's algorithm
- Nearest neighbour algorithm
- Prim's algorithm
- Depth-first search
- Breadth-first search

Subareas

- Algebraic graph theory
- Geometric graph theory
- Extremal graph theory
- Probabilistic graph theory
- Topological graph theory

Related areas of mathematics

- Combinatorics
- Group theory
- Knot theory
- Ramsey theory

Generalizations

- Hypergraph
- Abstract simplicial complex

Prominent graph theorists

- Alon, Noga
- Berge, Claude
- Bollobás, Béla
- Bondy, Adrian John
- Brightwell, Graham
- Chudnovsky, Maria
- Chung, Fan
- Dirac, Gabriel Andrew
- Erdős, Paul
- Euler, Leonhard
- Faudree, Ralph
- Golumbic, Martin
- Graham, Ronald
- Harary, Frank
- Heawood, Percy John
- Kotzig, Anton
- Kőnig, Dénes
- Lovász, László
- Murty, U. S. R.
- Nešetřil, Jaroslav
- Rényi, Alfréd
- Ringel, Gerhard
- Robertson, Neil
- Seymour, Paul
- Sudakov, Benny
- Szemerédi, Endre
- Thomas, Robin
- Thomassen, Carsten
- Turán, Pál
- Tutte, W. T.
- Whitney, Hassler

1.3.8 Notes

[1] See, for instance, Iyanaga and Kawada, **69 J**, p. 234 or Biggs, p. 4.

[2] See, for instance, Graham et al., p. 5.

[3] Hale, Scott A. (2013). "Multilinguals and Wikipedia Editing". arXiv:1312.0976 [cs.CY].

[4] Mashaghi, A.; et al. (2004). "Investigation of a protein complex network". *European Physical Journal B*. **41** (1): 113–121. doi:10.1140/epjb/e2004-00301-0.

[5] Grandjean, Martin (2016). "A social network analysis of Twitter: Mapping the digital humanities community". *Cogent Arts & Humanities*. **3** (1): 1171458. doi:10.1080/23311983.2016.1171458.

[6] Grandjean, Martin (2015). "Social network analysis and visualization: Moreno's Sociograms revisited". Redesigned network strictly based on Moreno (1934), *Who Shall Survive*.

[7] Rosen, Kenneth H. *Discrete mathematics and its applications* (7th ed.). New York: McGraw-Hill. ISBN 978-0-07-338309-5.

[8] Biggs, N.; Lloyd, E.; Wilson, R. (1986), *Graph Theory, 1736-1936*, Oxford University Press

[9] Cauchy, A.L. (1813), "Recherche sur les polyèdres - premier mémoire", *Journal de l'École Polytechnique*, 9 (Cahier 16): 66–86.

[10] L'Huillier, S.-A.-J. (1861), "Mémoire sur la polyèdrométrie", *Annales de Mathématiques*, **3**: 169–189.

[11] Cayley, A. (1857), "On the theory of the analytical forms called trees", *Philosophical Magazine*, Series IV, **13** (85): 172–176, doi:10.1017/CBO9780511703690.046

[12] Cayley, A. (1875), "Ueber die Analytischen Figuren, welche in der Mathematik Bäume genannt werden und ihre Anwendung auf die Theorie chemischer Verbindungen", *Berichte der deutschen Chemischen Gesellschaft*, **8** (2): 1056–1059, doi:10.1002/cber.18750080252.

[13] Sylvester, James Joseph (1878). "Chemistry and Algebra". *Nature*. **17**: 284. doi:10.1038/017284a0.

[14] Tutte, W.T. (2001), *Graph Theory*, Cambridge University Press, p. 30, ISBN 978-0-521-79489-3, retrieved 2016-03-14

[15] Gardner, Martin (1992), *Fractal Music, Hypercards, and more…Mathematical Recreations from Scientific American*, W. H. Freeman and Company, p. 203

[16] Society for Industrial and Applied Mathematics (2002), "The George Polya Prize", *Looking Back, Looking Ahead: A SIAM History* (PDF), p. 26, retrieved 2016-03-14

[17] Heinrich Heesch: Untersuchungen zum Vierfarbenproblem. Mannheim: Bibliographisches Institut 1969.

[18] Appel, K.; Haken, W. (1977), "Every planar map is four colorable. Part I. Discharging", *Illinois J. Math.*, **21**: 429–490.

[19] Appel, K.; Haken, W. (1977), "Every planar map is four colorable. Part II. Reducibility", *Illinois J. Math.*, **21**: 491–567.

[20] Robertson, N.; Sanders, D.; Seymour, P.; Thomas, R. (1997), "The four color theorem", *Journal of Combinatorial Theory Series B*, **70**: 2–44, doi:10.1006/jctb.1997.1750.

1.3.9 References

- Berge, Claude (1958), *Théorie des graphes et ses applications*, Collection Universitaire de Mathématiques, **II**, Paris: Dunod. English edition, Wiley 1961; Methuen & Co, New York 1962; Russian, Moscow 1961; Spanish, Mexico 1962; Roumanian, Bucharest 1969; Chinese, Shanghai 1963; Second printing of the 1962 first English edition, Dover, New York 2001.
- Biggs, N.; Lloyd, E.; Wilson, R. (1986), *Graph Theory, 1736–1936*, Oxford University Press.
- Bondy, J.A.; Murty, U.S.R. (2008), *Graph Theory*, Springer, ISBN 978-1-84628-969-9.
- Bollobás, Béla; Riordan, O.M (2003), *Mathematical results on scale-free random graphs in "Handbook of Graphs and Networks" (S. Bornholdt and H.G. Schuster (eds)), Wiley VCH, Weinheim, 1st ed.*.
- Chartrand, Gary (1985), *Introductory Graph Theory*, Dover, ISBN 0-486-24775-9.
- Gibbons, Alan (1985), *Algorithmic Graph Theory*, Cambridge University Press.
- Reuven Cohen, Shlomo Havlin (2010), *Complex Networks: Structure, Robustness and Function*, Cambridge University Press.
- Golumbic, Martin (1980), *Algorithmic Graph Theory and Perfect Graphs*, Academic Press.
- Harary, Frank (1969), *Graph Theory*, Reading, MA: Addison-Wesley.
- Harary, Frank; Palmer, Edgar M. (1973), *Graphical Enumeration*, New York, NY: Academic Press.
- Mahadev, N.V.R.; Peled, Uri N. (1995), *Threshold Graphs and Related Topics*, North-Holland.
- Mark Newman (2010), *Networks: An Introduction*, Oxford University Press.

1.3.10 External links

- Graph theory with examples
- Hazewinkel, Michiel, ed. (2001), "Graph theory", *Encyclopedia of Mathematics*, Springer, ISBN 978-1-55608-010-4
- Graph theory tutorial
- A searchable database of small connected graphs
- Image gallery: graphs at the Wayback Machine (archived February 6, 2006)
- Concise, annotated list of graph theory resources for researchers
- rocs — a graph theory IDE
- The Social Life of Routers — non-technical paper discussing graphs of people and computers
- Graph Theory Software — tools to teach and learn graph theory
- Online books, and library resources in your library and in other libraries about graph theory
- A list of graph algorithms with references and links to graph library implementations

Online textbooks

- Phase Transitions in Combinatorial Optimization Problems, Section 3: Introduction to Graphs (2006) by Hartmann and Weigt
- Digraphs: Theory Algorithms and Applications 2007 by Jorgen Bang-Jensen and Gregory Gutin
- Graph Theory, by Reinhard Diestel

Chapter 2

The Basics

2.1 Element

In mathematics, an **element**, or **member**, of a set is any one of the distinct objects that make up that set.

2.1.1 Sets

Writing $A = \{1, 2, 3, 4\}$ means that the elements of the set A are the numbers 1, 2, 3 and 4. Sets of elements of A, for example $\{1, 2\}$, are subsets of A.

Sets can themselves be elements. For example, consider the set $B = \{1, 2, \{3, 4\}\}$. The elements of B are *not* 1, 2, 3, and 4. Rather, there are only three elements of B, namely the numbers 1 and 2, and the set $\{3, 4\}$.

The elements of a set can be anything. For example, C = { red, green, blue }, is the set whose elements are the colors red, green and blue.

2.1.2 Notation and terminology

IV. *De classibus.*

Signo K significatur *classis*, sive entium aggregatio.
Signum ϵ significat *est*. Ita *a* ϵ *b* legitur *a est quoddam b*; *a* ϵ K significat *a est quaedam classis*; *a* ϵ P significat *a est quaedam propositio*.

First usage of the symbol ϵ in the work Arithmetices principia nova methodo exposita by Giuseppe Peano.

The relation "is an element of", also called **set membership**, is denoted by the symbol "∈". Writing

$$x \in A$$

means that "x is an element of A". Equivalent expressions are "x is a member of A", "x belongs to A", "x is in A" and "x lies in A". The expressions "A includes x" and "A contains x" are also used to mean set membership, however some authors use them to mean instead "x is a subset of A".[1] Logician George Boolos strongly urged that "contains" be used for membership only and "includes" for the subset relation only.[2]

Another possible notation for the same relation is

$$A \ni x,$$

meaning "A contains x", though it is used less often.

The negation of set membership is denoted by the symbol "∉". Writing

$$x \notin A$$

means that "x is not an element of A".

The symbol ϵ was first used by Giuseppe Peano 1889 in his work Arithmetices principia nova methodo exposita. Here he wrote on page X:

> "Signum ϵ significat est. Ita a ϵ b legitur a est quoddam b; ..."

which means

> "The symbol ϵ means *is*. So a ϵ b is read as a *is a* b; ..."

The symbol itself is a stylized lowercase Greek letter epsilon ("ε"), the first letter of the word ἐστί, which means "is".

The Unicode characters for these symbols are U+2208 ('element of'), U+220B ('contains as member') and U+2209 ('not an element of'). The equivalent LaTeX commands are "\in", "\ni" and "\notin". Mathematica has commands "\[Element]" and "\[NotElement]".

2.1.3 Cardinality of sets

Main article: Cardinality

The number of elements in a particular set is a property known as cardinality; informally, this is the size of a set. In the above examples the cardinality of the set A is 4, while the cardinality of either of the sets B and C is 3.

An infinite set is a set with an infinite number of elements, while a finite set is a set with a finite number of elements. The above examples are examples of finite sets. An example of an infinite set is the set of positive integers = { 1, 2, 3, 4, ... }.

2.1.4 Examples

Using the sets defined above, namely A = {1, 2, 3, 4 }, B = {1, 2, {3, 4}} and C = { red, green, blue }:

- $2 \in A$
- {3,4} $\in B$
- 3,4 $\notin B$
- {3,4} is a member of B
- Yellow $\notin C$
- The cardinality of D = { 2, 4, 8, 10, 12 } is finite and equal to 5.
- The cardinality of P = { 2, 3, 5, 7, 11, 13, ...} (the prime numbers) is infinite (this was proven by Euclid).

2.1.5 References

[1] Eric Schechter (1997). *Handbook of Analysis and Its Foundations*. Academic Press. ISBN 0-12-622760-8. p. 12

[2] George Boolos (February 4, 1992). *24.243 Classical Set Theory (lecture)*. (Speech). Massachusetts Institute of Technology, Cambridge, MA.

2.1.6 Further reading

- Halmos, Paul R. (1974) [1960], *Naive Set Theory*, Undergraduate Texts in Mathematics (Hardcover ed.), NY: Springer-Verlag, ISBN 0-387-90092-6 - "Naive" means that it is not fully axiomatized, not that it is silly or easy (Halmos's treatment is neither).
- Jech, Thomas (2002), "Set Theory", *Stanford Encyclopedia of Philosophy*
- Suppes, Patrick (1972) [1960], *Axiomatic Set Theory*, NY: Dover Publications, Inc., ISBN 0-486-61630-4 - Both the notion of set (a collection of members), membership or element-hood, the axiom of extension, the axiom of separation, and the union axiom (Suppes calls it the sum axiom) are needed for a more thorough understanding of "set element".

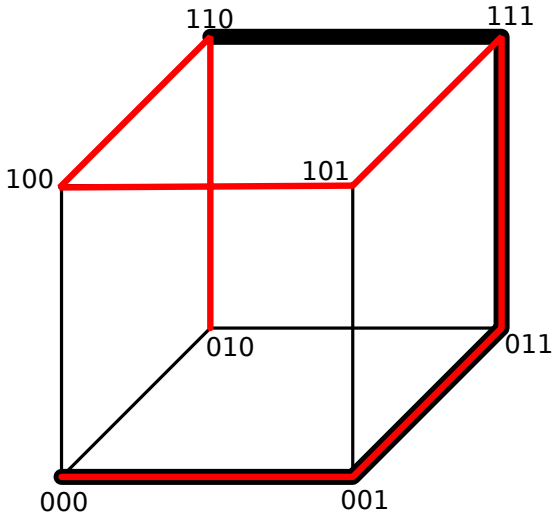
2.1.7 External links

- Weisstein, Eric W. "Element". *MathWorld*.

2.2 Path

For the family of graphs known as paths, see Path graph.

In graph theory, a **path** in a graph is a finite or infinite



A hypercube graph showing a Hamiltonian path in red, and a longest induced path in bold black.

sequence of edges which connect a sequence of vertices which, by most definitions, are all distinct from one another. In a directed graph, a **directed path** (sometimes called **dipath**[1]) is again a sequence of edges (or arcs) which connect a sequence of vertices, but with the added restriction that the edges all be directed in the same direction.

Paths are fundamental concepts of graph theory, described in the introductory sections of most graph theory texts. See e.g. Bondy and Murty (1976), Gibbons (1985), or Diestel (2005). Korte et al. (1990) cover more advanced algorithmic topics concerning paths in graphs.

2.2.1 Definitions

A path is a trail in which all vertices (except possibly the first and last) are distinct. A trail is a walk in which all edges are distinct. A walk of length k in a graph is an alternating sequence of vertices and edges, $v_0, e_0, v_1, e_1, v_2, \ldots, v_{k-1}, e_{k-1}, v_k$, which begins and ends with vertices. If the graph is undirected, then the endpoints of e_i are v_i and v_{i+1} . If the graph is directed, then e_i is an arc from v_i to v_{i+1} . An infinite path is an alternating sequence of the same type described here, but with no first or last vertex, and a semi-infinite path (also ray) has a first vertex, v_0 , but no last vertex. Most authors require that all of the edges and vertices be distinct from one another. However, some authors do not make this requirement, and instead use the term **simple path** to refer to a path which contains no repeated vertices.

A weighted graph associates a value (*weight*) with every

edge in the graph. The *weight of a path* in a weighted graph is the sum of the weights of the traversed edges. Sometimes the words *cost* or *length* are used instead of weight.

2.2.2 Examples

- A graph is connected if there are paths containing each pair of vertices.
- A directed graph is strongly connected if there are oppositely oriented directed paths containing each pair of vertices.
- A path such that no graph edges connect two non-consecutive path vertices is called an induced path.
- A path that includes every vertex of the graph is known as a Hamiltonian path.
- Two paths are *vertex-independent* (alternatively, *internally vertex-disjoint*) if they do not have any internal vertex in common. Similarly, two paths are *edge-independent* (or *edge-disjoint*) if they do not have any internal edge in common. Two internally vertex-disjoint paths are edge-disjoint, but the converse is not necessarily true.
- The distance between two vertices in a graph is the length of a shortest path between them, if one exists, and otherwise the distance is infinity.
- The diameter of a connected graph is the largest distance (defined above) between pairs of vertices of the graph.

2.2.3 Finding paths

Several algorithms exist to find shortest and longest paths in graphs, with the important distinction that the former problem is computationally much easier than the latter.

Dijkstra's algorithm produces a list of shortest paths from a source vertex to every other vertex in directed and undirected graphs with non-negative edge weights (or no edge weights), whilst the Bellman–Ford algorithm can be applied to directed graphs with negative edge weights. The Floyd–Warshall algorithm can be used to find the shortest paths between all pairs of vertices in weighted directed graphs.

2.2.4 See also

- Glossary of graph theory
- Path graph
- Polygonal chain
- Shortest path problem
- Longest path problem
- Dijkstra's algorithm
- Bellman–Ford algorithm
- Floyd–Warshall algorithm
- Self-avoiding walk

2.2.5 References

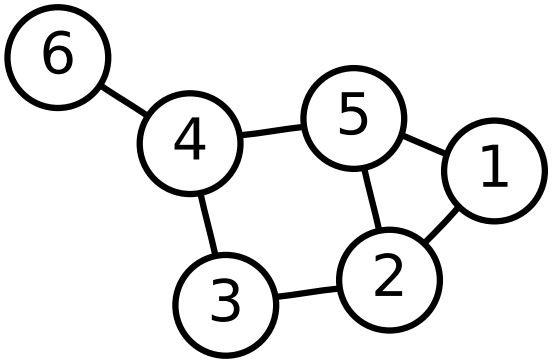
[1] Graph Structure Theory: Proceedings of the AMS-IMS-SIAM Joint Summer Research Conference on Graph Minors, Held June 22 to July 5, 1991,, p.205

- Bondy, J. A.; Murty, U. S. R. (1976). *Graph Theory with Applications*. North Holland. pp. 12–21. ISBN 0-444-19451-7.
- Diestel, Reinhard (2005). *Graph Theory* (3rd ed.). Graduate Texts in Mathematics, vol. 173, Springer-Verlag. pp. 6–9. ISBN 3-540-26182-6.
- Gibbons, A. (1985). *Algorithmic Graph Theory*. Cambridge University Press. pp. 5–6. ISBN 0-521-28881-9.
- Korte, Bernhard; Lovász, László; Prömel, Hans Jürgen; Schrijver, Alexander (Eds.) (1990). *Paths, Flows, and VLSI-Layout*. Algorithms and Combinatorics 9, Springer-Verlag. ISBN 0-387-52685-4.

2.3 Graph

This article is about sets of vertices connected by edges. For graphs of mathematical functions, see Graph of a function. For other uses, see Graph (disambiguation).

In mathematics, and more specifically in graph theory,



A drawing of a labeled graph on 6 vertices and 7 edges.

a **graph** is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related". The objects correspond to mathematical abstractions called *vertices* (also called *nodes* or *points*) and each

of the related pairs of vertices is called an *edge* (also called an *arc* or *line*).[1] Typically, a graph is depicted in diagrammatic form as a set of dots for the vertices, joined by lines or curves for the edges. Graphs are one of the objects of study in discrete mathematics.

The edges may be directed or undirected. For example, if the vertices represent people at a party, and there is an edge between two people if they shake hands, then this graph is undirected because any person A can shake hands with a person B only if B also shakes hands with A. In contrast, if any edge from a person A to a person B corresponds to A's admiring B, then this graph is directed, because admiration is not necessarily reciprocated. The former type of graph is called an *undirected graph* and the edges are called *undirected edges* while the latter type of graph is called a *directed graph* and the edges are called *directed edges*.

Graphs are the basic subject studied by graph theory. The word "graph" was first used in this sense by J. J. Sylvester in 1878.[2][3]

2.3.1 Definitions

Definitions in graph theory vary. The following are some of the more basic ways of defining graphs and related mathematical structures.

Graph

In the most common sense of the term,[4] a *graph* is an ordered pair $G = (V, E)$ comprising a set V of *vertices*, *nodes* or *points* together with a set E of *edges*, *arcs* or *lines*, which are 2-element subsets of V (i.e., an edge is associated with two vertices, and the association takes the form of the unordered pair of the vertices). To avoid ambiguity, this type of graph may be described precisely as *undirected* and *simple*.

Other senses of *graph* stem from different conceptions of the edge set. In one more general conception, [5] E is a set together with a relation of *incidence* that associates with each edge two vertices. In another generalized notion, E is a multiset of unordered pairs of (not necessarily distinct) vertices. Many authors call this type of object a multigraph or pseudograph.

All of these variants and others are described more fully below.

The vertices belonging to an edge are called the *ends* or *end vertices* of the edge. A vertex may exist in a graph and not belong to an edge.

V and E are usually taken to be finite, and many of the well-known results are not true (or are rather different) for *infinite graphs* because many of the arguments fail in the infinite case. Moreover, V is often assumed to be non-empty, but E is allowed to be the empty set. The *order* of a graph is $|V|$, its number of vertices. The *size* of a graph is $|E|$, its number of edges. The *degree* or *valency* of a vertex is the number of edges that connect to it, where an edge that connects to the vertex at both ends (a loop) is counted twice.

For an edge $\{x, y\}$, graph theorists usually use the somewhat shorter notation xy.

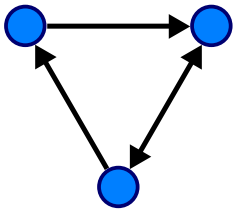
Adjacency relation

The edges E of an undirected graph G induce a symmetric binary relation ~ on V that is called the *adjacency relation* of G. Specifically, for each edge $\{x, y\}$, the vertices x and y are said to be *adjacent* to one another, which is denoted $x \sim y$.

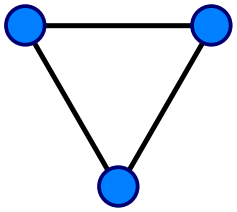
2.3.2 Types of graphs

Distinction in terms of the main definition

As stated above, in different contexts it may be useful to refine the term *graph* with different degrees of generality. Whenever it is necessary to draw a strict distinction, the following terms are used. Most commonly, in modern texts in graph theory, unless stated otherwise, *graph* means "undirected simple finite graph" (see the definitions below).



A directed graph.



A simple undirected graph with three vertices and three edges. Each vertex has degree two, so this is also a regular graph.

Undirected graph An *undirected graph* is a graph in which edges have no orientation. The edge (x, y) is identical to the edge (y, x), i.e., they are not ordered pairs, but sets $\{x, y\}$ (or 2-multisets) of vertices. The maximum number of edges in an undirected graph without a loop is $n(n-1)/2$.

Directed graph Main article: Directed graph

A *directed graph* or *digraph* is a graph in which edges have orientations. It is written as an ordered pair $G = (V, A)$ (sometimes $G = (V, E)$) with

- V a set whose elements are called *vertices*, *nodes*, or *points*;
- A a set of ordered pairs of vertices, called *arrows*, *directed edges* (sometimes simply *edges* with the corresponding set named E instead of A), *directed arcs*, or *directed lines*.

An arrow (x, y) is considered to be directed *from* x *to* y; y is called the *head* and x is called the *tail* of the arrow; y is said to be a *direct successor* of x and x is said to be a *direct predecessor* of y. If a path leads from x to y, then y is said to be a *successor* of x and *reachable* from x, and x is said to be a *predecessor* of y. The arrow (y, x) is called the *inverted arrow* of (x, y).

A directed graph G is called *symmetric* if, for every arrow in G, the corresponding inverted arrow also belongs to G. A symmetric loopless directed graph $G = (V, A)$ is equivalent to a simple undirected graph $G' = (V, E)$, where the pairs of inverse arrows in A correspond one-to-one with the edges in E; thus the number of edges in G' is $| E | = | A |/2$, that is half the number of arrows in G.

Oriented graph An *oriented graph* is a directed graph in which at most one of (x, y) and (y, x) may be arrows of the graph. That is, it is a directed graph that can be formed as an orientation of an undirected graph. However, some authors use "oriented graph" to mean the same as "directed graph".

Mixed graph Main article: Mixed graph

A *mixed graph* is a graph in which some edges may be directed and some may be undirected. It is written as an ordered triple $G = (V, E, A)$ with V, E, and A defined as above. Directed and undirected graphs are special cases.

Multigraph Main article: Multigraph

Multiple edges are two or more edges that connect the same two vertices. A *loop* is an edge (directed or undirected) that connects a vertex to itself; it may be permitted or not, according to the application. In this context, an edge with two different ends is called a *link*.

A *multigraph*, as opposed to a simple graph, is an undirected graph in which multiple edges (and sometimes loops) are allowed.

Where graphs are defined so as to *disallow* both multiple edges and loops, a multigraph is often defined to mean a graph which can have both multiple edges and loops,[6] although many use the term *pseudograph* for this meaning.[7] Where graphs are defined so as to *allow* both multiple edges and loops, a multigraph is often defined to mean a graph without loops.[8]

Simple graph A *simple graph*, as opposed to a multigraph, is an undirected graph in which both multiple edges and loops are disallowed. In a simple graph the edges form a *set* (rather than a multiset) and each edge is an unordered pair of *distinct* vertices. In a simple graph with n vertices, the degree of every vertex is at most $n - 1$.

Quiver Main article: Quiver (mathematics)

A *quiver* or *multidigraph* is a directed multigraph. A quiver may also have directed loops in it.

Weighted graph A *weighted graph* is a graph in which a number (the weight) is assigned to each edge.[9] Such weights might represent for example costs, lengths or capacities, depending on the problem at hand. Some authors call such a graph a *network*.[10] Weighted correlation networks can be defined by soft-thresholding the pairwise correlations among variables (e.g. gene measurements). Such graphs arise in many contexts, for example in shortest path problems such as the traveling salesman problem.

Half-edges, loose edges In certain situations it can be helpful to allow edges with only one end, called *half-edges*, or no ends, called *loose edges*; see the articles Signed graphs and Biased graphs.

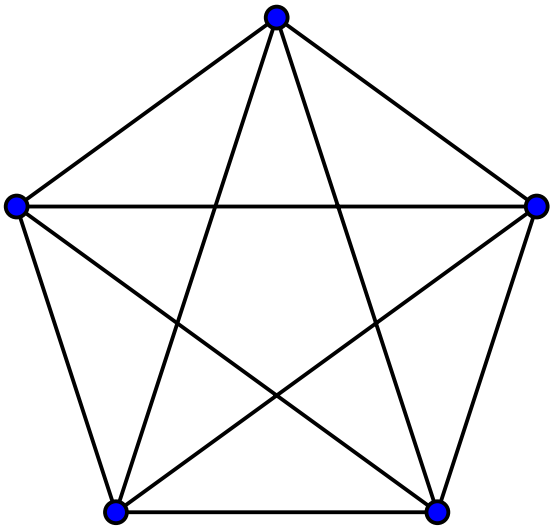
Important classes of graph

Regular graph Main article: Regular graph

A *regular graph* is a graph in which each vertex has the same number of neighbours, i.e., every vertex has the same degree. A regular graph with vertices of degree k is called a k-regular graph or regular graph of degree k.

Complete graph Main article: Complete graph
A *complete graph* is a graph in which each pair of vertices is joined by an edge. A complete graph contains all possible edges.

Finite graph A *finite graph* is a graph in which the vertex set and the edge set are finite sets. Otherwise, it is called an *infinite graph*.



A complete graph with 5 vertices. Each vertex has an edge to every other vertex.

Most commonly in graph theory it is implied that the graphs discussed are finite. If the graphs are infinite, that is usually specifically stated.

Connected graph Main article: Connectivity (graph theory)

In an undirected graph, an unordered pair of vertices $\{x, y\}$ is called *connected* if a path leads from x to y. Otherwise, the unordered pair is called *disconnected*.

A *connected graph* is an undirected graph in which every unordered pair of vertices in the graph is connected. Otherwise, it is called a *disconnected graph*.

In a directed graph, an ordered pair of vertices (x, y) is called *strongly connected* if a directed path leads from x to y. Otherwise, the ordered pair is called *weakly connected* if an undirected path leads from x to y after replacing all of its directed edges with undirected edges. Otherwise, the ordered pair is called *disconnected*.

A *strongly connected graph* is a directed graph in which every ordered pair of vertices in the graph is strongly connected. Otherwise, it is called a *weakly connected graph* if every ordered pair of vertices in the graph is weakly connected. Otherwise it is called a *disconnected graph*.

A *k-vertex-connected graph* or *k-edge-connected graph* is a graph in which no set of $k - 1$ vertices (respectively, edges) exists that, when removed, disconnects the graph. A k-vertex-connected graph is often called simply a *k-connected graph*.

Bipartite graph Main article: Bipartite graph

A *bipartite graph* is a graph in which the vertex set can be partitioned into two sets, W and X, so that no two vertices in W share a common edge and no two vertices in X share a common edge. Alternatively, it is a graph with a chromatic number of 2.

In a complete bipartite graph, the vertex set is the union of two disjoint sets, W and X, so that every vertex in W is adjacent to every vertex in X but there are no edges within W or X.

Path graph Main article: Path graph

A *path graph* or *linear graph* of order $n \geq 2$ is a graph in which the vertices can be listed in an order v_1, v_2, …, vn such that the edges are the $\{vi, vi_{+1}\}$ where $i = 1, 2, \ldots, n - 1$. Path graphs can be characterized as connected graphs in which the degree of all but two vertices is 2 and the degree of the two remaining vertices is 1. If a path graph occurs as a subgraph of another graph, it is a path in that graph.

Planar graph Main article: Planar graph

A *planar graph* is a graph whose vertices and edges can be drawn in a plane such that no two of the edges intersect.

Cycle graph Main article: Cycle graph

A *cycle graph* or *circular graph* of order $n \geq 3$ is a graph in which the vertices can be listed in an order v_1, v_2, …, vn such that the edges are the $\{vi, vi_{+1}\}$ where $i = 1, 2, \ldots, n - 1$, plus the edge $\{vn, v_1\}$. Cycle graphs can be characterized as connected graphs in which the degree of all vertices is 2. If a cycle graph occurs as a subgraph of another graph, it is a cycle or circuit in that graph.

Tree Main article: Tree (graph theory)

A *tree* is a connected graph with no cycles.

A *forest* is a graph with no cycles, i.e. the disjoint union of one or more trees.

Advanced classes More advanced kinds of graphs are:

- Petersen graph and its generalizations;
- perfect graphs;
- cographs;
- chordal graphs;
- other graphs with large automorphism groups: vertex-transitive, arc-transitive, and distance-transitive graphs;

- strongly regular graphs and their generalizations distance-regular graphs.

2.3.3 Properties of graphs

See also: Glossary of graph theory and Graph property

Two edges of a graph are called *adjacent* if they share a common vertex. Two arrows of a directed graph are called *consecutive* if the head of the first one is the tail of the second one. Similarly, two vertices are called *adjacent* if they share a common edge (*consecutive* if the first one is the tail and the second one is the head of an arrow), in which case the common edge is said to *join* the two vertices. An edge and a vertex on that edge are called *incident*.

The graph with only one vertex and no edges is called the *trivial graph*. A graph with only vertices and no edges is known as an *edgeless graph*. The graph with no vertices and no edges is sometimes called the *null graph* or *empty graph*, but the terminology is not consistent and not all mathematicians allow this object.

Normally, the vertices of a graph, by their nature as elements of a set, are distinguishable. This kind of graph may be called *vertex-labeled*. However, for many questions it is better to treat vertices as indistinguishable. (Of course, the vertices may be still distinguishable by the properties of the graph itself, e.g., by the numbers of incident edges.) The same remarks apply to edges, so graphs with labeled edges are called *edge-labeled*. Graphs with labels attached to edges or vertices are more generally designated as *labeled*. Consequently, graphs in which vertices are indistinguishable and edges are indistinguishable are called *unlabeled*. (Note that in the literature, the term *labeled* may apply to other kinds of labeling, besides that which serves only to distinguish different vertices or edges.)

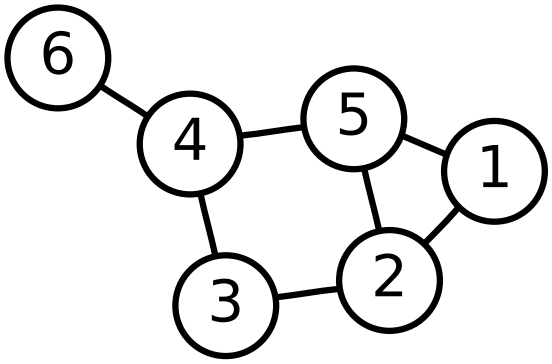
The category of all graphs is the slice category $\mathrm{Set} \downarrow D$ where D: Set → Set is the functor taking a set s to $s \times s$.

2.3.4 Examples

- The diagram at right is a graphic representation of the following graph:

$$V = \{1, 2, 3, 4, 5, 6\};$$
$$E = \{\{1, 2\}, \{1, 5\}, \{2, 3\}, \{2, 5\}, \{3, 4\}, \{4, 5\}, \{4, 6\}\}.$$

- In category theory, a small category can be represented by a directed multigraph in which the objects of the category are represented as vertices and the morphisms as directed edges. Then, the functors between categories induce some, but not necessarily all, of the digraph morphisms of the graph.



A graph with six nodes.

- In computer science, directed graphs are used to represent knowledge (e.g., conceptual graph), finite state machines, and many other discrete structures.
- A binary relation R on a set X defines a directed graph. An element x of X is a direct predecessor of an element y of X if and only if xRy.
- A directed graph can model information networks such as Twitter, with one user following another.[11][12]
- Particularly regular examples of directed graphs are given by the Cayley graphs of finitely-generated groups, as well as Schreier coset graphs

2.3.5 Graph operations

Main article: Graph operations

There are several operations that produce new graphs from initial ones, which might be classified into the following categories:

- *unary operations*, which create a new graph from an initial one, such as:
  - edge contraction,
  - line graph,
  - dual graph,
  - complement graph,
  - graph rewriting;
- *binary operations*, which create a new graph from two initial ones, such as:
  - disjoint union of graphs,
  - cartesian product of graphs,
  - tensor product of graphs,
  - strong product of graphs,
  - lexicographic product of graphs,
  - series-parallel graphs.

2.3.6 Generalizations

In a hypergraph, an edge can join more than two vertices.

An undirected graph can be seen as a simplicial complex consisting of 1-simplices (the edges) and 0-simplices (the vertices). As such, complexes are generalizations of graphs since they allow for higher-dimensional simplices.

Every graph gives rise to a matroid.

In model theory, a graph is just a structure. But in that case, there is no limitation on the number of edges: it can be any cardinal number, see continuous graph.

In computational biology, power graph analysis introduces power graphs as an alternative representation of undirected graphs.

In geographic information systems, geometric networks are closely modeled after graphs, and borrow many concepts from graph theory to perform spatial analysis on road networks or utility grids.

2.3.7 See also

- Conceptual graph
- Dual graph
- Glossary of graph theory
- Graph (abstract data type)
- Graph database
- Graph drawing
- Graph theory
- Hypergraph
- List of graph theory topics
- List of publications in graph theory
- Network theory

2.3.8 Notes

[1] Trudeau, Richard J. (1993). *Introduction to Graph Theory* (Corrected, enlarged republication. ed.). New York: Dover Pub. p. 19. ISBN 978-0-486-67870-2. Retrieved 8 August 2012. A graph is an object consisting of two sets called its *vertex set* and its *edge set*.

[2] See:

- J. J. Sylvester (February 7, 1878) "Chemistry and algebra," *Nature*, *17* : 284. From page 284: "Every invariant and covariant thus becomes expressible by a *graph* precisely identical with a Kekuléan diagram or chemicograph."
- J. J. Sylvester (1878) "On an application of the new atomic theory to the graphical representation of the invariants and covariants of binary quantics, — with three appendices," *American Journal of Mathematics, Pure and Applied*, *1* (1) : 64-90. The term "graph" first appears in this paper on page 65.

[3] Gross, Jonathan L.; Yellen, Jay (2004). *Handbook of graph theory*. CRC Press. p. 35. ISBN 978-1-58488-090-5.

[4] See, for instance, Iyanaga and Kawada, *69 J*, p. 234 or Biggs, p. 4.

[5] See, for instance, Graham et al., p. 5.

[6] For example, see. Bollobás, p. 7 and Diestel, p. 25.

[7] Gross (1998), p. 3, Gross (2003), p. 205, Harary, p.10, and Zwillinger, p. 220.

[8] For example, see Balakrishnan, p. 1, Gross (2003), p. 4, and Zwillinger, p. 220.

[9] Fletcher, Peter; Hoyle, Hughes; Patty, C. Wayne (1991). *Foundations of Discrete Mathematics* (International student ed.). Boston: PWS-KENT Pub. Co. p. 463. ISBN 0-53492-373-9. A *weighted graph* is a graph in which a number *w(e)*, called its *weight*, is assigned to each edge *e*.

[10] Strang, Gilbert (2005), *Linear Algebra and Its Applications* (4th ed.), Brooks Cole, ISBN 0-03-010567-6

[11] Grandjean, Martin (2016). "A social network analysis of Twitter: Mapping the digital humanities community". *Cogent Arts & Humanities*. **3** (1): 1171458. doi:10.1080/23311983.2016.1171458.

[12] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Bosagh Zadeh WTF: The who-to-follow system at Twitter, *Proceedings of the 22nd international conference on World Wide Web*

2.3.9 References

- Balakrishnan, V. K. (1997-02-01). *Graph Theory* (1st ed.). McGraw-Hill. ISBN 0-07-005489-4.
- Berge, Claude (1958). *Théorie des graphes et ses applications* (in French). Dunod, Paris: Collection Universitaire de Mathématiques, II. pp. viii+277. Translation: -. Dover, New York: Wiley. 2001 [1962].
- Biggs, Norman (1993). *Algebraic Graph Theory* (2nd ed.). Cambridge University Press. ISBN 0-521-45897-8.
- Bollobás, Béla (2002-08-12). *Modern Graph Theory* (1st ed.). Springer. ISBN 0-387-98488-7.
- Bang-Jensen, J.; Gutin, G. (2000). *Digraphs: Theory, Algorithms and Applications*. Springer.

- Diestel, Reinhard (2005). *Graph Theory* (3rd ed.). Berlin, New York: Springer-Verlag. ISBN 978-3-540-26183-4..
- Graham, R.L.; Grötschel, M.; Lovász, L, eds. (1995). *Handbook of Combinatorics*. MIT Press. ISBN 0-262-07169-X.
- Gross, Jonathan L.; Yellen, Jay (1998-12-30). *Graph Theory and Its Applications*. CRC Press. ISBN 0-8493-3982-0.
- Gross, Jonathan L.; Yellen, Jay, eds. (2003-12-29). *Handbook of Graph Theory*. CRC. ISBN 1-58488-090-2.
- Harary, Frank (January 1995). *Graph Theory*. Addison Wesley Publishing Company. ISBN 0-201-41033-8.
- Iyanaga, Shôkichi; Kawada, Yukiyosi (1977). *Encyclopedic Dictionary of Mathematics*. MIT Press. ISBN 0-262-09016-3.
- Zwillinger, Daniel (2002-11-27). *CRC Standard Mathematical Tables and Formulae* (31st ed.). Chapman & Hall/CRC. ISBN 1-58488-291-3.

2.3.10 Further reading

- Trudeau, Richard J. (1993). *Introduction to Graph Theory* (Corrected, enlarged republication. ed.). New York: Dover Publications. ISBN 978-0-486-67870-2. Retrieved 8 August 2012.

2.3.11 External links

- Weisstein, Eric W. "Graph". *MathWorld*.

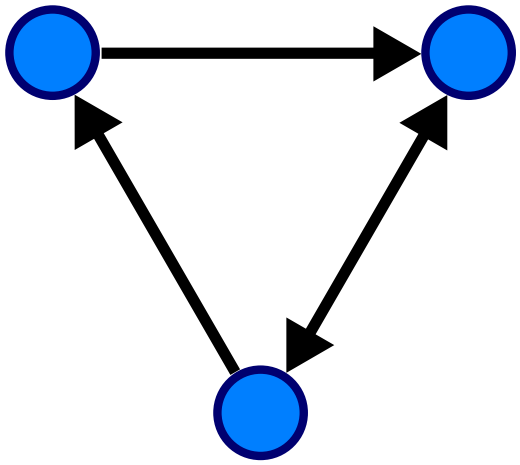
2.4 Directed graph

In mathematics, and more specifically in graph theory, a **directed graph** (or **digraph**) is a graph (that is a set of vertices connected by edges), where the edges have a direction associated with them.

2.4.1 Definition

In formal terms, a directed graph is an ordered pair $G = (V, A)$ where[1]

- V is a set whose elements are called *vertices*, *nodes*, or *points*;
- A is a set of ordered pairs of vertices, called *arrows*, *directed edges* (sometimes simply *edges* with the corresponding set named E instead of A), *directed arcs*, or *directed lines*.



A simple directed graph.

It differs from an ordinary or undirected graph, in that the latter is defined in terms of unordered pairs of vertices, which are usually called *edges*, *arcs*, or *lines*.

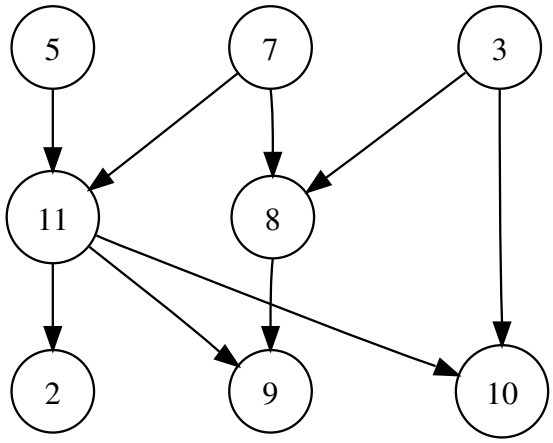
The aforementioned definition does not allow a directed graph to have multiple arrows with same source and target nodes, but some authors consider a broader definition that allow directed graphs to have such multiple arrows (namely, they allow the arrows set to be a multiset). More specifically, these entities are addressed as **directed multigraphs** (or **multidigraphs**).
On the other hand, the aforementioned definition allows a directed graph to have loops (that is, arrows that connect nodes with themselves), but some authors consider a narrower definition that doesn't allow directed graphs to have loops.[2] More specifically, directed graphs without loops are addressed as **simple directed graphs**, while directed graphs with loops are addressed as *loop-digraphs* (see section Types of directed graphs).

2.4.2 Types of directed graphs

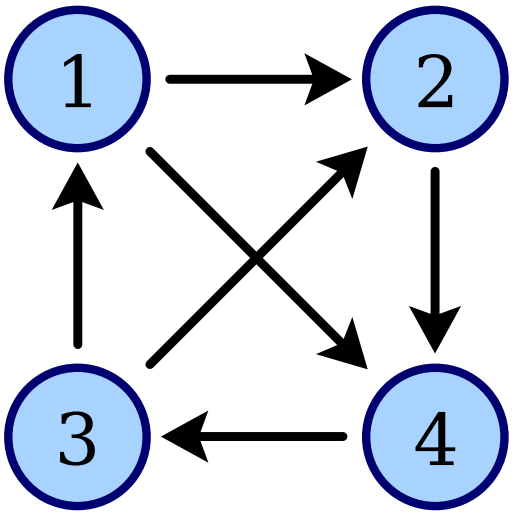
See also: Graph (discrete mathematics) § Types of graphs

Subclasses

- **Symmetric directed graphs** are directed graphs where all edges are bidirected (that is, for every arrow that belongs to the digraph, the corresponding inversed arrow also belongs to it).
- **Simple directed graphs** are directed graphs that have no loops (arrows that connect vertices to themselves) and no multiple arrows with same source and target nodes. As already introduced, in case of multiple arrows the entity is usually addressed



A simple acyclic directed graph.



A tournament on 4 vertices.

as *directed multigraph*. Some authors describe digraphs with loops as *loop-digraphs*.[2]

- **Complete directed graphs** are simple directed graphs where each pair of vertices is joined by a symmetric pair of directed arrows (it is equivalent to an undirected complete graph with the edges replaced by pairs of inverse arrows). It follows that a complete digraph is symmetric.
- **Oriented graphs** are directed graphs having no bidirected edges (i.e. at most one of (x, y) and (y, x) may be arrows of the graph). It follows that a directed graph is an oriented graph iff it hasn't any 2-cycle.[3]
  - **Tournaments** are oriented graphs obtained by choosing a direction for each edge in undirected complete graphs.
  - **Directed acyclic graphs** (DAGs) are directed graphs with no directed cycles.
    - *Multitrees* are DAGs in which no two directed paths from a single starting vertex meet back at the same ending vertex.
  - *Oriented trees* or *polytrees* are DAGs formed by orienting the edges of undirected acyclic graphs.
    - *Rooted trees* are oriented trees in which all edges of the underlying undirected tree are directed away from the roots.

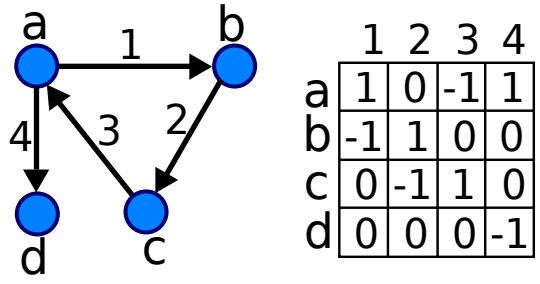
Digraphs with supplementary properties

This list is incomplete; you can help by expanding it.

- **Weighted directed graphs** (also known as **directed networks**) are (simple) directed graphs with *weights* assigned to their arrows, similarly to weighted graphs (which are also known as undirected networks or weighted networks).[2]
  - **Flow networks** are weighted directed graphs where two nodes are distinguished, a *source* and a *sink*.
- **Rooted directed graphs** (also known as **flow graphs**) are digraphs in which a vertex has been distinguished as the root.
  - *Control flow graphs* are rooted digraphs used in computer science as a representation of the paths that might be traversed through a program during its execution.
- **Signal-flow graphs** are directed graphs in which nodes represent system variables and branches (edges, arcs, or arrows) represent functional connections between pairs of nodes.
- **Flow graphs** are digraphs associated with a set of linear algebraic or differential equations.
- **State diagrams** are directed multigraphs that represent finite state machines.
- **Commutative diagrams** are digraphs used in category theory, where the vertices represent (mathematical) objects and the arrows represent morphisms, with the property that all directed paths with the same start and endpoints lead to the same result by composition.
- In the theory of Lie groups, a **quiver** Q is a directed graph serving as the domain of, and thus characterizing the shape of, a *representation* V defined as a functor, specifically an object of the functor category $\mathrm{FinVct}_K^{F(Q)}$ where $F(Q)$ is the free category on

Q consisting of paths in *Q* and FinVct*K* is the category of finite-dimensional vector spaces over a field *K*. Representations of a quiver label its vertices with vector spaces and its edges (and hence paths) compatibly with linear transformations between them, and transform via natural transformations.

2.4.3 Basic terminology



Oriented graph with corresponding incidence matrix.

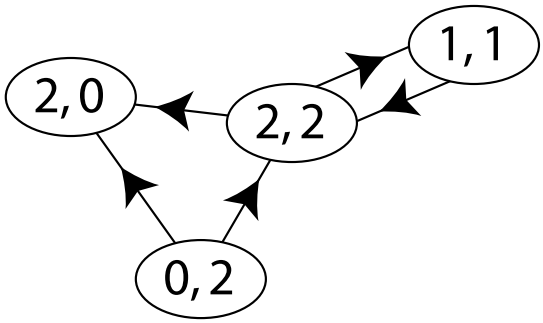
An arrow (x, y) is considered to be directed *from x to y*; *y* is called the *head* and *x* is called the *tail* of the arrow; *y* is said to be a *direct successor* of *x* and *x* is said to be a *direct predecessor* of *y*. If a path leads from *x* to *y*, then *y* is said to be a *successor* of *x* and *reachable* from *x*, and *x* is said to be a *predecessor* of *y*. The arrow (y, x) is called the *inverted arrow* of (x, y).

The adjacency matrix of a multidigraph with loops is the integer-valued matrix with rows and columns corresponding to the vertices, where a nondiagonal entry *aij* is the number of arrows from vertex *i* to vertex *j*, and the diagonal entry *aii* is the number of loops at vertex *i*. The adjacency matrix of a directed graph is unique up to identical permutation of rows and columns.

Another matrix representation for a directed graph is its incidence matrix.

See direction for more definitions.

2.4.4 Indegree and outdegree



A directed graph with vertices labeled (indegree, outdegree).

For a vertex, the number of head ends adjacent to a vertex is called the *indegree* of the vertex and the number of tail ends adjacent to a vertex is its *outdegree* (called "branching factor" in trees).

Let $G = (V, A)$ and $v \in V$. The indegree of v is denoted $\deg^-(v)$ and its outdegree is denoted $\deg^+(v)$.

A vertex with $\deg^-(v) = 0$ is called a *source*, as it is the origin of each of its outcoming arrows. Similarly, a vertex with $\deg^+(v) = 0$ is called a *sink*, since it is the end of each of its incoming arrows.

If a vertex is neither a *source* nor a *sink*, it is called an *internal*.

The *degree sum formula* states that, for a directed graph,

$$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = |A|.$$

If for every vertex $v \in V$, $\deg^+(v) = \deg^-(v)$, the graph is called a *balanced directed graph*.[4]

2.4.5 Degree sequence

The degree sequence of a directed graph is the list of its indegree and outdegree pairs; for the above example we have degree sequence ((2, 0), (2, 2), (0, 2), (1, 1)). The degree sequence is a directed graph invariant so isomorphic directed graphs have the same degree sequence. However, the degree sequence does not, in general, uniquely identify a directed graph; in some cases, non-isomorphic digraphs have the same degree sequence.

The directed graph realization problem is the problem of finding a directed graph with the degree sequence a given sequence of positive integer pairs. (Trailing pairs of zeros may be ignored since they are trivially realized by adding an appropriate number of isolated vertices to the directed graph.) A sequence which is the degree sequence of some directed graph, i.e. for which the directed graph realization problem has a solution, is called a directed graphic or directed graphical sequence. This problem can either be solved by the Kleitman–Wang algorithm or by the Fulkerson–Chen–Anstee theorem.

2.4.6 Directed graph connectivity

Main article: Connectivity (graph theory)

A directed graph is *weakly connected* (or just *connected*[5]) if the undirected *underlying graph* obtained by replacing all directed edges of the graph with undirected edges is a connected graph. A directed graph is *strongly connected* or *strong* if it contains a directed path from *x* to *y* and a directed path from *y* to *x* for every pair of vertices $\{x, y\}$. The *strong components* are the maximal strongly connected subgraphs.

2.4.7 See also

- Coates graph
- Flow chart
- Glossary of graph theory
- Graph theory
- Graph (abstract data type)
- Network theory
- Orientation
- Preorder
- Transpose graph
- Vertical constraint graph

2.4.8 Notes

[1] Bang-Jensen & Gutin (2000). Diestel (2005), Section 1.10. Bondy & Murty (1976), Section 10.

[2] Chartrand, Gary (1977). *Introductory Graph Theory*. Courier Corporation. ISBN 9780486247755.

[3] Diestel (2005), Section 1.10.

[4] Satyanarayana, Bhavanari; Prasad, Kuncham Syam, *Discrete Mathematics and Graph Theory*, PHI Learning Pvt. Ltd., p. 460, ISBN 978-81-203-3842-5; Brualdi, Richard A. (2006), *Combinatorial Matrix Classes*, Encyclopedia of Mathematics and Its Applications, **108**, Cambridge University Press, p. 51, ISBN 978-0-521-86565-4.

[5] Bang-Jensen & Gutin (2000) p. 19 in the 2007 edition; p. 20 in the 2nd edition (2009).

2.4.9 References

- Bang-Jensen, Jørgen; Gutin, Gregory (2000), *directed graphs: Theory, Algorithms and Applications*, Springer, ISBN 1-85233-268-9
(the corrected 1st edition of 2007 is now freely available on the authors' site; the 2nd edition appeared in 2009 ISBN 1-84800-997-6).
- Bondy, John Adrian; Murty, U. S. R. (1976), *Graph Theory with Applications*, North-Holland, ISBN 0-444-19451-7.
- Diestel, Reinhard (2005), *Graph Theory* (3rd ed.), Springer, ISBN 3-540-26182-6 (the electronic 3rd edition is freely available on author's site).
- Harary, Frank; Norman, Robert Z.; Cartwright, Dorwin (1965), *Structural Models: An Introduction to the Theory of Directed Graphs*, New York: Wiley.
- Number of directed graphs (or directed graphs) with n nodes from On-Line Encyclopedia of Integer Sequences

2.5 Complete graph

In the mathematical field of graph theory, a **complete graph** is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge. A **complete digraph** is a directed graph in which every pair of distinct vertices is connected by a pair of unique edges (one in each direction).

Graph theory itself is typically dated as beginning with Leonhard Euler's 1736 work on the Seven Bridges of Königsberg. However, drawings of complete graphs, with their vertices placed on the points of a regular polygon, appeared already in the 13th century, in the work of Ramon Llull.[1] Such a drawing is sometimes referred to as a **mystic rose**.[2]

2.5.1 Properties

The complete graph on n vertices is denoted by K_n. Some sources claim that the letter K in this notation stands for the German word *komplett*,[3] but the German name for a complete graph, *vollständiger Graph*, does not contain the letter K, and other sources state that the notation honors the contributions of Kazimierz Kuratowski to graph theory.[4]

K_n has $n(n-1)/2$ edges (a triangular number), and is a regular graph of degree $n-1$. All complete graphs are their own maximal cliques. They are maximally connected as the only vertex cut which disconnects the graph is the complete set of vertices. The complement graph of a complete graph is an empty graph.

If the edges of a complete graph are each given an orientation, the resulting directed graph is called a tournament.

The number of matchings of the complete graphs are given by the telephone numbers

> 1, 1, 2, 4, 10, 26, 76, 232, 764, 2620, 9496, 35696, 140152, 568504, 2390480, 10349536, 46206736, ... (sequence A000085 in the OEIS).

These numbers give the largest possible value of the Hosoya index for an n-vertex graph.[5] The number of perfect matchings of the complete graph K_n (with n even) is given by the double factorial $(n-1)!!$.[6]

The crossing numbers up to K_{27} are known, with K_{28} requiring either 7233 or 7234 crossings. Further values are collected by the Rectilinear Crossing Number project.[7] Crossing numbers for K_n are

> 0, 0, 0, 0, 1, 3, 9, 19, 36, 62, 102, 153, 229, 324, 447, 603, 798, 1029, 1318, 1657, 2055, 2528, 3077, 3699, 4430, 5250, 6180, ... (sequence A014540 in the OEIS).

2.5.2 Geometry and topology

A complete graph with n nodes represents the edges of an $(n-1)$-simplex. Geometrically K_3 forms the edge set of a triangle, K_4 a tetrahedron, etc. The Császár polyhedron, a nonconvex polyhedron with the topology of a torus, has the complete graph K_7 as its skeleton. Every neighborly polytope in four or more dimensions also has a complete skeleton.

K_1 through K_4 are all planar graphs. However, every planar drawing of a complete graph with five or more vertices must contain a crossing, and the nonplanar complete graph K_5 plays a key role in the characterizations of planar graphs: by Kuratowski's theorem, a graph is planar if and only if it contains neither K_5 nor the complete bipartite graph $K_{3,3}$ as a subdivision, and by Wagner's theorem the same result holds for graph minors in place of subdivisions. As part of the Petersen family, K_6 plays a similar role as one of the forbidden minors for linkless embedding.[8] In other words, and as Conway and Gordon[9] proved, every embedding of K_6 into three-dimensional space is intrinsically linked, with at least one pair of linked triangles. Conway and Gordon also showed that any three-dimensional embedding of K_7 contains a Hamiltonian cycle that is embedded in space as a nontrivial knot.

2.5.3 Examples

Complete graphs on n vertices, for n between 1 and 12, are shown below along with the numbers of edges:

2.5.4 See also

- Complete bipartite graph

2.5.5 References

[1] Knuth, Donald E. (2013), "Two thousand years of combinatorics", in Wilson, Robin; Watkins, John J., *Combinatorics: Ancient and Modern*, Oxford University Press, pp. 7–37.

[2] *Mystic Rose*, nrich.maths.org, retrieved 23 January 2012.

[3] Gries, David; Schneider, Fred B. (1993), *A Logical Approach to Discrete Math*, Springer-Verlag, p. 436.

[4] Pirnot, Thomas L. (2000), *Mathematics All Around*, Addison Wesley, p. 154, ISBN 9780201308150.

[5] Tichy, Robert F.; Wagner, Stephan (2005), "Extremal problems for topological indices in combinatorial chemistry" (PDF), *Journal of Computational Biology*, **12** (7): 1004–1013, doi:10.1089/cmb.2005.12.1004.

[6] Callan, David (2009), *A combinatorial survey of identities for the double factorial*, arXiv:0906.1317.

[7] Oswin Aichholzer. "Rectilinear Crossing Number project".

[8] Robertson, Neil; Seymour, P. D.; Thomas, Robin (1993), "Linkless embeddings of graphs in 3-space", *Bulletin of the American Mathematical Society*, **28** (1): 84–89, arXiv:math/9301216, doi:10.1090/S0273-0979-1993-00335-5, MR 1164063.

[9] Conway, J. H.; Cameron Gordon (1983). "Knots and Links in Spatial Graphs". *J. Graph Th.* **7** (4): 445–453. doi:10.1002/jgt.3190070410.

2.5.6 External links

- Weisstein, Eric W. "Complete Graph". *MathWorld*.

Chapter 3

Elaborations

3.1 Tree

In mathematics, and more specifically in graph theory, a **tree** is an undirected graph in which any two vertices are connected by *exactly one* path. In other words, any acyclic connected graph is a tree. A **forest** is a disjoint union of trees.

The various kinds of data structures referred to as trees in computer science have underlying graphs that are trees in graph theory, although such data structures are generally **rooted trees**. A rooted tree may be directed, called a **directed rooted tree**,[1][2] either making all its edges point away from the root—in which case it is called an **arborescence**,[3] **branching**,[4] or **out-tree**[4]—or making all its edges point towards the root—in which case it is called an **anti-arborescence**[5] or **in-tree**.[6] A rooted tree itself has been defined by some authors as a directed graph.[7][8][9]

The term "tree" was coined in 1857 by the British mathematician Arthur Cayley.[10]

3.1.1 Definitions

Tree

A *tree* is an undirected graph G that satisfies any of the following equivalent conditions:

- G is connected and has no cycles.
- G is acyclic, and a simple cycle is formed if any edge is added to G.
- G is connected, but is not connected if any single edge is removed from G.
- G is connected and the 3-vertex complete graph K_3 is not a minor of G.
- Any two vertices in G can be connected by a unique simple path.

If G has finitely many vertices, say n of them, then the above statements are also equivalent to any of the following conditions:

- G is connected and has $n - 1$ edges.
- G has no simple cycles and has $n - 1$ edges.

As elsewhere in graph theory, the order-zero graph (graph with no vertices) is generally excluded from consideration: while it is vacuously connected as a graph (any two vertices can be connected by a path), it is not 0-connected (or even (-1)-connected) in algebraic topology, unlike non-empty trees, and violates the "one more vertex than edges" relation.

An **internal vertex** (or **inner vertex** or **branch vertex**) is a vertex of degree at least 2. Similarly, an **external vertex** (or *outer vertex*, *terminal vertex* or *leaf*) is a vertex of degree 1.

An *irreducible tree* (or *series-reduced tree*) is a tree in which there is no vertex of degree 2.

Forest

A *forest* is an undirected graph, all of whose connected components are trees; in other words, the graph consists of a disjoint union of trees. Equivalently, a forest is an undirected acyclic graph. As special cases, an empty graph, a single tree, and the discrete graph on a set of vertices (that is, the graph with these vertices that has no edges), are examples of forests.

Polytree

Main article: Polytree

A *polytree*[11] (or *oriented tree*[12][13] or *singly connected network*[14]) is a directed acyclic graph (DAG) whose underlying undirected graph is a tree. In other words, if we replace its directed edges with undirected edges, we obtain an undirected graph that is both connected and acyclic.

A *directed tree* is a directed graph which would be a tree if the directions on the edges were ignored, i.e. a polytree. Some authors restrict the phrase to the case where the edges are all directed towards a particular vertex, or all directed away from a particular vertex (see arborescence).

Rooted tree

A *rooted tree* is a tree in which one vertex has been designated the *root*. The edges of a rooted tree can be assigned a natural orientation, either *away from* or *towards* the root, in which case the structure becomes a *directed rooted tree*. When a directed rooted tree has an orientation away from the root, it is called an *arborescence*, *branching*, or *out-tree*; when it has an orientation towards the root, it is called an *anti-arborescence* or *in-tree*. The *tree-order* is the partial ordering on the vertices of a tree with $u < v$ if and only if the unique path from the root to v passes through u. A rooted tree which is a subgraph of some graph G is a normal tree if the ends of every edge in G are comparable in this tree-order whenever those ends are vertices of the tree (Diestel 2005, p. 15). Rooted trees, often with additional structure such as ordering of the neighbors at each vertex, are a key data structure in computer science; see tree data structure.

In a context where trees are supposed to have a root, a tree without any designated root is called a *free tree*.

A *labeled tree* is a tree in which each vertex is given a unique label. The vertices of a labeled tree on n vertices are typically given the labels 1, 2, …, n. A *recursive tree* is a labeled rooted tree where the vertex labels respect the tree order (i.e., if $u < v$ for two vertices u and v, then the label of u is smaller than the label of v).

In a rooted tree, the *parent* of a vertex is the vertex connected to it on the path to the root; every vertex except the root has a unique parent. A *child* of a vertex v is a vertex of which v is the parent. A *descendent* of any vertex v is any vertex which is either the child of v or is (recursively) the descendent of any of the children of v. A *sibling* to a vertex v is any other vertex on the tree which has the same parent as v. The root is an external vertex if it has precisely one child. A leaf is different from the root.

The *height* of a vertex in a rooted tree is the length of the longest downward path to a leaf from that vertex. The *height* of the tree is the height of the root. The *depth* of a vertex is the length of the path to its root (*root path*). This is commonly needed in the manipulation of the various self-balancing trees, AVL trees in particular. The root has depth zero, leaves have height zero, and a tree with only a single vertex (hence both a root and leaf) has depth and height zero. Conventionally, an empty tree (tree with no vertices, if such are allowed) has depth and height −1.

A *k-ary tree* is a rooted tree in which each vertex has at most k children.[15] 2-ary trees are often called *binary trees*, while 3-ary trees are sometimes called *ternary trees*.

Ordered tree

An *ordered tree* (or *plane tree*) is a rooted tree in which an ordering is specified for the children of each vertex. This is called a "plane tree" because an ordering of the children is equivalent to an embedding of the tree in the plane, with the root at the top and the children of each vertex lower than that vertex. Given an embedding of a rooted tree in the plane, if one fixes a direction of children, say left to right, then an embedding gives an ordering of the children. Conversely, given an ordered tree, and conventionally drawing the root at the top, then the child vertices in an ordered tree can be drawn left-to-right, yielding an essentially unique planar embedding .

3.1.2 Facts

- Every tree is a bipartite graph and a median graph. Every tree with only countably many vertices is a planar graph.
- Every connected graph G admits a spanning tree, which is a tree that contains every vertex of G and whose edges are edges of G.
- Every connected graph with only countably many vertices admits a normal spanning tree (Diestel 2005, Prop. 8.2.4).
- There exist connected graphs with uncountably many vertices which do not admit a normal spanning tree (Diestel 2005, Prop. 8.5.2).
- Every finite tree with n vertices, with $n > 1$, has at least two terminal vertices (leaves). This minimal number of leaves is characteristic of path graphs; the maximal number, $n - 1$, is attained only by star graphs. The number of leaves is at least the maximal vertex degree.
- For any three vertices in a tree, the three paths between them have exactly one vertex in common.
- Otter showed that any n-vertex tree has either a unique center vertex, whose removal splits the tree into subtrees of fewer than n/2 vertices, or a unique center edge, whose removal splits the tree into two subtrees of exactly n/2 vertices.

3.1.3 Enumeration

Labeled trees

Cayley's formula states that there are n^{n-2} trees on n labeled vertices. A classic proof uses Prüfer sequences, which naturally show a stronger result: the number of trees with vertices 1, 2, …, n of degrees d_1, d_2, …, dn respectively, is the multinomial coefficient

$$\binom{n-2}{d_1-1, d_2-1, \ldots, d_n-1}.$$

A more general problem is to count spanning trees in an undirected graph, which is addressed by the matrix tree

theorem. (Cayley's formula is the special case of spanning trees in a complete graph.) The similar problem of counting all the subtrees regardless of size has been shown to be #P-complete in the general case (Jerrum (1994)).

Unlabeled trees

See also: Combinatorial species § Unlabelled structures

Counting the number of unlabeled free trees is a harder problem. No closed formula for the number $t(n)$ of trees with n vertices up to graph isomorphism is known. The first few values of $t(n)$ are

> 1, 1, 1, 1, 2, 3, 6, 11, 23, 47, 106, 235, 551, 1301, 3159, … (sequence A000055 in the OEIS).

Otter (1948) proved the asymptotic estimate

$$t(n) \sim C\alpha^n n^{-5/2} \quad \text{as} n \to \infty,$$

with the values C and α known to be approximately 0.534949606… and 2.95576528565… (sequence A051491 in the OEIS), respectively. (Here, $f \sim g$ means that $\lim_{n\to\infty} f/g = 1$.) This is a consequence of his asymptotic estimate for the number $r(n)$ of unlabeled rooted trees with n vertices:

$$r(n) \sim D\alpha^n n^{-3/2} \quad \text{as} n \to \infty,$$

with D around 0.43992401257… and the same α as above (cf. Knuth (1997), chap. 2.3.4.4 and Flajolet & Sedgewick (2009), chap. VII.5, p. 475).

The first few values of $r(n)$ are[16]

> 1, 1, 2, 4, 9, 20, 48, 115, 286, 719, 1842, 4766, 12486, 32973, …

3.1.4 Types of trees

- A *star tree* is a tree which consists of a single internal vertex (and $n - 1$ leaves). In other words, a star tree of order n is a tree of order n with as many leaves as possible.
- A *caterpillar tree* is a tree in which all vertices are within distance 1 of a central path subgraph.
- A *lobster tree* is a tree in which all vertices are within distance 2 of a central path subgraph.

3.1.5 See also

- Hypertree
- Tree structure
- Tree (data structure)
- Decision tree
- Pseudoforest
- Unrooted binary tree

3.1.6 Notes

[1] Stanley Gill Williamson (1985). *Combinatorics for Computer Science*. Courier Dover Publications. p. 288. ISBN 978-0-486-42076-9.

[2] Mehran Mesbahi; Magnus Egerstedt (2010). *Graph Theoretic Methods in Multiagent Networks*. Princeton University Press. p. 38. ISBN 1-4008-3535-6.

[3] Ding-Zhu Du; Ker-I Ko; Xiaodong Hu (2011). *Design and Analysis of Approximation Algorithms*. Springer Science & Business Media. p. 108. ISBN 978-1-4614-1701-9.

[4] Jonathan L. Gross; Jay Yellen; Ping Zhang (2013). *Handbook of Graph Theory, Second Edition*. CRC Press. p. 116. ISBN 978-1-4398-8018-0.

[5] Bernhard Korte; Jens Vygen (2012). *Combinatorial Optimization: Theory and Algorithms* (5th ed.). Springer Science & Business Media. p. 28. ISBN 978-3-642-24488-9.

[6] Kurt Mehlhorn; Peter Sanders (2008). *Algorithms and Data Structures: The Basic Toolbox* (PDF). Springer Science & Business Media. p. 52. ISBN 978-3-540-77978-0.

[7] David Makinson (2012). *Sets, Logic and Maths for Computing*. Springer Science & Business Media. pp. 167–168. ISBN 978-1-4471-2499-3.

[8] Kenneth Rosen (2011). *Discrete Mathematics and Its Applications, 7th edition*. McGraw-Hill Science. p. 747. ISBN 978-0-07-338309-5.

[9] Alexander Schrijver (2003). *Combinatorial Optimization: Polyhedra and Efficiency*. Springer. p. 34. ISBN 3-540-44389-4.

[10] Cayley (1857) "On the theory of the analytical forms called trees," *Philosophical Magazine*, 4th series, **13** : 172–176.
However it should be mentioned that in 1847, K.G.C. von Staudt, in his book *Geometrie der Lage* (Nürnberg, (Germany): Bauer und Raspe, 1847), presented a proof of Euler's polyhedron theorem which relies on trees on pages 20–21. Also in 1847, the German physicist Gustav Kirchhoff investigated electrical circuits and found a relation between the number (n) of wires/resistors (branches), the number (m) of junctions (vertices), and the number (μ) of

loops (faces) in the circuit. He proved the relation via an argument relying on trees. See: Kirchhoff, G. R. (1847) "Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird" (On the solution of equations to which one is led by the investigation of the linear distribution of galvanic currents), *Annalen der Physik und Chemie*, **72** (12) : 497–508.

[11] See Dasgupta (1999).

[12] See Harary & Sumner (1980).

[13] See Simion (1991).

[14] See Kim & Pearl (1983).

[15] See Black, Paul E. (4 May 2007). "k-ary tree". U.S. National Institute of Standards and Technology. Retrieved 8 February 2015.

[16] See Li (1996).

3.1.7 References

- Dasgupta, Sanjoy (1999), "Learning polytrees", *in Proc. 15th Conference on Uncertainty in Artificial Intelligence (UAI 1999), Stockholm, Sweden, July–August 1999* (PDF), pp. 134–141.
- Harary, Frank; Sumner, David (1980), "The dichromatic number of an oriented tree", *Journal of Combinatorics, Information & System Sciences*, **5** (3): 184–187, MR 603363.
- Kim, Jin H.; Pearl, Judea (1983), "A computational model for causal and diagnostic reasoning in inference engines", *in Proc. 8th International Joint Conference on Artificial Intelligence (IJCAI 1983), Karlsruhe, Germany, August 1983* (PDF), pp. 190–193.
- Li, Gang (1996), "Generation of Rooted Trees and Free Trees", *M.S. Thesis, Dept. of Computer Science, University of Victoria, BC, Canada* (PDF), p. 9.
- Simion, Rodica (1991), "Trees with 1-factors and oriented trees", *Discrete Mathematics*, **88** (1): 93–104, doi:10.1016/0012-365X(91)90061-6, MR 1099270.

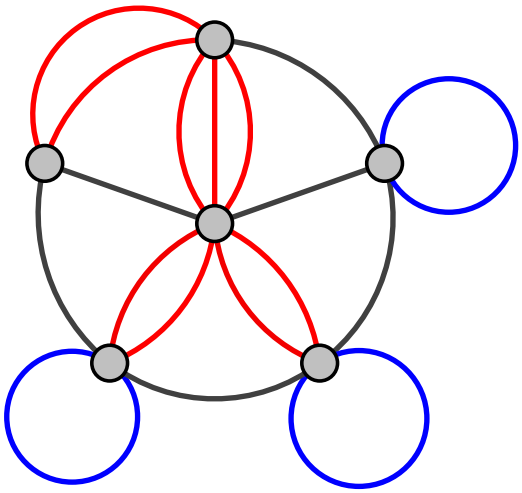
3.1.8 Further reading

- Diestel, Reinhard (2005), *Graph Theory* (3rd ed.), Berlin, New York: Springer-Verlag, ISBN 978-3-540-26183-4.
- Flajolet, Philippe; Sedgewick, Robert (2009), *Analytic Combinatorics*, Cambridge University Press, ISBN 978-0-521-89806-5
- Hazewinkel, Michiel, ed. (2001), "Tree", *Encyclopedia of Mathematics*, Springer, ISBN 978-1-55608-010-4
- Knuth, Donald E. (November 14, 1997), *The Art of Computer Programming Volume 1: Fundamental Algorithms* (3rd ed.), Addison-Wesley Professional
- Jerrum, Mark (1994), "Counting trees in a graph is #P-complete", *Information Processing Letters*, **51** (3): 111–116, doi:10.1016/0020-0190(94)00085-9, ISSN 0020-0190.
- Otter, Richard (1948), "The Number of Trees", *Annals of Mathematics. Second Series*, **49** (3): 583–599, doi:10.2307/1969046, JSTOR 1969046.

3.2 Multigraph

This article is about the mathematical concept. For other uses, see Multigraph (disambiguation).
"Pseudograph" redirects here. It is not to be confused with Pseudepigraph.

In mathematics, and more specifically in graph theory, a **multigraph** is a graph which is permitted to have multiple edges (also called *parallel edges*[1]), that is, edges that have the same end nodes. Thus two vertices may be connected by more than one edge.



A multigraph with multiple edges (red) and several loops (blue). Not all authors allow multigraphs to have loops.

There are two distinct notions of multiple edges:

- *Edges without own identity*: The identity of an edge is defined solely by the two nodes it connects. In this case, the term "multiple edges" means that the same edge can occur several times between these two nodes.
- *Edges with own identity*: Edges are primitive entities just like nodes. When multiple edges connect two nodes, these are different edges.

A multigraph is different from a hypergraph, which is a graph in which an edge can connect any number of nodes, not just two.

For some authors, the terms *pseudograph* and *multigraph* are synonymous. For others, a **pseudograph** is a multigraph with loops.

3.2.1 Undirected multigraph (edges without own identity)

A multigraph G is an ordered pair $G:=(V, E)$ with

- V a set of *vertices* or *nodes*,
- E a multiset of unordered pairs of vertices, called *edges* or *lines*.

3.2.2 Undirected multigraph (edges with own identity)

A multigraph G is an ordered triple $G:=(V, E, r)$ with

- V a set of *vertices* or *nodes*,
- E a set of *edges* or *lines*,
- $r : E \to \{\{x,y\} : x, y \in V\}$, assigning to each edge an unordered pair of endpoint nodes.

Some authors allow multigraphs to have loops, that is, an edge that connects a vertex to itself,[2] while others call these **pseudographs**, reserving the term multigraph for the case with no loops.[3]

3.2.3 Directed multigraph (edges without own identity)

A **multidigraph** is a directed graph which is permitted to have *multiple arcs,* i.e., arcs with the same source and target nodes. A multidigraph G is an ordered pair $G:=(V,A)$ with

- V a set of *vertices* or *nodes*,
- A a multiset of ordered pairs of vertices called *directed edges*, *arcs* or *arrows*.

A **mixed multigraph** $G:=(V,E, A)$ may be defined in the same way as a mixed graph.

3.2.4 Directed multigraph (edges with own identity)

A multidigraph or quiver G is an ordered 4-tuple $G:=(V, A, s, t)$ with

- V a set of *vertices* or *nodes*,
- A a set of *edges* or *lines*,
- $s : A \to V$, assigning to each edge its source node,
- $t : A \to V$, assigning to each edge its target node.

This notion might be used to model the possible flight connections offered by an airline. In this case the multigraph would be a directed graph with pairs of directed parallel edges connecting cities to show that it is possible to fly both *to* and *from* these locations.

In category theory a small category can be defined as a multidigraph (with edges having their own identity) equipped with an associative composition law and a distinguished self-loop at each vertex serving as the left and right identity for composition. For this reason, in category theory the term *graph* is standardly taken to mean "multidigraph", and the underlying multidigraph of a category is called its **underlying digraph**.

3.2.5 Labeling

Multigraphs and multidigraphs also support the notion of graph labeling, in a similar way. However there is no unity in terminology in this case.

The definitions of **labeled multigraphs** and **labeled multidigraphs** are similar, and we define only the latter ones here.

Definition 1: A labeled multidigraph is a labeled graph with *labeled* arcs.

Formally: A labeled multidigraph G is a multigraph with *labeled* vertices and arcs. Formally it is an 8-tuple $G = (\Sigma_V, \Sigma_A, V, A, s, t, \ell_V, \ell_A)$ where

- V is a set of vertices and A is a set of arcs.
- Σ_V and Σ_A are finite alphabets of the available vertex and arc labels,
- $s\colon A \to V$ and $t\colon A \to V$ are two maps indicating the *source* and *target* vertex of an arc,
- $\ell_V\colon V \to \Sigma_V$ and $\ell_A\colon A \to \Sigma_A$ are two maps describing the labeling of the vertices and arcs.

Definition 2: A labeled multidigraph is a labeled graph with multiple *labeled* arcs, i.e. arcs with the same end vertices and the same arc label (note that this notion of a labeled graph is different from the notion given by the article graph labeling).

3.2.6 See also

- Multidimensional network
- Glossary of graph theory
- Graph theory

3.2.7 Notes

[1] For example, see Balakrishnan 1997, p. 1 or Chartrand and Zhang 2012, p. 26.

[2] For example, see Bollobás 2002, p. 7 or Diestel 2010, p. 28.

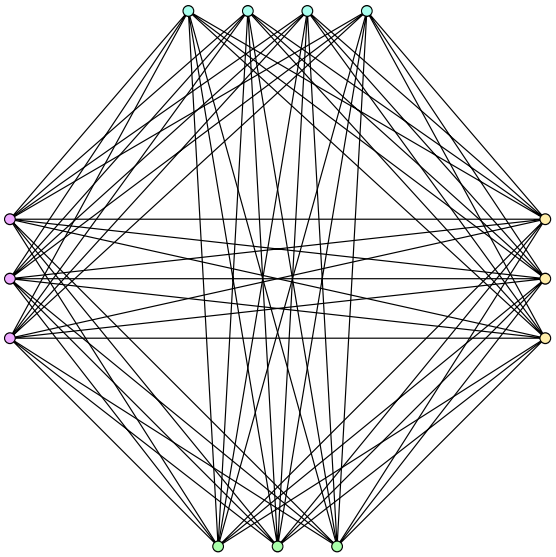
[3] For example, see Wilson 2002, p. 6 or Chartrand and Zhang 2012, pp. 26-27.

3.2.8 References

- Balakrishnan, V. K. (1997). *Graph Theory*. McGraw-Hill. ISBN 0-07-005489-4.
- Bollobás, Béla (2002). *Modern Graph Theory*. Graduate Texts in Mathematics. **184**. Springer. ISBN 0-387-98488-7.
- Chartrand, Gary; Zhang, Ping (2012). *A First Course in Graph Theory*. Dover. ISBN 978-0-486-48368-9.
- Diestel, Reinhard (2010). *Graph Theory*. Graduate Texts in Mathematics. **173** (4th ed.). Springer. ISBN 978-3-642-14278-9.
- Gross, Jonathan L.; Yellen, Jay (1998). *Graph Theory and Its Applications*. CRC Press. ISBN 0-8493-3982-0.
- Gross, Jonathan L.; Yellen, Jay, eds. (2003). *Handbook of Graph Theory*. CRC. ISBN 1-58488-090-2.
- Harary, Frank (1995). *Graph Theory*. Addison Wesley. ISBN 0-201-41033-8.
- Janson, Svante; Knuth, Donald E.; Luczak, Tomasz; Pittel, Boris (1993). "The birth of the giant component". *Random Structures and Algorithms*. **4** (3): 231–358. doi:10.1002/rsa.3240040303. ISSN 1042-9832. MR 1220220.
- Wilson, Robert A. (2002). *Graphs, Colourings and the Four-Colour Theorem*. Oxford Science Publ. ISBN 0-19-851062-4.
- Zwillinger, Daniel (2002). *CRC Standard Mathematical Tables and Formulae* (31st ed.). Chapman & Hall/CRC. ISBN 1-58488-291-3.

3.2.9 External links

- This article incorporates public domain material from the NIST document: Black, Paul E. "Multigraph". *Dictionary of Algorithms and Data Structures*.



The Turán graph T(n,r) is an example of an extremal graph. It has the maximum possible number of edges for a graph on n vertices without (r+1)-cliques. This is T(13,4).

3.3 Extremal graph theory

Extremal graph theory is a branch of the mathematical field of graph theory. Extremal graph theory studies extremal (maximal or minimal) graphs which satisfy a certain property. Extremality can be taken with respect to different graph invariants, such as order, size or girth. More abstractly, it studies how global properties of a graph influence local substructures of the graph.[1]

3.3.1 Examples

For example, a simple extremal graph theory question is "which acyclic graphs on n vertices have the maximum number of edges?" The extremal graphs for this question are trees on n vertices, which have $n - 1$ edges.[2] More generally, a typical question is the following: given a graph property P, an invariant u, and a set of graphs H, we wish to find the minimum value of m such that every graph in H which has u larger than m possess property P. In the example above, H was the set of n-vertex graphs, P was the property of being cyclic, and u was the number of edges in the graph. Thus every graph on n vertices with more than $n - 1$ edges must contain a cycle.

Several foundational results in extremal graph theory are questions of the above-mentioned form. For instance, the question of how many edges an n-vertex graph can have before it must contain as subgraph a clique of size k is answered by Turán's theorem. Instead of cliques, if the same question is asked for complete multi-partite graphs, the answer is given by the Erdős–Stone theorem.

3.3.2 History

Extremal graph theory, in its strictest sense, is a branch of graph theory developed and loved by Hungarians.

Bollobás (2004)

Extremal graph theory started in 1941 when Turán proved his theorem determining those graphs of order *n*, not containing the complete graph *Kk* of order k, and extremal with respect to size (that is, with as many edges as possible).[3] Another crucial year for the subject was 1975 when Szemerédi proved his result a vital tool in attacking extremal problems.[3]

3.3.3 Density results

A typical result in extremal graph theory is Turán's theorem. It answers the following question. What is the maximum possible number of edges in an undirected graph *G* with *n* vertices which does not contain *K3* (three vertices *A*, *B*, *C* with edges *AB*, *AC*, *BC*; i.e. a triangle) as a subgraph? The complete bipartite graph where the partite sets differ in their size by at most 1, is the only extremal graph with this property. It contains

$$\left\lfloor \frac{n^2}{4} \right\rfloor$$

edges. Similar questions have been studied with various other subgraphs *H* instead of *K3*; for instance, the Zarankiewicz problem concerns the largest graph that does not contain a fixed complete bipartite graph as a subgraph, and the even circuit theorem concerns the largest graph without a fixed-length even cycle. Turán also found the (unique) largest graph not containing *Kk* which is named after him, namely the Turán graph. This graph is the complete join of "k-1" independent sets (as equisized as possible) and has at most

$$\left\lfloor \frac{(k-2)n^2}{2(k-1)} \right\rfloor = \left\lfloor \left(1 - \frac{1}{k-1}\right) \frac{n^2}{2} \right\rfloor$$

edges. For *C4*, the largest graph on *n* vertices not containing *C4* has

$$\left(\frac{1}{2} + o(1)\right) n^{3/2}$$

edges.

3.3.4 Minimum degree conditions

The preceding theorems give conditions for a small object to appear within a (perhaps) very large graph. At the opposite extreme, one might search for conditions which force the existence of a structure which covers every vertex. But it is possible for a graph with

$$\binom{n-1}{2}$$

edges to have an isolated vertex - even though almost every possible edge is present in the graph - which means that even a graph with very high density may have no interesting structure covering every vertex. Simple edge counting conditions, which give no indication as to how the edges in the graph are distributed, thus often tend to give uninteresting results for very large structures. Instead, we introduce the concept of minimum degree. The minimum degree of a graph *G* is defined to be

$$\delta(G) = \min_{v \in G} d(v).$$

Specifying a large minimum degree removes the objection that there may be a few 'pathological' vertices; if the minimum degree of a graph *G* is 1, for example, then there can be no isolated vertices (even though *G* may have very few edges).

A classic result is Dirac's theorem, which states that every graph *G* with *n* vertices and minimum degree at least *n/2* contains a Hamilton cycle.

3.3.5 See also

- Ramsey theory

3.3.6 Notes

[1] Diestel 2010

[2] Bollobás 2004, p. 9

[3] Bollobás 1998, p. 104

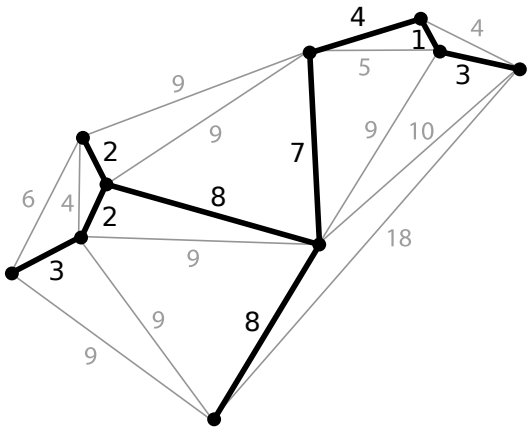
3.3.7 References

- Bollobás, Béla (2004), *Extremal Graph Theory*, New York: Dover Publications, ISBN 978-0-486-43596-1.
- Bollobás, Béla (1998), *Modern Graph Theory*, Berlin, New York: Springer-Verlag, pp. 103–144, ISBN 978-0-387-98491-9.
- Diestel, Reinhard (2010), *Graph Theory* (4th ed.), Berlin, New York: Springer-Verlag, pp. 169–198, ISBN 978-3-642-14278-9.
- M. Simonovits, Slides from the Chorin summer school lectures, 2006.

Chapter 4

Graph Traversal

4.1 Minimum spanning tree



A planar graph and its minimum spanning tree. Each edge is labeled with its weight, which here is roughly proportional to its length.

A **minimum spanning tree** (**MST**) or **minimum weight spanning tree** is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible. More generally, any undirected graph (not necessarily connected) has a **minimum spanning forest**, which is a union of the minimum spanning trees for its connected components.

There are quite a few use cases for minimum spanning trees. One example would be a telecommunications company which is trying to lay out cables in new neighborhood. If it is constrained to bury the cable only along certain paths (e.g. along roads), then there would be a graph representing which points are connected by those paths. Some of those paths might be more expensive, because they are longer, or require the cable to be buried deeper; these paths would be represented by edges with larger weights. Currency is an acceptable unit for edge weight – there is no requirement for edge lengths to obey normal rules of geometry such as the triangle inequality. A *spanning tree* for that graph would be a subset of those paths that has no cycles but still connects to every house; there might be several spanning trees possible. A *minimum spanning tree* would be one with the lowest total cost, thus would represent the least expensive path for laying the cable.

4.1.1 Properties

Possible multiplicity

If there are n vertices in the graph, then each spanning tree has $n - 1$ edges.

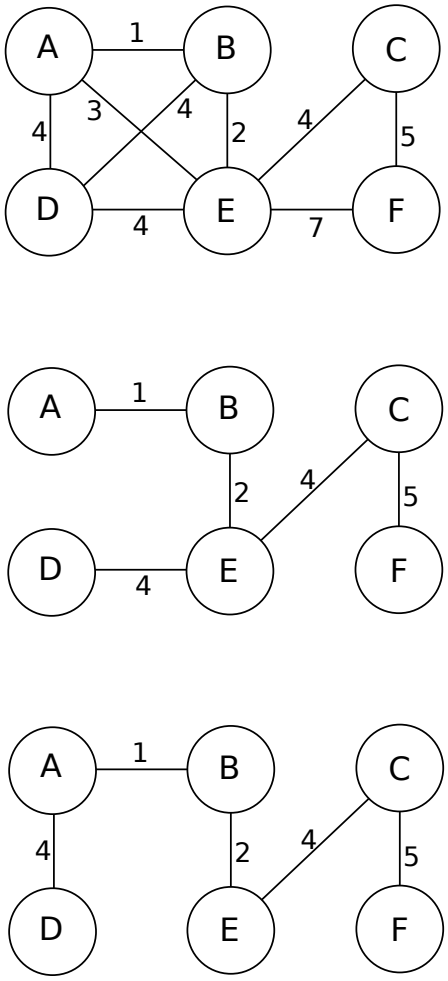
There may be several minimum spanning trees of the same weight; in particular, if all the edge weights of a given graph are the same, then every spanning tree of that graph is minimum.

Uniqueness

If each edge has a distinct weight then there will be only one, unique minimum spanning tree. This is true in many realistic situations, such as the telecommunications company example above, where it's unlikely any two paths have *exactly* the same cost. This generalizes to spanning forests as well.

Proof:

1. Assume the contrary, that there are two different MSTs A and B.
2. Since A and B differ despite containing the same nodes, there is at least one edge that belongs to one but not the other. Among such edges, let e_1 be the one with least weight; this choice is unique because the edge weights are all distinct. Without loss of generality, assume e_1 is in A.
3. As B is a MST, $\{e_1\} \cup B$ must contain a cycle C.
4. As a tree, A contains no cycles, therefore C must have an edge e_2 that is not in A.
5. Since e_1 was chosen as the unique lowest-weight edge among those belonging to exactly one of A and



This figure shows there may be more than one minimum spanning tree in a graph. In the figure, the two trees below the graph are two possibilities of minimum spanning tree of the given graph.

B, the weight of e_2 must be greater than the weight of e_1.

6. Replacing e_2 with e_1 in *B* therefore yields a spanning tree with a smaller weight.

7. This contradicts the assumption that *B* is a MST.

More generally, if the edge weights are not all distinct then only the (multi-)set of weights in minimum spanning trees is certain to be unique; it is the same for all minimum spanning trees.[1]

Minimum-cost subgraph

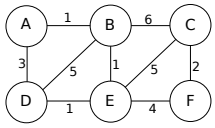
If the weights are *positive*, then a minimum spanning tree is in fact a minimum-cost subgraph connecting all vertices, since subgraphs containing cycles necessarily have more total weight.

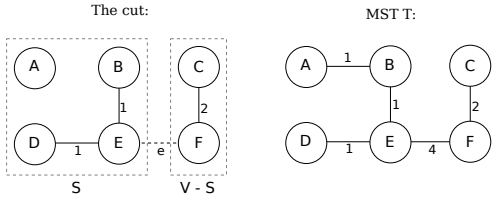
Cycle property

For any cycle C *in the graph, if the weight of an edge* e *of* C *is larger than the individual weights of all other edges of* C*, then this edge cannot belong to a MST.*

Proof: Assume the contrary, i.e. that *e* belongs to an MST T1. Then deleting *e* will break T1 into two subtrees with the two ends of *e* in different subtrees. The remainder of *C* reconnects the subtrees, hence there is an edge *f* of *C* with ends in different subtrees, i.e., it reconnects the subtrees into a tree T2 with weight less than that of T1, because the weight of *f* is less than the weight of *e*.

Cut property





This figure shows the cut property of MSTs. T is the only MST of the given graph. If S = {A,B,D,E}, thus V-S = {C,F}, then there are 3 possibilities of the edge across the cut(S,V-S), they are edges BC, EC, EF of the original graph. Then, e is one of the minimum-weight-edge for the cut, therefore S ∪ {e} is part of the MST T.

For any cut C *of the graph, if the weight of an edge* e *in the cut-set of* C *is strictly smaller than the weights of all other edges of the cut-set of* C*, then this edge belongs to all MSTs of the graph.*

Proof: assume the contrary, i.e., in the figure at right, make edge BC (weight 6) part of the MST T instead of edge e (weight 4). Adding e to T will produce a cycle, while replacing BC with e would produce MST of smaller weight. Thus, a tree containing BC is not a MST, a contradiction that violates our assumption. By a similar argument, if more than one edge is of minimum weight across a cut, then each such edge is contained in some minimum spanning tree.

Minimum-cost edge

If the minimum cost edge e *of a graph is unique, then this edge is included in any MST.*

Proof: if *e* was not included in the MST, removing any of the (larger cost) edges in the cycle formed after adding *e*

to the MST, would yield a spanning tree of smaller weight.

Contraction

If T is a tree of MST edges, then we can contract *T into a single vertex while maintaining the invariant that the MST of the contracted graph plus T gives the MST for the graph before contraction.*[2]

4.1.2 Algorithms

In all of the algorithms below, "m" is the number of edges in the graph and "n" is the number of vertices.

Classic algorithms

The first algorithm for finding a minimum spanning tree was developed by Czech scientist Otakar Borůvka in 1926 (see Borůvka's algorithm). Its purpose was an efficient electrical coverage of Moravia. The algorithm proceeds in a sequence of stages. In each stage, called *Boruvka step*, it identifies a forest F consisting of the minimum-weight edge incident to each vertex in the graph G, then forms the graph as the input to the next step. Here denotes the graph derived from G by contracting edges in F (by the Cut property, these edges belong to the MST). Each Boruvka step takes linear time. Since the number of vertices is reduced by at least half in each step, Boruvka's algorithm takes O(m log n) time.[2]

A second algorithm is Prim's algorithm, which was invented by Jarnik in 1930 and rediscovered by Prim in 1957 and Dijkstra in 1959. Basically, it grows the MST (T) one edge at a time. Initially, T contains an arbitrary vertex. In each step, T is augmented with a least-weight edge (x,y) such that x is in T and y is not yet in T. By the Cut property, all edges added to T are in the MST. Its run-time is either O(m log n) or O(m + n log n), depending on the data-structures used.

A third algorithm commonly in use is Kruskal's algorithm, which also takes O(m log n) time.

A fourth algorithm, not as commonly used, is the reverse-delete algorithm, which is the reverse of Kruskal's algorithm. Its runtime is O(m log n (log log n)^3).

All these four are greedy algorithms. Since they run in polynomial time, the problem of finding such trees is in **FP**, and related decision problems such as determining whether a particular edge is in the MST or determining if the minimum total weight exceeds a certain value are in **P**.

Faster algorithms

Several researchers have tried to find more computationally-efficient algorithms.

In a comparison model, in which the only allowed operations on edge weights are pairwise comparisons, Karger, Klein & Tarjan (1995) found a linear time randomized algorithm based on a combination of Borůvka's algorithm and the reverse-delete algorithm.[3][4]

The fastest non-randomized comparison-based algorithm with known complexity, by Bernard Chazelle, is based on the soft heap, an approximate priority queue.[5][6] Its running time is $O(m\ \alpha(m,n))$, where α is the classical functional inverse of the Ackermann function. The function α grows extremely slowly, so that for all practical purposes it may be considered a constant no greater than 4; thus Chazelle's algorithm takes very close to linear time.

Linear-time algorithms in special cases

Dense graphs If the graph is dense (i.e. $m/n \geq \log\log n$), then a deterministic algorithm by Fredman and Tarjan finds the MST in time O(m).[7] The algorithm executes a number of phases. Each phase executes Prim's algorithm many times, each for a limited number of steps. The run-time of each phase is O(m+n). If the number of vertices before a phase is n' , the number of vertices remaining after a phase is at most $n'/2^{m/n'}$. Hence, at most $\log * n$ phases are needed, which gives a linear run-time for dense graphs.[2]

There are other algorithms that work in linear time on dense graphs.[5][8]

Integer weights If the edge weights are integers represented in binary, then deterministic algorithms are known that solve the problem in $O(m + n)$ integer operations.[9] Whether the problem can be solved *deterministically* for a *general graph* in *linear time* by a comparison-based algorithm remains an open question.

Decision trees

Given graph G where the nodes and edges are fixed but the weights are unknown, it is possible to construct a binary decision tree (DT) for calculating the MST for any permutation of weights. Each internal node of the DT contains a comparison between two edges, e.g. "Is the weight of the edge between x and y larger than the weight of the edge between w and z?". The two children of the node correspond to the two possible answers "yes" or "no". In each leaf of the DT, there is a list of edges from G that correspond to an MST. The runtime complexity of a DT is the largest number of queries required to find the MST, which is just the depth of the DT. A DT for a graph G is called *optimal* if it has the smallest depth of all correct DTs for G.

For every integer r, it is possible to find optimal decision trees for all graphs on r vertices by brute-force search. This search proceeds in two steps.

A. Generating all potential DTs

- There are $2^{\binom{r}{2}}$ different graphs on *r* vertices.
- For each graph, an MST can always be found using $r(r-1)$ comparisons, e.g. by Prim's algorithm.
- Hence, the depth of an optimal DT is less than r^2 .
- Hence, the number of internal nodes in an optimal DT is less than 2^{r^2} .
- Every internal node compares two edges. The number of edges is at most r^2 so the different number of comparisons is at most r^4 .
- Hence, the number of potential DTs is less than: $\left(r^4\right)^{\left(2^{r^2}\right)} = r^{2^{(r^2+2)}}$.

B. Identifying the correct DTs To check if a DT is correct, it should be checked on all possible permutations of the edge weights.

- The number of such permutations is at most $(r^2)!$.
- For each permutation, solve the MST problem on the given graph using any existing algorithm, and compare the result to the answer given by the DT.
- The running time of any MST algorithm is at most (r^2) , so the total time required to check all permutations is at most $(r^2+1)!$.

Hence, the total time required for finding an optimal DT for *all* graphs with *r* vertices is: $2^{\binom{r}{2}} \cdot r^{2^{(r^2+2)}} \cdot (r^2+1)!$, which is less than: $2^{2^{r^2+o(r)}}$.[2]

See also: Decision tree model

Optimal algorithm

Seth Pettie and Vijaya Ramachandran have found a provably optimal deterministic comparison-based minimum spanning tree algorithm.[2] The following is a simplified description of the algorithm.

1. Let $r = \log\log\log n$, where *n* is the number of vertices. Find all optimal decision trees on *r* vertices. This can be done in time O(*n*) (see Decision trees above).
2. Partition the graph to components with at most *r* vertices in each component. This partition can be done in time O(*m*).
3. Use the optimal decision trees to find an MST for each component.
4. Contract each connected component spanned by the MSTs to a single vertex.
5. It is possible to prove that the resulting graph has at most *n*/*r* vertices. Hence, the graph is dense and we can use any algorithm which works on Dense graphs in time O(*m*).

The runtime of all steps in the algorithm is O(*m*), *except for the step of using the decision trees*. We don't know the runtime of this step, but we know that it is optimal - no algorithm can do better than the optimal decision tree.

Thus, this algorithm has the peculiar property that it is *provably optimal* although its runtime complexity is *unknown*.

Parallel and distributed algorithms

Research has also considered parallel algorithms for the minimum spanning tree problem. With a linear number of processors it is possible to solve the problem in $O(\log n)$ time.[10][11] Bader & Cong (2006) demonstrate an algorithm that can compute MSTs 5 times faster on 8 processors than an optimized sequential algorithm.[12]

Other specialized algorithms have been designed for computing minimum spanning trees of a graph so large that most of it must be stored on disk at all times. These *external storage* algorithms, for example as described in "Engineering an External Memory Minimum Spanning Tree Algorithm" by Roman, Dementiev et al.,[13] can operate, by authors' claims, as little as 2 to 5 times slower than a traditional in-memory algorithm. They rely on efficient external storage sorting algorithms and on graph contraction techniques for reducing the graph's size efficiently.

The problem can also be approached in a distributed manner. If each node is considered a computer and no node knows anything except its own connected links, one can still calculate the distributed minimum spanning tree.

4.1.3 MST on complete graphs

Alan M. Frieze showed that given a complete graph on *n* vertices, with edge weights that are independent identically distributed random variables with distribution function F satisfying $F'(0) > 0$, then as *n* approaches $+\infty$ the expected weight of the MST approaches $\zeta(3)/F'(0)$, where ζ is the Riemann zeta function. Frieze and Steele also proved convergence in probability. Svante Janson proved a central limit theorem for weight of the MST.

For uniform random weights in $[0,1]$, the exact expected size of the minimum spanning tree has been computed for small complete graphs.[14]

4.1.4 Applications

Minimum spanning trees have direct applications in the design of networks, including computer networks, telecommunications networks, transportation networks, water supply networks, and electrical grids (which they were first invented for, as mentioned above).[15] They are invoked as subroutines in algorithms for other problems, including the Christofides algorithm for approximating the traveling salesman problem,[16] approximating the multi-terminal minimum cut problem (which is equivalent in the single-terminal case to the maximum flow problem),[17] and approximating the minimum-cost weighted perfect matching.[18]

Other practical applications based on minimal spanning trees include:

- Taxonomy.[19]
- Cluster analysis: clustering points in the plane,[20] single-linkage clustering (a method of hierarchical clustering),[21] graph-theoretic clustering,[22] and clustering gene expression data.[23]
- Constructing trees for broadcasting in computer networks.[24] On Ethernet networks this is accomplished by means of the Spanning tree protocol.
- Image registration[25] and segmentation[26] – see minimum spanning tree-based segmentation.
- Curvilinear feature extraction in computer vision.[27]
- Handwriting recognition of mathematical expressions.[28]
- Circuit design: implementing efficient multiple constant multiplications, as used in finite impulse response filters.[29]
- Regionalisation of socio-geographic areas, the grouping of areas into homogeneous, contiguous regions.[30]
- Comparing ecotoxicology data.[31]
- Topological observability in power systems.[32]
- Measuring homogeneity of two-dimensional materials.[33]
- Minimax process control.[34]
- Minimum spanning trees can also be used to describe financial markets.[35] A correlation matrix can be created by calculating a coefficient of correlation between any two stocks.This matrix can be represented topologically as a complex network and a minimum spanning tree can be constructed to visualize relationships.

4.1.5 Related problems

The problem of finding the Steiner tree of a subset of the vertices, that is, minimum tree that spans the given subset, is known to be NP-Complete.[36]

A related problem is the k-minimum spanning tree (k-MST), which is the tree that spans some subset of k vertices in the graph with minimum weight.

A set of *k-smallest spanning trees* is a subset of k spanning trees (out of all possible spanning trees) such that no spanning tree outside the subset has smaller weight.[37][38][39] (Note that this problem is unrelated to the k-minimum spanning tree.)

The Euclidean minimum spanning tree is a spanning tree of a graph with edge weights corresponding to the Euclidean distance between vertices which are points in the plane (or space).

The rectilinear minimum spanning tree is a spanning tree of a graph with edge weights corresponding to the rectilinear distance between vertices which are points in the plane (or space).

In the distributed model, where each node is considered a computer and no node knows anything except its own connected links, one can consider distributed minimum spanning tree. The mathematical definition of the problem is the same but there are different approaches for a solution.

The capacitated minimum spanning tree is a tree that has a marked node (origin, or root) and each of the subtrees attached to the node contains no more than a c nodes. c is called a tree capacity. Solving CMST optimally is NP-hard,[40] but good heuristics such as Esau-Williams and Sharma produce solutions close to optimal in polynomial time.

The degree constrained minimum spanning tree is a minimum spanning tree in which each vertex is connected to no more than d other vertices, for some given number d. The case $d = 2$ is a special case of the traveling salesman problem, so the degree constrained minimum spanning tree is NP-hard in general.

For directed graphs, the minimum spanning tree problem is called the Arborescence problem and can be solved in quadratic time using the Chu–Liu/Edmonds algorithm.

A **maximum spanning tree** is a spanning tree with weight greater than or equal to the weight of every other spanning tree. Such a tree can be found with algorithms such as Prim's or Kruskal's after multiplying the edge weights by -1 and solving the MST problem on the new graph. A path in the maximum spanning tree is the widest path in the graph between its two endpoints: among all possible paths, it maximizes the weight of the minimum-weight edge.[41] Maximum spanning trees find applications in parsing algorithms for natural languages[42] and in training algorithms for conditional random fields.

The **dynamic MST** problem concerns the update of a previously computed MST after an edge weight change in the original graph or the insertion/deletion of a vertex.[43][44][45]

The minimum labeling spanning tree problem is to find a spanning tree with least types of labels if each edge in a graph is associated with a label from a finite label set instead of a weight.[46]

A bottleneck edge is the highest weighted edge in a spanning tree. A spanning tree is a **minimum bottleneck spanning tree** (or **MBST**) if the graph does not contain a spanning tree with a smaller bottleneck edge weight. A MST is necessarily a MBST (provable by the cut property), but a MBST is not necessarily a MST.[47][48]

4.1.6 References

[1] Do the minimum spanning trees of a weighted graph have the same number of edges with a given weight?

[2] Pettie, Seth; Ramachandran, Vijaya (2002), "An optimal minimum spanning tree algorithm", *Journal of the Association for Computing Machinery*, **49** (1): 16–34, doi:10.1145/505241.505243, MR 2148431.

[3] Karger, David R.; Klein, Philip N.; Tarjan, Robert E. (1995), "A randomized linear-time algorithm to find minimum spanning trees", *Journal of the Association for Computing Machinery*, **42** (2): 321–328, doi:10.1145/201019.201022, MR 1409738

[4] Pettie, Seth; Ramachandran, Vijaya (2002), "Minimizing randomness in minimum spanning tree, parallel connectivity, and set maxima algorithms", *Proc. 13th ACM-SIAM Symposium on Discrete Algorithms (SODA '02)*, San Francisco, California, pp. 713–722.

[5] Chazelle, Bernard (2000), "A minimum spanning tree algorithm with inverse-Ackermann type complexity", *Journal of the Association for Computing Machinery*, **47** (6): 1028–1047, doi:10.1145/355541.355562, MR 1866456.

[6] Chazelle, Bernard (2000), "The soft heap: an approximate priority queue with optimal error rate", *Journal of the Association for Computing Machinery*, **47** (6): 1012–1027, doi:10.1145/355541.355554, MR 1866455.

[7] Fredman, M. L.; Tarjan, R. E. (1987). "Fibonacci heaps and their uses in improved network optimization algorithms". *Journal of the ACM*. **34** (3): 596. doi:10.1145/28869.28874.

[8] Gabow, H. N.; Galil, Z.; Spencer, T.; Tarjan, R. E. (1986). "Efficient algorithms for finding minimum spanning trees in undirected and directed graphs". *Combinatorica*. **6** (2): 109. doi:10.1007/bf02579168.

[9] Fredman, M. L.; Willard, D. E. (1994), "Trans-dichotomous algorithms for minimum spanning trees and shortest paths", *Journal of Computer and System Sciences*, **48** (3): 533–551, doi:10.1016/S0022-0000(05)80064-9, MR 1279413.

[10] Chong, Ka Wong; Han, Yijie; Lam, Tak Wah (2001), "Concurrent threads and optimal parallel minimum spanning trees algorithm", *Journal of the Association for Computing Machinery*, **48** (2): 297–323, doi:10.1145/375827.375847, MR 1868718.

[11] Pettie, Seth; Ramachandran, Vijaya (2002), "A randomized time-work optimal parallel algorithm for finding a minimum spanning forest", *SIAM Journal on Computing*, **31** (6): 1879–1895, doi:10.1137/S0097539700371065, MR 1954882.

[12] Bader, David A.; Cong, Guojing (2006), "Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs", *Journal of Parallel and Distributed Computing*, **66** (11): 1366–1378, doi:10.1016/j.jpdc.2006.06.001.

[13] Dementiev, Roman; Sanders, Peter; Schultes, Dominik; Sibeyn, Jop F. (2004), "Engineering an external memory minimum spanning tree algorithm", *Proc. IFIP 18th World Computer Congress, TC1 3rd International Conference on Theoretical Computer Science (TCS2004)* (PDF), pp. 195–208.

[14] Steele, J. Michael (2002), "Minimal spanning trees for graphs with random edge lengths", *Mathematics and computer science, II (Versailles, 2002)*, Trends Math., Basel: Birkhäuser, pp. 223–245, MR 1940139

[15] Graham, R. L.; Hell, Pavol (1985), "On the history of the minimum spanning tree problem", *Annals of the History of Computing*, **7** (1): 43–57, doi:10.1109/MAHC.1985.10011, MR 783327

[16] Nicos Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, Report 388, Graduate School of Industrial Administration, CMU, 1976.

[17] Dahlhaus, E.; Johnson, D. S.; Papadimitriou, C. H.; Seymour, P. D.; Yannakakis, M. (August 1994). "The complexity of multiterminal cuts" (PDF). *SIAM Journal on Computing*. **23** (4): 864–894. doi:10.1137/S0097539792225297. Retrieved 17 December 2012.

[18] Supowit, Kenneth J.; Plaisted, David A.; Reingold, Edward M. (1980). *Heuristics for weighted perfect matching*. 12th Annual ACM Symposium on Theory of Computing (STOC '80). New York, NY, USA: ACM. pp. 398–419. doi:10.1145/800141.804689.

[19] Sneath, P. H. A. (1 August 1957). "The Application of Computers to Taxonomy". *Journal of General Microbiology*. **17** (1): 201–226. doi:10.1099/00221287-17-1-201. PMID 13475686.

[20] Asano, T.; Bhattacharya, B.; Keil, M.; Yao, F. (1988). *Clustering algorithms based on minimum and maximum spanning trees*. Fourth Annual Symposium on Computational Geometry (SCG '88). **1**. pp. 252–257. doi:10.1145/73393.73419.

[21] Gower, J. C.; Ross, G. J. S. (1969). "Minimum Spanning Trees and Single Linkage Cluster Analysis". *Journal of the Royal Statistical Society*. C (Applied Statistics). **18** (1): 54–64. doi:10.2307/2346439.

[22] Päivinen, Niina (1 May 2005). "Clustering with a minimum spanning tree of scale-free-like structure". *Pattern Recognition Letters*. **26** (7): 921–930. doi:10.1016/j.patrec.2004.09.039.

[23] Xu, Y.; Olman, V.; Xu, D. (1 April 2002). "Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees". *Bioinformatics*. **18** (4): 536–545. doi:10.1093/bioinformatics/18.4.536. PMID 12016051.

[24] Dalal, Yogen K.; Metcalfe, Robert M. (1 December 1978). "Reverse path forwarding of broadcast packets". *Communications of the ACM*. **21** (12): 1040–1048. doi:10.1145/359657.359665.

[25] Ma, B.; Hero, A.; Gorman, J.; Michel, O. (2000). *Image registration with minimum spanning tree algorithm* (PDF). International Conference on Image Processing. **1**. pp. 481–484. doi:10.1109/ICIP.2000.901000.

[26] P. Felzenszwalb, D. Huttenlocher: Efficient Graph-Based Image Segmentation. IJCV 59(2) (September 2004)

[27] Suk, Minsoo; Song, Ohyoung (1 June 1984). "Curvilinear feature extraction using minimum spanning trees". *Computer Vision, Graphics, and Image Processing*. **26** (3): 400–411. doi:10.1016/0734-189X(84)90221-4.

[28] Tapia, Ernesto; Rojas, Raúl (2004). "Recognition of On-line Handwritten Mathematical Expressions Using a Minimum Spanning Tree Construction and Symbol Dominance". *Graphics Recognition. Recent Advances and Perspectives* (PDF). Lecture Notes in Computer Science. **3088**. Berlin Heidelberg: Springer-Verlag. pp. 329–340. ISBN 3540224785.

[29] Ohlsson, H. (2004). *Implementation of low complexity FIR filters using a minimum spanning tree*. 12th IEEE Mediterranean Electrotechnical Conference (MELECON 2004). **1**. pp. 261–264. doi:10.1109/MELCON.2004.1346826.

[30] AssunÇão, R. M.; M. C. Neves; G. Câmara; C. Da Costa Freitas (2006). "Efficient regionalization techniques for socio-economic geographical units using minimum spanning trees". *International Journal of Geographical Information Science*. **20** (7): 797–811. doi:10.1080/13658810600665111.

[31] Devillers, J.; Dore, J.C. (1 April 1989). "Heuristic potency of the minimum spanning tree (MST) method in toxicology". *Ecotoxicology and Environmental Safety*. **17** (2): 227–235. doi:10.1016/0147-6513(89)90042-0. PMID 2737116.

[32] Mori, H.; Tsuzuki, S. (1 May 1991). "A fast method for topological observability analysis using a minimum spanning tree technique". *IEEE Transactions on Power Systems*. **6** (2): 491–500. doi:10.1109/59.76691.

[33] Filliben, James J.; Kafadar, Karen; Shier, Douglas R. (1 January 1983). "Testing for homogeneity of two-dimensional surfaces". *Mathematical Modelling*. **4** (2): 167–189. doi:10.1016/0270-0255(83)90026-X.

[34] Kalaba, Robert E. (1963), *Graph Theory and Automatic Control* (PDF)

[35] Djauhari, M., & Gan, S. (2015). Optimality problem of network topology in stocks market analysis. Physica A: Statistical Mechanics and Its Applications, 419, 108-114.

[36] Garey, Michael R.; Johnson, David S. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, ISBN 0-7167-1045-5. ND12

[37] Gabow, Harold N. (1977), "Two algorithms for generating weighted spanning trees in order", *SIAM Journal on Computing*, **6** (1): 139–150, doi:10.1137/0206011, MR 0441784.

[38] Eppstein, David (1992), "Finding the k smallest spanning trees", *BIT*, **32** (2): 237–248, doi:10.1007/BF01994879, MR 1172188.

[39] Frederickson, Greg N. (1997), "Ambivalent data structures for dynamic 2-edge-connectivity and k smallest spanning trees", *SIAM Journal on Computing*, **26** (2): 484–538, doi:10.1137/S0097539792226825, MR 1438526.

[40] Jothi, Raja; Raghavachari, Balaji (2005), "Approximation Algorithms for the Capacitated Minimum Spanning Tree Problem and Its Variants in Network Design", *ACM Trans. Algorithms*, **1** (2): 265–282, doi:10.1145/1103963.1103967

[41] Hu, T. C. (1961), "The maximum capacity route problem", *Operations Research*, **9** (6): 898–900, doi:10.1287/opre.9.6.898, JSTOR 167055.

[42] McDonald, Ryan; Pereira, Fernando; Ribarov, Kiril; Hajič, Jan (2005). "Non-projective dependency parsing using spanning tree algorithms" (PDF). *Proc. HLT/EMNLP*.

[43] Spira, P. M.; Pan, A. (1975), "On finding and updating spanning trees and shortest paths", *SIAM Journal on Computing*, **4** (3): 375–380, doi:10.1137/0204032, MR 0378466.

[44] Holm, Jacob; de Lichtenberg, Kristian; Thorup, Mikkel (2001), "Poly-logarithmic deterministic fully dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity", *Journal of the Association for Computing Machinery*, **48** (4): 723–760, doi:10.1145/502090.502095, MR 2144928.

[45] Chin, F.; Houck, D. (1978), "Algorithms for updating minimal spanning trees", *Journal of Computer and System Sciences*, **16** (3): 333–344, doi:10.1016/0022-0000(78)90022-3.

[46] Chang, R.S.; Leu, S.J. (1997), "The minimum labeling spanning trees", *Information Processing Letters*, **63** (5): 277–282, doi:10.1016/s0020-0190(97)00127-0.

[47] http://flashing-thoughts.blogspot.ru/2010/06/everything-about-bottleneck-spanning.html

[48] http://pages.cpsc.ucalgary.ca/~{}dcatalin/413/t4.pdf

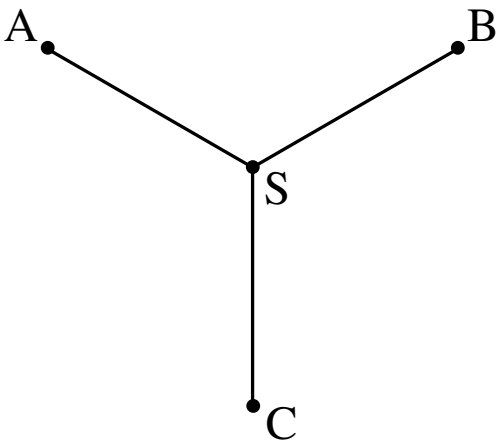
4.1.7 Additional reading

- Otakar Boruvka on Minimum Spanning Tree Problem (translation of the both 1926 papers, comments, history) (2000) Jaroslav Nešetřil, Eva Milková, Helena Nesetrilová. (Section 7 gives his algorithm, which looks like a cross between Prim's and Kruskal's.)
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Chapter 23: Minimum Spanning Trees, pp. 561–579.
- Eisner, Jason (1997). State-of-the-art algorithms for minimum spanning trees: A tutorial discussion. Manuscript, University of Pennsylvania, April. 78 pp.
- Kromkowski, John David. "Still Unmelted after All These Years", in Annual Editions, Race and Ethnic Relations, 17/e (2009 McGraw Hill) (Using minimum spanning tree as method of demographic analysis of ethnic diversity across the United States).

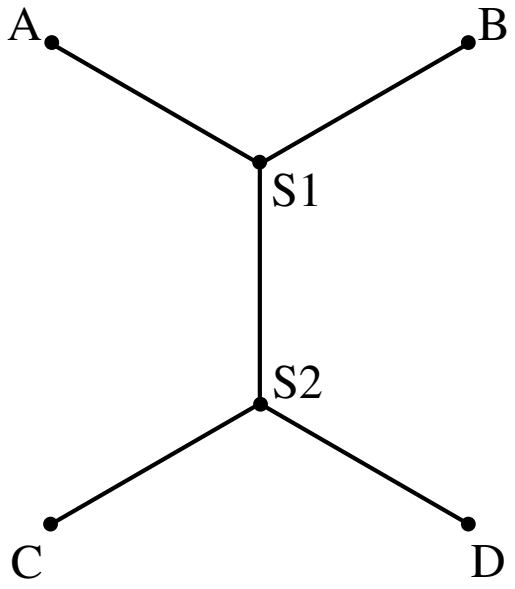
4.1.8 External links

- Implemented in BGL, the Boost Graph Library
- The Stony Brook Algorithm Repository - Minimum Spanning Tree codes
- Implemented in QuickGraph for .Net

4.2 Steiner tree problem



Steiner tree for three points A, B, *and* C *(note there are no direct connections between* A, B, C*). The Steiner point* S *is located at the Fermat point of the triangle* ABC.



Solution for four points—there are two Steiner points, S_1 *and* S_2

Steiner tree problem, or **minimum Steiner tree problem**, named after Jakob Steiner, is an umbrella term for a class of problems in combinatorial optimization. While Steiner tree problems may be formulated in a number of settings, they all require an optimal interconnect for a given set of objects and a predefined objective function. One well-known variant, which is often used synonymously with the term Steiner tree problem, is the **Steiner tree problem in graphs**. Given an undirected graph with non-negative edge weights and a subset of vertices, usually referred to as **terminals,** the Steiner tree problem in graphs requires a tree of minimum weight that contains all terminals (but may include additional vertices). Further well-known variants are the **Euclidean Steiner tree problem** and the **rectilinear minimum Steiner tree problem**.

The Steiner tree problem in graphs can be seen as a generalization of two other famous combinatorial optimization problems: the (non-negative) shortest path problem and the minimum spanning tree problem. If a Steiner tree problem in graphs contains exactly two terminals, it reduces to finding a shortest path. If, on the other hand, all vertices are terminals, the Steiner tree problem in graphs is equivalent to the minimum spanning tree. However, while both the non-negative shortest path and the minimum spanning tree problem are solvable in polynomial time, the decision variant of the Steiner tree problem in graphs is NP-complete (which implies that the optimization variant is NP-hard); in fact, the decision variant was among Karp's original 21 NP-complete problems. The Steiner tree problem in graphs has applications in circuit layout or network design. However, practical applications usually require variations, giving rise to a multitude of Steiner tree problem variants.

Most versions of the Steiner tree problem are NP-hard, but some restricted cases can be solved in polynomial time. Despite the pessimistic worst-case complexity, several Steiner tree problem variants, including the Steiner tree problem in graphs and the recitilinear Steiner tree problem, can be solved efficiently in practice, even for large-scale real-world problems. [1][2]

4.2.1 Euclidean Steiner tree

The original problem was stated in the form that has become known as the **Euclidean Steiner tree problem** or **geometric Steiner tree problem**: Given N points in the plane, the goal is to connect them by lines of minimum total length in such a way that any two points may be interconnected by line segments either directly or via other points and line segments. It may be shown that the connecting line segments do not intersect each other except at the endpoints and form a tree, hence the name of the problem.

For the Euclidean Steiner problem, points added to the graph (Steiner points) must have a degree of three, and the three edges incident to such a point must form three 120 degree angles (see Fermat point). It follows that the maximum number of Steiner points that a Steiner tree can have is $N - 2$, where N is the initial number of given points.

For $N = 3$ there are two possible cases: if the triangle formed by the given points has all angles which are less than 120 degrees, the solution is given by a Steiner point located at the Fermat point; otherwise the solution is given by the two sides of the triangle which meet on the angle with 120 or more degrees.

For general N, the Euclidean Steiner tree problem is NP-hard, and hence it is not known whether an optimal solution can be found by using a polynomial-time algorithm. However, there is a polynomial-time approximation scheme (PTAS) for Euclidean Steiner trees, i.e., a *near-optimal* solution can be found in polynomial time.[3] It is not known whether the Euclidean Steiner tree problem is NP-complete, since membership to the complexity class NP is not known.

4.2.2 Rectilinear Steiner tree

Main article: Rectilinear Steiner tree

The rectilinear Steiner tree problem is a variant of the geometric Steiner tree problem in the plane, in which the Euclidean distance is replaced with the rectilinear distance. The problem arises in the physical design of electronic design automation. In VLSI circuits, wire routing is carried out by wires that are often constrained by design rules to run only in vertical and horizontal directions, so the rectilinear Steiner tree problem can be used to model the routing of nets with more than two terminals.[4]

4.2.3 Steiner tree in graphs and variants

Steiner trees have been extensively studied in the context of weighted graphs. The prototype is, arguably, the **Steiner tree problem in graphs**. Let $G = (V, E)$ be an undirected graph with non-negative edge weights c and let $S \subseteq V$ be a subset of vertices, called **terminals**. A **Steiner tree** is a tree in G that spans S. There are two versions of the problem: in the optimization problem associated with Steiner trees, the task is to find a minimum-weight Steiner tree; in the decision problem the edge weights are integer and the task is to determine whether a Steiner tree exists whose total weight does not exceed a predefined natural number k. The decision problem is one of Karp's 21 NP-complete problems; hence the optimization problem is NP-hard.

A special case of this problem is when G is a complete graph, each vertex $v \in V$ corresponds to a point in a metric space, and the edge weights $w(e)$ for each $e \in E$ correspond to distances in the space. Put otherwise, the edge weights satisfy the triangle inequality. This variant is known as the **metric Steiner tree problem**. Given an instance of the (non-metric) Steiner tree problem, we can transform it in polynomial time into an equivalent instance of the metric Steiner tree problem; the transformation preserves the approximation factor.[5]

While the Euclidean version admits a PTAS, it is known that the metric Steiner tree problem is APX-complete, i.e., unless P = NP, it is impossible to achieve approximation ratios that are arbitrarily close to 1 in polynomial time. There is a polynomial-time algorithm that approximates the minimum Steiner tree to within a factor of $\ln(4)+\epsilon \approx 1.386$;[6] however, approximating within a factor $96/95 \approx 1.0105$ is NP-hard.[7] For the restricted case of Steiner Tree problem with distances 1 and 2, a 1.25-approximation algorithm is known.[8]

In a special case of the graph problem, the Steiner tree problem for quasi-bipartite graphs, S is required to include at least one endpoint of every edge in G.

The Steiner tree problem has also been investigated in higher dimensions and on various surfaces. Algorithms to find the Steiner minimal tree have been found on the sphere, torus, projective plane, wide and narrow cones, and others.[9]

Another generalizations of the Steiner tree problem are the **k-edge-connected Steiner network problem** and the **k-vertex-connected Steiner network problem**, where the goal is to find a k-edge-connected graph or a k-vertex-connected graph rather than any connected graph.

The Steiner problem has also been stated in the general setting of metric spaces and for possibly infinitely many points.[10]

4.2.4 Approximating the Steiner tree

The general graph Steiner tree problem can be approximated by computing the minimum spanning tree of the subgraph of the metric closure of the graph induced by the terminal vertices. The metric closure of a graph G is the complete graph in which each edge is weighted by the shortest path distance between the nodes in G. This algorithm produces a tree whose weight is within a 2 - 2 / t factor of the weight of the optimal Steiner tree; this can be proven by considering a traveling salesperson tour on the optimal Steiner tree. The approximate solution is computable in polynomial time by first solving the all-pairs shortest paths problem to compute the metric closure, then by solving the minimum spanning tree problem.

A series of papers provided approximation algorithms for the minimum Steiner tree problem with approximation ratios that improved upon the 2 - 2 / t ratio. This sequence culminated with Robins and Zelikovsky's algorithm in 2000 which improved the ratio to 1.55 by iteratively improving upon the minimum cost terminal spanning tree. More recently, however, Jaroslaw Byrka et al. proved an $\ln(4) + \epsilon \leq 1.39$ approximation using a linear programming relaxation and a technique called iterative, randomized rounding.[6]

4.2.5 Steiner ratio

The **Steiner ratio** is the supremum of the ratio of the total length of the minimum spanning tree to the minimum Steiner tree for a set of points in the Euclidean plane.[11]

In the Euclidean Steiner tree problem, the Steiner ratio is conjectured to be $\frac{2}{\sqrt{3}}$. Despite earlier claims of a proof,[12] the conjecture is still open.[13] In the Rectilinear Steiner tree problem, the Steiner ratio is $\frac{3}{2}$.[14]

4.2.6 See also

- Opaque forest problem

4.2.7 Notes

[1] "Report by Polzin and Vahdati Daneshmand" (PDF). Retrieved 11 November 2016.

[2] Juhl et al. (2014).

[3] Crescenzi et al. (2000).

[4] Sherwani (1993), p. 228.

[5] Vazirani (2003), pp. 27–28.

[6] Byrka et al. (2010).

[7] Chlebík & Chlebíková (2008).

[8] Berman, Karpinski & Zelikovsky (2009).

[9] Smith & Winter (1995), p. 361.

[10] Paolini & Stepanov (2012).

[11] Ganley (2004).

[12] The New York Times, Oct 30, 1990, reported that a proof had been found, and that Ronald Graham, who had offered $500 for a proof, was about to mail a check to the authors.

[13] Ivanov & Tuzhilin (2012).

[14] Hwang (1976).

4.2.8 References

- Berman, Piotr; Karpinski, Marek; Zelikovsky, Alexander (2009). "1.25-approximation algorithm for Steiner tree problem with distances 1 and 2". *Algorithms and Data Structures: 11th International Symposium, WADS 2009, Banff, Canada, August 21-23, 2009, Proceedings*. Lecture Notes in Computer Science. **5664**. pp. 86–97. doi:10.1007/978-3-642-03367-4_8.
- Bern, Marshall W.; Graham, Ronald L. (1989). "The shortest-network problem". *Scientific American*. **260** (1): 84–89. doi:10.1038/scientificamerican0189-84.
- Juhl, D.; Warme, D.M.; Winter, P.; Zachariasen, M. (2014). "The GeoSteiner Software Package for computing Steiner trees in the plane: an updated computational study". *Proceedings of the 11th DIMACS Implementation Challenge*.
- Byrka, J.; Grandoni, F.; Rothvoß, T.; Sanita, L. (2010). "An improved LP-based approximation for Steiner tree". *Proceedings of the 42nd ACM Symposium on Theory of Computing*. pp. 583–592. doi:10.1145/1806689.1806769.
- Chlebík, Miroslav; Chlebíková, Janka (2008). "The Steiner tree problem on graphs: Inapproximability results". *Theoretical Computer Science*. doi:10.1016/j.tcs.2008.06.046.
- Cieslik, Dietmar (1998). *Steiner Minimal Trees*. p. 319. ISBN 0-7923-4983-0.
- Crescenzi, Pierluigi; Kann, Viggo; Halldórsson, Magnús; Karpinski, Marek; Woeginger, Gerhard (2000). "Minimum geometric Steiner tree". *A Compendium of NP Optimization Problems*.
- Ganley, Joseph L. (2004). "Steiner ratio". In Black, Paul E. *Dictionary of Algorithms and Data Structures*. U.S. National Institute of Standards and Technology. Retrieved 2012-05-24.

- Garey, Michael R.; Johnson, David S. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, ISBN 0-7167-1045-5, pp. 208–209, problems ND12 and ND13.
- Hwang, F. K. (1976). "On Steiner minimal trees with rectilinear distance". *SIAM Journal on Applied Mathematics*. **30** (1): 104–114. doi:10.1137/0130013.
- Hwang, F. K.; Richards, D. S.; Winter, P. (1992). *The Steiner Tree Problem*. Annals of Discrete Mathematics. **53**. North-Holland: Elsevier. ISBN 0-444-89098-X.
- Ivanov, Alexander; Tuzhilin, Alexey (1994). *Minimal Networks: The Steiner Problem and Its Generalizations*. N.W., Boca Raton, Florida: CRC Press. ISBN 978-0-8493-8642-8.
- Ivanov, Alexander; Tuzhilin, Alexey (2000). *Branching solutions to one-dimensional variational problems*. Singapore-New Jersey-London-Hong Kong: World Scientific. ISBN 978-981-02-4060-8.
- Ivanov, Alexander; Tuzhilin, Alexey (2003). *Extreme Networks Theory* (in Russian). Moscow-Izhevsk: Institute of Computer Investigations. ISBN 5-93972-292-X.
- Ivanov, Alexander; Tuzhilin, Alexey (2012). "The Steiner ratio Gilbert–Pollak conjecture is still open: Clarification statement". *Algorithmica*. **62** (1-2): 630–632. doi:10.1007/s00453-011-9508-3.
- Ivanov, Alexander; Tuzhilin, Alexey (2015). "Branched coverings and Steiner ratio". *International Transactions in Operational Research (ITOR)*. doi:10.1111/itor.12182.
- Korte, Bernhard; Vygen, Jens (2006). "Section 20.1". *Combinatorial Optimization: Theory and Algorithms* (3rd ed.). Springer. ISBN 3-540-25684-9.
- Paolini, E.; Stepanov, E. (2012). "Existence and regularity results for the Steiner problem". *Calc. Var. Partial Diff. Equations*. doi:10.1007/s00526-012-0505-4.
- Robins, Gabriel; Zelikovsky, Alexander (2000). "Improved Steiner Tree Approximation in Graphs". *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '00)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics. pp. 770–779. ISBN 0-89871-453-2.
- Sherwani, Naveed A. (1993). *Algorithms for VLSI Physical Design Automation*. Kluwer Academic Publishers. ISBN 9781475722192.
- Smith, J. M.; Winter, P. (1995). "Computational geometry and topological network design". In Du, Ding-Zhu; Hwang, Frank. *Computing in Euclidean geometry*. Lecture Notes Series on Computing. **4** (2nd ed.). River Edge, NJ: World Scientific Publishing Co. pp. 351–451. ISBN 981-02-1876-1.
- Vazirani, Vijay V. (2003). *Approximation Algorithms*. Berlin: Springer. ISBN 3-540-65367-8.
- Wu, Bang Ye; Chao, Kun-Mao (2004). "Chapter 7". *Spanning Trees and Optimization Problems*. Chapman & Hall/CRC. ISBN 1-58488-436-3.

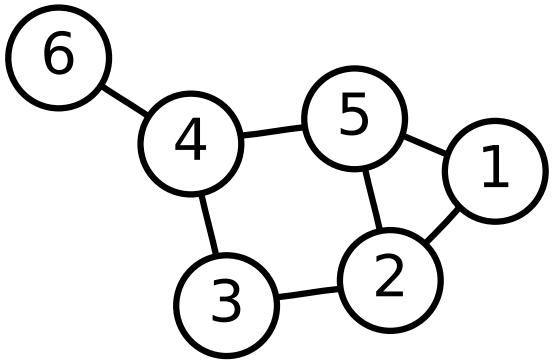
4.2.9 External links

- Hazewinkel, M. (2001), "Steiner tree problem", in Hazewinkel, Michiel, *Encyclopedia of Mathematics*, Springer, ISBN 978-1-55608-010-4
- M. Hauptmann, M. Karpinski (2013): A Compendium on Steiner Tree Problems
- GeoSteiner (Euclidean and rectilinear Steiner tree solver, Source available, for non commercial use)
- SCIP-Jack (Solver for the Steiner tree problem and ten variants)
- https://archive.org/details/RonaldLG1988 (Movie: Ronald L Graham: The Shortest Network Problem (1988)
- Fortran subroutine for finding the Steiner vertex of a triangle (i.e., Fermat point), its distances from the triangle vertices, and the relative vertex weights.
- Phylomurka (Solver for the Steiner tree problem in graphs)
- https://www.youtube.com/watch?v=PI6rAOWu-Og (Movie: solving the Steiner tree problem with water and soap)

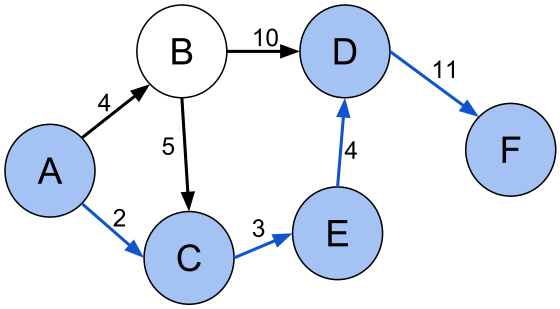
4.3 Shortest path problem

In graph theory, the **shortest path problem** is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized.

The problem of finding the shortest path between two intersections on a road map (the graph's vertices correspond to intersections and the edges correspond to road segments, each weighted by the length of its road segment) may be modeled by a special case of the shortest path problem in graphs.



(6, 4, 5, 1) and (6, 4, 3, 2, 1) are both paths between vertices 6 and 1



Shortest path (A, C, E, D, F) between vertices A and F in the weighted directed graph

4.3.1 Definition

The shortest path problem can be defined for graphs whether undirected, directed, or mixed. It is defined here for undirected graphs; for directed graphs the definition of path requires that consecutive vertices be connected by an appropriate directed edge.

Two vertices are adjacent when they are both incident to a common edge. A path in an undirected graph is a sequence of vertices $P = (v_1, v_2, \ldots, v_n) \in V \times V \times \cdots \times V$ such that v_i is adjacent to v_{i+1} for $1 \leq i < n$. Such a path P is called a path of length $n - 1$ from v_1 to v_n . (The v_i are variables; their numbering here relates to their position in the sequence and needs not to relate to any canonical labeling of the vertices.)

Let $e_{i,j}$ be the edge incident to both v_i and v_j . Given a real-valued weight function $f : E \to \mathbb{R}$, and an undirected (simple) graph G , the shortest path from v to v' is the path $P = (v_1, v_2, \ldots, v_n)$ (where $v_1 = v$ and $v_n = v'$) that over all possible n minimizes the sum $\sum_{i=1}^{n-1} f(e_{i,i+1})$. When each edge in the graph has unit weight or $f : E \to \{1\}$, this is equivalent to finding the path with fewest edges.

The problem is also sometimes called the **single-pair shortest path problem**, to distinguish it from the following variations:

- The **single-source shortest path problem**, in which we have to find shortest paths from a source vertex v to all other vertices in the graph.
- The **single-destination shortest path problem**, in which we have to find shortest paths from all vertices in the directed graph to a single destination vertex v. This can be reduced to the single-source shortest path problem by reversing the arcs in the directed graph.
- The **all-pairs shortest path problem**, in which we have to find shortest paths between every pair of vertices v, v' in the graph.

These generalizations have significantly more efficient algorithms than the simplistic approach of running a single-pair shortest path algorithm on all relevant pairs of vertices.

4.3.2 Algorithms

The most important algorithms for solving this problem are:

- Dijkstra's algorithm solves the single-source shortest path problem.
- Bellman–Ford algorithm solves the single-source problem if edge weights may be negative.
- A* search algorithm solves for single pair shortest path using heuristics to try to speed up the search.
- Floyd–Warshall algorithm solves all pairs shortest paths.
- Johnson's algorithm solves all pairs shortest paths, and may be faster than Floyd–Warshall on sparse graphs.
- Viterbi algorithm solves the shortest stochastic path problem with an additional probabilistic weight on each node.

Additional algorithms and associated evaluations may be found in Cherkassky, Goldberg & Radzik (1996).

4.3.3 Single-source shortest paths

Undirected graphs

Unweighted graphs

Directed acyclic graphs

An algorithm using topological sorting can solve the single-source shortest path problem in linear time, $\Theta(E + V)$, in weighted DAGs.

Directed graphs with nonnegative weights

The following table is taken from Schrijver (2004). A green background indicates an asymptotically best bound in the table.

This list is incomplete; you can help by expanding it.

Planar directed graphs with nonnegative weights

Directed graphs with arbitrary weights without negative cycles

This list is incomplete; you can help by expanding it.

Planar directed graphs with arbitrary weights

4.3.4 All-pairs shortest paths

The all-pairs shortest path problem finds the shortest paths between every pair of vertices v, v' in the graph. The all-pairs shortest paths problem for unweighted directed graphs was introduced by Shimbel (1953), who observed that it could be solved by a linear number of matrix multiplications that takes a total time of $O(V^4)$.

Undirected graph

Directed graph

4.3.5 Applications

Shortest path algorithms are applied to automatically find directions between physical locations, such as driving directions on web mapping websites like MapQuest or Google Maps. For this application fast specialized algorithms are available.[1]

If one represents a nondeterministic abstract machine as a graph where vertices describe states and edges describe possible transitions, shortest path algorithms can be used to find an optimal sequence of choices to reach a certain goal state, or to establish lower bounds on the time needed to reach a given state. For example, if vertices represent the states of a puzzle like a Rubik's Cube and each directed edge corresponds to a single move or turn, shortest path algorithms can be used to find a solution that uses the minimum possible number of moves.

In a networking or telecommunications mindset, this shortest path problem is sometimes called the min-delay path problem and usually tied with a widest path problem. For example, the algorithm may seek the shortest (min-delay) widest path, or widest shortest (min-delay) path.

A more lighthearted application is the games of "six degrees of separation" that try to find the shortest path in graphs like movie stars appearing in the same film.

Other applications, often studied in operations research, include plant and facility layout, robotics, transportation, and VLSI design".[2]

Road networks

A road network can be considered as a graph with positive weights. The nodes represent road junctions and each edge of the graph is associated with a road segment between two junctions. The weight of an edge may correspond to the length of the associated road segment, the time needed to traverse the segment, or the cost of traversing the segment. Using directed edges it is also possible to model one-way streets. Such graphs are special in the sense that some edges are more important than others for long distance travel (e.g. highways). This property has been formalized using the notion of highway dimension.[3] There are a great number of algorithms that exploit this property and are therefore able to compute the shortest path a lot quicker than would be possible on general graphs.

All of these algorithms work in two phases. In the first phase, the graph is preprocessed without knowing the source or target node. The second phase is the query phase. In this phase, source and target node are known.The idea is that the road network is static, so the preprocessing phase can be done once and used for a large number of queries on the same road network.

The algorithm with the fastest known query time is called hub labeling and is able to compute shortest path on the road networks of Europe or the USA in a fraction of a microsecond.[4] Other techniques that have been used are:

- ALT (A* search, landmarks, and triangle inequality)
- Arc flags
- Contraction hierarchies
- Transit node routing
- Reach-based pruning
- Labeling

4.3.6 Related problems

For shortest path problems in computational geometry, see Euclidean shortest path.

The travelling salesman problem is the problem of finding the shortest path that goes through every vertex exactly once, and returns to the start. Unlike the shortest path problem, which can be solved in polynomial time in graphs without negative cycles, the travelling salesman

problem is NP-complete and, as such, is believed not to be efficiently solvable for large sets of data (see P = NP problem). The problem of finding the longest path in a graph is also NP-complete.

The Canadian traveller problem and the stochastic shortest path problem are generalizations where either the graph isn't completely known to the mover, changes over time, or where actions (traversals) are probabilistic.

The shortest multiple disconnected path [5] is a representation of the primitive path network within the framework of Reptation theory.

The widest path problem seeks a path so that the minimum label of any edge is as large as possible.

Strategic shortest-paths

Sometimes, the edges in a graph have personalities: each edge has its own selfish interest. An example is a communication network, in which each edge is a computer that possibly belongs to a different person. Different computers have different transmission speeds, so every edge in the network has a numeric weight equal to the number of milliseconds it takes to transmit a message. Our goal is to send a message between two points in the network in the shortest time possible. If we know the transmission-time of each computer (the weight of each edge), then we can use a standard shortest-paths algorithm. If we do not know the transmission times, then we have to ask each computer to tell us its transmission-time. But, the computers may be selfish: a computer might tell us that its transmission time is very long, so that we will not bother it with our messages. A possible solution to this problem is to use a variant of the VCG mechanism, which gives the computers an incentive to reveal their true weights.

4.3.7 Linear programming formulation

There is a natural linear programming formulation for the shortest path problem, given below. It is very simple compared to most other uses of linear programs in discrete optimization, however it illustrates connections to other concepts.

Given a directed graph (V, A) with source node s, target node t, and cost w_{ij} for each edge (i, j) in A, consider the program with variables x_{ij}

$$\text{minimize } \sum_{ij \in A} w_{ij} x_{ij} \text{ subject to } x \geq 0 \text{ and for all } i, \sum_j x_{ij} - \sum_j x_{ji} = \begin{cases} 1, & \text{if } i = s; \\ -1, & \text{if } i = t; \\ 0, & \text{otherwise.} \end{cases}$$

The intuition behind this is that x_{ij} is an indicator variable for whether edge (i, j) is part of the shortest path: 1 when it is, and 0 if it is not. We wish to select the set of edges with minimal weight, subject to the constraint that this set forms a path from s to t (represented by the equality constraint: for all vertices except s and t the number of incoming and outcoming edges that are part of the path must be the same (i.e., that it should be a path from s to t).

This LP has the special property that it is integral; more specifically, every basic optimal solution (when one exists) has all variables equal to 0 or 1, and the set of edges whose variables equal 1 form an s-t dipath. See Ahuja et al.[6] for one proof, although the origin of this approach dates back to mid-20th century.

The dual for this linear program is

$$\text{maximize } y_t - y_s \text{ subject to for all } ij,\ y_j - y_i \leq w_{ij}$$

and feasible duals correspond to the concept of a consistent heuristic for the A* algorithm for shortest paths. For any feasible dual y the reduced costs $w'_{ij} = w_{ij} - y_j + y_i$ are nonnegative and A* essentially runs Dijkstra's algorithm on these reduced costs.

4.3.8 General algebraic framework on semirings: the algebraic path problem

Many problems can be framed as a form of the shortest path for some suitably substituted notions of addition along a path and taking the minimum. The general approach to these is to consider the two operations to be those of a semiring. Semiring multiplication is done along the path, and the addition is between paths. This general framework is known as the algebraic path problem.[7][8][9]

Most of the classic shortest-path algorithms (and new ones) can be formulated as solving linear systems over such algebraic structures.[10]

More recently, an even more general framework for solving these (and much less obviously related problems) has been developed under the banner of valuation algebras.[11]

4.3.9 Shortest path in stochastic time-dependent networks

In real-life situations, the transportation network is usually stochastic and time-dependent. In fact, a traveler traversing a link daily may experiences different travel times on that link due not only to the fluctuations in travel demand (origin-destination matrix) but also due to such incidents as work zones, bad weather conditions, accidents and vehicle breakdowns. As a result, a stochastic

time-dependent (STD) network is a more realistic representation of an actual road network compared with the deterministic one.[12]

Despite considerable progress during the course of the past decade, it remains a controversial question how an optimal path should be defined and identified in stochastic road networks. In other words, there is no unique definition of an optimal path under uncertainty. One possible and common answer to this question is to find a path with the minimum expected travel time. The main advantage of using this approach is that efficient shortest path algorithms introduced for the deterministic networks can be readily employed to identify the path with the minimum expected travel time in a stochastic network. However, the resulting optimal path identified by this approach may not be reliable, because this approach fails to address travel time variability. To tackle this issue some researchers use distribution of travel time instead of expected value of it so they find the probability distribution of total traveling time using different optimization methods such as dynamic programming and Dijkstra's algorithm .[13] These methods use stochastic optimization, specifically stochastic dynamic programming to find the shortest path in networks with probabilistic arc length.[14] It should be noted that the concept of travel time reliability is used interchangeably with travel time variability in the transportation research literature, so that, in general, one can say that the higher the variability in travel time, the lower the reliability would be, and vice versa.

In order to account for travel time reliability more accurately, two common alternative definitions for an optimal path under uncertainty have been suggested. Some have introduced the concept of the most reliable path, aiming to maximize the probability of arriving on time or earlier than a given travel time budget. Others, alternatively, have put forward the concept of an α-reliable path based on which they intended to minimize the travel time budget required to ensure a pre-specified on-time arrival probability.

4.3.10 See also

- Pathfinding
- IEEE 802.1aq
- Flow network
- Shortest path tree
- Euclidean shortest path
- Min-plus matrix multiplication
- Bidirectional search, an algorithm that finds the shortest path between two vertices on a directed graph
- Travel time reliability

4.3.11 References

Notes

[1] Sanders, Peter (March 23, 2009). "Fast route planning". Google Tech Talk.

[2] Chen, Danny Z. (December 1996). "Developing algorithms and software for geometric path planning problems". *ACM Computing Surveys*. **28** (4es): 18. doi:10.1145/242224.242246.

[3] Abraham, Ittai; Fiat, Amos; Goldberg, Andrew V.; Werneck, Renato F. "Highway Dimension, Shortest Paths, and Provably Efficient Algorithms". ACM-SIAM Symposium on Discrete Algorithms, pages 782-793, 2010.

[4] Abraham, Ittai; Delling, Daniel; Goldberg, Andrew V.; Werneck, Renato F. research.microsoft.com/pubs/142356/HL-TR.pdf "A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks". Symposium on Experimental Algorithms, pages 230-241, 2011.

[5] Kroger, Martin (2005). "Shortest multiple disconnected path for the analysis of entanglements in two- and three-dimensional polymeric systems". *Computer Physics Communications*. **168** (168): 209–232. doi:10.1016/j.cpc.2005.01.020.

[6] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin (1993). *Network Flows: Theory, Algorithms and Applications*. Prentice Hall. ISBN 0-13-617549-X.

[7] John Baras; George Theodorakopoulos (4 April 2010). *Path Problems in Networks*. Morgan & Claypool Publishers. pp. 9–. ISBN 978-1-59829-924-3.

[8] Mehryar Mohri, "Semiring frameworks and algorithms for shortest-distance problems", Journal of Automata, Languages and Combinatorics, Volume 7 Issue 3, January 2002, Pages 321 - 350

[9] http://www.iam.unibe.ch/~{}run/talks/2008-06-05-Bern-Jonczy.pdf

[10] Michel Gondran; Michel Minoux (2008). *Graphs, Dioids and Semirings: New Models and Algorithms*. Springer Science & Business Media. chapter 4. ISBN 978-0-387-75450-5.

[11] Marc Pouly; Jürg Kohlas (2011). *Generic Inference: A Unifying Theory for Automated Reasoning*. John Wiley & Sons. Chapter 6. Valuation Algebras for Path Problems. ISBN 978-1-118-01086-0.

[12] Rajabi-Bahaabadi, Mojtaba; Shariat-Mohaymany, Afshin; Babaei, Mohsen; Ahn, Chang Wook (2015). "Multi-objective path finding in stochastic time-dependent road networks using non-dominated sorting genetic algorithm". *Expert Systems with Applications*. **42** (12): 5056–5064. doi:10.1016/j.eswa.2015.02.046.

[13] Olya, Mohammad Hessam (2014). "Finding shortest path in a combined exponential – gamma probability distribution arc length". *International Journal of Operational Research*. **21** (1): 25–37. doi:10.1504/IJOR.2014.064020.

[14] Olya, Mohammad Hessam (2014). "Applying Dijkstra's algorithm for general shortest path problem with normal probability distribution arc length". *International Journal of Operational Research*. **21** (2): 143–154. doi:10.1504/IJOR.2014.064541.

Bibliography

- Ahuja, Ravindra K.; Mehlhorn, Kurt; Orlin, James; Tarjan, Robert E. (April 1990). "Faster algorithms for the shortest path problem". *Journal of the ACM*. ACM. **37** (2): 213–223. doi:10.1145/77600.77615.
- Bellman, Richard (1958). "On a routing problem". *Quarterly of Applied Mathematics*. **16**: 87–90. MR 0102435.
- Cherkassky, Boris V.; Goldberg, Andrew V.; Radzik, Tomasz (1996). "Shortest paths algorithms: theory and experimental evaluation". *Mathematical Programming*. Ser. A. **73** (2): 129–174. doi:10.1016/0025-5610(95)00021-6. MR 1392160.
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001) [1990]. "Single-Source Shortest Paths and All-Pairs Shortest Paths". *Introduction to Algorithms* (2nd ed.). MIT Press and McGraw-Hill. pp. 580–642. ISBN 0-262-03293-7.
- Dantzig, G. B. (January 1960). "On the Shortest Route through a Network". *Management Science*. **6** (2): 187–190. doi:10.1287/mnsc.6.2.187.
- Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs" (PDF). *Numerische Mathematik*. **1**: 269–271. doi:10.1007/BF01386390.
- Ford, L. R. (1956). "Network Flow Theory". Rand Corporation. P-923.
- Fredman, Michael Lawrence; Tarjan, Robert E. (1984). *Fibonacci heaps and their uses in improved network optimization algorithms*. 25th Annual Symposium on Foundations of Computer Science. IEEE. pp. 338–346. doi:10.1109/SFCS.1984.715934. ISBN 0-8186-0591-X.
- Fredman, Michael Lawrence; Tarjan, Robert E. (1987). "Fibonacci heaps and their uses in improved network optimization algorithms". *Journal of the Association for Computing Machinery*. **34** (3): 596–615. doi:10.1145/28869.28874.
- Gabow, H. N. (1983). "Scaling algorithms for network problems". *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS 1983)*. pp. 248–258. doi:10.1109/SFCS.1983.68.
- Gabow, Harold N. (1985). "Scaling algorithms for network problems". *Journal of Computer and System Sciences*. **31** (2): 148–168. doi:10.1016/0022-0000(85)90039-X. MR 828519.
- Hagerup, Torben (2000). Montanari, Ugo; Rolim, José D. P.; Welzl, Emo, eds. *Improved Shortest Paths on the Word RAM*. *Proceedings of the 27th International Colloquium on Automata, Languages and Programming*. pp. 61–72. ISBN 3-540-67715-1.
- Johnson, Donald B. (December 1981). "A priority queue in which initialization and queue operations take $O(\log \log D)$ time". *Mathematical Systems Theory*. **15** (1): 295–309. doi:10.1007/BF01786986. MR 683047.
- Karlsson, Rolf G.; Poblete, Patricio V. (1983). "An $O(m \log \log D)$ algorithm for shortest paths". *Discrete Applied Mathematics*. **6** (1): 91–93. doi:10.1016/0166-218X(83)90104-X. MR 700028.
- Leyzorek, M.; Gray, R. S.; Johnson, A. A.; Ladew, W. C.; Meaker, S. R., Jr.; Petry, R. M.; Seitz, R. N. (1957). *Investigation of Model Techniques — First Annual Report — 6 June 1956 — 1 July 1957 — A Study of Model Techniques for Communication Systems*. Cleveland, Ohio: Case Institute of Technology.
- Moore, E. F. (1959). "The shortest path through a maze". *Proceedings of an International Symposium on the Theory of Switching (Cambridge, Massachusetts, 2–5 April 1957)*. Cambridge: Harvard University Press. pp. 285–292.
- Pettie, Seth; Ramachandran, Vijaya (2002). *Computing shortest paths with comparisons and additions*. *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*. pp. 267–276. ISBN 0-89871-513-X.
- Pettie, Seth (26 January 2004). "A new approach to all-pairs shortest paths on real-weighted graphs". *Theoretical Computer Science*. **312** (1): 47–74. doi:10.1016/s0304-3975(03)00402-x.
- Pollack, M.; Wiebenson, W. (March–April 1960). "Solution of the Shortest-Route Problem—A Review". *Op. Res.* **8** (2): 224–230. doi:10.1287/opre.8.2.224.
- Schrijver, Alexander (2004). *Combinatorial Optimization — Polyhedra and Efficiency*. Algorithms and Combinatorics. **24**. Springer. ISBN 3-540-20456-3. Here: vol.A, sect.7.5b, p. 103
- Shimbel, Alfonso (1953). "Structural parameters of communication networks". *Bulletin of Mathematical Biophysics*. **15** (4): 501–507. doi:10.1007/BF02476438.

- Thorup, Mikkel (1999). "Undirected single-source shortest paths with positive integer weights in linear time". *Journal of the ACM (JACM)*. **46** (3): 362–394. doi:10.1145/316542.316548. Retrieved 28 November 2014.
- Thorup, Mikkel (2004). "Integer priority queues with decrease key in constant time and the single source shortest paths problem". *Journal of Computer and System Sciences*. **69** (3): 330–353. doi:10.1016/j.jcss.2004.04.003.
- Whiting, P. D.; Hillier, J. A. (March–June 1960). "A Method for Finding the Shortest Route through a Road Network". *Operational Research Quarterly*. **11** (1/2): 37–40. doi:10.1057/jors.1960.32.
- Williams, Ryan (2014). "Faster all-pairs shortest paths via circuit complexity". *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC '14)*. New York: ACM. pp. 664–673. arXiv:1312.6680. doi:10.1145/2591796.2591811. MR 3238994.

Missing references

- Dantzig (1958). Missing or empty |title= (help)

4.3.12 Further reading

- Frigioni, D.; Marchetti-Spaccamela, A.; Nanni, U. (1998). "Fully dynamic output bounded single source shortest path problem". *Proc. 7th Annu. ACM-SIAM Symp. Discrete Algorithms*. Atlanta, GA. pp. 212–221.
- Dreyfus, S. E. (October 1967). An Appraisal of Some Shortest Path Algorithms (PDF) (Report). Project Rand. United States Air Force. RM-5433-PR. DTIC AD-661265.

4.4 Dijkstra's algorithm

Not to be confused with Dykstra's projection algorithm.

Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.[1][2]

The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes,[2] but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

For a given source node in the graph, the algorithm finds the shortest path between that node and every other.[3]:196–206 It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path algorithm is widely used in network routing protocols, most notably IS-IS and Open Shortest Path First (OSPF). It is also employed as a subroutine in other algorithms such as Johnson's.

Dijkstra's original algorithm does not use a min-priority queue and runs in time $O(|V|^2)$ (where $|V|$ is the number of nodes). The idea of this algorithm is also given in Leyzorek et al. 1957. The implementation based on a min-priority queue implemented by a Fibonacci heap and running in $O(|E| + |V| \log |V|)$ (where $|E|$ is the number of edges) is due to Fredman & Tarjan 1984. This is asymptotically the fastest known single-source shortest-path algorithm for arbitrary directed graphs with unbounded non-negative weights. However, specialized cases (such as bounded/integer weights, directed acyclic graphs etc.) can indeed be improved further as detailed in § Specialized variants.

In some fields, artificial intelligence in particular, Dijkstra's algorithm or a variant of it is known as **uniform-cost search** and formulated as an instance of the more general idea of best-first search.[4]

4.4.1 History

Dijkstra thought about the shortest path problem when working at the Mathematical Center in Amsterdam in 1956 as a programmer to demonstrate capabilities of a new computer called ARMAC. His objective was to choose both a problem as well as an answer (that would be produced by computer) that non-computing people could understand. He designed the shortest path algorithm and later implemented it for ARMAC for a slightly simplified transportation map of 64 cities in the Netherlands (64, so that 6 bits would be sufficient to encode the city number).[1] A year later, he came across another problem from hardware engineers working on the institute's next computer: minimize the amount of wire needed to connect the pins on the back panel of the machine. As a solution, he re-discovered the algorithm known as Prim's minimal spanning tree algorithm (known earlier to Jarník, and also rediscovered by Prim).[5][6] Dijkstra published the algorithm in 1959, two years after Prim and 29 years after Jarník.[7][8]

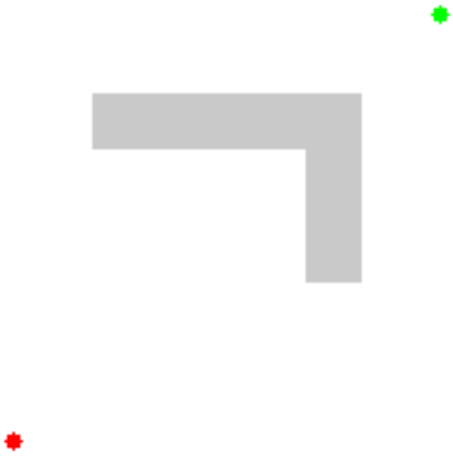


Illustration of Dijkstra's algorithm search for finding path from a start node (lower left, red) to a goal node (upper right, green) in a robot motion planning problem. Open nodes represent the "tentative" set. Filled nodes are visited ones, with color representing the distance: the greener, the farther. Nodes in all the different directions are explored uniformly, appearing as a more-or-less circular wavefront as Dijkstra's algorithm uses a heuristic identically equal to 0.

4.4.2 Algorithm

Let the node at which we are starting be called the **initial node**. Let the **distance of node *Y*** be the distance from the **initial node** to *Y*. Dijkstra's algorithm will assign some initial distance values and will try to improve them step by step.

1. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes.

2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.

3. For the current node, consider all of its unvisited neighbors and calculate their *tentative* distances. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbor *B* has length 2, then the distance to *B* (through *A*) will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, keep the current value.

4. When we are done considering all of the neighbors of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.

5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.

6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

4.4.3 Description

> ***Note:*** *For ease of understanding, this discussion uses the terms* ***intersection****,* ***road*** *and* ***map*** *— however, in formal terminology these terms are* ***vertex****,* ***edge*** *and* ***graph****, respectively.*

Suppose you would like to find the *shortest path* between two intersections on a city map: a *starting point* and a *destination*. Dijkstra's algorithm initially marks the distance (from the starting point) to every other intersection on the map with *infinity*. This is done not to imply there is an infinite distance, but to note that those intersections have not yet been visited; some variants of this method simply leave the intersections' distances *unlabeled*. Now, at each iteration, select the *current intersection*. For the first iteration, the current intersection will be the starting point, and the distance to it (the intersection's label) will be *zero*. For subsequent iterations (after the first), the current intersection will be the *closest unvisited intersection* to the starting point (this will be easy to find).

From the current intersection, *update* the distance to every unvisited intersection that is directly connected to it. This is done by determining the *sum* of the distance between an unvisited intersection and the value of the current intersection, and relabeling the unvisited intersection with this value (the sum), if it is less than its current value. In effect, the intersection is relabeled if the path to it through the current intersection is shorter than the previously known paths. To facilitate shortest path identification, in pencil, mark the road with an arrow pointing to the relabeled intersection if you label/relabel it, and erase all others pointing to it. After you have updated the distances to each neighboring intersection, mark the current intersection as *visited*, and select the unvisited intersection with lowest distance (from the starting point) – or the lowest label—as the current intersection. Nodes marked as visited are labeled with the shortest path from the starting point to it and will not be revisited or returned to.

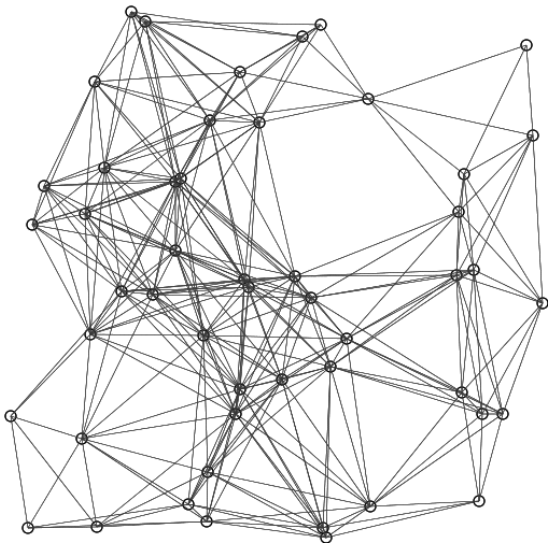
Continue this process of updating the neighboring intersections with the shortest distances, then marking the current intersection as visited and moving onto the closest unvisited intersection until you have marked the destination as visited. Once you have marked the destination as

visited (as is the case with any visited intersection) you have determined the shortest path to it, from the starting point, and can *trace your way back, following the arrows in reverse*; in the algorithm's implementations, this is usually done (after the algorithm has reached the destination node) by following the nodes' parents from the destination node up to the starting node; that's why we keep also track of each node's parent.

This algorithm makes no attempt to direct "exploration" towards the destination as one might expect. Rather, the sole consideration in determining the next "current" intersection is its distance from the starting point. This algorithm therefore expands outward from the starting point, interactively considering every node that is closer in terms of shortest path distance until it reaches the destination. When understood in this way, it is clear how the algorithm necessarily finds the shortest path. However, it may also reveal one of the algorithm's weaknesses: its relative slowness in some topologies.

4.4.4 Pseudocode

In the following algorithm, the code u ← vertex in *Q* with min dist[u], searches for the vertex u in the vertex set Q that has the least dist[u] value. length(u, v) returns the length of the edge joining (i.e. the distance between) the two neighbor-nodes u and v. The variable alt on line 17 is the length of the path from the root node to the neighbor node v if it were to go through u. If this path is shorter than the current shortest path recorded for v, that current path is replaced with this alt path. The prev array is populated with a pointer to the "next-hop" node on the source graph to get the shortest route to the source.



A demo of Dijkstra's algorithm based on Euclidean distance. Red lines are the shortest path covering, i.e., connecting u *and prev[*u*]. Blue lines indicate where relaxing happens, i.e., connecting* v *with a node* u *in* Q*, which gives a shorter path from the source to* v.

1 **function** Dijkstra(*Graph*, *source*): 2 3 create vertex set Q 4 5 **for each** vertex *v* in *Graph*: *// Initialization* 6 dist[*v*] ← INFINITY *// Unknown distance from source to v* 7 prev[*v*] ← UNDEFINED *// Previous node in optimal path from source* 8 add *v* to *Q* *// All nodes initially in Q (unvisited nodes)* 9 10 dist[*source*] ← 0 *// Distance from source to source* 11 12 **while** *Q* is not empty: 13 *u* ← vertex in *Q* with min dist[u] *// Node with the least distance will be selected first* 14 remove *u* from *Q* 15 16 **for each** neighbor *v* of *u*: *// where v is still in Q.* 17 *alt* ← dist[*u*] + length(*u*, *v*) 18 **if** *alt* < dist[*v*]: *// A shorter path to v has been found* 19 dist[*v*] ← *alt* 20 prev[*v*] ← *u* 21 22 **return** dist[], prev[]

If we are only interested in a shortest path between vertices source and target, we can terminate the search after line 13 if u = target. Now we can read the shortest path from source to target by reverse iteration:

1 *S* ← empty sequence 2 *u* ← *target* 3 **while** prev[*u*] is defined: *// Construct the shortest path with a stack S* 4 insert *u* at the beginning of *S* *// Push the vertex onto the stack* 5 *u* ← prev[*u*] *// Traverse from target to source* 6 insert *u* at the beginning of *S* *// Push the source onto the stack*

Now sequence S is the list of vertices constituting one of the shortest paths from source to target, or the empty sequence if no path exists.

A more general problem would be to find all the shortest paths between source and target (there might be several different ones of the same length). Then instead of storing only a single node in each entry of prev[] we would store all nodes satisfying the relaxation condition. For example, if both r and source connect to target and both of them lie on different shortest paths through target (because the edge cost is the same in both cases), then we would add both r and source to prev[target]. When the algorithm completes, prev[] data structure will actually describe a graph that is a subset of the original graph with some edges removed. Its key property will be that if the algorithm was run with some starting node, then every path from that node to any other node in the new graph will be the shortest path between those nodes in the original graph, and all paths of that length from the original graph will be present in the new graph. Then to actually find all these shortest paths between two given nodes we would use a path finding algorithm on the new graph, such as depth-first search.

Using a priority queue

A min-priority queue is an abstract data type that provides 3 basic operations : add_with_priority(), decrease_priority() and extract_min(). As mentioned earlier, using such a data structure can lead to faster computing times than using a basic queue. Notably, Fibonacci heap (Fredman & Tarjan 1984) or Brodal queue offer optimal implementations for those 3 operations. As the algorithm is slightly different, we mention it here, in

pseudo-code as well :

1 **function** Dijkstra(*Graph*, *source*): 2 dist[*source*] ← 0 *// Initialization* 3 4 create vertex set Q 5 6 **for each** vertex *v* in *Graph*: 7 **if** *v* ≠ *source* 8 dist[*v*] ← INFINITY *// Unknown distance from source to v* 9 prev[*v*] ← UNDEFINED *// Predecessor of v* 10 11 *Q*.add_with_priority(*v*, dist[*v*]) 12 13 14 **while** *Q* is not empty: *// The main loop* 15 *u* ← *Q*.extract_min() *// Remove and return best vertex* 16 **for each** neighbor *v* of *u*: *// only v that is still in Q* 17 *alt* = dist[*u*] + length(*u*, *v*) 18 **if** *alt* < dist[*v*] 19 dist[*v*] ← *alt* 20 prev[*v*] ← *u* 21 *Q*.decrease_priority(*v*, *alt*) 22 23 **return** dist[], prev[]

Instead of filling the priority queue with all nodes in the initialization phase, it is also possible to initialize it to contain only *source*; then, inside the **if** *alt* < dist[*v*] block, the node must be inserted if not already in the queue (instead of performing a decrease_priority operation).[3]:198

Other data structures can be used to achieve even faster computing times in practice.[9]

4.4.5 Proof of correctness

Proof is by induction on the number of visited nodes.

Invariant hypothesis: For each visited node v, dist[v] is the shortest distance from source to v; and for each unvisited node u, dist[u] is the shortest distance via visited nodes only from source to u (if such a path exists, otherwise infinity; note we do not assume dist[u] is the actual shortest distance for unvisited nodes).

The base case is when there is just one visited node, namely the initial node source, and the hypothesis is trivial.

Assume the hypothesis for *n-1* visited nodes. Now we choose an edge vu where u has the least dist[u] of any unvisited node and the edge vu is such that dist[u] = dist[v] + length[v,u]. dist[u] must be the shortest distance from source to u because if there were a shorter path, and if w was the first unvisited node on that path then by hypothesis dist[w] < dist[u] creating a contradiction. Similarly if there was a shorter path to u without using unvisited nodes then dist[u] would have been less than dist[v] + length[v,u].

After processing u it will still be true that for each unvisited node w, dist[w] is the shortest distance from source to w using visited nodes only, since if there were a shorter path which doesn't visit u we would have found it previously, and if there is a shorter path using u we update it when processing u .

4.4.6 Running time

Bounds of the running time of Dijkstra's algorithm on a graph with edges E and vertices V can be expressed as a function of the number of edges, denoted $|E|$, and the number of vertices, denoted $|V|$, using big-O notation. How tight a bound is possible depends on the way the vertex set Q is implemented. In the following, upper bounds can be simplified because $|E| = O(|V|^2)$ for any graph, but that simplification disregards the fact that in some problems, other upper bounds on $|E|$ may hold.

For any implementation of the vertex set Q , the running time is in

$$O(|E| \cdot T_{\text{dk}} + |V| \cdot T_{\text{em}})$$

where T_{dk} and T_{em} are the complexities of the *decrease-key* and *extract-minimum* operations in Q , respectively. The simplest implementation of Dijkstra's algorithm stores the vertex set Q as an ordinary linked list or array, and extract-minimum is simply a linear search through all vertices in Q . In this case, the running time is $O(|E| + |V|^2) = O(|V|^2)$.

For sparse graphs, that is, graphs with far fewer than $|V|^2$ edges, Dijkstra's algorithm can be implemented more efficiently by storing the graph in the form of adjacency lists and using a self-balancing binary search tree, binary heap, pairing heap, or Fibonacci heap as a priority queue to implement extracting minimum efficiently. To perform decrease-key steps in a binary heap efficiently, it is necessary to use an auxiliary data structure that maps each vertex to its position in the heap, and to keep this structure up to date as the priority queue Q changes. With a self-balancing binary search tree or binary heap, the algorithm requires

$$\Theta((|E| + |V|) \log |V|)$$

time in the worst case; for connected graphs this time bound can be simplified to $\Theta(|E| \log |V|)$. The Fibonacci heap improves this to

$$O(|E| + |V| \log |V|)$$

When using binary heaps, the average case time complexity is lower than the worst-case: assuming edge costs are drawn independently from a common probability distribution, the expected number of *decrease-key* operations is bounded by $O(|V| \log(|E|/|V|))$, giving a total running time of[3]:199–200

$$O(|E| + |V| \log \frac{|E|}{|V|} \log |V|)$$

Practical optimizations and infinite graphs

In common presentations of Dijkstra's algorithm, initially all nodes are entered into the priority queue. This is,

however, not necessary: the algorithm can start with a priority queue that contains only one item, and insert new items as they are discovered (instead of doing a decrease-key, check whether the key is in the queue; if it is, decrease its key, otherwise insert it).[3]:198 This variant has the same worst-case bounds as the common variant, but maintains a smaller priority queue in practice, speeding up the queue operations.[4]

Moreover, not inserting all nodes in a graph makes it possible to extend the algorithm to find the shortest path from a single source to the closest of a set of target nodes on infinite graphs or those too large to represent in memory. The resulting algorithm is called *uniform-cost search* (UCS) in the artificial intelligence literature[4][10][11] and can be expressed in pseudocode as

procedure *UniformCostSearch*(Graph, start, goal) node ← start cost ← 0 frontier ← priority queue containing node only explored ← empty set **do if** frontier is empty **return** failure node ← frontier.pop() **if** node is goal **return** solution explored.add(node) **for each** of node's neighbors n **if** n is not in explored **if** n is not in frontier frontier.add(n) **else if** n is in frontier with higher cost replace existing node with n

The complexity of this algorithm can be expressed in an alternative way for very large graphs: when C^* is the length of the shortest path from the start node to any node satisfying the "goal" predicate, each edge has cost at least ε, and the number of neighbors per node is bounded by b, then the algorithm's worst-case time and space complexity are both in $O(b^{1+\lfloor C^*/\varepsilon \rfloor})$.[10]

Further optimizations of Dijkstra's algorithm for the single-target case include bidirectional variants, goal-directed variants such as the A* algorithm (see § Related problems and algorithms), graph pruning to determine which nodes are likely to form the middle segment of shortest paths (reach-based routing), and hierarchical decompositions of the input graph that reduce *s*–*t* routing to connecting s and t to their respective "transit nodes" followed by shortest-path computation between these transit nodes using a "highway".[12] Combinations of such techniques may be needed for optimal practical performance on specific problems.[13]

Specialized variants

When arc weights are small integers (bounded by a parameter C), a monotone priority queue can be used to speed up Dijkstra's algorithm. The first algorithm of this type was **Dial's algorithm**, which used a bucket queue to obtain a running time $O(|E| + \operatorname{diam}(G))$ that depends on the weighted diameter of a graph with integer edge weights (Dial 1969). The use of a Van Emde Boas tree as the priority queue brings the complexity to $O(|E| \log\log C)$ (Ahuja et al. 1990). Another interesting implementation based on a combination of a new radix heap and the well-known Fibonacci heap runs in time $O(|E| + |V|\sqrt{\log C})$ (Ahuja et al. 1990). Finally, the best algorithms in this special case are as follows. The algorithm given by (Thorup 2000) runs in $O(|E| \log\log |V|)$ time and the algorithm given by (Raman 1997) runs in $O(|E| + |V| \min\{(\log |V|)^{1/3+\epsilon}, (\log C)^{1/4+\epsilon}\})$ time.

Also, for directed acyclic graphs, it is possible to find shortest paths from a given starting vertex in linear $O(|E| + |V|)$ time, by processing the vertices in a topological order, and calculating the path length for each vertex to be the minimum length obtained via any of its incoming edges.[14][15]

In the special case of integer weights and undirected graphs, Dijkstra's algorithm can be completely countered with a linear $O(|V| + |E|)$ complexity algorithm, given by (Thorup 1999).

4.4.7 Related problems and algorithms

The functionality of Dijkstra's original algorithm can be extended with a variety of modifications. For example, sometimes it is desirable to present solutions which are less than mathematically optimal. To obtain a ranked list of less-than-optimal solutions, the optimal solution is first calculated. A single edge appearing in the optimal solution is removed from the graph, and the optimum solution to this new graph is calculated. Each edge of the original solution is suppressed in turn and a new shortest-path calculated. The secondary solutions are then ranked and presented after the first optimal solution.

Dijkstra's algorithm is usually the working principle behind link-state routing protocols, OSPF and IS-IS being the most common ones.

Unlike Dijkstra's algorithm, the Bellman–Ford algorithm can be used on graphs with negative edge weights, as long as the graph contains no negative cycle reachable from the source vertex *s*. The presence of such cycles means there is no shortest path, since the total weight becomes lower each time the cycle is traversed. It is possible to adapt Dijkstra's algorithm to handle negative weight edges by combining it with the Bellman-Ford algorithm (to remove negative edges and detect negative cycles), such an algorithm is called Johnson's algorithm.

The A* algorithm is a generalization of Dijkstra's algorithm that cuts down on the size of the subgraph that must be explored, if additional information is available that provides a lower bound on the "distance" to the target. This approach can be viewed from the perspective of linear programming: there is a natural linear program for computing shortest paths, and solutions to its dual linear program are feasible if and only if they form a consistent heuristic (speaking roughly, since the sign conventions differ from place to place in the literature). This feasible dual / consistent heuristic defines a non-negative reduced cost and A* is essentially running Dijkstra's algorithm

with these reduced costs. If the dual satisfies the weaker condition of admissibility, then A* is instead more akin to the Bellman–Ford algorithm.

The process that underlies Dijkstra's algorithm is similar to the greedy process used in Prim's algorithm. Prim's purpose is to find a minimum spanning tree that connects all nodes in the graph; Dijkstra is concerned with only two nodes. Prim's does not evaluate the total weight of the path from the starting node, only the individual path.

Breadth-first search can be viewed as a special-case of Dijkstra's algorithm on unweighted graphs, where the priority queue degenerates into a FIFO queue.

Fast marching method can be viewed as a continuous version of Dijkstra's algorithm which computes the geodesic distance on a triangle mesh.

Dynamic programming perspective

From a dynamic programming point of view, Dijkstra's algorithm is a successive approximation scheme that solves the dynamic programming functional equation for the shortest path problem by the **Reaching** method.[16][17][18]

In fact, Dijkstra's explanation of the logic behind the algorithm,[19] namely

> **Problem 2.** Find the path of minimum total length between two given nodes P and Q .
>
> We use the fact that, if R is a node on the minimal path from P to Q , knowledge of the latter implies the knowledge of the minimal path from P to R .

is a paraphrasing of Bellman's famous Principle of Optimality in the context of the shortest path problem.

4.4.8 See also

- A* search algorithm
- Bellman–Ford algorithm
- Euclidean shortest path
- Flood fill
- Floyd–Warshall algorithm
- Johnson's algorithm
- Longest path problem

4.4.9 Notes

[1] Frana, Phil (August 2010). "An Interview with Edsger W. Dijkstra". *Communications of the ACM*. **53** (8): 41–47. doi:10.1145/1787234.1787249. What is the shortest way to travel from Rotterdam to Groningen? It is the algorithm for the shortest path which I designed in about 20 minutes. One morning I was shopping with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path.

[2] Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs" (PDF). *Numerische Mathematik*. **1**: 269–271. doi:10.1007/BF01386390.

[3] Mehlhorn, Kurt; Sanders, Peter (2008). *Algorithms and Data Structures: The Basic Toolbox* (PDF). Springer.

[4] Felner, Ariel (2011). *Position Paper: Dijkstra's Algorithm versus Uniform Cost Search or a Case Against Dijkstra's Algorithm*. Proc. 4th Int'l Symp. on Combinatorial Search. In a route-finding problem, Felner finds that the queue can be a factor 500–600 smaller, taking some 40% of the running time.

[5] Dijkstra, Edsger W., *Reflections on "A note on two problems in connexion with graphs* (PDF)

[6] Tarjan, Robert Endre (1983), *Data Structures and Network Algorithms*, CBMS_NSF Regional Conference Series in Applied Mathematics, **44**, Society for Industrial and Applied Mathematics, p. 75, The third classical minimum spanning tree algorithm was discovered by Jarník and rediscovered by Prim and Dikstra; it is commonly known as Prim's algorithm.

[7] R. C. Prim: *Shortest connection networks and some generalizations.* In: *Bell System Technical Journal*, 36 (1957), pp. 1389–1401.

[8] V. Jarník: *O jistém problému minimálním* [About a certain minimal problem], Práce Moravské Přírodovědecké Společnosti, 6, 1930, pp. 57–63. (in Czech)

[9] Chen, M.; Chowdhury, R. A.; Ramachandran, V.; Roche, D. L.; Tong, L. (2007). *Priority Queues and Dijkstra's Algorithm — UTCS Technical Report TR-07-54 — 12 October 2007* (PDF). Austin, Texas: The University of Texas at Austin, Department of Computer Sciences.

[10] Russell, Stuart; Norvig, Peter (2009) [1995]. *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall. pp. 75, 81. ISBN 978-0-13-604259-4.

[11] Sometimes also *least-cost-first search*: Nau, Dana S. (1983). "Expert computer systems" (PDF). *Computer*. IEEE. **16** (2): 63–85. doi:10.1109/mc.1983.1654302.

[12] Wagner, Dorothea; Willhalm, Thomas (2007). *Speed-up techniques for shortest-path computations*. STACS. pp. 23–36.

[13] Bauer, Reinhard; Delling, Daniel; Sanders, Peter; Schieferdecker, Dennis; Schultes, Dominik; Wagner,

Dorothea (2010). "Combining hierarchical and goal-directed speed-up techniques for Dijkstra's algorithm". *J. Experimental Algorithmics*. **15**.

[14] http://www.boost.org/doc/libs/1_44_0/libs/graph/doc/dag_shortest_paths.html

[15] Cormen et al. 2001, p. 655

[16] Sniedovich, M. (2006). "Dijkstra's algorithm revisited: the dynamic programming connexion" (PDF). *Journal of Control and Cybernetics*. **35** (3): 599–620. Online version of the paper with interactive computational modules.

[17] Denardo, E.V. (2003). *Dynamic Programming: Models and Applications*. Mineola, NY: Dover Publications. ISBN 978-0-486-42810-9.

[18] Sniedovich, M. (2010). *Dynamic Programming: Foundations and Principles*. Francis & Taylor. ISBN 978-0-8247-4099-3.

[19] Dijkstra 1959, p. 270

4.4.10 References

- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Section 24.3: Dijkstra's algorithm". *Introduction to Algorithms* (Second ed.). MIT Press and McGraw–Hill. pp. 595–601. ISBN 0-262-03293-7.
- Dial, Robert B. (1969). "Algorithm 360: Shortest-path forest with topological ordering [H]". *Communications of the ACM*. **12** (11): 632–633. doi:10.1145/363269.363610.
- Fredman, Michael Lawrence; Tarjan, Robert E. (1984). *Fibonacci heaps and their uses in improved network optimization algorithms*. 25th Annual Symposium on Foundations of Computer Science. IEEE. pp. 338–346. doi:10.1109/SFCS.1984.715934.
- Fredman, Michael Lawrence; Tarjan, Robert E. (1987). "Fibonacci heaps and their uses in improved network optimization algorithms". *Journal of the Association for Computing Machinery*. **34** (3): 596–615. doi:10.1145/28869.28874.
- Zhan, F. Benjamin; Noon, Charles E. (February 1998). "Shortest Path Algorithms: An Evaluation Using Real Road Networks". *Transportation Science*. **32** (1): 65–73. doi:10.1287/trsc.32.1.65.
- Leyzorek, M.; Gray, R. S.; Johnson, A. A.; Ladew, W. C.; Meaker, Jr., S. R.; Petry, R. M.; Seitz, R. N. (1957). *Investigation of Model Techniques — First Annual Report — 6 June 1956 — 1 July 1957 — A Study of Model Techniques for Communication Systems*. Cleveland, Ohio: Case Institute of Technology.
- Knuth, D.E. (1977). "A Generalization of Dijkstra's Algorithm". *Information Processing Letters*. **6** (1): 1–5. doi:10.1016/0020-0190(77)90002-3.
- Ahuja, Ravindra K.; Mehlhorn, Kurt; Orlin, James B.; Tarjan, Robert E. (April 1990). "Faster Algorithms for the Shortest Path Problem". *Journal of Association for Computing Machinery (ACM)*. **37** (2): 213–223. doi:10.1145/77600.77615.
- Raman, Rajeev (1997). "Recent results on the single-source shortest paths problem". *SIGACT News*. **28** (2): 81–87. doi:10.1145/261342.261352.
- Thorup, Mikkel (2000). "On RAM priority Queues". *SIAM Journal on Computing*. **30** (1): 86–109. doi:10.1137/S0097539795288246.
- Thorup, Mikkel (1999). "Undirected single-source shortest paths with positive integer weights in linear time". *journal of the ACM*. **46** (3): 362–394. doi:10.1145/316542.316548.

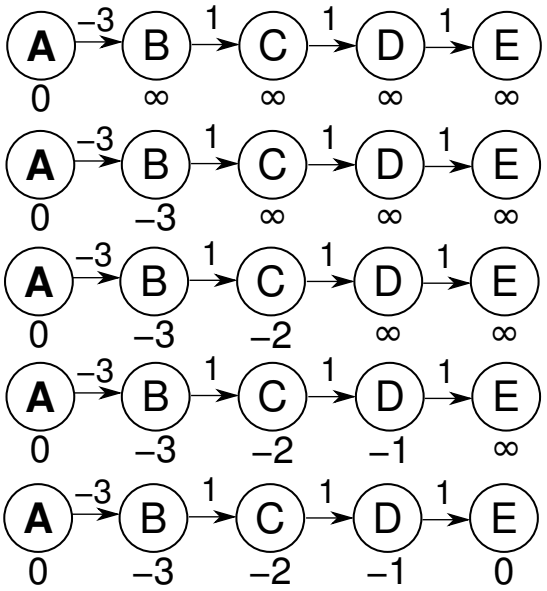
4.4.11 External links

- Oral history interview with Edsger W. Dijkstra, Charles Babbage Institute University of Minnesota, Minneapolis.
- Implementation of Dijkstra's algorithm using TDD, Robert Cecil Martin, The Clean Code Blog

4.5 Bellman–Ford algorithm

The **Bellman–Ford algorithm** is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph.[1] It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. The algorithm was first proposed by Alfonso Shimbel in 1955, but is instead named after Richard Bellman and Lester Ford, Jr., who published it in 1958 and 1956, respectively.[2] Edward F. Moore also published the same algorithm in 1957, and for this reason it is also sometimes called the **Bellman–Ford–Moore algorithm**.[1]

Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm.[3] If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, then there is no *cheapest* path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman–Ford algorithm can detect negative cycles and report their existence.[1][4]



−1 or 4 iterations for the distance estimates to converge. Conversely, if the edges are processed in the best order, from left to right, the algorithm converges in a single iteration.

4.5.1 Algorithm

Like Dijkstra's Algorithm, Bellman–Ford is based on the principle of relaxation, in which an approximation to the correct distance is gradually replaced by more accurate values until eventually reaching the optimum solution. In both algorithms, the approximate distance to each vertex is always an overestimate of the true distance, and is replaced by the minimum of its old value with the length of a newly found path. However, Dijkstra's algorithm uses a priority queue to greedily select the closest vertex that has not yet been processed, and performs this relaxation process on all of its outgoing edges; by contrast, the Bellman–Ford algorithm simply relaxes *all* the edges, and does this $|V| - 1$ times, where $|V|$ is the number of vertices in the graph. In each of these repetitions, the number of vertices with correctly calculated distances grows, from which it follows that eventually all vertices will have their correct distances. This method allows the Bellman–Ford algorithm to be applied to a wider class of inputs than Dijkstra.

Bellman–Ford runs in $O(|V| \cdot |E|)$ time, where $|V|$ and $|E|$ are the number of vertices and edges respectively.

function BellmanFord(*list* vertices, *list* edges, *vertex* source) ::distance[],predecessor[] *// This implementation takes in a graph, represented as // lists of vertices and edges, and fills two arrays // (distance and predecessor) with shortest-path // (less cost/distance/metric) information // Step 1: initialize graph* **for each** vertex v **in** vertices: distance[v] := **inf** // At the beginning , all vertices have a weight of infinity predecessor[v] := **null** // And a null predecessor distance[source] := 0 // Except for the Source, where the Weight is zero *// Step 2: relax edges repeatedly* **for** i **from** 1 **to** size(vertices)−1: **for each** edge (u, v) **with** weight w **in** edges: **if** distance[u] + w < distance[v]: distance[v] := distance[u] + w predecessor[v] := u *// Step 3: check for negative-weight cycles* **for each** edge (u, v) **with** weight w **in** edges: **if** distance[u] + w < distance[v]: **error** "Graph contains a negative-weight cycle" **return** distance[], predecessor[]

Simply put, the algorithm initializes the distance to the source to 0 and all other nodes to infinity. Then for all edges, if the distance to the destination can be shortened by taking the edge, the distance is updated to the new lower value. At each iteration i that the edges are scanned, the algorithm finds all shortest paths of at most length i edges (and possibly some paths longer than i edges). Since the longest possible path without a cycle can be $|V| - 1$ edges, the edges must be scanned $|V| - 1$ times to ensure the shortest path has been found for all nodes. A final scan of all the edges is performed and if any distance is updated, then a path of length $|V|$ edges has been found which can only occur if at least one negative cycle exists in the graph.

4.5.2 Proof of correctness

The correctness of the algorithm can be shown by induction. The precise statement shown by induction is:

Lemma. After *i* repetitions of *for* loop:

- If Distance(*u*) is not infinity, it is equal to the length of some path from *s* to *u*;
- If there is a path from *s* to *u* with at most *i* edges, then Distance(u) is at most the length of the shortest path from *s* to *u* with at most *i* edges.

Proof. For the base case of induction, consider i=0 and the moment before *for* loop is executed for the first time. Then, for the source vertex, source.distance = 0, which is correct. For other vertices *u*, u.distance = **infinity**, which is also correct because there is no path from *source* to *u* with 0 edges.

For the inductive case, we first prove the first part. Consider a moment when a vertex's distance is updated by v.distance := u.distance + uv.weight. By inductive assumption, u.distance is the length of some path from *source* to *u*. Then u.distance + uv.weight is the length of the path from *source* to *v* that follows the path from *source* to *u* and then goes to *v*.

For the second part, consider the shortest path from *source* to *u* with at most *i* edges. Let *v* be the last vertex before *u* on this path. Then, the part of the path from *source* to *v* is the shortest path from *source* to *v* with at most *i-1* edges. By inductive assumption, v.distance after *i*−1 iterations is at most the length of this path. Therefore, uv.weight + v.distance is at most the length of the

path from *s* to *u*. In the i^{th} iteration, u.distance gets compared with uv.weight + v.distance, and is set equal to it if uv.weight + v.distance was smaller. Therefore, after *i* iteration, u.distance is at most the length of the shortest path from *source* to *u* that uses at most *i* edges.

If there are no negative-weight cycles, then every shortest path visits each vertex at most once, so at step 3 no further improvements can be made. Conversely, suppose no improvement can be made. Then for any cycle with vertices *v*[0], ..., *v*[*k*−1],

v[i].distance <= v[i-1 (mod k)].distance + v[i-1 (mod k)]v[i].weight

Summing around the cycle, the *v*[*i*].distance and *v*[*i*−1 (mod *k*)].distance terms cancel, leaving

0 <= sum from 1 to k of v[i-1 (mod k)]v[i].weight

I.e., every cycle has nonnegative weight.

4.5.3 Finding negative cycles

When the algorithm is used to find shortest paths, the existence of negative cycles is a problem, preventing the algorithm from finding a correct answer. However, since it terminates upon finding a negative cycle, the Bellman–Ford algorithm can be used for applications in which this is the target to be sought - for example in cycle-cancelling techniques in network flow analysis.[1]

4.5.4 Applications in routing

A distributed variant of the Bellman–Ford algorithm is used in distance-vector routing protocols, for example the Routing Information Protocol (RIP). The algorithm is distributed because it involves a number of nodes (routers) within an Autonomous system, a collection of IP networks typically owned by an ISP. It consists of the following steps:

1. Each node calculates the distances between itself and all other nodes within the AS and stores this information as a table.
2. Each node sends its table to all neighboring nodes.
3. When a node receives distance tables from its neighbors, it calculates the shortest routes to all other nodes and updates its own table to reflect any changes.

The main disadvantages of the Bellman–Ford algorithm in this setting are as follows:

- It does not scale well.
- Changes in network topology are not reflected quickly since updates are spread node-by-node.
- Count to infinity if link or node failures render a node unreachable from some set of other nodes, those nodes may spend forever gradually increasing their estimates of the distance to it, and in the meantime there may be routing loops.

4.5.5 Improvements

The Bellman–Ford algorithm may be improved in practice (although not in the worst case) by the observation that, if an iteration of the main loop of the algorithm terminates without making any changes, the algorithm can be immediately terminated, as subsequent iterations will not make any more changes. With this early termination condition, the main loop may in some cases use many fewer than $|V| - 1$ iterations, even though the worst case of the algorithm remains unchanged.

Yen (1970) described two more improvements to the Bellman–Ford algorithm for a graph without negative-weight cycles; again, while making the algorithm faster in practice, they do not change its $O(|V| \cdot |E|)$ worst case time bound. His first improvement reduces the number of relaxation steps that need to be performed within each iteration of the algorithm. If a vertex *v* has a distance value that has not changed since the last time the edges out of *v* were relaxed, then there is no need to relax the edges out of *v* a second time. In this way, as the number of vertices with correct distance values grows, the number whose outgoing edges that need to be relaxed in each iteration shrinks, leading to a constant-factor savings in time for dense graphs.

Yen's second improvement first assigns some arbitrary linear order on all vertices and then partitions the set of all edges into two subsets. The first subset, *Ef*, contains all edges (*vi*, *vj*) such that $i < j$; the second, *Eb*, contains edges (*vi*, *vj*) such that $i > j$. Each vertex is visited in the order v_1, v_2, ..., $v|V|$, relaxing each outgoing edge from that vertex in *Ef*. Each vertex is then visited in the order $v|V|$, $v|V|_{-1}$, ..., v_1, relaxing each outgoing edge from that vertex in *Eb*. Each iteration of the main loop of the algorithm, after the first one, adds at least two edges to the set of edges whose relaxed distances match the correct shortest path distances: one from *Ef* and one from *Eb*. This modification reduces the worst-case number of iterations of the main loop of the algorithm from $|V| - 1$ to $|V|/2$.[5][6]

Another improvement, by Bannister & Eppstein (2012), replaces the arbitrary linear order of the vertices used in Yen's second improvement by a random permutation. This change makes the worst case for Yen's improvement (in which the edges of a shortest path strictly alternate between the two subsets *Ef* and *Eb*) very unlikely to happen. With a randomly permuted vertex ordering, the expected number of iterations needed in the main loop is at most $|V|/3$.[6]

4.5.6 Notes

[1] Bang-Jensen & Gutin (2000)

[2] Schrijver (2005)

[3] Sedgewick (2002).

[4] Kleinberg & Tardos (2006).

[5] Cormen et al., 2nd ed., Problem 24-1, pp. 614–615.

[6] See Sedgewick's web exercises for *Algorithms*, 4th ed., exercises 5 and 11 (retrieved 2013-01-30).

4.5.7 References

Original sources

- Shimbel, A. (1955). *Structure in communication nets*. Proceedings of the Symposium on Information Networks. New York, NY: Polytechnic Press of the Polytechnic Institute of Brooklyn. pp. 199–203.
- Bellman, Richard (1958). "On a routing problem". *Quarterly of Applied Mathematics*. **16**: 87–90. MR 0102435.
- Ford Jr., Lester R. (August 14, 1956). *Network Flow Theory*. Paper P-923. Santa Monica, California: RAND Corporation.
- Moore, Edward F. (1959). *The shortest path through a maze*. Proc. Internat. Sympos. Switching Theory 1957, Part II. Cambridge, Mass.: Harvard Univ. Press. pp. 285–292. MR 0114710.
- Yen, Jin Y. (1970). "An algorithm for finding shortest routes from all source nodes to a given destination in general networks". *Quarterly of Applied Mathematics*. **27**: 526–530. MR 0253822.
- Bannister, M. J.; Eppstein, D. (2012). *Randomized speedup of the Bellman–Ford algorithm* (PDF). Analytic Algorithmics and Combinatorics (ANALCO12), Kyoto, Japan. pp. 41–47. arXiv:1111.5414.

Secondary sources

- Bang-Jensen, Jørgen; Gutin, Gregory (2000). "Section 2.3.4: The Bellman-Ford-Moore algorithm". *Digraphs: Theory, Algorithms and Applications* (First ed.). ISBN 978-1-84800-997-4.
- Schrijver, Alexander (2005). "On the history of combinatorial optimization (till 1960)" (PDF). *Handbook of Discrete Optimization*. Elsevier: 1–68.
- Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. *Introduction to Algorithms*. MIT Press and McGraw-Hill., Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 24.1: The Bellman–Ford algorithm, pp. 588–592. Problem 24-1, pp. 614–615. Third Edition. MIT Press, 2009. ISBN 978-0-262-53305-8. Section 24.1: The Bellman–Ford algorithm, pp. 651–655.
- Heineman, George T.; Pollice, Gary; Selkow, Stanley (2008). "Chapter 6: Graph Algorithms". *Algorithms in a Nutshell*. O'Reilly Media. pp. 160–164. ISBN 978-0-596-51624-6.
- Kleinberg, Jon; Tardos, Éva (2006). *Algorithm Design*. New York: Pearson Education, Inc.
- Sedgewick, Robert (2002). "Section 21.7: Negative Edge Weights". *Algorithms in Java* (3rd ed.). ISBN 0-201-36121-3.

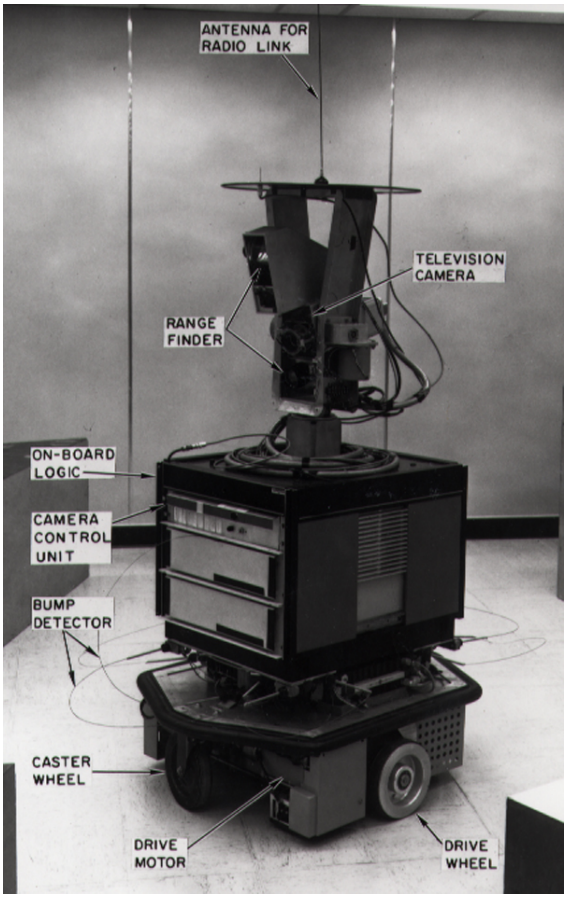
4.6 A* search algorithm

In computer science, **A*** (pronounced as "A star" (listen)) is a computer algorithm that is widely used in pathfinding and graph traversal, the process of plotting an efficiently directed path between multiple points, called nodes. It enjoys widespread use due to its performance and accuracy. However, in practical travel-routing systems, it is generally outperformed by algorithms which can pre-process the graph to attain better performance,[1] although other work has found A* to be superior to other approaches.[2]

Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute (now SRI International) first described the algorithm in 1968.[3] It is an extension of Edsger Dijkstra's 1959 algorithm. A* achieves better performance by using heuristics to guide its search.

4.6.1 History

In 1968, AI researcher Nils Nilsson was trying to improve the path planning done by Shakey the Robot, a prototype robot that could navigate through a room containing obstacles. This path-finding algorithm, which Nilsson called A1, was a faster version of the then best known method, Dijkstra's algorithm, for finding shortest paths in graphs. Bertram Raphael suggested some significant improvements upon this algorithm, calling the revised version A2. Then Peter E. Hart introduced an argument that established A2, with only minor changes, to be the best possible algorithm for finding shortest paths. Hart, Nilsson and Raphael then jointly developed a proof that the revised A2 algorithm was *optimal* for finding shortest paths under certain well-defined conditions.



A was invented by researchers working on Shakey the Robot's path planning.*

4.6.2 Description

A* is an informed search algorithm, or a best-first search, meaning that it solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that *appear* to lead most quickly to the solution. It is formulated in terms of weighted graphs: starting from a specific node of a graph, it constructs a tree of paths starting from that node, expanding paths one step at a time, until one of its paths ends at the predetermined goal node.

At each iteration of its main loop, A* needs to determine which of its partial paths to expand into one or more longer paths. It does so based on an estimate of the cost (total weight) still to go to the goal node. Specifically, A* selects the path that minimizes

$$f(n) = g(n) + h(n)$$

where n is the last node on the path, $g(n)$ is the cost of the path from the start node to n, and $h(n)$ is a heuristic that estimates the cost of the cheapest path from n to the goal. The heuristic is problem-specific. For the algorithm to find the actual shortest path, the heuristic function must be admissible, meaning that it never overestimates the actual cost to get to the nearest goal node.

Typical implementations of A* use a priority queue to perform the repeated selection of minimum (estimated) cost nodes to expand. This priority queue is known as the *open set* or *fringe*. At each step of the algorithm, the node with the lowest $f(x)$ value is removed from the queue, the f and g values of its neighbors are updated accordingly, and these neighbors are added to the queue. The algorithm continues until a goal node has a lower f value than any node in the queue (or until the queue is empty).[lower-alpha 1] The f value of the goal is then the length of the shortest path, since h at the goal is zero in an admissible heuristic.

The algorithm described so far gives us only the length of the shortest path. To find the actual sequence of steps, the algorithm can be easily revised so that each node on the path keeps track of its predecessor. After this algorithm is run, the ending node will point to its predecessor, and so on, until some node's predecessor is the start node.

As an example, when searching for the shortest route on a map, $h(x)$ might represent the straight-line distance to the goal, since that is physically the smallest possible distance between any two points.

If the heuristic h satisfies the additional condition $h(x) \leq d(x, y) + h(y)$ for every edge (x, y) of the graph (where d denotes the length of that edge), then h is called monotone, or consistent. In such a case, A* can be implemented more efficiently—roughly speaking, no node needs to be processed more than once (see *closed set* below)—and A* is equivalent to running Dijkstra's algorithm with the reduced cost $d'(x, y) = d(x, y) + h(y) - h(x)$.

Pseudocode

The following pseudocode describes the algorithm:

function A*(start, goal) // The set of nodes already evaluated. closedSet := {} // The set of currently discovered nodes that are already evaluated. // Initially, only the start node is known. openSet := {start} // For each node, which node it can most efficiently be reached from. // If a node can be reached from many nodes, cameFrom will eventually contain the // most efficient previous step. cameFrom := the empty map // For each node, the cost of getting from the start node to that node. gScore := map with default value of Infinity // The cost of going from start to start is zero. gScore[start] := 0 // For each node, the total cost of getting from the start node to the goal // by passing by that node. That value is partly known, partly heuristic. fScore := map with default value of Infinity // For the first node, that value is completely heuristic. fScore[start] := heuristic_cost_estimate(start, goal) while openSet is not empty current := the node in openSet having the lowest fScore[] value if current

= goal return reconstruct_path(cameFrom, current) openSet.Remove(current) closedSet.Add(current) for each neighbor of current if neighbor in closedSet continue // Ignore the neighbor which is already evaluated. // The distance from start to a neighbor tentative_gScore := gScore[current] + dist_between(current, neighbor) if neighbor not in openSet // Discover a new node openSet.Add(neighbor) else if tentative_gScore >= gScore[neighbor] continue // This is not a better path. // This path is the best until now. Record it! cameFrom[neighbor] := current gScore[neighbor] := tentative_gScore fScore[neighbor] := gScore[neighbor] + heuristic_cost_estimate(neighbor, goal) return failure function reconstruct_path(cameFrom, current) total_path := [current] while current in cameFrom.Keys: current := cameFrom[current] total_path.append(current) return total_path

Remark: the above pseudocode assumes that the heuristic function is *monotonic* (or consistent, see below), which is a frequent case in many practical problems, such as the Shortest Distance Path in road networks. However, if the assumption is not true, nodes in the **closed** set may be rediscovered and their cost improved. In other words, the closed set can be omitted (yielding a tree search algorithm) if a solution is guaranteed to exist, or if the algorithm is adapted so that new nodes are added to the open set only if they have a lower f value than at any previous iteration.

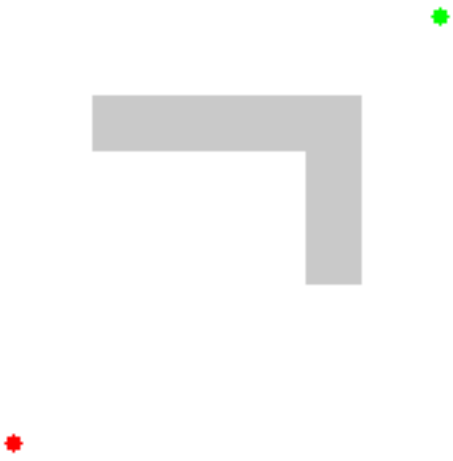
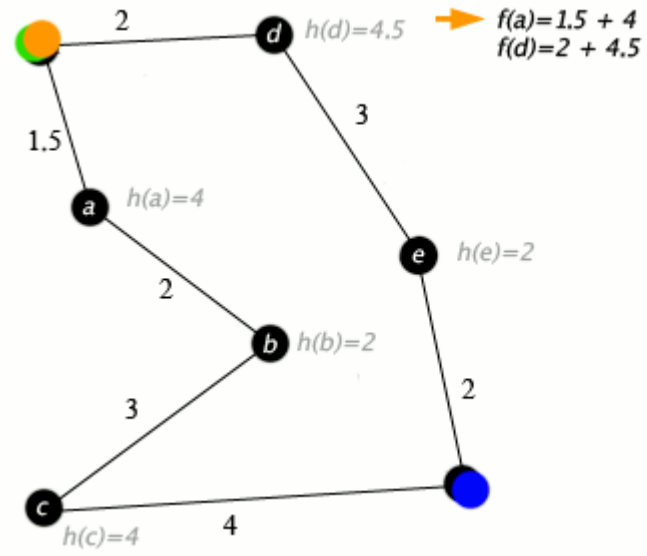


Illustration of A search for finding path from a start node to a goal node in a robot motion planning problem. The empty circles represent the nodes in the* open set*, i.e., those that remain to be explored, and the filled ones are in the closed set. Color on each closed node indicates the distance from the start: the greener, the farther. One can first see the A* moving in a straight line in the direction of the goal, then when hitting the obstacle, it explores alternative routes through the nodes from the open set.*
See also: Dijkstra's algorithm

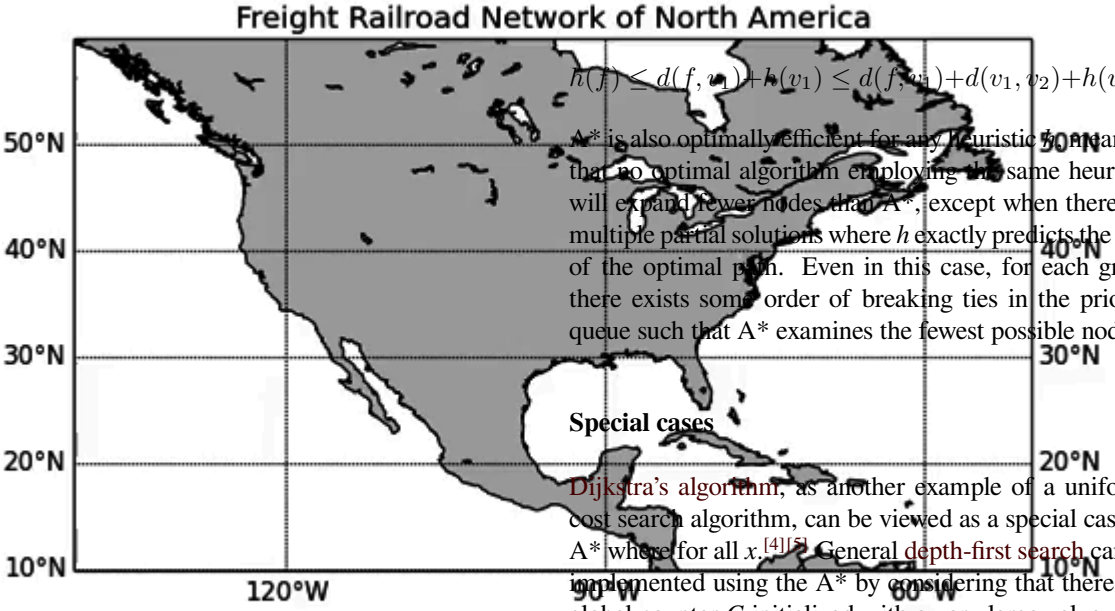
Example

An example of an A* algorithm in action where nodes are cities connected with roads and h(x) is the straight-line distance to target point:



Key: green: start; blue: goal; orange: visited

The A* algorithm also has real-world applications. In this example, edges are railroads and h(x) is the great-circle distance (the shortest possible distance on a sphere) to the target. The algorithm is searching for a path between Washington, D.C. and Los Angeles.



4.6.3 Properties

Like breadth-first search, A* is *complete* and will always find a solution if one exists provided $d(x,y) > \varepsilon > 0$ for fixed ε .

If the heuristic function *h* is admissible, meaning that it never overestimates the actual minimal cost of reaching the goal, then A* is itself admissible (or *optimal*) if we do not use a closed set. If a closed set is used, then *h* must also be *monotonic* (or consistent) for A* to be optimal. This means that for any pair of adjacent nodes *x* and *y*, where denotes the length of the edge between them, we must have:

$$h(x) \leq d(x,y) + h(y)$$

This ensures that for any path *X* from the initial node to *x*:

$$L(X)+h(x) \leq L(X)+d(x,y)+h(y) = L(Y)+h(y)$$

where *L* is a function that denotes the length of a path, and *Y* is the path *X* extended to include *y*. In other words, it is impossible to decrease (total distance so far + estimated remaining distance) by extending a path to include a neighboring node. (This is analogous to the restriction to nonnegative edge weights in Dijkstra's algorithm.)

Monotonicity implies admissibility when the heuristic estimate at any goal node itself is zero, since (letting $P = (f,v_1,v_2,...,vn,g)$ be a shortest path from any node *f* to the nearest goal *g*):

$$h(f) \leq d(f,v_1)+h(v_1) \leq d(f,v_1)+d(v_1,v_2)+h(v_2) \leq \ldots \leq L(P)+h($$

A* is also optimally efficient for any heuristic function, meaning that no optimal algorithm employing the same heuristic will expand fewer nodes than A*, except when there are multiple partial solutions where *h* exactly predicts the cost of the optimal path. Even in this case, for each graph there exists some order of breaking ties in the priority queue such that A* examines the fewest possible nodes.

Special cases

Dijkstra's algorithm, as another example of a uniform-cost search algorithm, can be viewed as a special case of A* where for all *x*.[4][5] General depth-first search can be implemented using the A* by considering that there is a global counter *C* initialized with a very large value. Every time we process a node we assign *C* to all of its newly discovered neighbors. After each single assignment, we decrease the counter *C* by one. Thus the earlier a node is discovered, the higher its value. It should be noted, however, that both Dijkstra's algorithm and depth-first search can be implemented more efficiently without including a value at each node.

Implementation details

There are a number of simple optimizations or implementation details that can significantly affect the performance of an A* implementation. The first detail to note is that the way the priority queue handles ties can have a significant effect on performance in some situations. If ties are broken so the queue behaves in a LIFO manner, A* will behave like depth-first search among equal cost paths (avoiding exploring more than one equally optimal solution).

When a path is required at the end of the search, it is common to keep with each node a reference to that node's parent. At the end of the search these references can be used to recover the optimal path. If these references are being kept then it can be important that the same node doesn't appear in the priority queue more than once (each entry corresponding to a different path to the node, and each with a different cost). A standard approach here is to check if a node about to be added already appears in the priority queue. If it does, then the priority and parent pointers are changed to correspond to the lower cost path. A standard binary heap based priority queue does not directly support the operation of searching for one of its elements, but it can be augmented with a hash

table that maps elements to their position in the heap, allowing this decrease-priority operation to be performed in logarithmic time. Alternatively, a Fibonacci heap can perform the same decrease-priority operations in constant amortized time.

4.6.4 Admissibility and optimality

A* is admissible and considers fewer nodes than any other admissible search algorithm with the same heuristic. This is because A* uses an "optimistic" estimate of the cost of a path through every node that it considers—optimistic in that the true cost of a path through that node to the goal will be at least as great as the estimate. But, critically, as far as A* "knows", that optimistic estimate might be achievable.

To prove the admissibility of A*, the solution path returned by the algorithm is used as follows:

When A* terminates its search, it has found a path whose actual cost is lower than the estimated cost of any path through any open node. But since those estimates are optimistic, A* can safely ignore those nodes. In other words, A* will never overlook the possibility of a lower-cost path and so is admissible.

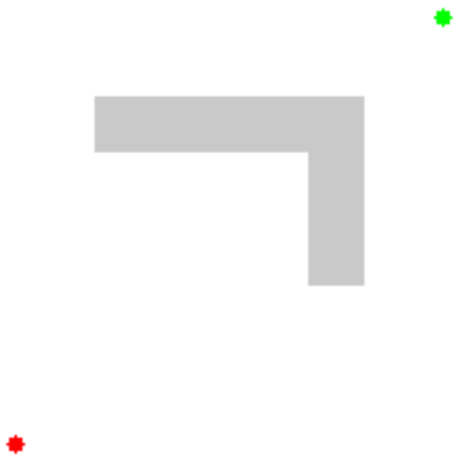
Suppose now that some other search algorithm B terminates its search with a path whose actual cost is *not* less than the estimated cost of a path through some open node. Based on the heuristic information it has, Algorithm B cannot rule out the possibility that a path through that node has a lower cost. So while B might consider fewer nodes than A*, it cannot be admissible. Accordingly, A* considers the fewest nodes of any admissible search algorithm.

This is only true if both:

- A* uses an admissible heuristic. Otherwise, A* is not guaranteed to expand fewer nodes than another search algorithm with the same heuristic.[6]
- A* solves only one search problem rather than a series of similar search problems. Otherwise, A* is not guaranteed to expand fewer nodes than incremental heuristic search algorithms.[7]

Bounded relaxation

While the admissibility criterion guarantees an optimal solution path, it also means that A* must examine all equally meritorious paths to find the optimal path. It is possible to speed up the search at the expense of optimality by relaxing the admissibility criterion. Oftentimes we want to bound this relaxation, so that we can guarantee that the solution path is no worse than $(1 + \varepsilon)$ times the optimal solution path. This new guarantee is referred to as ε-admissible.



A search that uses a heuristic that is 5.0(=ε) times a consistent heuristic, and obtains a suboptimal path.*

There are a number of ε-admissible algorithms:

- Weighted A*/Static Weighting.[8] If *ha*(*n*) is an admissible heuristic function, in the weighted version of the A* search one uses *hw*(*n*) = ε *ha*(*n*), $\varepsilon > 1$ as the heuristic function, and perform the A* search as usual (which eventually happens faster than using *ha* since fewer nodes are expanded). The path hence found by the search algorithm can have a cost of at most ε times that of the least cost path in the graph.[9]
- Dynamic Weighting[10] uses the cost function , where $w(n) = \begin{cases} 1 - \frac{d(n)}{N} & d(n) \leq N \\ 0 & \text{otherwise} \end{cases}$, and where is the depth of the search and *N* is the anticipated length of the solution path.
- Sampled Dynamic Weighting[11] uses sampling of nodes to better estimate and debias the heuristic error.
- A^*_ε .[12] uses two heuristic functions. The first is the FOCAL list, which is used to select candidate nodes, and the second *hF* is used to select the most promising node from the FOCAL list.
- Aε[13] selects nodes with the function , where *A* and *B* are constants. If no nodes can be selected, the algorithm will backtrack with the function , where *C* and *D* are constants.
- AlphA*[14] attempts to promote depth-first exploitation by preferring recently expanded nodes. AlphA* uses the cost function $f_\alpha(n) = (1 + w_\alpha(n))f(n)$

, where $w_\alpha(n) = \begin{cases} \lambda & g(\pi(n)) \leq g(\tilde{n}) \\ \Lambda & \text{otherwise} \end{cases}$, where λ and Λ are constants with $\lambda \leq \Lambda$, $\pi(n)$ is the parent of n, and $\tilde{n}$ is the most recently expanded node.

4.6.5 Complexity

The time complexity of A* depends on the heuristic. In the worst case of an unbounded search space, the number of nodes expanded is exponential in the depth of the solution (the shortest path) d: $O(b^d)$, where b is the branching factor (the average number of successors per state).[15] This assumes that a goal state exists at all, and is reachable from the start state; if it is not, and the state space is infinite, the algorithm will not terminate.

The heuristic function has a major effect on the practical performance of A* search, since a good heuristic allows A* to prune away many of the b^d nodes that an uninformed search would expand. Its quality can be expressed in terms of the *effective* branching factor b^*, which can be determined empirically for a problem instance by measuring the number of nodes expanded, N, and the depth of the solution, then solving[16]

$$N + 1 = 1 + b^* + (b^*)^2 + \cdots + (b^*)^d.$$

Good heuristics are those with low effective branching factor (the optimal being $b^* = 1$).

The time complexity is polynomial when the search space is a tree, there is a single goal state, and the heuristic function h meets the following condition:

$$|h(x) - h^*(x)| = O(\log h^*(x))$$

where h^* is the optimal heuristic, the exact cost to get from x to the goal. In other words, the error of h will not grow faster than the logarithm of the "perfect heuristic" h^* that returns the true distance from x to the goal.[9][15]

4.6.6 Applications

A* is commonly used for the common pathfinding problem in applications such as games, but was originally designed as a general graph traversal algorithm.[3] It finds applications to diverse problems, including the problem of parsing using stochastic grammars in NLP.[17] Other cases include an Informational search with online learning [18]

4.6.7 Relations to other algorithms

What sets A* apart from a greedy best-first search algorithm is that it takes the cost/distance already traveled, $g(n)$, into account.

Some common variants of Dijkstra's algorithm can be viewed as a special case of A* where the heuristic $h(n) = 0$ for all nodes;[4][5] in turn, both Dijkstra and A* are special cases of dynamic programming.[19] A* itself is a special case of a generalization of branch and bound[20] and can be derived from the primal-dual algorithm for linear programming.[21]

Variants of A*

- Anytime Repairing A* (ARA*)[22]
- Block A*
- D*
- Field D*
- Fringe
- Fringe Saving A* (FSA*)
- Generalized Adaptive A* (GAA*)
- IDA*
- Informational search [18]
- Jump point search
- Lifelong Planning A* (LPA*)
- Simplified Memory bounded A* (SMA*)
- Theta*
- Anytime A* [23]
- Realtime A* [24]
- Anytime Dynamic A*

A* can also be adapted to a bidirectional search algorithm. Special care needs to be taken for the stopping criterion.[25]

4.6.8 See also

- Pathfinding
- Breadth-first search
- Depth-first search
- Any-angle path planning, search for paths that are not limited to move along graph edges but rather can take on any angle

4.6.9 Notes

[1] Goal nodes may be passed over multiple times if there remain other nodes with lower f values, as they may lead to a shorter path to a goal.

4.6.10 References

[1] Delling, D.; Sanders, P.; Schultes, D.; Wagner, D. (2009). "Engineering route planning ...algorithms". *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*. Springer. pp. 117–139. doi:10.1007/978-3-642-02094-0_7.

[2] Zeng, W.; Church, R. L. (2009). "Finding shortest paths on real road networks: the case for A*". *International Journal of Geographical Information Science*. **23** (4): 531–543. doi:10.1080/13658810801949850.

[3] Hart, P. E.; Nilsson, N. J.; Raphael, B. (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". *IEEE Transactions on Systems Science and Cybernetics SSC4*. **4** (2): 100–107. doi:10.1109/TSSC.1968.300136.

[4] De Smith, Michael John; Goodchild, Michael F.; Longley, Paul (2007), *Geospatial Analysis: A Comprehensive Guide to Principles, Techniques and Software Tools*, Troubadour Publishing Ltd, p. 344, ISBN 9781905886609.

[5] Hetland, Magnus Lie (2010), *Python Algorithms: Mastering Basic Algorithms in the Python Language*, Apress, p. 214, ISBN 9781430232377.

[6] Dechter, Rina; Judea Pearl (1985). "Generalized best-first search strategies and the optimality of A*". *Journal of the ACM*. **32** (3): 505–536. doi:10.1145/3828.3830.

[7] Koenig, Sven; Maxim Likhachev; Yaxin Liu; David Furcy (2004). "Incremental heuristic search in AI". *AI Magazine*. **25** (2): 99–112.

[8] Pohl, Ira (1970). "First results on the effect of error in heuristic search". *Machine Intelligence*. **5**: 219–236.

[9] Pearl, Judea (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley. ISBN 0-201-05594-5.

[10] Pohl, Ira (August 1973). "The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving" (PDF). *Proceedings of the Third International Joint Conference on Artificial Intelligence (IJCAI-73)*. **3**. California, USA. pp. 11–17.

[11] Köll, Andreas; Hermann Kaindl (August 1992). "A new approach to dynamic weighting". *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*. Vienna, Austria. pp. 16–17.

[12] Pearl, Judea; Jin H. Kim (1982). "Studies in semi-admissible heuristics". *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*. **4** (4): 392–399.

[13] Ghallab, Malik; Dennis Allard (August 1983). "*Aε* – an efficient near admissible heuristic search algorithm" (PDF). *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*. **2**. Karlsruhe, Germany. pp. 789–791.

[14] Reese, Bjørn (1999). "AlphA*: An ε-admissible heuristic search algorithm" (PDF).

[15] Russell, Stuart; Norvig, Peter (2003) [1995]. *Artificial Intelligence: A Modern Approach* (2nd ed.). Prentice Hall. pp. 97–104. ISBN 978-0137903955.

[16] Russell, Stuart; Norvig, Peter (2009) [1995]. *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall. p. 103. ISBN 978-0-13-604259-4.

[17] Klein, Dan; Manning, Christopher D. (2003). *A* parsing: fast exact Viterbi parse selection*. Proc. NAACL-HLT.

[18] Kagan E. and Ben-Gal I. (2014). "A Group-Testing Algorithm with Online Informational Learning" (PDF). IIE Transactions, 46:2, 164-184,.

[19] Ferguson, Dave; Likhachev, Maxim; Stentz, Anthony (2005). *A Guide to Heuristic-based Path Planning* (PDF). Proc. ICAPS Workshop on Planning under Uncertainty for Autonomous Systems.

[20] Nau, Dana S.; Kumar, Vipin; Kanal, Laveen (1984). "General branch and bound, and its relation to A∗ and AO∗" (PDF). *Artificial Intelligence*. **23** (1): 29–58. doi:10.1016/0004-3702(84)90004-3.

[21] Ye, Xugang; Han, Shih-Ping; Lin, Anhua (2010). "A Note on the Connection Between the Primal-Dual and the A* Algorithm". *Int'l J. Operations Research and Information Systems*. **1** (1): 73–85.

[22] Likhachev, Maxim; Gordon, Geoff; Thrun, Sebastian. "ARA*: Anytime A* search with provable bounds on sub-optimality". In S. Thrun, L. Saul, and B. Schölkopf, editors, *Proceedings of Conference on Neural Information Processing Systems (NIPS)*, Cambridge, MA, 2003. MIT Press.

[23] Hansen, Eric A., and Rong Zhou. "Anytime Heuristic Search." J. Artif. Intell. Res.(JAIR) 28 (2007): 267-297.

[24] Korf, Richard E. "Real-time heuristic search." Artificial intelligence 42.2-3 (1990): 189-211.

[25] "Efficient Point-to-Point Shortest Path Algorithms" (PDF). from Princeton University

4.6.11 Further reading

- Hart, P. E.; Nilsson, N. J.; Raphael, B. (1972). "Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths"". *SIGART Newsletter*. **37**: 28–29. doi:10.1145/1056777.1056779.
- Nilsson, N. J. (1980). *Principles of Artificial Intelligence*. Palo Alto, California: Tioga Publishing Company. ISBN 0-935382-01-1.

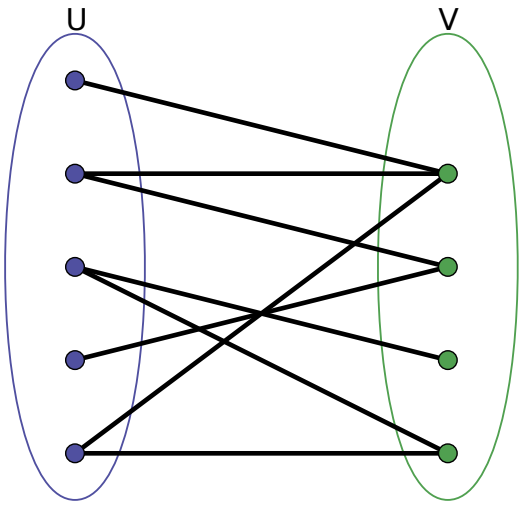
4.6.12 External links

- Clear visual A* explanation, with advice and thoughts on path-finding
- Variation on A* called Hierarchical Path-Finding A* (HPA*)

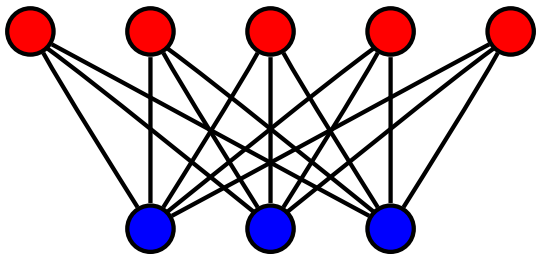
Chapter 5

Analysis

5.1 Bipartite graph



Example of a bipartite graph without cycles



A complete bipartite graph with m = 5 and n = 3

In the mathematical field of graph theory, a **bipartite graph** (or **bigraph**) is a graph whose vertices can be divided into two disjoint sets U and V (that is, U and V are each independent sets) such that every edge connects a vertex in U to one in V . Vertex sets U and V are usually called the *parts* of the graph. Equivalently, a bipartite graph is a graph that does not contain any odd-length cycles.[1][2]

The two sets U and V may be thought of as a coloring of the graph with two colors: if one colors all nodes in U blue, and all nodes in V green, each edge has endpoints of differing colors, as is required in the graph coloring problem.[3][4] In contrast, such a coloring is impossible in the case of a non-bipartite graph, such as a triangle: after one node is colored blue and another green, the third vertex of the triangle is connected to vertices of both colors, preventing it from being assigned either color.

One often writes $G = (U, V, E)$ to denote a bipartite graph whose partition has the parts U and V , with E denoting the edges of the graph. If a bipartite graph is not connected, it may have more than one bipartition;[5] in this case, the (U, V, E) notation is helpful in specifying one particular bipartition that may be of importance in an application. If $|U| = |V|$, that is, if the two subsets have equal cardinality, then G is called a *balanced* bipartite graph.[3] If all vertices on the same side of the bipartition have the same degree, then G is called biregular.

5.1.1 Examples

When modelling relations between two different classes of objects, bipartite graphs very often arise naturally. For instance, a graph of football players and clubs, with an edge between a player and a club if the player has played for that club, is a natural example of an *affiliation network*, a type of bipartite graph used in social network analysis.[6]

Another example where bipartite graphs appear naturally is in the (NP-complete) railway optimization problem, in which the input is a schedule of trains and their stops, and the goal is to find a set of train stations as small as possible such that every train visits at least one of the chosen stations. This problem can be modeled as a dominating set problem in a bipartite graph that has a vertex for each train and each station and an edge for each pair of a station and a train that stops at that station.[7]

A third example is in the academic field of numismatics. Ancient coins are made using two positive impressions of the design (the obverse and reverse). The charts numismatists produce to represent the production of coins are bipartite graphs. [8]

More abstract examples include the following:

- Every tree is bipartite.[4]

- Cycle graphs with an even number of vertices are bipartite.[4]
- Every planar graph whose faces all have even length is bipartite.[9] Special cases of this are grid graphs and squaregraphs, in which every inner face consists of 4 edges and every inner vertex has four or more neighbors.[10]
- The complete bipartite graph on *m* and *n* vertices, denoted by *Kn,m* is the bipartite graph *G = (U, V, E)*, where *U* and *V* are disjoint sets of size *m* and *n*, respectively, and *E* connects every vertex in *U* with all vertices in *V*. It follows that *Km,n* has *mn* edges.[11] Closely related to the complete bipartite graphs are the crown graphs, formed from complete bipartite graphs by removing the edges of a perfect matching.[12]
- Hypercube graphs, partial cubes, and median graphs are bipartite. In these graphs, the vertices may be labeled by bitvectors, in such a way that two vertices are adjacent if and only if the corresponding bitvectors differ in a single position. A bipartition may be formed by separating the vertices whose bitvectors have an even number of ones from the vertices with an odd number of ones. Trees and squaregraphs form examples of median graphs, and every median graph is a partial cube.[13]

5.1.2 Properties

Characterization

Bipartite graphs may be characterized in several different ways:

- A graph is bipartite if and only if it does not contain an odd cycle.[14]
- A graph is bipartite if and only if it is 2-colorable, (i.e. its chromatic number is less than or equal to 2).[3]
- The spectrum of a graph is symmetric if and only if it's a bipartite graph.[15]

König's theorem and perfect graphs

In bipartite graphs, the size of minimum vertex cover is equal to the size of the maximum matching; this is König's theorem.[16][17] An alternative and equivalent form of this theorem is that the size of the maximum independent set plus the size of the maximum matching is equal to the number of vertices. In any graph without isolated vertices the size of the minimum edge cover plus the size of a maximum matching equals the number of vertices.[18] Combining this equality with König's theorem leads to the facts that, in bipartite graphs, the size of the minimum edge cover is equal to the size of the maximum independent set, and the size of the minimum edge cover plus the size of the minimum vertex cover is equal to the number of vertices.

Another class of related results concerns perfect graphs: every bipartite graph, the complement of every bipartite graph, the line graph of every bipartite graph, and the complement of the line graph of every bipartite graph, are all perfect. Perfection of bipartite graphs is easy to see (their chromatic number is two and their maximum clique size is also two) but perfection of the complements of bipartite graphs is less trivial, and is another restatement of König's theorem. This was one of the results that motivated the initial definition of perfect graphs.[19] Perfection of the complements of line graphs of perfect graphs is yet another restatement of König's theorem, and perfection of the line graphs themselves is a restatement of an earlier theorem of König, that every bipartite graph has an edge coloring using a number of colors equal to its maximum degree.

According to the strong perfect graph theorem, the perfect graphs have a forbidden graph characterization resembling that of bipartite graphs: a graph is bipartite if and only if it has no odd cycle as a subgraph, and a graph is perfect if and only if it has no odd cycle or its complement as an induced subgraph. The bipartite graphs, line graphs of bipartite graphs, and their complements form four out of the five basic classes of perfect graphs used in the proof of the strong perfect graph theorem.[20]

Degree

For a vertex, the number of adjacent vertices is called the degree of the vertex and is denoted $\deg(v)$. The *degree sum formula* for a bipartite graph states that

$$\sum_{v \in V} \deg(v) = \sum_{u \in U} \deg(u) = |E| \,.$$

The degree sequence of a bipartite graph is the pair of lists each containing the degrees of the two parts U and V . For example, the complete bipartite graph $K_{3,5}$ has degree sequence $(5, 5, 5), (3, 3, 3, 3, 3)$. Isomorphic bipartite graphs have the same degree sequence. However, the degree sequence does not, in general, uniquely identify a bipartite graph; in some cases, non-isomorphic bipartite graphs may have the same degree sequence.

The bipartite realization problem is the problem of finding a simple bipartite graph with the degree sequence being two given lists of natural numbers. (Trailing zeros may be ignored since they are trivially realized by adding an appropriate number of isolated vertices to the digraph.)

Relation to hypergraphs and directed graphs

The biadjacency matrix of a bipartite graph (U, V, E) is a (0,1)-matrix of size $|U| \times |V|$ that has a one for each pair of adjacent vertices and a zero for nonadjacent vertices.[21] Biadjacency matrices may be used to describe equivalences between bipartite graphs, hypergraphs, and directed graphs.

A hypergraph is a combinatorial structure that, like an undirected graph, has vertices and edges, but in which the edges may be arbitrary sets of vertices rather than having to have exactly two endpoints. A bipartite graph (U, V, E) may be used to model a hypergraph in which U is the set of vertices of the hypergraph, V is the set of hyperedges, and E contains an edge from a hypergraph vertex v to a hypergraph edge e exactly when v is one of the endpoints of v . Under this correspondence, the biadjacency matrices of bipartite graphs are exactly the incidence matrices of the corresponding hypergraphs. As a special case of this correspondence between bipartite graphs and hypergraphs, any multigraph (a graph in which there may be two or more edges between the same two vertices) may be interpreted as a hypergraph in which some hyperedges have equal sets of endpoints, and represented by a bipartite graph that does not have multiple adjacencies and in which the vertices on one side of the bipartition all have degree two.[22]

A similar reinterpretation of adjacency matrices may be used to show a one-to-one correspondence between directed graphs (on a given number of labeled vertices, allowing self-loops) and balanced bipartite graphs, with the same number of vertices on both sides of the bipartition. For, the adjacency matrix of a directed graph with n vertices can be any $(0, 1)$ -matrix of size $n \times n$, which can then be reinterpreted as the adjacency matrix of a bipartite graph with n vertices on each side of its bipartition.[23] In this construction, the bipartite graph is the bipartite double cover of the directed graph.

5.1.3 Algorithms

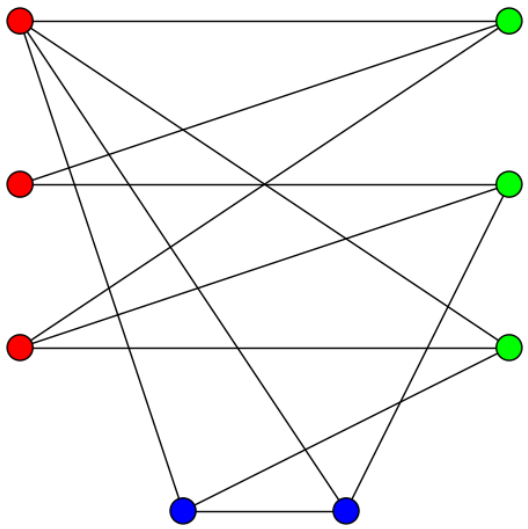
Testing bipartiteness

It is possible to test whether a graph is bipartite, and to return either a two-coloring (if it is bipartite) or an odd cycle (if it is not) in linear time, using depth-first search. The main idea is to assign to each vertex the color that differs from the color of its parent in the depth-first search tree, assigning colors in a preorder traversal of the depth-first-search tree. This will necessarily provide a two-coloring of the spanning tree consisting of the edges connecting vertices to their parents, but it may not properly color some of the non-tree edges. In a depth-first search tree, one of the two endpoints of every non-tree edge is an ancestor of the other endpoint, and when the depth first search discovers an edge of this type it should check that these two vertices have different colors. If they do not, then the path in the tree from ancestor to descendant, together with the miscolored edge, form an odd cycle, which is returned from the algorithm together with the result that the graph is not bipartite. However, if the algorithm terminates without detecting an odd cycle of this type, then every edge must be properly colored, and the algorithm returns the coloring together with the result that the graph is bipartite.[24]

Alternatively, a similar procedure may be used with breadth-first search in place of depth-first search. Again, each node is given the opposite color to its parent in the search tree, in breadth-first order. If, when a vertex is colored, there exists an edge connecting it to a previously-colored vertex with the same color, then this edge together with the paths in the breadth-first search tree connecting its two endpoints to their lowest common ancestor forms an odd cycle. If the algorithm terminates without finding an odd cycle in this way, then it must have found a proper coloring, and can safely conclude that the graph is bipartite.[25]

For the intersection graphs of n line segments or other simple shapes in the Euclidean plane, it is possible to test whether the graph is bipartite and return either a two-coloring or an odd cycle in time $O(n \log n)$, even though the graph itself may have as many as $\Omega(n^2)$ edges.[26]

Odd cycle transversal



A graph with an odd cycle transversal of size 2: removing the two blue bottom vertices leaves a bipartite graph.

Odd cycle transversal is an NP-complete algorithmic problem that asks, given a graph $G = (V,E)$ and a number k, whether there exists a set of k vertices whose removal from G would cause the resulting graph to be bipartite.[27] The problem is fixed-parameter tractable, meaning that there is an algorithm whose running time can be bounded by a polynomial function of the size of the graph multi-

plied by a larger function of k.[28] More specifically, the time for this algorithm is $O(3^k\ mn)$, although this was not stated in that paper.[29] The result by Reed et al. was obtained using a completely new method, which later was called iterative compression and turned out to be an extremely useful algorithmic tool, especially in the field of fixed-parameter tractability. This tool is now considered one of the fundamental tools for parameterized algorithmics.

The name *odd cycle transversal* comes from the fact that a graph is bipartite if and only if it has no odd cycles. Hence, to delete vertices from a graph in order to obtain a bipartite graph, one needs to "hit all odd cycle", or find a so-called odd cycle transversal set. In the illustration, one can observe that every odd cycle in the graph contains the blue (the bottommost) vertices, hence removing those vertices kills all odd cycles and leaves a bipartite graph.

The *edge bipartization* problem is the algorithmic problem of deleting as few edges as possible to make a graph bipartite and is also an important problem in graph modification algorithmics. This problem is also fixed-parameter tractable, and can be solved in time $O(2^k\ m^2)$,[30] where k is the number of edges to delete and m is the number of edges in the input graph.

Matching

A matching in a graph is a subset of its edges, no two of which share an endpoint. Polynomial time algorithms are known for many algorithmic problems on matchings, including maximum matching (finding a matching that uses as many edges as possible), maximum weight matching, and stable marriage.[31] In many cases, matching problems are simpler to solve on bipartite graphs than on non-bipartite graphs,[32] and many matching algorithms such as the Hopcroft–Karp algorithm for maximum cardinality matching[33] work correctly only on bipartite inputs.

As a simple example, suppose that a set P of people are all seeking jobs from among a set of J jobs, with not all people suitable for all jobs. This situation can be modeled as a bipartite graph (P, J, E) where an edge connects each job-seeker with each suitable job.[34] A perfect matching describes a way of simultaneously satisfying all job-seekers and filling all jobs; Hall's marriage theorem provides a characterization of the bipartite graphs which allow perfect matchings. The National Resident Matching Program applies graph matching methods to solve this problem for U.S. medical student job-seekers and hospital residency jobs.[35]

The Dulmage–Mendelsohn decomposition is a structural decomposition of bipartite graphs that is useful in finding maximum matchings.[36]

5.1.4 Additional applications

Bipartite graphs are extensively used in modern coding theory, especially to decode codewords received from the channel. Factor graphs and Tanner graphs are examples of this. A Tanner graph is a bipartite graph in which the vertices on one side of the bipartition represent digits of a codeword, and the vertices on the other side represent combinations of digits that are expected to sum to zero in a codeword without errors.[37] A factor graph is a closely related belief network used for probabilistic decoding of LDPC and turbo codes.[38]

In computer science, a Petri net is a mathematical modeling tool used in analysis and simulations of concurrent systems. A system is modeled as a bipartite directed graph with two sets of nodes: A set of "place" nodes that contain resources, and a set of "event" nodes which generate and/or consume resources. There are additional constraints on the nodes and edges that constrain the behavior of the system. Petri nets utilize the properties of bipartite directed graphs and other properties to allow mathematical proofs of the behavior of systems while also allowing easy implementation of simulations of the system.[39]

In projective geometry, Levi graphs are a form of bipartite graph used to model the incidences between points and lines in a configuration. Corresponding to the geometric property of points and lines that every two lines meet in at most one point and every two points be connected with a single line, Levi graphs necessarily do not contain any cycles of length four, so their girth must be six or more.[40]

5.1.5 See also

- Bipartite dimension, the minimum number of complete bipartite graphs whose union is the given graph
- Bipartite double cover, a way of transforming any graph into a bipartite graph by doubling its vertices
- Bipartite matroid, a class of matroids that includes the graphic matroids of bipartite graphs
- Bipartite network projection, a weighting technique for compressing information about bipartite networks
- Convex bipartite graph, a bipartite graph whose vertices can be ordered so that the vertex neighborhoods are contiguous
- Multipartite graph, a generalization of bipartite graphs to more than two subsets of vertices
- Parity graph, a generalization of bipartite graphs in which every two induced paths between the same two points have the same parity

- Quasi-bipartite graph, a type of Steiner tree problem instance in which the terminals form an independent set, allowing approximation algorithms that generalize those for bipartite graphs
- Split graph, a graph in which the vertices can be partitioned into two subsets, one of which is independent and the other of which is a clique
- Zarankiewicz problem on the maximum number of edges in a bipartite graph with forbidden subgraphs

5.1.6 References

[1] Diestel, Reinard (2005). *Graph Theory, Grad. Texts in Math*. Springer. ISBN 978-3-642-14278-9.

[2] Asratian, Armen S.; Denley, Tristan M. J.; Häggkvist, Roland (1998), *Bipartite Graphs and their Applications*, Cambridge Tracts in Mathematics, **131**, Cambridge University Press, ISBN 9780521593458.

[3] Asratian, Denley & Häggkvist (1998), p. 7.

[4] Scheinerman, Edward R. (2012), *Mathematics: A Discrete Introduction* (3rd ed.), Cengage Learning, p. 363, ISBN 9780840049421.

[5] Chartrand, Gary; Zhang, Ping (2008), *Chromatic Graph Theory*, Discrete Mathematics And Its Applications, **53**, CRC Press, p. 223, ISBN 9781584888000.

[6] Wasserman, Stanley; Faust, Katherine (1994), *Social Network Analysis: Methods and Applications*, Structural Analysis in the Social Sciences, **8**, Cambridge University Press, pp. 299–302, ISBN 9780521387071.

[7] Niedermeier, Rolf (2006). *Invitation to Fixed Parameter Algorithms (Oxford Lecture Series in Mathematics and Its Applications)*. Oxford. pp. 20–21. ISBN 978-0-19-856607-6.

[8] Bracey, Robert (2012). "On the Graphical Interpreation of Herod's Coinage in Judaea and Rome in Coins". pp. 65–84.

[9] Soifer, Alexander (2008), *The Mathematical Coloring Book*, Springer-Verlag, pp. 136–137, ISBN 978-0-387-74640-1. This result has sometimes been called the "two color theorem"; Soifer credits it to a famous 1879 paper of Alfred Kempe containing a false proof of the four color theorem.

[10] Bandelt, H.-J.; Chepoi, V.; Eppstein, D. (2010), "Combinatorics and geometry of finite and infinite squaregraphs", *SIAM Journal on Discrete Mathematics*, **24** (4): 1399–1440, arXiv:0905.4537, doi:10.1137/090760301.

[11] Asratian, Denley & Häggkvist (1998), p. 11.

[12] Archdeacon, D.; Debowsky, M.; Dinitz, J.; Gavlas, H. (2004), "Cycle systems in the complete bipartite graph minus a one-factor", *Discrete Mathematics*, **284** (1–3): 37–43, doi:10.1016/j.disc.2003.11.021.

[13] Ovchinnikov, Sergei (2011), *Graphs and Cubes*, Universitext, Springer. See especially Chapter 5, "Partial Cubes", pp. 127–181.

[14] Asratian, Denley & Häggkvist (1998), Theorem 2.1.3, p. 8. Asratian et al. attribute this characterization to a 1916 paper by Dénes Kőnig. For infinite graphs, this result requires the axiom of choice.

[15] Biggs, Norman (1994), *Algebraic Graph Theory*, Cambridge Mathematical Library (2nd ed.), Cambridge University Press, p. 53, ISBN 9780521458979.

[16] Kőnig, Dénes (1931). "Gráfok és mátrixok". *Matematikai és Fizikai Lapok*. **38**: 116–119..

[17] Gross, Jonathan L.; Yellen, Jay (2005), *Graph Theory and Its Applications*, Discrete Mathematics And Its Applications (2nd ed.), CRC Press, p. 568, ISBN 9781584885054.

[18] Chartrand, Gary; Zhang, Ping (2012), *A First Course in Graph Theory*, Courier Dover Publications, pp. 189–190, ISBN 9780486483689.

[19] Béla Bollobás (1998), *Modern Graph Theory*, Graduate Texts in Mathematics, **184**, Springer, p. 165, ISBN 9780387984889.

[20] Chudnovsky, Maria; Robertson, Neil; Seymour, Paul; Thomas, Robin (2006), "The strong perfect graph theorem", *Annals of Mathematics*, **164** (1): 51–229, doi:10.4007/annals.2006.164.51.

[21] Asratian, Denley & Häggkvist (1998), p. 17.

[22] A. A. Sapozhenko (2001), "Hypergraph", in Hazewinkel, Michiel, *Encyclopedia of Mathematics*, Springer, ISBN 978-1-55608-010-4

[23] Brualdi, Richard A.; Harary, Frank; Miller, Zevi (1980), "Bigraphs versus digraphs via matrices", *Journal of Graph Theory*, **4** (1): 51–73, doi:10.1002/jgt.3190040107, MR 558453. Brualdi et al. credit the idea for this equivalence to Dulmage, A. L.; Mendelsohn, N. S. (1958), "Coverings of bipartite graphs", *Canadian Journal of Mathematics*, **10**: 517–534, doi:10.4153/CJM-1958-052-0, MR 0097069.

[24] Sedgewick, Robert (2004), *Algorithms in Java, Part 5: Graph Algorithms* (3rd ed.), Addison Wesley, pp. 109–111.

[25] Kleinberg, Jon; Tardos, Éva (2006), *Algorithm Design*, Addison Wesley, pp. 94–97.

[26] Eppstein, David (2009), "Testing bipartiteness of geometric intersection graphs", *ACM Transactions on Algorithms*, **5** (2): Art. 15, arXiv:cs.CG/0307023, doi:10.1145/1497290.1497291, MR 2561751.

[27] Yannakakis, Mihalis (1978), "Node-and edge-deletion NP-complete problems", *Proceedings of the 10th ACM Symposium on Theory of Computing (STOC '78)*, pp. 253–264, doi:10.1145/800133.804355

[28] Reed, Bruce; Smith, Kaleigh; Vetta, Adrian (2004), "Finding odd cycle transversals", *Operations Research Letters*, **32** (4): 299–301, doi:10.1016/j.orl.2003.10.009, MR 2057781.

[29] Hüffner, Falk (2005), "Algorithm engineering for optimal graph bipartization", *Experimental and Efficient Algorithms*: 240–252, doi:10.1007/11427186_22.

[30] Guo, Jiong; Gramm, Jens; Hüffner, Falk; Niedermeier, Rolf; Wernicke, Sebastian (2006), "Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization", *JCSS*: 1386–1396, doi:10.1016/j.jcss.2006.02.001

[31] Ahuja, Ravindra K.; Magnanti, Thomas L.; Orlin, James B. (1993), "12. Assignments and Matchings", *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, pp. 461–509.

[32] Ahuja, Magnanti & Orlin (1993), p. 463: "Nonbipartite matching problems are more difficult to solve because they do not reduce to standard network flow problems."

[33] Hopcroft, John E.; Karp, Richard M. (1973), "An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs", *SIAM Journal on Computing*, **2** (4): 225–231, doi:10.1137/0202019.

[34] Ahuja, Magnanti & Orlin (1993), Application 12.1 Bipartite Personnel Assignment, pp. 463–464.

[35] Robinson, Sara (April 2003), "Are Medical Students Meeting Their (Best Possible) Match?" (PDF), *SIAM News* (3): 36.

[36] Dulmage & Mendelsohn (1958).

[37] Moon, Todd K. (2005), *Error Correction Coding: Mathematical Methods and Algorithms*, John Wiley & Sons, p. 638, ISBN 9780471648000.

[38] Moon (2005), p. 686.

[39] Cassandras, Christos G.; Lafortune, Stephane (2007), *Introduction to Discrete Event Systems* (2nd ed.), Springer, p. 224, ISBN 9780387333328.

[40] Grünbaum, Branko (2009), *Configurations of Points and Lines*, Graduate Studies in Mathematics, **103**, American Mathematical Society, p. 28, ISBN 9780821843086.

5.1.7 External links

- Hazewinkel, Michiel, ed. (2001), "Graph, bipartite", *Encyclopedia of Mathematics*, Springer, ISBN 978-1-55608-010-4
- Information System on Graph Classes and their Inclusions: bipartite graph
- Weisstein, Eric W. "Bipartite Graph". *MathWorld*.

5.2 Complete bipartite graph

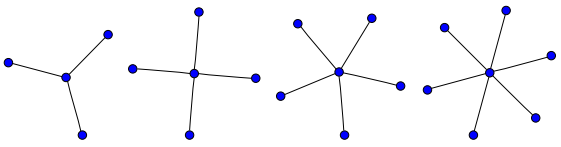
In the mathematical field of graph theory, a **complete bipartite graph** or **biclique** is a special kind of bipartite graph where every vertex of the first set is connected to every vertex of the second set.[1][2]

Graph theory itself is typically dated as beginning with Leonhard Euler's 1736 work on the Seven Bridges of Königsberg. However, drawings of complete bipartite graphs were already printed as early as 1669, in connection with an edition of the works of Ramon Llull edited by Athanasius Kircher.[3][4] Llull himself had made similar drawings of complete graphs three centuries earlier.[3]

5.2.1 Definition

A **complete bipartite graph** is a graph whose vertices can be partitioned into two subsets V_1 and V_2 such that no edge has both endpoints in the same subset, and every possible edge that could connect vertices in different subsets is part of the graph. That is, it is a bipartite graph (V_1, V_2, E) such that for every two vertices $v_1 \in V_1$ and $v_2 \in V_2$, v_1v_2 is an edge in E. A complete bipartite graph with partitions of size $|V_1|=m$ and $|V_2|=n$, is denoted $K_{m,n}$;[1][2] every two graphs with the same notation are isomorphic.

5.2.2 Examples



The star graphs K*1,3,* K*1,4,* K*1,5, and* K*1,6.*

- For any k, $K_{1,k}$ is called a star.[2] All complete bipartite graphs which are trees are stars.
- The graph $K_{1,3}$ is called a claw, and is used to define the claw-free graphs.[5]
- The graph $K_{3,3}$ is called the utility graph. This usage comes from a standard mathematical puzzle in which three utilities must each be connected to three buildings; it is impossible to solve without crossings due to the nonplanarity of $K_{3,3}$.[6]

5.2.3 Properties

- Given a bipartite graph, testing whether it contains a complete bipartite subgraph Ki,i for a parameter i is an NP-complete problem.[8]
- A planar graph cannot contain $K_{3,3}$ as a minor; an outerplanar graph cannot contain $K_{3,2}$ as a minor

(These are not sufficient conditions for planarity and outerplanarity, but necessary). Conversely, every nonplanar graph contains either $K_{3,3}$ or the complete graph K_5 as a minor; this is Wagner's theorem.[9]

- Every complete bipartite graph. $\mathbf{K}n,n$ is a Moore graph and a $(n,4)$-cage.[10]
- The complete bipartite graphs $\mathbf{K}n,n$ and $\mathbf{K}n,n_{+1}$ have the maximum possible number of edges among all triangle-free graphs with the same number of vertices; this is Mantel's theorem. Mantel's result was generalized to k-partite graphs and graphs that avoid larger cliques as subgraphs in Turán's theorem, and these two complete bipartite graphs are examples of Turán graphs, the extremal graphs for this more general problem.[11]
- The complete bipartite graph $\mathbf{K}m,n$ has a vertex covering number of $\mathbf{min}\{m,n\}$ and an edge covering number of $\mathbf{max}\{m,n\}$.
- The complete bipartite graph $\mathbf{K}m,n$ has a maximum independent set of size $\mathbf{max}\{m,n\}$.
- The adjacency matrix of a complete bipartite graph $\mathbf{K}m,n$ has eigenvalues $\sqrt{(nm)}$, $-\sqrt{(nm)}$ and 0; with multiplicity 1, 1 and $n+m-2$ respectively.[12]
- The Laplacian matrix of a complete bipartite graph $\mathbf{K}m,n$ has eigenvalues $n+m$, n, m, and 0; with multiplicity 1, $m-1$, $n-1$ and 1 respectively.
- A complete bipartite graph $\mathbf{K}m,n$ has m^{n-1} n^{m-1} spanning trees.[13]
- A complete bipartite graph $\mathbf{K}m,n$ has a maximum matching of size $\mathbf{min}\{m,n\}$.
- A complete bipartite graph $\mathbf{K}n,n$ has a proper n-edge-coloring corresponding to a Latin square.[14]
- Every complete bipartite graph is a modular graph: every triple of vertices has a median that belongs to shortest paths between each pair of vertices.[15]

5.2.4 See also

- Biclique-free graph, a class of sparse graphs defined by avoidance of complete bipartite subgraphs
- Crown graph, a graph formed by removing a perfect matching from a complete bipartite graph
- Complete multipartite graph, a generalization of complete bipartite graphs to more than two sets of vertices

5.2.5 References

[1] Bondy, John Adrian; Murty, U. S. R. (1976), *Graph Theory with Applications*, North-Holland, p. 5, ISBN 0-444-19451-7.

[2] Diestel, Reinhard (2005), *Graph Theory* (3rd ed.), Springer, ISBN 3-540-26182-6. Electronic edition, page 17.

[3] Knuth, Donald E. (2013), "Two thousand years of combinatorics", in Wilson, Robin; Watkins, John J., *Combinatorics: Ancient and Modern*, Oxford University Press, pp. 7–37.

[4] Read, Ronald C.; Wilson, Robin J. (1998), *An Atlas of Graphs*, Clarendon Press, p. ii, ISBN 9780198532897.

[5] Lovász, László; Plummer, Michael D. (2009), *Matching theory*, AMS Chelsea Publishing, Providence, RI, p. 109, ISBN 978-0-8218-4759-6, MR 2536865. Corrected reprint of the 1986 original.

[6] Gries, David; Schneider, Fred B. (1993), *A Logical Approach to Discrete Math*, Springer, p. 437, ISBN 9780387941158.

[7] Coxeter, *Regular Complex Polytopes*, second edition, p.114

[8] Garey, Michael R.; Johnson, David S. (1979), "[GT24] Balanced complete bipartite subgraph", *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, p. 196, ISBN 0-7167-1045-5.

[9] Diestel, elect. ed. p. 105.

[10] Biggs, Norman (1993), *Algebraic Graph Theory*, Cambridge University Press, p. 181, ISBN 9780521458979.

[11] Bollobás, Béla (1998), *Modern Graph Theory*, Graduate Texts in Mathematics, **184**, Springer, p. 104, ISBN 9780387984889.

[12] Bollobás (1998), p. 266.

[13] Jungnickel, Dieter (2012), *Graphs, Networks and Algorithms*, Algorithms and Computation in Mathematic, **5**, Springer, p. 557, ISBN 9783642322785.

[14] Jensen, Tommy R.; Toft, Bjarne (2011), *Graph Coloring Problems*, Wiley Series in Discrete Mathematics and Optimization, **39**, John Wiley & Sons, p. 16, ISBN 9781118030745.

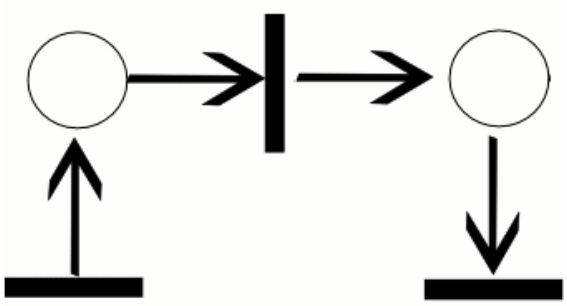
[15] Bandelt, H.-J.; Dählmann, A.; Schütte, H. (1987), "Absolute retracts of bipartite graphs", *Discrete Applied Mathematics*, **16** (3): 191–215, doi:10.1016/0166-218X(87)90058-8, MR 878021.

5.3 Petri net

A **Petri net** (also known as a **place/transition net** or **P/T net**) is one of several mathematical modeling languages for the description of distributed systems. A Petri net is

a directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur, represented by bars) and places (i.e. conditions, represented by circles). The directed arcs describe which places are pre- and/or postconditions for which transitions (signified by arrows). Some sources[1] state that Petri nets were invented in August 1939 by Carl Adam Petri—at the age of 13—for the purpose of describing chemical processes.

Like industry standards such as UML activity diagrams, Business Process Model and Notation and EPCs, Petri nets offer a graphical notation for stepwise processes that include choice, iteration, and concurrent execution. Unlike these standards, Petri nets have an exact mathematical definition of their execution semantics, with a well-developed mathematical theory for process analysis.



(a) Petri net trajectory example

5.3.1 Petri net basics

A Petri net consists of *places*, *transitions*, and *arcs*. Arcs run from a place to a transition or vice versa, never between places or between transitions. The places from which an arc runs to a transition are called the *input places* of the transition; the places to which arcs run from a transition are called the *output places* of the transition.

Graphically, places in a Petri net may contain a discrete number of marks called *tokens*. Any distribution of tokens over the places will represent a configuration of the net called a *marking*. In an abstract sense relating to a Petri net diagram, a transition of a Petri net may *fire* if it is *enabled*, i.e. there are sufficient tokens in all of its input places; when the transition fires, it consumes the required input tokens, and creates tokens in its output places. A firing is atomic, i.e. a single non-interruptible step.

Unless an *execution policy* is defined, the execution of Petri nets is nondeterministic: when multiple transitions are enabled at the same time, any one of them may fire.

Since firing is nondeterministic, and multiple tokens may be present anywhere in the net (even in the same place), Petri nets are well suited for modeling the concurrent behavior of distributed systems.

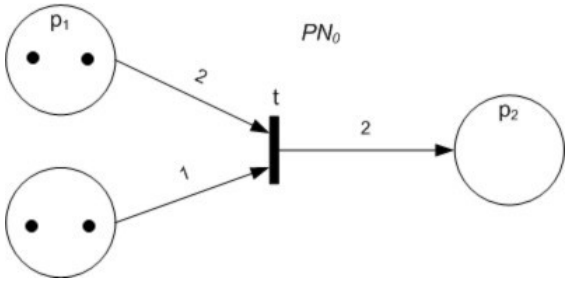
5.3.2 Formal definition and basic terminology

Petri nets are state-transition systems that extend a class of nets called elementary nets.[2]

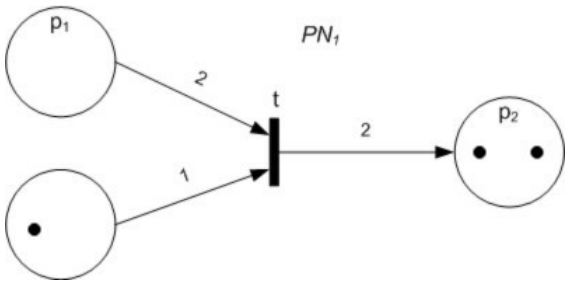
Definition 1. A *net* is a triple $N = (P, T, F)$ where:

1. P and T are *disjoint* finite sets of *places* and *transitions*, respectively.
2. $F \subset (P \times T) \cup (T \times P)$ is a set of *arcs* (or flow relations).

Definition 2. Given a net $N = (P, T, F)$, a *configuration* is a set C so that $C \subseteq P$.



A Petri net with an enabled transition.



The Petri net that follows after the transition fires (Initial Petri net in the figure above).

Definition 3. An *elementary net* is a net of the form $EN = (N, C)$ where:

1. $N = (P, T, F)$ is a net.
2. C is such that $C \subseteq P$ is a *configuration*.

Definition 4. A *Petri net* is a net of the form $PN = (N, M, W)$, which extends the elementary net so that:

1. $N = (P, T, F)$ is a net.
2. $M : P \to Z$ is a place multiset, where Z is a countable set. M extends the concept of *configuration* and is commonly described with reference to Petri net diagrams as a *marking*.

3. $W : F \to Z$ is an arc multiset, so that the count (or weight) for each arc is a measure of the arc *multiplicity*.

If a Petri net is equivalent to an elementary net, then Z can be the countable set {0,1} and those elements in P that map to 1 under M form a configuration. Similarly, if a Petri net is not an elementary net, then the multiset M can be interpreted as representing a non-singleton set of configurations. In this respect, M extends the concept of configuration for elementary nets to Petri nets.

In the diagram of a Petri net (see top figure right), places are conventionally depicted with circles, transitions with long narrow rectangles and arcs as one-way arrows that show connections of places to transitions or transitions to places. If the diagram were of an elementary net, then those places in a configuration would be conventionally depicted as circles, where each circle encompasses a single dot called a *token*. In the given diagram of a Petri net (see right), the place circles may encompass more than one token to show the number of times a place appears in a configuration. The configuration of tokens distributed over an entire Petri net diagram is called a *marking*.

In the top figure (see right), the place p_1 is an input place of transition t; whereas, the place p_2 is an output place to the same transition. Let PN_0 (Fig. top) be a Petri net with a marking configured M_0 and PN_1 (Fig. bottom) be a Petri net with a marking configured M_1. The configuration of PN_0 *enable* transition t through the property that all input places have sufficient number of tokens (shown in the figures as dots) "equal to or greater" than the multiplicities on their respective arcs to t. Once and only once a transition is enabled will the transition fire. In this example, the *firing* of transition t generates a map that has the marking configured M_1 in the image of M_0 and results in Petri net PN_1, seen in the bottom figure. In the diagram, the firing rule for a transition can be characterised by subtracting a number of tokens from its input places equal to the multiplicity of the respective input arcs and accumulating a new number of tokens at the output places equal to the multiplicity of the respective output arcs.

Remark 1. The precise meaning of "equal to or greater" will depend on the precise algebraic properties of addition being applied on Z in the firing rule, where subtle variations on the algebraic properties can lead to other classes of Petri nets; for example, Algebraic Petri nets.[3]

The following formal definition is loosely based on (Peterson 1981). Many alternative definitions exist.

Syntax

A **Petri net graph** (called *Petri net* by some, but see below) is a 3-tuple (S, T, W), where

- S is a finite set of *places*
- T is a finite set of *transitions*
- S and T are disjoint, i.e. no object can be both a place and a transition
- $W : (S \times T) \cup (T \times S) \to \mathbb{N}$ is a multiset of arcs, i.e. it assigns to each arc a non-negative integer *arc multiplicity* (or weight); note that no arc may connect two places or two transitions.

The *flow relation* is the set of arcs: $F = \{(x, y) \mid W(x, y) > 0\}$. In many textbooks, arcs can only have multiplicity 1. These texts often define Petri nets using F instead of W. When using this convention, a Petri net graph is a bipartite multigraph $(S \cup T, F)$ with node partitions S and T.

The *preset* of a transition t is the set of its *input places*: ${}^\bullet t = \{s \in S \mid W(s, t) > 0\}$; its *postset* is the set of its *output places*: $t^\bullet = \{s \in S \mid W(t, s) > 0\}$. Definitions of pre- and postsets of places are analogous.

A *marking* of a Petri net (graph) is a multiset of its places, i.e., a mapping $M : S \to \mathbb{N}$. We say the marking assigns to each place a number of *tokens*.

A **Petri net** (called *marked Petri net* by some, see above) is a 4-tuple (S, T, W, M_0), where

- (S, T, W) is a Petri net graph;
- M_0 is the *initial marking*, a marking of the Petri net graph.

Execution semantics

In words:

- firing a transition t in a marking M consumes $W(s, t)$ tokens from each of its input places s, and produces $W(t, s)$ tokens in each of its output places s
- a transition is *enabled* (it may *fire*) in M if there are enough tokens in its input places for the consumptions to be possible, i.e. iff $\forall s : M(s) \geq W(s, t)$.

We are generally interested in what may happen when transitions may continually fire in arbitrary order.

We say that a marking M' *is reachable from* a marking M *in one step* if $M \xrightarrow[G]{} M'$; we say that it *is reachable from M* if $M \xrightarrow[G]{*} M'$, where $\xrightarrow[G]{*}$ is the reflexive transitive closure of $\xrightarrow[G]{}$; that is, if it is reachable in 0 or more steps.

For a (marked) Petri net $N = (S, T, W, M_0)$, we are interested in the firings that can be performed starting with the initial marking M_0 . Its set of *reachable markings* is the set $R(N) \stackrel{D}{=} \left\{ M' \middle| M_0 \xrightarrow[(S,T,W)]{*} M' \right\}$

The *reachability graph* of N is the transition relation $\xrightarrow[G]{}$ restricted to its reachable markings $R(N)$. It is the state space of the net.

A *firing sequence* for a Petri net with graph G and initial marking M_0 is a sequence of transitions $\vec{\sigma} = \langle t_{i_1} \dots t_{i_n} \rangle$ such that $M_0 \xrightarrow[G,t_{i_1}]{} M_1 \wedge \dots \wedge M_{n-1} \xrightarrow[G,t_{i_n}]{} M_n$. The set of firing sequences is denoted as $L(N)$.

5.3.3 Variations on the definition

As already remarked, a common variation is to disallow arc multiplicities and replace the bag of arcs *W* with a simple set, called the *flow relation*, $F \subseteq (S \times T) \cup (T \times S)$. This doesn't limit expressive power as both can represent each other.

Another common variation, e.g. in, Desel and Juhás (2001),[4] is to allow *capacities* to be defined on places. This is discussed under *extensions* below.

5.3.4 Formulation in terms of vectors and matrices

The markings of a Petri net (S, T, W, M_0) can be regarded as vectors of nonnegative integers of length $|S|$.

Its transition relation can be described as a pair of $|S|$ by $|T|$ matrices:

- W^- , defined by $\forall s, t : W^-[s, t] = W(s, t)$
- W^+ , defined by $\forall s, t : W^+[s, t] = W(t, s)$.

Then their difference

- $W^T = W^+ - W^-$

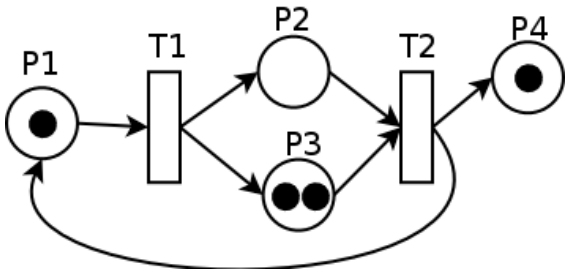
can be used to describe the reachable markings in terms of matrix multiplication, as follows. For any sequence of transitions w, write $o(w)$ for the vector that maps every transition to its number of occurrences in w. Then, we have

- $R(N) = \{M \mid \exists w : M = M_0 + W^T \cdot o(w) \wedge w$ of sequence firing a is $N\}$.

Note that it must be required that w is a firing sequence; allowing arbitrary sequences of transitions will generally produce a larger set.

$$W^+ = \begin{bmatrix} * & t1 & t2 \\ p1 & 0 & 1 \\ p2 & 1 & 0 \\ p3 & 1 & 0 \\ p4 & 0 & 1 \end{bmatrix}, W^- = \begin{bmatrix} * & t1 & t2 \\ p1 & 1 & 0 \\ p2 & 0 & 1 \\ p3 & 0 & 1 \\ p4 & 0 & 0 \end{bmatrix}, W^T = \begin{bmatrix} * & t1 & t2 \\ p1 & -1 & 1 \\ p2 & 1 & -1 \\ p3 & 1 & -1 \\ p4 & 0 & 1 \end{bmatrix}$$

$$M_0 = \begin{bmatrix} 1 & 0 & 2 & 1 \end{bmatrix}$$



(b) Petri net Example

5.3.5 Mathematical properties of Petri nets

One thing that makes Petri nets interesting is that they provide a balance between modeling power and analyzability: many things one would like to know about concurrent systems can be automatically determined for Petri nets, although some of those things are very expensive to determine in the general case. Several subclasses of Petri nets have been studied that can still model interesting classes of concurrent systems, while these problems become easier.

An overview of such decision problems, with decidability and complexity results for Petri nets and some subclasses, can be found in Esparza and Nielsen (1995).[5]

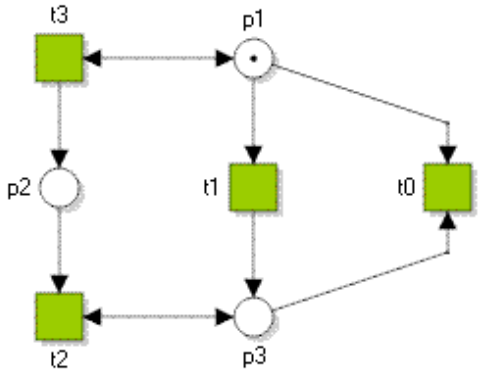
Reachability

The reachability problem for Petri nets is to decide, given a Petri net *N* and a marking *M*, whether $M \in R(N)$.

Clearly, this is a matter of walking the reachability graph defined above, until either we reach the requested marking or we know it can no longer be found. This is harder than it may seem at first: the reachability graph is generally infinite, and it is not easy to determine when it is safe to stop.

In fact, this problem was shown to be EXPSPACE-hard[6] years before it was shown to be decidable at all (Mayr, 1981). Papers continue to be published on how to do it efficiently.[7]

While reachability seems to be a good tool to find erroneous states, for practical problems the constructed graph usually has far too many states to calculate. To alleviate this problem, linear temporal logic is usually used in conjunction with the tableau method to prove that such states cannot be reached. LTL uses the semi-decision technique to find if indeed a state can be reached, by finding a set of necessary conditions for the state to be reached then proving that those conditions cannot be satisfied.



A Petri net in which transition t_0 is dead, while for all $j > 0$, t_j is L_j -live

Liveness

Petri nets can be described as having different degrees of liveness $L_1 - L_4$. A Petri net (N, M_0) is called L_k -live iff all of its transitions are L_k -live, where a transition is

- *dead*, if it can never fire, i.e. it is not in any firing sequence in $L(N, M_0)$
- L_1 -live (*potentially fireable*), iff it may fire, i.e. it is in some firing sequence in $L(N, M_0)$
- L_2 -live iff it can fire arbitrarily often, i.e. if for every positive integer k, it occurs at least k times in some firing sequence in $L(N, M_0)$
- L_3 -live iff it can fire infinitely often, i.e. if for every positive integer k, it occurs at least k times in V, for some prefix-closed set of firing sequences $V \subseteq L(N, M_0)$
- L_4 -live (*live*) iff it may always fire, i.e., it is L_1 -live in every reachable marking in $R(N, M_0)$

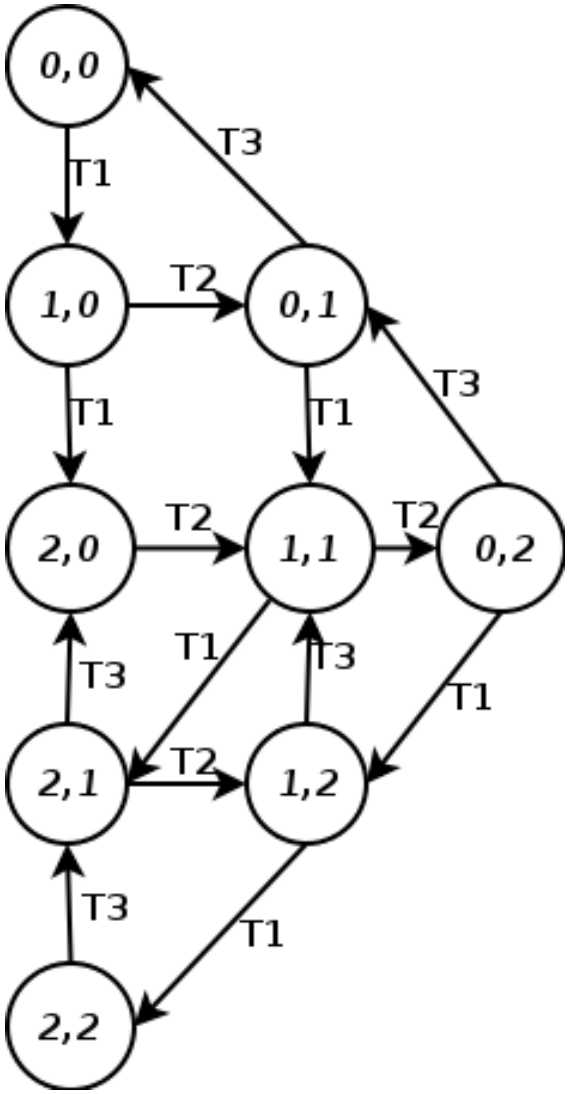
Note that these are increasingly stringent requirements: L_{j+1} -liveness implies L_j -liveness, for $j \in 1, 2, 3$.

These definitions are in accordance with Murata's overview,[8] which additionally uses L_0 *-live* as a term for *dead*.

Boundedness

A place in a Petri net is called *k-bounded* if it does not contain more than *k* tokens in all reachable markings, including the initial marking; it is said to be *safe* if it is 1-bounded; it is *bounded* if it is *k-bounded* for some *k*.

A (marked) Petri net is called *k*-bounded, *safe*, or *bounded* when all of its places are. A Petri net (graph) is called *(structurally) bounded* if it is bounded for every possible initial marking.



The reachability graph of N2.

Note that a Petri net is bounded if and only if its reachability graph is finite.

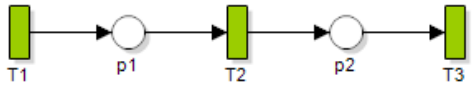
Boundedness is decidable by looking at covering, by constructing the Karp–Miller Tree.

It can be useful to explicitly impose a bound on places in a given net. This can be used to model limited system resources.

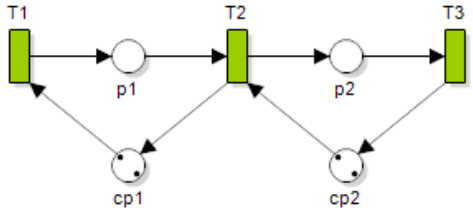
Some definitions of Petri nets explicitly allow this as a syntactic feature.[9] Formally, *Petri nets with place capacities* can be defined as tuples (S, T, W, C, M_0) , where (S, T, W, M_0) is a Petri net, $C : P \nrightarrow IN$ an assignment of capacities to (some or all) places, and the transition relation is the usual one restricted to the markings in which each place with a capacity has at most that many tokens.

For example, if in the net *N*, both places are assigned capacity 2, we obtain a Petri net with place capacities, say *N2*; its reachability graph is displayed on the right.

Alternatively, places can be made bounded by extending



An unbounded Petri net, N.



A two-bounded Petri net, obtained by extending N *with "counter-places".*

the net. To be exact, a place can be made k-bounded by adding a "counter-place" with flow opposite to that of the place, and adding tokens to make the total in both places k.

5.3.6 Discrete, continuous, and hybrid Petri nets

As well as for discrete events, there are Petri nets for continuous and hybrid discrete-continuous processes that are useful in discrete, continuous and hybrid control theory,[10] and related to discrete, continuous and hybrid automata.

5.3.7 Extensions

There are many extensions to Petri nets. Some of them are completely backwards-compatible (e.g. coloured Petri nets) with the original Petri net, some add properties that cannot be modelled in the original Petri net formalism (e.g. timed Petri nets). Although backwards-compatible models do not extend the computational power of Petri nets, they may have more succinct representations and may be more convenient for modeling.[11] Extensions that cannot be transformed into Petri nets are sometimes very powerful, but usually lack the full range of mathematical tools available to analyse ordinary Petri nets.

The term high-level Petri net is used for many Petri net formalisms that extend the basic P/T net formalism; this includes coloured Petri nets, hierarchical Petri nets such as Nets within Nets, and all other extensions sketched in this section. The term is also used specifically for the type of coloured nets supported by CPN Tools.

A short list of possible extensions:

- Additional types of arcs; two common types are:
  - a *reset arc* does not impose a precondition on firing, and empties the place when the transition fires; this makes reachability undecidable,[12] while some other properties, such as termination, remain decidable;[13]
  - an *inhibitor arc* imposes the precondition that the transition may only fire when the place is empty; this allows arbitrary computations on numbers of tokens to be expressed, which makes the formalism Turing complete and implies existence of a universal net.[14]
- In a standard Petri net, tokens are indistinguishable. In a Coloured Petri net, every token has a value.[15] In popular tools for coloured Petri nets such as CPN Tools, the values of tokens are typed, and can be tested (using *guard* expressions) and manipulated with a functional programming language. A subsidiary of coloured Petri nets are the well-formed Petri nets, where the arc and guard expressions are restricted to make it easier to analyse the net.
- Another popular extension of Petri nets is hierarchy; this in the form of different views supporting levels of refinement and abstraction was studied by Fehling. Another form of hierarchy is found in so-called object Petri nets or object systems where a Petri net can contain Petri nets as its tokens inducing a hierarchy of nested Petri nets that communicate by synchronisation of transitions on different levels. See[16] for an informal introduction to object Petri nets.
- A vector addition system with states (VASS) is an equivalent formalism to Petri nets. However, it can be superficially viewed as a generalisation of Petri nets. Consider a finite state automaton where each transition is labelled by a transition from the Petri net. The Petri net is then synchronised with the finite state automaton, i.e., a transition in the automaton is taken at the same time as the corresponding transition in the Petri net. It is only possible to take a transition in the automaton if the corresponding transition in the Petri net is enabled, and it is only possible to fire a transition in the Petri net if there is a transition from the current state in the automaton labelled by it. (The definition of VASS is usually formulated slightly differently.)
- Prioritised Petri nets add priorities to transitions, whereby a transition cannot fire, if a higher-priority transition is enabled (i.e. can fire). Thus, transitions are in priority groups, and e.g. priority group 3 can only fire if all transitions are disabled in groups 1 and 2. Within a priority group, firing is *still* non-deterministic.
- The non-deterministic property has been a very valuable one, as it lets the user abstract a large

number of properties (depending on what the net is used for). In certain cases, however, the need arises to also model the timing, not only the structure of a model. For these cases, timed Petri nets have evolved, where there are transitions that are timed, and possibly transitions which are not timed (if there are, transitions that are not timed have a higher priority than timed ones). A subsidiary of timed Petri nets are the stochastic Petri nets that add nondeterministic time through adjustable randomness of the transitions. The exponential random distribution is usually used to 'time' these nets. In this case, the nets' reachability graph can be used as a continuous time Markov chain (CTMC).

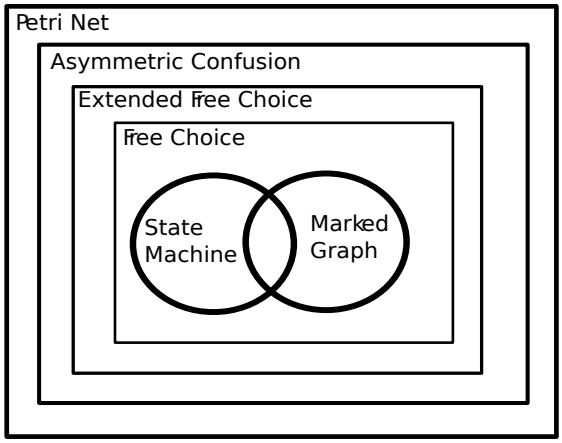
- Dualistic Petri Nets (dP-Nets) is a Petri Net extension developed by E. Dawis, et al.[17] to better represent real-world process. dP-Nets balance the duality of change/no-change, action/passivity, (transformation) time/space, etc., between the bipartite Petri Net constructs of transformation and place resulting in the unique characteristic of *transformation marking*, i.e., when the transformation is "working" it is marked. This allows for the transformation to fire (or be marked) multiple times representing the real-world behavior of process throughput. Marking of the transformation assumes that transformation time must be greater than zero. A zero transformation time used in many typical Petri Nets may be mathematically appealing but impractical in representing real-world processes. dP-Nets also exploit the power of Petri Nets' hierarchical abstraction to depict Process architecture. Complex process systems are modeled as a series of simpler nets interconnected through various levels of hierarchical abstraction. The process architecture of a packet switch is demonstrated in,[18] where development requirements are organized around the structure of the designed system.

There are many more extensions to Petri nets, however, it is important to keep in mind, that as the complexity of the net increases in terms of extended properties, the harder it is to use standard tools to evaluate certain properties of the net. For this reason, it is a good idea to use the most simple net type possible for a given modelling task.

5.3.8 Restrictions

Instead of extending the Petri net formalism, we can also look at restricting it, and look at particular types of Petri nets, obtained by restricting the syntax in a particular way. Ordinary Petri nets are the nets where all arc weights are 1. Restricting further, the following types of ordinary Petri nets are commonly used and studied:

1. In a state machine (SM), every transition has one incoming arc, and one outgoing arc, and all markings have exactly one token. As a consequence, there can *not* be *concurrency*, but there can be *conflict* (i.e. nondeterminism). Mathematically: $\forall t \in T : |t^\bullet| = |^\bullet t| = 1$



Petri net types graphically

2. In a marked graph (MG), every place has one incoming arc, and one outgoing arc. This means, that there can *not* be *conflict*, but there can be *concurrency*. Mathematically: $\forall s \in S : |s^\bullet| = |^\bullet s| = 1$

3. In a *free choice* net (FC), - every arc from a place to a transition is either the only arc from that place or the only arc to that transition. I.e. there can be *both concurrency and conflict, but not at the same time*. Mathematically: $\forall s \in S : (|s^\bullet| \leq 1) \vee (^\bullet(s^\bullet) = \{s\})$

4. Extended free choice (EFC) - a Petri net that can be *transformed into an FC*.

5. In an *asymmetric choice* net (AC), concurrency and conflict (in sum, *confusion*) may occur, but *not symmetrically*. Mathematically: $\forall s_1, s_2 \in S : (s_1{}^\bullet \cap s_2{}^\bullet \neq \emptyset) \rightarrow [(s_1{}^\bullet \subseteq s_2{}^\bullet) \vee (s_2{}^\bullet \subseteq s_1{}^\bullet)]$

5.3.9 Workflow nets

Workflow nets (WF-nets) are a subclass of Petri nets intending to model the workflow of process activities.[19] The WF-net transitions are assigned to tasks or activities, and places are assigned to the pre/post conditions. The WF-nets have additional structural and operational requirements, mainly the addition of a single input (source) place with no previous transitions, and output place (sink) with no following transitions. Accordingly start and termination markings can be defined that represent the process status.

WF-nets have the **soundness** property,[19] indicating that a process with a start marking of k tokens in its source place, can reach the termination state marking with k tokens in its sink place (defined as K-sound WF-net). Ad-

ditionally, all the transitions in the process could fire (i.e., for each transition there is a reachable state in which the transition is enabled). A general sound (G-sound) WF-net is defined as being K-sound for every k>0.[20]

A directed **path** in the Petri net is defined as the sequence of nodes (places and transitions) linked by the directed arcs. An **elementary path** includes every node in the sequence only once.

A **Well-handled** Petri net is a net in which there are no fully distinct elementary paths between a place and a transition (or transition and a place), i.e., if there are two paths between the pair of node then these paths share a node. An acyclic well-handled WF-net is sound (G-sound).[21]

Extended WF-net is a Petri net that is composed of a WF-net with additional transition t (feedback transition). The sink place is connected as the input place of transition t and the source place as its output place. Firing of the transition causes iteration of the process (Note: the extended WF-net is not a WF-net).[19]

WRI (Well-handled with Regular Iteration) WF-net, is an extended acyclic well-handled WF-net. WRI-WF-net can be built as composition of nets, i.e., replacing a transition within a WRI-WF-net with a subnet which is a WRI-WF-net. The result is also WRI-WF-net. WRI-WF-nets are G-sound,[21] therefore by using only WRI-WF-net building blocks, one can get WF-nets that are G-sound by construction.

The Design structure matrix (DSM) can model process relations, and be utilized for process planning. The **DSM-nets** are realization of DSM-based plans into workflow processes by Petri nets, and are equivalent to WRI-WF-nets. The DSM-net construction process ensures the soundness property of the resulting net.

5.3.10 Other models of concurrency

Other ways of modelling concurrent computation have been proposed, including process algebra, the actor model, and trace theory.[22] Different models provide tradeoffs of concepts such as compositionality, modularity, and locality.

An approach to relating some of these models of concurrency is proposed in the chapter by Winskel and Nielsen.[23]

5.3.11 Application areas

- Business Process Modeling
- Concurrent programming
- Data analysis
- Diagnosis (Artificial intelligence)
- Discrete process control
- Kahn process networks
- Process modeling
- Reliability engineering
- Simulation
- Software design
- Workflow management systems

5.3.12 See also

- Communicating finite-state machine
- Finite state machine
- Kahn process networks
- Petri Net Markup Language
- Petriscript
- Process architecture

5.3.13 References

[1] Petri, Carl Adam; Reisig, Wolfgang (2008). "Petri net". *Scholarpedia*. **3** (4): 6477. doi:10.4249/scholarpedia.6477.

[2] Rozenburg, G.; Engelfriet, J. (1998). "Elementary Net Systems". In Reisig, W.; Rozenberg, G. *Lectures on Petri Nets I: Basic Models - Advances in Petri Nets*. Lecture Notes in Computer Science. **1491**. Springer. pp. 12–121.

[3] Reisig, Wolfgang (1991). "Petri Nets and Algebraic Specifications". *Theoretical Computer Science*. **80** (1): 1–34. doi:10.1016/0304-3975(91)90203-e.

[4] Desel, Jörg; Juhás, Gabriel (2001). "What Is a Petri Net? Informal Answers for the Informed Reader". In Ehrig, Hartmut; et al. *Unifying Petri Nets*. LNCS. **2128**. Springerlink.com. pp. 1–25. Retrieved 2014-05-14.

[5] Esparza, Javier; Nielsen, Mogens (1995) [1994]. "Decidability issues for Petri nets - a survey". *Bulletin of the EATCS* (Revised ed.). Retrieved 2014-05-14.

[6] Lipton, R. (1976). "The Reachability Problem Requires Exponential Space". *Technical Report 62*. Yale University.

[7] Küngas, P. (July 26–29, 2005). *Petri Net Reachability Checking Is Polynomial with Optimal Abstraction Hierarchies*. Proceedings of the 6th International Symposium on Abstraction, Reformulation and Approximation—SARA 2005. Airth Castle, Scotland, UK.

[8] Murata, Tadao (April 1989). "Petri Nets: Properties, Analysis and Applications". *Proceedings of the IEEE*. **77** (4): 541–558. doi:10.1109/5.24143. Retrieved 2014-10-13.

[9] "Petri Nets". *www.techfak.uni-bielefeld.de*.

[10] David, René; Alla, Hassane (2005). *Discrete, continuous, and hybrid Petri Nets*. Springer. ISBN 978-3-540-22480-8.

[11] Jensen, Kurt. "A brief introduction to colored Petri nets" (PDF).

[12] Araki, T.; Kasami, T. (1977). "Some Decision Problems Related to the Reachability Problem for Petri Nets". *Theoretical Computer Science*. **3** (1): 85–104. doi:10.1016/0304-3975(76)90067-0.

[13] Dufourd, C.; Finkel, A.; Schnoebelen, Ph. (1998). "Reset Nets Between Decidability and Undecidability". *Proceedings of the 25th International Colloquium on Automata, Languages and Programming*. LNCS. **1443**. pp. 103–115.

[14] Zaitsev, D. A. (2013). "Toward the Minimal Universal Petri Net". *IEEE Transactions on Systems, Man, and Cybernetics: Systems*. **44**: 1–12. doi:10.1109/TSMC.2012.2237549.

[15] "Very Brief Introduction to CP-nets". Department of Computer Science, University of Aarhus, Denmark.

[16] http://llpn.com/OPNs.html

[17] Dawis, E. P.; Dawis, J. F.; Koo, Wei-Pin (2001). *Architecture of Computer-based Systems using Dualistic Petri Nets*. 2001 IEEE International Conference on Systems, Man, and Cybernetics. **3**. pp. 1554–1558.

[18] Dawis, E. P. (2001). *Architecture of an SS7 Protocol Stack on a Broadband Switch Platform using Dualistic Petri Nets*. 2001 IEEE Pacific Rim Conference on Communications, Computers and signal Processing. **1**. pp. 323–326.

[19] van der Aalst, W. M. P. (1998). "The application of Petri nets to workflow management" (PDF). *Journal of Circuits, Systems and Computers*. **8** (1): 21–66. doi:10.1142/s0218126698000043.

[20] van Hee, K.; Sidorova, N.; Voorhoeve, M. (2003). "Soundness and separability of workflow nets in the stepwise refinement approach" (PDF). In van der Aalst, W. M. P.; Best, E. *Application and Theory of Petri Nets 2003*. Lect Notes in Comput Sci. **2678**. Springer. pp. 337–356.

[21] Ping, L.; Hao, H.; Jian, L. (2004). Moldt, Daniel, ed. *On 1-soundness and soundness of workflow nets*. Proc of the 3rd Workshop on Modelling of Objects, Components, and Agents. **571**. Aarhus, Denmark: DAIMI PB. pp. 21–36.

[22] Mazurkiewicz, Antoni (1995). "Introduction to Trace Theory". In Diekert, V.; Rozenberg, G. *The Book of Traces*. Singapore: World Scientific. pp. 3–67.

[23] Winskel, G.; Nielsen, M. "Models for Concurrency" (PDF). *Handbook of Logic and the Foundations of Computer Science*. **4**. OUP. pp. 1–148.

5.3.14 Further reading

- Cardoso, Janette; Camargo, Heloisa (1999). *Fuzziness in Petri Nets*. Physica-Verlag. ISBN 3-7908-1158-0.
- Grobelna, Iwona (2011). "Formal verification of embedded logic controller specification with computer deduction in temporal logic". *Przeglad Elektrotechniczny*. **87** (12a): 47–50.
- Jensen, Kurt (1997). *Coloured Petri Nets*. Springer Verlag. ISBN 3-540-62867-3.
- Котов, Вадим (1984). *Сети Петри (Petri Nets, in Russian)*. Наука, Москва.
- Pataricza, András (2004). *Formális módszerek az informatikában (Formal methods in informatics)*. TYPOTEX Kiadó. ISBN 963-9548-08-1.
- Peterson, James L. (1977). "Petri Nets". *ACM Computing Surveys*. **9** (3): 223–252. doi:10.1145/356698.356702.
- Peterson, James Lyle (1981). "Petri Net Theory and the Modeling of Systems". Prentice Hall. ISBN 0-13-661983-5.
- Petri, Carl A. (1962). *Kommunikation mit Automaten* (Ph. D. thesis). University of Bonn.
- Petri, Carl Adam; Reisig, Wolfgang. "Petri net". *Scholarpedia*. **3** (4): 6477. doi:10.4249/scholarpedia.6477. Retrieved 2008-07-13.
- Reisig, Wolfgang (1992). *A Primer in Petri Net Design*. Springer-Verlag. ISBN 3-540-52044-9.
- Riemann, Robert-Christoph (1999). *Modelling of Concurrent Systems: Structural and Semantical Methods in the High Level Petri Net Calculus*. Herbert Utz Verlag. ISBN 3-89675-629-X.
- Störrle, Harald (2000). *Models of Software Architecture - Design and Analysis with UML and Petri-Nets* (PDF). Books on Demand. ISBN 3-8311-1330-0.
- Zhou, Mengchu; Dicesare, Frank (1993). *Petri Net Synthesis for Discrete Event Control of Manufacturing Systems*. Kluwer Academic Publishers. ISBN 0-7923-9289-2.
- Zhou, Mengchu; Venkatesh, Kurapati (1998). *Modeling, Simulation, & Control of Flexible Manufacturing Systems: A Petri Net Approach*. World Scientific Publishing. ISBN 981-02-3029-X.
- Zaitsev, Dmitry (2013). *Clans of Petri Nets: Verification of protocols and performance evaluation of networks*. LAP LAMBERT Academic Publishing. ISBN 978-3-659-42228-7.

5.3.15 External links

- Petri Nets World
- Petri Net Markup Language
- Java implementation of Petri nets in the jBPT library (see jbpt-petri module)
- Java Petri net simulator
- Petia Wohed's Flash-based tutorial introduction to Workflow Technology with Petri Nets
- List of Petri net tools

5.4 Adjacency matrix

In graph theory and computer science, an **adjacency matrix** is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.

In the special case of a finite simple graph, the adjacency matrix is a (0,1)-matrix with zeros on its diagonal. If the graph is undirected, the adjacency matrix is symmetric. The relationship between a graph and the eigenvalues and eigenvectors of its adjacency matrix is studied in spectral graph theory.

The adjacency matrix should be distinguished from the incidence matrix for a graph, a different matrix representation whose elements indicate whether vertex–edge pairs are incident or not.

5.4.1 Definition

For a simple graph with vertex set V, the adjacency matrix is a square $|V| \times |V|$ matrix A such that its element A_{ij} is one when there is an edge from vertex i to vertex j, and zero when there is no edge.[1] The diagonal elements of the matrix are all zero, since edges from a vertex to itself (loops) are not allowed in simple graphs. It is also sometimes useful in algebraic graph theory to replace the nonzero elements with algebraic variables.[2]

The same concept can be extended to multigraphs and graphs with loops by storing the number of edges between each two vertices in the corresponding matrix element, and by allowing nonzero diagonal elements. Loops may be counted either once (as a single edge) or twice (as two vertex-edge incidences), as long as a consistent convention is followed. Undirected graphs often use the latter convention of counting loops twice, whereas directed graphs typically use the former convention.

Adjacency matrix of a bipartite graph

The adjacency matrix A of a bipartite graph whose two parts have r and s vertices can be written in the form

$$A = \begin{pmatrix} 0_{r,r} & B \\ B^T & 0_{s,s} \end{pmatrix},$$

where B is an $r \times s$ matrix, and $0_{r,r}$ and $0_{s,s}$ represent the $r \times r$ and $s \times s$ zero matrices. In this case, the smaller matrix B uniquely represents the graph, and the remaining parts of A can be discarded as redundant. B is sometimes called the biadjacency matrix.

Formally, let $G = (U, V, E)$ be a bipartite graph with parts $U = \{u_1, \ldots, u_r\}$ and $V = \{v_1, \ldots, v_s\}$. The *biadjacency matrix* is the $r \times s$ 0–1 matrix B in which $b_{i,j} = 1$ if and only if $(u_i, v_j) \in E$.

If G is a bipartite multigraph or weighted graph then the elements $b_{i,j}$ are taken to be the number of edges between the vertices or the weight of the edge (u_i, v_j), respectively.

Variations

An (a, b, c)-*adjacency matrix* A of a simple graph has $A_{i,j} = a$ if (i, j) is an edge, b if it is not, and c on the diagonal. The Seidel adjacency matrix is a $(-1, 1, 0)$-*adjacency matrix*. This matrix is used in studying strongly regular graphs and two-graphs.[3]

The **distance matrix** has in position (i, j) the distance between vertices v_i and v_j. The distance is the length of a shortest path connecting the vertices. Unless lengths of edges are explicitly provided, the length of a path is the number of edges in it. The distance matrix resembles a high power of the adjacency matrix, but instead of telling only whether or not two vertices are connected (i.e., the connection matrix, which contains boolean values), it gives the exact distance between them.

5.4.2 Examples

The convention followed here (for an undirected graph) is that each edge adds 1 to the appropriate cell in the matrix, and each loop adds 2. This allows the degree of a vertex to be easily found by taking the sum of the values in either its respective row or column in the adjacency matrix.

The adjacency matrix of a complete graph contains all ones except along the diagonal where there are only zeros. The adjacency matrix of an empty graph is a zero matrix.

5.4.3 Properties

Spectrum

The adjacency matrix of an undirected simple graph is symmetric, and therefore has a complete set of real eigenvalues and an orthogonal eigenvector basis. The set of eigenvalues of a graph is the **spectrum** of the graph.[4]

For d-regular graphs, d is also an eigenvalue of A for the vector $v = (1, \ldots, 1)$, and G is connected if and only if the multiplicity of the eigenvalue d is 1. It can be shown that $-d$ is also an eigenvalue of A if G is a connected bipartite graph. The above are results of the Perron–Frobenius theorem.

Isomorphism and invariants

Suppose two directed or undirected graphs G_1 and G_2 with adjacency matrices A_1 and A_2 are given. G_1 and G_2 are isomorphic if and only if there exists a permutation matrix P such that

$$PA_1P^{-1} = A_2.$$

In particular, A_1 and A_2 are similar and therefore have the same minimal polynomial, characteristic polynomial, eigenvalues, determinant and trace. These can therefore serve as isomorphism invariants of graphs. However, two graphs may possess the same set of eigenvalues but not be isomorphic.[5] Such linear operators are said to be isospectral.

Matrix powers

If A is the adjacency matrix of the directed or undirected graph G, then the matrix A^n (i.e., the matrix product of n copies of A) has an interesting interpretation: the element (i, j) gives the number of (directed or undirected) walks of length n from vertex i to vertex j. If n is the smallest nonnegative integer, such that for all i, j, the element (i, j) of A^n is positive, then n is the distance between vertex i and vertex j. This implies, for example, that the number of triangles in an undirected graph G is exactly the trace of A^3 divided by 6. Note that the adjacency matrix can be used to determine whether or not the graph is connected.

5.4.4 Data structures

The adjacency matrix may be used as a data structure for the representation of graphs in computer programs for manipulating graphs. The main alternative data structure, also in use for this application, is the adjacency list.[6][7]

Because each entry in the adjacency matrix requires only one bit, it can be represented in a very compact way, occupying only $|V|^2/8$ bytes to represent a directed graph, or (by using a packed triangular format and only storing the lower triangular part of the matrix) approximately $|V|^2/16$ bytes to represent an undirected graph. Although slightly more succinct representations are possible, this method gets close to the information-theoretic lower bound for the minimum number of bits needed to represent all n-vertex graphs.[8] For storing graphs in text files, fewer bits per byte can be used to ensure that all bytes are text characters, for instance by using a Base64 representation.[9] Besides avoiding wasted space, this compactness encourages locality of reference. However, for a large sparse graph, adjacency lists require less storage space, because they do not waste any space to represent edges that are *not* present.[7][10]

An alternative form of adjacency matrix, which however requires a larger amount of space, replaces the numbers in each element of the matrix with pointers to edge objects (when edges are present) or null pointers (when there is no edge).[10] It is also possible to store edge weights directly in the elements of an adjacency matrix.[7]

Besides the space tradeoff, the different data structures also facilitate different operations. Finding all vertices adjacent to a given vertex in an adjacency list is as simple as reading the list, and takes time proportional to the number of neighbors. With an adjacency matrix, an entire row must instead be scanned, which takes a larger amount of time, proportional to the number of vertices in the whole graph. On the other hand, testing whether there is an edge between two given vertices can be determined at once with an adjacency matrix, while requiring time proportional to the minimum degree of the two vertices with the adjacency list.[7][10]

5.4.5 References

[1] Biggs, Norman (1993), *Algebraic Graph Theory*, Cambridge Mathematical Library (2nd ed.), Cambridge University Press, Definition 2.1, p. 7.

[2] Harary, Frank (1962), "The determinant of the adjacency matrix of a graph", *SIAM Review*, **4**: 202–210, doi:10.1137/1004057, MR 0144330.

[3] Seidel, J. J. (1968). "Strongly Regular Graphs with (−1, 1, 0) Adjacency Matrix Having Eigenvalue 3". *Lin. Alg. Appl.* **1** (2): 281–298. doi:10.1016/0024-3795(68)90008-6.

[4] Biggs (1993), Chapter 2 ("The spectrum of a graph"), pp. 7–13.

[5] Godsil, Chris; Royle, Gordon *Algebraic Graph Theory*, Springer (2001), ISBN 0-387-95241-1, p.164

[6] Goodrich & Tamassia (2015), p. 361: "There are two data structures that people often use to represent graphs, the adjacency list and the adjacency matrix."

[7] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001), "Section 22.1: Representations of graphs", *Introduction to Algorithms* (Second ed.), MIT Press and McGraw-Hill, pp. 527–531, ISBN 0-262-03293-7.

[8] Turán, György (1984), "On the succinct representation of graphs", *Discrete Applied Mathematics*, **8** (3): 289–294, doi:10.1016/0166-218X(84)90126-4, MR 749658.

[9] McKay, Brendan, *Description of graph6 and sparse6 encodings.*

[10] Goodrich, Michael T.; Tamassia, Roberto (2015), *Algorithm Design and Applications*, Wiley, p. 363.

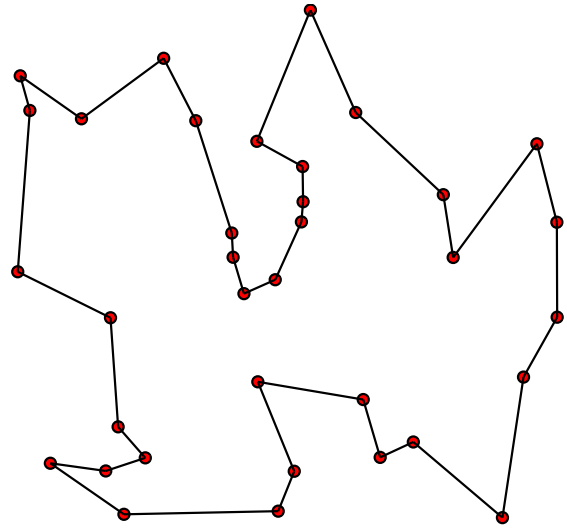
5.4.6 External links

- Weisstein, Eric W. "Adjacency matrix". *MathWorld*.
- Fluffschack — an educational Java web start game demonstrating the relationship between adjacency matrices and graphs.
- Open Data Structures - Section 12.1 - Adjacency-Matrix: Representing a Graph by a Matrix
- Café math : Adjacency Matrices of Graphs : Application of the adjacency matrices to the computation generating series of walks.

Chapter 6

Example Applications of Graph Theory

6.1 Travelling salesman problem



Solution of a travelling salesman problem: the black line shows the shortest possible loop that connects every red dot

The **travelling salesman problem** (**TSP**) asks the following question: *"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"* It is an NP-hard problem in combinatorial optimization, important in operations research and theoretical computer science.

TSP is a special case of the travelling purchaser problem and the vehicle routing problem.

In the theory of computational complexity, the decision version of the TSP (where, given a length L, the task is to decide whether the graph has any tour shorter than L) belongs to the class of NP-complete problems. Thus, it is possible that the worst-case running time for any algorithm for the TSP increases superpolynomially (but no more than exponentially) with the number of cities.

The problem was first formulated in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, a large number of heuristics and exact algorithms are known, so that some instances with tens of thousands of cities can be solved completely and even problems with millions of cities can be approximated within a small fraction of 1%.[1]

The TSP has several applications even in its purest formulation, such as planning, logistics, and the manufacture of microchips. Slightly modified, it appears as a sub-problem in many areas, such as DNA sequencing. In these applications, the concept *city* represents, for example, customers, soldering points, or DNA fragments, and the concept *distance* represents travelling times or cost, or a similarity measure between DNA fragments. The TSP also appears in astronomy, as astronomers observing many sources will want to minimize the time spent moving the telescope between the sources. In many applications, additional constraints such as limited resources or time windows may be imposed.

6.1.1 History

The origins of the travelling salesman problem are unclear. A handbook for travelling salesmen from 1832 mentions the problem and includes example tours through Germany and Switzerland, but contains no mathematical treatment.[2]

The travelling salesman problem was mathematically formulated in the 1800s by the Irish mathematician W.R. Hamilton and by the British mathematician Thomas Kirkman. Hamilton's Icosian Game was a recreational puzzle based on finding a Hamiltonian cycle.[3] The general form of the TSP appears to have been first studied by mathematicians during the 1930s in Vienna and at Harvard, notably by Karl Menger, who defines the problem, considers the obvious brute-force algorithm, and observes the non-optimality of the nearest neighbour heuristic:

> We denote by *messenger problem* (since in practice this question should be solved by each postman, anyway also by many travelers) the task to find, for finitely many points whose pairwise distances are known, the shortest route connecting the points. Of course, this prob-

William Rowan Hamilton

> lem is solvable by finitely many trials. Rules which would push the number of trials below the number of permutations of the given points, are not known. The rule that one first should go from the starting point to the closest point, then to the point closest to this, etc., in general does not yield the shortest route. [4]

It was first considered mathematically in the 1930s by Merrill Flood who was looking to solve a school bus routing problem.[5] Hassler Whitney at Princeton University introduced the name *travelling salesman problem* soon after.[6]

In the 1950s and 1960s, the problem became increasingly popular in scientific circles in Europe and the USA after the RAND Corporation in Santa Monica, offered prizes for steps in solving the problem.[5] Notable contributions were made by George Dantzig, Delbert Ray Fulkerson and Selmer M. Johnson from the RAND Corporation, who expressed the problem as an integer linear program and developed the cutting plane method for its solution. They wrote what is considered the seminal paper on the subject in which with these new methods they solved an instance with 49 cities to optimality by constructing a tour and proving that no other tour could be shorter. Dantzig, Fulkerson and Johnson, however, speculated that given a near optimal solution we may be able to find optimality or prove optimality by adding a small amount of extra inequalities (cuts). They used this idea to solve their initial 49 city problem using a string model. They found they only needed 26 cuts to come to a solution for their 49 city problem. While this paper did not give an algorithmic approach to TSP problems, the ideas that lay within it were indispensable to later creating exact solution methods for the TSP, though it would take 15 years to find an algorithmic approach in creating these cuts.[5] As well as cutting plane methods, Dantzig, Fulkerson and Johnson used branch and bound algorithms perhaps for the first time.[5]

In the following decades, the problem was studied by many researchers from mathematics, computer science, chemistry, physics, and other sciences. In the 1960s however a new approach was created, instead of finding optimal solutions, people tried to instead find the worst solutions and in doing so, created lower bounds for the problem. These may then be used with branch and bound approaches. One method of doing this was to create the minimum spanning tree of the graph and then multiply the cost of this by 2.[5]

Christofides made a big advance in this approach of giving an approach for which we know the worst-case scenario. His algorithm given in 1976, at worst is 1.5 times longer than the optimal solution. As the algorithm was so simple and quick, many hoped it would give way to a near optimal solution method. However, until 2011 when it was beaten by less than a billionth of a percent, this remained the method with the best worst-case scenario.[7]

Richard M. Karp showed in 1972 that the Hamiltonian cycle problem was NP-complete, which implies the NP-hardness of TSP. This supplied a mathematical explanation for the apparent computational difficulty of finding optimal tours.

Great progress was made in the late 1970s and 1980, when Grötschel, Padberg, Rinaldi and others managed to exactly solve instances with up to 2392 cities, using cutting planes and branch-and-bound.

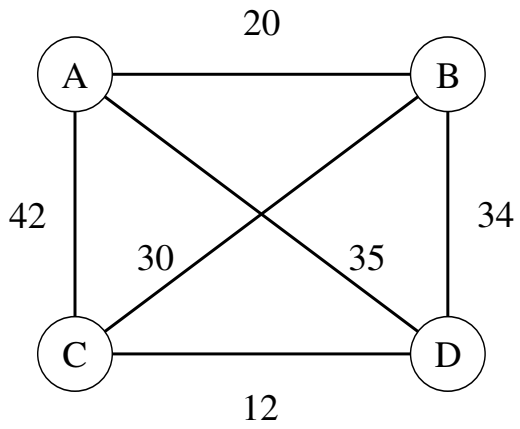
In the 1990s, Applegate, Bixby, Chvátal, and Cook developed the program *Concorde* that has been used in many recent record solutions. Gerhard Reinelt published the TSPLIB in 1991, a collection of benchmark instances of varying difficulty, which has been used by many research groups for comparing results. In 2006, Cook and others computed an optimal tour through an 85,900-city instance given by a microchip layout problem, currently the largest solved TSPLIB instance. For many other instances with millions of cities, solutions can be found that are guaranteed to be within 2-3% of an optimal tour.[8]

The problem is sometimes, especially in newer publications, referred to as *Travelling Salesperson Problem*.

6.1.2 Description

As a graph problem

TSP can be modelled as an undirected weighted graph, such that cities are the graph's vertices, paths are the graph's edges, and a path's distance is the edge's weight. It is a minimization problem starting and finishing at a spec-



Symmetric TSP with four cities

ified vertex after having visited each other vertex exactly once. Often, the model is a complete graph (*i.e.* each pair of vertices is connected by an edge). If no path exists between two cities, adding an arbitrarily long edge will complete the graph without affecting the optimal tour.

Asymmetric and symmetric

In the *symmetric TSP*, the distance between two cities is the same in each opposite direction, forming an undirected graph. This symmetry halves the number of possible solutions. In the *asymmetric TSP*, paths may not exist in both directions or the distances might be different, forming a directed graph. Traffic collisions, one-way streets, and airfares for cities with different departure and arrival fees are examples of how this symmetry could break down.

Related problems

- An equivalent formulation in terms of graph theory is: Given a complete weighted graph (where the vertices would represent the cities, the edges would represent the roads, and the weights would be the cost or distance of that road), find a Hamiltonian cycle with the least weight.
- The requirement of returning to the starting city does not change the computational complexity of the problem, see Hamiltonian path problem.
- Another related problem is the bottleneck travelling salesman problem (bottleneck TSP): Find a Hamiltonian cycle in a weighted graph with the minimal weight of the weightiest edge. The problem is of considerable practical importance, apart from evident transportation and logistics areas. A classic example is in printed circuit manufacturing: scheduling of a route of the drill machine to drill holes in a PCB. In robotic machining or drilling applications, the "cities" are parts to machine or holes (of different sizes) to drill, and the "cost of travel" includes time for retooling the robot (single machine job sequencing problem).[9]
- The generalized travelling salesman problem, also known as the "travelling politician problem", deals with "states" that have (one or more) "cities" and the salesman has to visit exactly one "city" from each "state". One application is encountered in ordering a solution to the cutting stock problem in order to minimize knife changes. Another is concerned with drilling in semiconductor manufacturing, see e.g., U.S. Patent 7,054,798. Noon and Bean demonstrated that the generalized travelling salesman problem can be transformed into a standard travelling salesman problem with the same number of cities, but a modified distance matrix.
- The sequential ordering problem deals with the problem of visiting a set of cities where precedence relations between the cities exist.
- A common interview question at Google is how to route data among data processing nodes; routes vary by time to transfer the data, but nodes also differ by their computing power and storage, compunding the problem of where to send data.
- The travelling purchaser problem deals with a purchaser who is charged with purchasing a set of products. He can purchase these products in several cities, but at different prices and not all cities offer the same products. The objective is to find a route between a subset of the cities, which minimizes total cost (travel cost + purchasing cost) and which enables the purchase of all required products.

6.1.3 Integer linear programming formulation

TSP can be formulated as an integer linear program.[10][11][12] Label the cities with the numbers 1, …, *n* and define:

$$x_{ij} = \begin{cases} 1 & \text{city from goes path the} i \text{ city to } j \\ 0 & \text{otherwise} \end{cases}$$

For *i* = 1, …, *n*, let u_i be a dummy variable, and finally take c_{ij} to be the distance from city *i* to city *j*. Then TSP can be written as the following integer linear programming problem:

$$\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} :$$

$$\begin{aligned}
& 0 \leq x_{ij} \leq 1 && i, j = 1, \ldots, n; \\
& u_i \in \mathbf{Z} && i = 1, \ldots, n; \\
& \sum_{i=1, i \neq j}^{n} x_{ij} = 1 && j = 1, \ldots, n; \\
& \sum_{j=1, j \neq i}^{n} x_{ij} = 1 && i = 1, \ldots, n; \\
& u_i - u_j + n x_{ij} \leq n - 1 && 1 \leq i \neq j \leq n.
\end{aligned}$$

The first set of equalities requires that each city be arrived at from exactly one other city, and the second set of equalities requires that from each city there is a departure to exactly one other city. The last constraints enforce that there is only a single tour covering all cities, and not two or more disjointed tours that only collectively cover all cities. To prove this, it is shown below (1) that every feasible solution contains only one closed sequence of cities, and (2) that for every single tour covering all cities, there are values for the dummy variables u_i that satisfy the constraints.

To prove that every feasible solution contains only one closed sequence of cities, it suffices to show that every subtour in a feasible solution passes through city 1 (noting that the equalities ensure there can only be one such tour). For if we sum all the inequalities corresponding to $x_{ij} = 1$ for any subtour of *k* steps not passing through city 1, we obtain:

$$nk \leq (n-1)k,$$

which is a contradiction.

It now must be shown that for every single tour covering all cities, there are values for the dummy variables u_i that satisfy the constraints.

Without loss of generality, define the tour as originating (and ending) at city 1. Choose $u_i = t$ if city *i* is visited in step *t* (*i*, *t* = 1, 2, ..., n). Then

$$u_i - u_j \leq n - 1,$$

since u_i can be no greater than *n* and u_j can be no less than 1; hence the constraints are satisfied whenever $x_{ij} = 0$. For $x_{ij} = 1$, we have:

$$u_i - u_j + n x_{ij} = (t) - (t+1) + n = n - 1,$$

satisfying the constraint.

6.1.4 Computing a solution

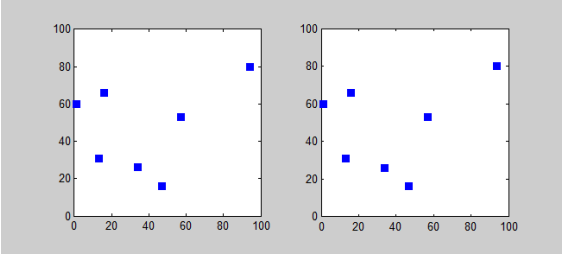
The traditional lines of attack for the NP-hard problems are the following:

- Devising exact algorithms, which work reasonably fast only for small problem sizes.
- Devising "suboptimal" or heuristic algorithms, i.e., algorithms that deliver either seemingly or probably good solutions, but which could not be proved to be optimal.
- Finding special cases for the problem ("subproblems") for which either better or exact heuristics are possible.

Exact algorithms

The most direct solution would be to try all permutations (ordered combinations) and see which one is cheapest (using brute force search). The running time for this approach lies within a polynomial factor of $O(n!)$, the factorial of the number of cities, so this solution becomes impractical even for only 20 cities.

One of the earliest applications of dynamic programming is the Held–Karp algorithm that solves the problem in time $O(n^2 2^n)$.[13]

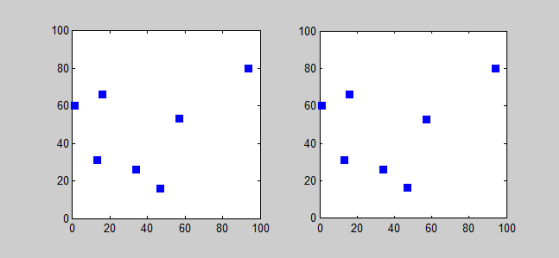


Solution to a symmetric TSP with 7 cities using brute force search. Note: Number of permutations: (7-1)!/2 = 360

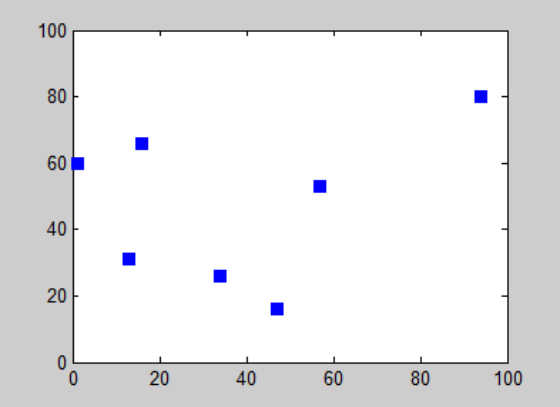
Improving these time bounds seems to be difficult. For example, it has not been determined whether an exact algorithm for TSP that runs in time $O(1.9999^n)$ exists.[14]

Other approaches include:

- Various branch-and-bound algorithms, which can be used to process TSPs containing 40–60 cities.
- Progressive improvement algorithms which use techniques reminiscent of linear programming. Works well for up to 200 cities.
- Implementations of branch-and-bound and problem-specific cut generation (branch-and-cut[15]); this is the method of choice for solving large instances. This approach holds the current



Solution of a TSP with 7 cities using a simple Branch and bound algorithm. Note: The number of permutations is much less than Brute force search



Nearest Neighbour algorithm for a TSP with 7 cities. The solution changes as the starting point is changed

record, solving an instance with 85,900 cities, see Applegate et al. (2006).

An exact solution for 15,112 German towns from TSPLIB was found in 2001 using the cutting-plane method proposed by George Dantzig, Ray Fulkerson, and Selmer M. Johnson in 1954, based on linear programming. The computations were performed on a network of 110 processors located at Rice University and Princeton University (see the Princeton external link). The total computation time was equivalent to 22.6 years on a single 500 MHz Alpha processor. In May 2004, the travelling salesman problem of visiting all 24,978 towns in Sweden was solved: a tour of length approximately 72,500 kilometres was found and it was proven that no shorter tour exists.[16] In March 2005, the travelling salesman problem of visiting all 33,810 points in a circuit board was solved using *Concorde TSP Solver*: a tour of length 66,048,945 units was found and it was proven that no shorter tour exists. The computation took approximately 15.7 CPU-years (Cook et al. 2006). In April 2006 an instance with 85,900 points was solved using *Concorde TSP Solver*, taking over 136 CPU-years, see Applegate et al. (2006).

Heuristic and approximation algorithms

Various heuristics and approximation algorithms, which quickly yield good solutions have been devised. Modern methods can find solutions for extremely large problems (millions of cities) within a reasonable time which are with a high probability just 2–3% away from the optimal solution.[8]

Several categories of heuristics are recognized.

Constructive heuristics The nearest neighbour (NN) algorithm (a greedy algorithm) lets the salesman choose the nearest unvisited city as his next move. This algorithm quickly yields an effectively short route. For N cities randomly distributed on a plane, the algorithm on average yields a path 25% longer than the shortest possible path.[17] However, there exist many specially arranged city distributions which make the NN algorithm give the worst route (Gutin, Yeo, and Zverovich, 2002). This is true for both asymmetric and symmetric TSPs (Gutin and Yeo, 2007). Rosenkrantz et al. [1977] showed that the NN algorithm has the approximation factor $\Theta(\log |V|)$ for instances satisfying the triangle inequality. A variation of NN algorithm, called Nearest Fragment (NF) operator, which connects a group (fragment) of nearest unvisited cities, can find shorter route with successive iterations.[18] The NF operator can also be applied on an initial solution obtained by NN algorithm for further improvement in an elitist model, where only better solutions are accepted.

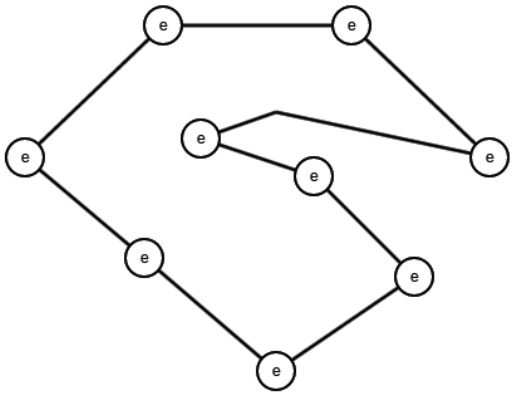
The bitonic tour of a set of points is the minimum-perimeter monotone polygon that has the points as its vertices; it can be computed efficiently by dynamic programming.

Another constructive heuristic, Match Twice and Stitch (MTS) (Kahng, Reda 2004 [19]), performs two sequential matchings, where the second matching is executed after deleting all the edges of the first matching, to yield a set of cycles. The cycles are then stitched to produce the final tour.

Christofides' algorithm for the TSP The Christofides algorithm follows a similar outline but combines the minimum spanning tree with a solution of another problem, minimum-weight perfect matching. This gives a TSP tour which is at most 1.5 times the optimal. The Christofides algorithm was one of the first approximation algorithms, and was in part responsible for drawing attention to approximation algorithms as a practical approach to intractable problems. As a matter of fact, the term "algorithm" was not commonly extended to approximation algorithms until later; the Christofides algorithm was initially referred to as the Christofides heuristic.

This algorithm looks at things differently by using a result from graph theory which helps improve on the LB of the TSP which originated from doubling the cost of the minimum spanning tree. Given an Eulerian graph we can

find an Eulerian tour in *O(n)* time.[5] So if we had an Eulerian graph with cities from a TSP as vertices then we can easily see that we could use such a method for finding an Eulerian tour to find a TSP solution. By triangular inequality we know that the TSP tour can be no longer than the Eulerian tour and as such we have a LB for the TSP. Such a method is described below.



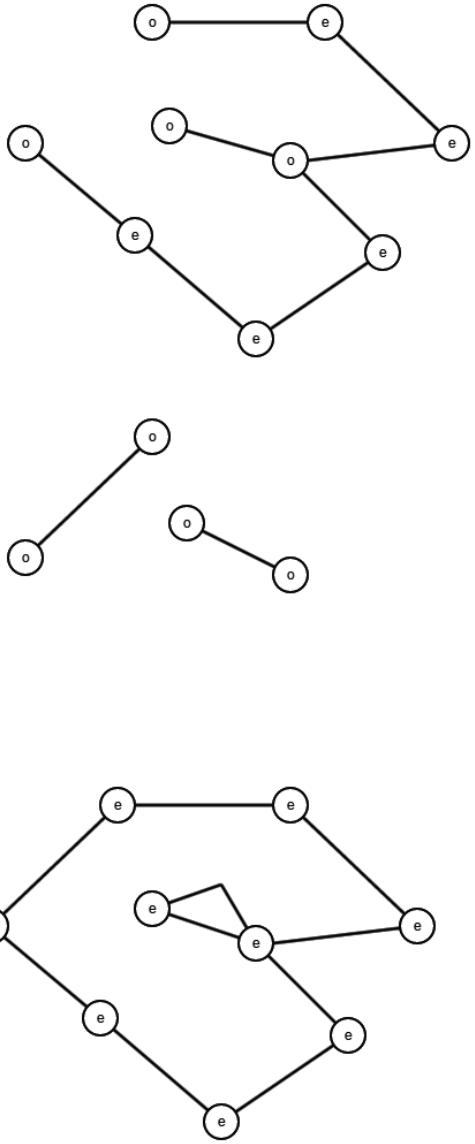
Using a shortcut heuristic on the graph created by the matching below

1. Find a minimum spanning tree for the problem
2. Create duplicates for every edge to create an Eulerian graph
3. Find an Eulerian tour for this graph
4. Convert to TSP: if a city is visited twice, create a shortcut from the city before this in the tour to the one after this.

To improve our lower bound, we therefore need a better way of creating an Eulerian graph. But by triangular inequality, the best Eulerian graph must have the same cost as the best travelling salesman tour, hence finding optimal Eulerian graphs is at least as hard as TSP. One way of doing this that has been proposed is by the concept of minimum weight matching for the creation of which there exist algorithms of $O(n^3)$.[5]

To make a graph into an Eulerian graph, one starts with the minimum spanning tree. Then all the vertices of odd order must be made even. So a matching for the odd degree vertices must be added which increases the order of every odd degree vertex by one.[5] This leaves us with a graph where every vertex is of even order which is thus Eulerian. Now we can adapt the above method to give Christofides' algorithm,

1. Find a minimum spanning tree for the problem
2. Create a matching for the problem with the set of cities of odd order.
3. Find an Eulerian tour for this graph
4. Convert to TSP using shortcuts.

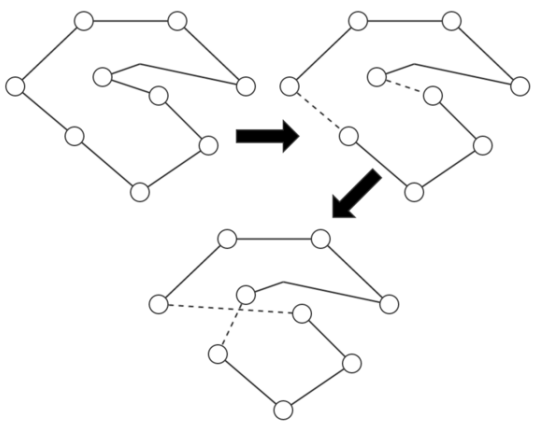


Creating a matching

Iterative improvement

Pairwise exchange The pairwise exchange or *2-opt* technique involves iteratively removing two edges and replacing these with two different edges that reconnect the fragments created by edge removal into a new and shorter tour. This is a special case of the *k*-opt method. Note that the label *Lin–Kernighan* is an often heard misnomer for 2-opt. Lin–Kernighan is actually the more general k-opt method.

For Euclidean instances, 2-opt heuristics give on average solutions that are about 5% better than Christofides' algorithm. If we start with an initial solution made with a greedy algorithm, the average number of moves greatly decreases again and is *O(n)*. For random starts however, the average number of moves is *O(n* log*(n))*. However



An example of a 2-opt iteration

whilst in order this is a small increase in size, the initial number of moves for small problems is 10 times as big for a random start compared to one made from a greedy heuristic. This is because such 2-opt heuristics exploit `bad' parts of a solution such as crossings. These types of heuristics are often used within Vehicle routing problem heuristics to reoptimize route solutions.[17]

***k*-opt heuristic, or Lin–Kernighan heuristics** Take a given tour and delete k mutually disjoint edges. Reassemble the remaining fragments into a tour, leaving no disjoint subtours (that is, don't connect a fragment's endpoints together). This in effect simplifies the TSP under consideration into a much simpler problem. Each fragment endpoint can be connected to $2k - 2$ other possibilities: of $2k$ total fragment endpoints available, the two endpoints of the fragment under consideration are disallowed. Such a constrained $2k$-city TSP can then be solved with brute force methods to find the least-cost recombination of the original fragments. The k-opt technique is a special case of the V-opt or variable-opt technique. The most popular of the k-opt methods are 3-opt, and these were introduced by Shen Lin of Bell Labs in 1965. There is a special case of 3-opt where the edges are not disjoint (two of the edges are adjacent to one another). In practice, it is often possible to achieve substantial improvement over 2-opt without the combinatorial cost of the general 3-opt by restricting the 3-changes to this special subset where two of the removed edges are adjacent. This so-called two-and-a-half-opt typically falls roughly midway between 2-opt and 3-opt, both in terms of the quality of tours achieved and the time required to achieve those tours.

***V*-opt heuristic** The variable-opt method is related to, and a generalization of the k-opt method. Whereas the k-opt methods remove a fixed number (k) of edges from the original tour, the variable-opt methods do not fix the size of the edge set to remove. Instead they grow the set as the search process continues. The best known method in this family is the Lin–Kernighan method (mentioned above as a misnomer for 2-opt). Shen Lin and Brian Kernighan first published their method in 1972, and it was the most reliable heuristic for solving travelling salesman problems for nearly two decades. More advanced variable-opt methods were developed at Bell Labs in the late 1980s by David Johnson and his research team. These methods (sometimes called Lin–Kernighan–Johnson) build on the Lin–Kernighan method, adding ideas from tabu search and evolutionary computing. The basic Lin–Kernighan technique gives results that are guaranteed to be at least 3-opt. The Lin–Kernighan–Johnson methods compute a Lin–Kernighan tour, and then perturb the tour by what has been described as a mutation that removes at least four edges and reconnecting the tour in a different way, then V-opting the new tour. The mutation is often enough to move the tour from the local minimum identified by Lin–Kernighan. V-opt methods are widely considered the most powerful heuristics for the problem, and are able to address special cases, such as the Hamilton Cycle Problem and other non-metric TSPs that other heuristics fail on. For many years Lin–Kernighan–Johnson had identified optimal solutions for all TSPs where an optimal solution was known and had identified the best known solutions for all other TSPs on which the method had been tried.

Randomized improvement Optimized Markov chain algorithms which use local searching heuristic sub-algorithms can find a route extremely close to the optimal route for 700 to 800 cities.

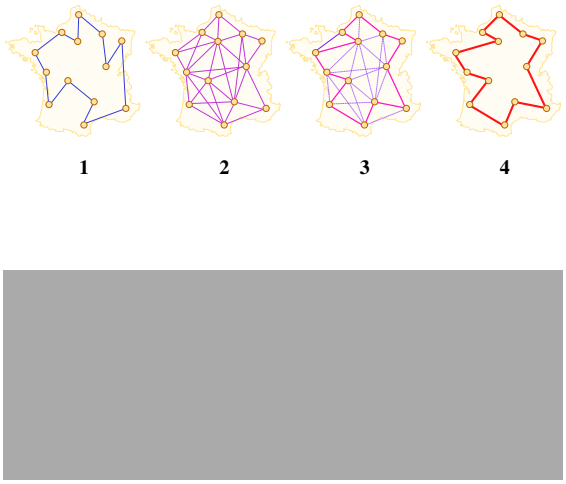
TSP is a touchstone for many general heuristics devised for combinatorial optimization such as genetic algorithms, simulated annealing, Tabu search, ant colony optimization, river formation dynamics (see swarm intelligence) and the cross entropy method.

Ant colony optimization Main article: Ant colony optimization algorithms

Artificial intelligence researcher Marco Dorigo described in 1993 a method of heuristically generating "good solutions" to the TSP using a simulation of an ant colony called *ACS* (Ant Colony System).[20] It models behaviour observed in real ants to find short paths between food sources and their nest, an emergent behaviour resulting from each ant's preference to follow trail pheromones deposited by other ants.

ACS sends out a large number of virtual ant agents to explore many possible routes on the map. Each ant probabilistically chooses the next city to visit based on a heuris-

tic combining the distance to the city and the amount of virtual pheromone deposited on the edge to the city. The ants explore, depositing pheromone on each edge that they cross, until they have all completed a tour. At this point the ant which completed the shortest tour deposits virtual pheromone along its complete tour route (*global trail updating*). The amount of pheromone deposited is inversely proportional to the tour length: the shorter the tour, the more it deposits.



Ant Colony Optimization Algorithm for a TSP with 7 cities: Red and thick lines in the pheromone map indicate presence of more pheromone

6.1.5 Special cases of the TSP

Metric TSP

In the *metric TSP*, also known as *delta-TSP* or Δ-TSP, the intercity distances satisfy the triangle inequality.

A very natural restriction of the TSP is to require that the distances between cities form a metric to satisfy the triangle inequality; that is the direct connection from A to B is never farther than the route via intermediate C:

$$d_{AB} \leq d_{AC} + d_{CB}$$

The edge spans then build a metric on the set of vertices. When the cities are viewed as points in the plane, many natural distance functions are metrics, and so many natural instances of TSP satisfy this constraint.

The following are some examples of metric TSPs for various metrics.

- In the Euclidean TSP (see below) the distance between two cities is the Euclidean distance between the corresponding points.
- In the rectilinear TSP the distance between two cities is the sum of the differences of their x- and y-coordinates. This metric is often called the Manhattan distance or city-block metric.
- In the maximum metric, the distance between two points is the maximum of the absolute values of differences of their x- and y-coordinates.

The last two metrics appear for example in routing a machine that drills a given set of holes in a printed circuit board. The Manhattan metric corresponds to a machine that adjusts first one co-ordinate, and then the other, so the time to move to a new point is the sum of both movements. The maximum metric corresponds to a machine that adjusts both co-ordinates simultaneously, so the time to move to a new point is the slower of the two movements.

In its definition, the TSP does not allow cities to be visited twice, but many applications do not need this constraint. In such cases, a symmetric, non-metric instance can be reduced to a metric one. This replaces the original graph with a complete graph in which the inter-city distance d_{AB} is replaced by the shortest path between A and B in the original graph.

Euclidean TSP

When the input numbers can be arbitrary real numbers, Euclidean TSP is a particular case of metric TSP, since distances in a plane obey the triangle inequality. When the input numbers must be integers, comparing lengths of tours involves comparing sums of square-roots.

Like the general TSP, Euclidean TSP is NP-hard in either case. With rational coordinates and discretized metric (distances rounded up to an integer), the problem is NP-complete.[21] With rational coordinates and the actual Euclidean metric, Euclidean TSP is known to be in the Counting Hierarchy,[22] a subclass of PSPACE. With arbitrary real coordinates, Euclidean TSP cannot be in such classes, since there are uncountably many possible inputs. However, Euclidean TSP is probably the easiest version for approximation.[23] For example, the minimum spanning tree of the graph associated with an instance of the Euclidean TSP is a Euclidean minimum spanning tree, and so can be computed in expected O (n log n) time for n points (considerably less than the number of edges). This enables the simple 2-approximation algorithm for TSP with triangle inequality above to operate more quickly.

In general, for any $c > 0$, where d is the number of dimensions in the Euclidean space, there is a polynomial-time algorithm that finds a tour of length at most $(1 + 1/c)$ times the optimal for geometric instances of TSP in

$$O\left(n(\log n)^{(O(c\sqrt{d}))^{d-1}}\right),$$

time; this is called a polynomial-time approximation

scheme (PTAS).[24] Sanjeev Arora and Joseph S. B. Mitchell were awarded the Gödel Prize in 2010 for their concurrent discovery of a PTAS for the Euclidean TSP.

In practice, simpler heuristics with weaker guarantees continue to be used.

Asymmetric TSP

In most cases, the distance between two nodes in the TSP network is the same in both directions. The case where the distance from *A* to *B* is not equal to the distance from *B* to *A* is called asymmetric TSP. A practical application of an asymmetric TSP is route optimization using street-level routing (which is made asymmetric by one-way streets, slip-roads, motorways, etc.).

Solving by conversion to symmetric TSP Solving an asymmetric TSP graph can be somewhat complex. The following is a 3×3 matrix containing all possible path weights between the nodes *A*, *B* and *C*. One option is to turn an asymmetric matrix of size *N* into a symmetric matrix of size 2*N*.[25]

To double the size, each of the nodes in the graph is duplicated, creating a second *ghost node*, linked to the original node with a "ghost" edge of very low (possibly negative) weight, here denoted −w. (Alternatively, the ghost edges have weight 0, and weight w is added to all other edges.) The original 3×3 matrix shown above is visible in the bottom left and the inverse of the original in the top-right. Both copies of the matrix have had their diagonals replaced by the low-cost hop paths, represented by −w. In the new graph, no edge directly links original nodes and no edge directly links ghost nodes.

The weight −w of the "ghost" edges linking the ghost nodes to the corresponding original nodes must be low enough to ensure that all ghost edges must belong to any optimal symmetric TSP solution on the new graph (w=0 is not always low enough). As a consequence, in the optimal symmetric tour, each original node appears next to its ghost node (e.g. a possible path is *A* -> *A′* -> *C* -> *C′* -> *B* -> *B′* -> *A*) and by mergeing the original and ghost nodes again we get an (optimal) solution of the original asymmetric problem (in our example, *A* -> *C* -> *B* -> *A*).

Analyst's travelling salesman problem

There is an analogous problem in geometric measure theory which asks the following: under what conditions may a subset *E* of Euclidean space be contained in a rectifiable curve (that is, when is there a curve with finite length that visits every point in *E*)? This problem is known as the analyst's travelling salesman problem

TSP path length for random sets of points in a square

Suppose $X_1, \ldots, X_n$ are n independent random variables with uniform distribution in the square $[0,1]^2$, and let L_n^* be the shortest path length (i.e. TSP solution) for this set of points, according to the usual Euclidean distance. It is known[26] that, almost surely,

$$\frac{L_n^*}{\sqrt{n}} \to \beta \qquad \text{when} n \to \infty,$$

where β is a positive constant that is not known explicitly. Since $L_n^* \leq 2\sqrt{n} + 2$ (see below), it follows from bounded convergence theorem that $\beta = \lim_{n\to\infty} \mathbb{E}[L_n^*]/\sqrt{n}$, hence lower and upper bounds on β follow from bounds on $\mathbb{E}[L_n^*]$.

The almost sure limit $\frac{L_n^*}{\sqrt{n}} \to \beta$ as $n \to \infty$ may not exist if the independent locations $X_1, \ldots, X_n$ are replaced with observations from a stationary ergodic process with uniform marginals. [27]

Upper bound

- One has $L^* \leq 2\sqrt{n} + 2$, and therefore $\beta \leq 2$, by using a naive path which visits monotonically the points inside each of $\sqrt{n}$ slices of width $1/\sqrt{n}$ in the square.
- Few [28] proved $L_n^* \leq \sqrt{2n} + 1.75$, hence $\beta \leq \sqrt{2}$, later improved by Karloff (1987): $\beta \leq 0.984\sqrt{2}$.
- The currently [29] best upper bound is $\beta \leq 0.92\ldots$.

Lower bound

- By observing that $\mathbb{E}[L_n^*]$ is greater than n times the distance between X_0 and the closest point $X_i \neq X_0$, one gets (after a short computation)

$$\mathbb{E}[L_n^*] \geq \tfrac{1}{2}\sqrt{n}.$$

- A better lower bound is obtained[26] by observing that $\mathbb{E}[L_n^*]$ is greater than $\frac{1}{2}n$ times the sum of the distances between X_0 and the closest and second closest points $X_i, X_j \neq X_0$, which gives

$$\mathbb{E}[L_n^*] \geq \left(\tfrac{1}{4} + \tfrac{3}{8}\right)\sqrt{n} = \tfrac{5}{8}\sqrt{n},$$

- The currently [29] best lower bound is

$$\mathbb{E}[L_n^*] \geq (\tfrac{5}{8} + \tfrac{19}{5184})\sqrt{n},$$

- Held and Karp[30] gave a polynomial-time algorithm that provides numerical lower bounds for L_n^* , and thus for $\beta(\simeq\ L_n^*/\sqrt{n})$ which seem to be good up to more or less 1%.[31] In particular, David S. Johnson[32] obtained a lower bound by computer experiment:

$$L_n^* \gtrsim 0.7080\sqrt{n} + 0.522,$$

where 0.522 comes from the points near square boundary which have fewer neighbours, and Christine L. Valenzuela and Antonia J. Jones [33] obtained the following other numerical lower bound:

$$L_n^* \gtrsim 0.7078\sqrt{n} + 0.551$$

6.1.6 Computational complexity

The problem has been shown to be NP-hard (more precisely, it is complete for the complexity class $\mathrm{FP}^{\mathrm{NP}}$; see function problem), and the decision problem version ("given the costs and a number *x*, decide whether there is a round-trip route cheaper than *x*") is NP-complete. The bottleneck travelling salesman problem is also NP-hard. The problem remains NP-hard even for the case when the cities are in the plane with Euclidean distances, as well as in a number of other restrictive cases. Removing the condition of visiting each city "only once" does not remove the NP-hardness, since it is easily seen that in the planar case there is an optimal tour that visits each city only once (otherwise, by the triangle inequality, a shortcut that skips a repeated visit would not increase the tour length).

Complexity of approximation

In the general case, finding a shortest travelling salesman tour is NPO-complete.[34] If the distance measure is a metric and symmetric, the problem becomes APX-complete[35] and Christofides's algorithm approximates it within 1.5.[36] The best known inapproximability bound is 123/122 .[37]

If the distances are restricted to 1 and 2 (but still are a metric) the approximation ratio becomes 8/7.[38] In the asymmetric, metric case, only logarithmic performance guarantees are known, the best current algorithm achieves performance ratio 0.814 log(*n*);[39] it is an open question if a constant factor approximation exists.

The corresponding maximization problem of finding the *longest* travelling salesman tour is approximable within 63/38.[40] If the distance function is symmetric, the longest tour can be approximated within 4/3 by a deterministic algorithm[41] and within $\frac{1}{25}(33 + \varepsilon)$ by a randomized algorithm.[42]

6.1.7 Human performance on TSP

The TSP, in particular the Euclidean variant of the problem, has attracted the attention of researchers in cognitive psychology. It has been observed that humans are able to produce good quality solutions quickly.[43] These results suggest that computer performance on the TSP may be improved by understanding and emulating the methods used by humans for these problems, and have also led to new insights into the mechanisms of human thought.[44] The first issue of the *Journal of Problem Solving* was devoted to the topic of human performance on TSP,[45] and a 2011 review listed dozens of papers on the subject.[44]

6.1.8 Natural computation

When presented with a spatial configuration of food sources, the amoeboid Physarum polycephalum adapts its morphology to create an efficient path between the food sources which can also be viewed as an approximate solution to TSP.[46] It's considered to present interesting possibilities and it has been studied in the area of natural computing.

6.1.9 Benchmarks

For benchmarking of TSP algorithms, **TSPLIB** is a library of sample instances of the TSP and related problems is maintained, see the TSPLIB external reference. Many of them are lists of actual cities and layouts of actual printed circuits.

6.1.10 Popular culture

- The thriller novel *The Steradian Trail* by M. N. Krish weaves The Traveling Salesman Problem and mathematician Srinivasa Ramanujan and his accidental discovery into its plot connecting religion, mathematics, finance and economics.[47][48]
- *Travelling Salesman*, by director Timothy Lanzone, is the story of four mathematicians hired by the U.S. government to solve the most elusive problem in computer-science history: P vs. NP.[49]

6.1.11 See also

- Canadian traveller problem

- Exact algorithm
- Route inspection problem (also known as "Chinese postman problem")
- Set TSP problem
- Seven Bridges of Königsberg
- Tube Challenge
- Vehicle routing problem
- Graph Exploration

6.1.12 Notes

[1] See the TSP world tour problem which has already been solved to within 0.05% of the optimal solution.

[2] "Der Handlungsreisende – wie er sein soll und was er zu tun hat, um Aufträge zu erhalten und eines glücklichen Erfolgs in seinen Geschäften gewiß zu sein – von einem alten Commis-Voyageur" (The travelling salesman — how he must be and what he should do in order to get commissions and be sure of the happy success in his business — by an old *commis-voyageur*)

[3] A discussion of the early work of Hamilton and Kirkman can be found in Graph Theory 1736–1936

[4] Cited and English translation in Schrijver (2005). Original German: "Wir bezeichnen als *Botenproblem* (weil diese Frage in der Praxis von jedem Postboten, übrigens auch von vielen Reisenden zu lösen ist) die Aufgabe, für endlich viele Punkte, deren paarweise Abstände bekannt sind, den kürzesten die Punkte verbindenden Weg zu finden. Dieses Problem ist natürlich stets durch endlich viele Versuche lösbar. Regeln, welche die Anzahl der Versuche unter die Anzahl der Permutationen der gegebenen Punkte herunterdrücken würden, sind nicht bekannt. Die Regel, man solle vom Ausgangspunkt erst zum nächstgelegenen Punkt, dann zu dem diesem nächstgelegenen Punkt gehen usw., liefert im allgemeinen nicht den kürzesten Weg."

[5] al.], edited by E.L. Lawler ... [et (1985). *The Traveling salesman problem : a guided tour of combinatorial optimization* (Repr. with corrections. ed.). Chichester [West Sussex]: Wiley. ISBN 0471904139.

[6] A detailed treatment of the connection between Menger and Whitney as well as the growth in the study of TSP can be found in Alexander Schrijver's 2005 paper "On the history of combinatorial optimization (till 1960). Handbook of Discrete Optimization (K. Aardal, G.L. Nemhauser, R. Weismantel, eds.), Elsevier, Amsterdam, 2005, pp. 1–68.PS,PDF

[7] Klarreich, Erica. "Computer Scientists Find New Shortcuts for Infamous Traveling Salesman Problem". *WIRED*. Simons Science News. Retrieved 2015-06-14.

[8] Rego, César; Gamboa, Dorabela; Glover, Fred; Osterman, Colin (2011), "Traveling salesman problem heuristics: leading methods, implementations and latest advances", *European Journal of Operational Research*, **211** (3): 427–441, doi:10.1016/j.ejor.2010.09.010, MR 2774420.

[9] Behzad, Arash; Modarres, Mohammad (2002), "New Efficient Transformation of the Generalized Traveling Salesman Problem into Traveling Salesman Problem", *Proceedings of the 15th International Conference of Systems Engineering (Las Vegas)*

[10] Papadimitriou, C.H.; Steiglitz, K. (1998), *Combinatorial optimization: algorithms and complexity*, Mineola, NY: Dover, pp.308-309.

[11] Tucker, A. W. (1960), "On Directed Graphs and Integer Programs", IBM Mathematical research Project (Princeton University)

[12] Dantzig, George B. (1963), *Linear Programming and Extensions*, Princeton, NJ: PrincetonUP, pp. 545–7, ISBN 0-691-08000-3, sixth printing, 1974.

[13] Bellman (1960), Bellman (1962), Held & Karp (1962)

[14] Woeginger (2003)

[15] Padberg & Rinaldi (1991)

[16] Work by David Applegate, AT&T Labs – Research, Robert Bixby, ILOG and Rice University, Vašek Chvátal, Concordia University, William Cook, University of Waterloo, and Keld Helsgaun, Roskilde University is discussed on their project web page hosted by the University of Waterloo and last updated in June 2004, here

[17] Johnson, D. S.; McGeoch, L. A. (1997). "The Traveling Salesman Problem: A Case Study in Local Optimization" (PDF). In Aarts, E. H. L.; Lenstra, J. K. *Local Search in Combinatorial Optimisation*. London: John Wiley and Sons Ltd. pp. 215–310.

[18] Ray, S. S.; Bandyopadhyay, S.; Pal, S. K. (2007). "Genetic Operators for Combinatorial Optimization in TSP and Microarray Gene Ordering". *Applied Intelligence*. **26** (3): 183–195. doi:10.1007/s10489-006-0018-y.

[19] Kahng, A. B.; Reda, S. (2004). "Match Twice and Stitch: A New TSP Tour Construction Heuristic". *Operations Research Letters*. **32** (6): 499–509. doi:10.1016/j.orl.2004.04.001.

[20] Marco Dorigo. "Ant Colonies for the Traveling Salesman Problem. IRIDIA, Université Libre de Bruxelles. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66. 1997. http://citeseer.ist.psu.edu/86357.html

[21] Papadimitriou (1977).

[22] Allender et al. (2007)

[23] Larson & Odoni (1981)

[24] Arora (1998).

[25] Jonker, Roy; Volgenant, Ton. "Transforming asymmetric into symmetric traveling salesman problems". *Operations Research Letters*. **2** (161–163): 1983. doi:10.1016/0167-6377(83)90048-2.

[26] Beardwood, Halton & Hammersley (1959)

[27] Arlotto, Alessandro; Steele, J. Michael (2016), "Beardwood–Halton–Hammersley theorem for stationary ergodic sequences: a counterexample", *The Annals of Applied Probability*, **26** (4): 2141–2168, doi:10.1214/15-AAP1142

[28] Few, L. (1955). "The shortest path and the shortest road through n points". *Mathematika*. **2** (02): 141–144. doi:10.1112/s0025579300000784.

[29] Steinerberger, S. (2015). "New bounds for the traveling salesman constant". *Advances in Applied Probability*. **47.1**.

[30] Held, M.; Karp, R.M. (1970). "The Traveling Salesman Problem and Minimum Spanning Trees". *Operations Research*. **18**: 1138–1162. doi:10.1287/opre.18.6.1138.

[31] Goemans, M.; Bertsimas, D. (1991). "Probabilistic analysis of the Held and Karp lower bound for the Euclidean traveling salesman problem". *Mathematics of operation research*. **16** (1): 72–89. doi:10.1287/moor.16.1.72.

[32] David S. Johnson

[33] Christine L. Valenzuela and Antonia J. Jones

[34] Orponen (1987)

[35] Papadimitriou (1983)

[36] Christofides (1976)

[37] Karpinski, Lampis & Schmied (2015)

[38] Berman & Karpinski (2006).

[39] Kaplan (2004)

[40] Kosaraju (1994)

[41] Serdyukov (1984)

[42] Hassin (2000)

[43] Macgregor, J. N.; Ormerod, T. (June 1996), "Human performance on the traveling salesman problem", *Perception & Psychophysics*, **58** (4): 527–539, doi:10.3758/BF03213088.

[44] MacGregor, James N.; Chu, Yun (2011), "Human performance on the traveling salesman and related problems: A review", *Journal of Problem Solving*, **3** (2).

[45] *Journal of Problem Solving* 1(1), 2006, retrieved 2014-06-06.

[46] Jones, Jeff; Adamatzky, Andrew (2014), "Computation of the travelling salesman problem by a shrinking blob" (PDF), *Natural Computing*: 2, 13

[47] "Racy read". 7 December 2014. Retrieved 30 April 2016.

[48] "Crime in a World of High Science". 16 September 2014. Retrieved 30 April 2016.

[49] Geere, Duncan. "'Travelling Salesman' movie considers the repercussions if P equals NP". Wired. Retrieved 26 April 2012.

6.1.13 References

- Applegate, D. L.; Bixby, R. M.; Chvátal, V.; Cook, W. J. (2006), *The Traveling Salesman Problem*, ISBN 0-691-12993-2.
- Allender, Eric; Bürgisser, Peter; Kjeldgaard-Pedersen, Johan; Mitersen, Peter Bro (2007), "On the Complexity of Numerical Analysis" (PDF), *SIAM J. Comput.*, **38** (5), doi:10.1137/070697926.
- Arora, Sanjeev (1998), "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems", *Journal of the ACM*, **45** (5): 753–782, doi:10.1145/290179.290180, MR 1668147.
- Beardwood, J.; Halton, J.H.; Hammersley, J.M. (1959), "The Shortest Path Through Many Points", *Proceedings of the Cambridge Philosophical Society*, **55**: 299–327, doi:10.1017/s0305004100034095.
- Bellman, R. (1960), "Combinatorial Processes and Dynamic Programming", in Bellman, R.; Hall, M. Jr., *Combinatorial Analysis, Proceedings of Symposia in Applied Mathematics 10*, American Mathematical Society, pp. 217–249.
- Bellman, R. (1962), "Dynamic Programming Treatment of the Travelling Salesman Problem", *J. Assoc. Comput. Mach.*, **9**: 61–63, doi:10.1145/321105.321111.
- Berman, Piotr; Karpinski, Marek (2006), "8/7-approximation algorithm for (1,2)-TSP", *Proc. 17th ACM-SIAM Symposium on Discrete Algorithms (SODA '06)*, pp. 641–648, doi:10.1145/1109557.1109627, ISBN 0898716055, ECCC TR05-069.
- Christofides, N. (1976), *Worst-case analysis of a new heuristic for the travelling salesman problem*, Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh.
- Hassin, R.; Rubinstein, S. (2000), "Better approximations for max TSP", *Information Processing Letters*, **75** (4): 181–186, doi:10.1016/S0020-0190(00)00097-1.
- Held, M.; Karp, R. M. (1962), "A Dynamic Programming Approach to Sequencing Problems", *Journal of the Society for Industrial and Applied Mathematics*, **10** (1): 196–210, doi:10.1137/0110015.
- Kaplan, H.; Lewenstein, L.; Shafrir, N.; Sviridenko, M. (2004), "Approximation Algorithms for Asymmetric TSP by Decomposing Directed Regular Multigraphs", *In Proc. 44th IEEE Symp. on Foundations of Comput. Sci*, pp. 56–65.

- Karpinski, M.; Lampis, M.; Schmied, R. (2015), "New Inapproximability bounds for TSP", *Journal of Computer and System Sciences*, **81** (8): 1665–1677, doi:10.1016/j.jcss.2015.06.003
- Kosaraju, S. R.; Park, J. K.; Stein, C. (1994), "Long tours and short superstrings'", *Proc. 35th Ann. IEEE Symp. on Foundations of Comput. Sci*, IEEE Computer Society, pp. 166–177.
- Orponen, P.; Mannila, H. (1987), "On approximation preserving reductions: Complete problems and robust measures'", *Technical Report C-1987–28, Department of Computer Science, University of Helsinki*.
- Larson, Richard C.; Odoni, Amedeo R. (1981), "6.4.7: Applications of Network Models § Routing Problems §§ Euclidean TSP", *Urban Operations Research*, Prentice-Hall, ISBN 9780139394478, OCLC 6331426.
- Padberg, M.; Rinaldi, G. (1991), "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems", *Siam Review*: 60–100, doi:10.1137/1033004.
- Papadimitriou, Christos H. (1977), "The Euclidean traveling salesman problem is NP-complete", *Theoretical Computer Science*, **4** (3): 237–244, doi:10.1016/0304-3975(77)90012-3, MR 0455550.
- Papadimitriou, C. H.; Yannakakis, M. (1993), "The traveling salesman problem with distances one and two", *Math. Oper. Res.*, **18**: 1–11, doi:10.1287/moor.18.1.1.
- Serdyukov, A. I. (1984), "An algorithm with an estimate for the traveling salesman problem of the maximum'", *Upravlyaemye Sistemy*, **25**: 80–86.
- Steinerberger, Stefan (2015), "New Bounds for the Traveling Salesman Constant", *Advances in Applied Probability*, **47**.
- Woeginger, G.J. (2003), "Exact Algorithms for NP-Hard Problems: A Survey", *Combinatorial Optimization – Eureka, You Shrink! Lecture notes in computer science, vol. 2570*, Springer, pp. 185–207.

6.1.14 Further reading

- Adleman, Leonard (1994), "Molecular Computation of Solutions To Combinatorial Problems" (PDF), *Science*, **266** (5187): 1021–4, Bibcode:1994Sci...266.1021A, doi:10.1126/science.7973651, PMID 7973651
- Arora, S. (1998), "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems" (PDF), *Journal of the ACM*, **45** (5): 753–782, doi:10.1145/290179.290180.
- Babin, Gilbert; Deneault, Stéphanie; Laportey, Gilbert (2005), *Improvements to the Or-opt Heuristic for the Symmetric Traveling Salesman Problem*, Cahiers du GERAD, G-2005-02, Montreal: Group for Research in Decision Analysis.
- Cook, William (2011), *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*, Princeton University Press, ISBN 978-0-691-15270-7.
- Cook, William; Espinoza, Daniel; Goycoolea, Marcos (2007), "Computing with domino-parity inequalities for the TSP", *INFORMS Journal on Computing*, **19** (3): 356–365, doi:10.1287/ijoc.1060.0204.
- Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. (2001), "35.2: The traveling-salesman problem", *Introduction to Algorithms* (2nd ed.), MIT Press and McGraw-Hill, pp. 1027–1033, ISBN 0-262-03293-7.
- Dantzig, G. B.; Fulkerson, R.; Johnson, S. M. (1954), "Solution of a large-scale traveling salesman problem", *Operations Research*, **2** (4): 393–410, doi:10.1287/opre.2.4.393, JSTOR 166695.
- Garey, M. R.; Johnson, D. S. (1979), "A2.3: ND22–24", *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman, pp. 211–212, ISBN 0-7167-1045-5.
- Goldberg, D. E. (1989), "Genetic Algorithms in Search, Optimization & Machine Learning", *Reading: Addison-Wesley*, New York: Addison-Wesley, Bibcode:1989gaso.book.....G, ISBN 0-201-15767-5.
- Gutin, G.; Yeo, A.; Zverovich, A. (2002), "Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP", *Discrete Applied Mathematics*, **117** (1–3): 81–86, doi:10.1016/S0166-218X(01)00195-0.
- Gutin, G.; Punnen, A. P. (2006), *The Traveling Salesman Problem and Its Variations*, Springer, ISBN 0-387-44459-9.
- Johnson, D. S.; McGeoch, L. A. (1997), "The Traveling Salesman Problem: A Case Study in Local Optimization" (PDF), in Aarts, E. H. L.; Lenstra, J. K., *Local Search in Combinatorial Optimisation*, John Wiley and Sons Ltd, pp. 215–310.
- Lawler, E. L.; Lenstra, J. K.; Rinnooy Kan, A. H. G.; Shmoys, D. B. (1985), *The Traveling Salesman*

Problem: A Guided Tour of Combinatorial Optimization, John Wiley & Sons, ISBN 0-471-90413-9.

- MacGregor, J. N.; Ormerod, T. (1996), "Human performance on the traveling salesman problem" (PDF), *Perception & Psychophysics*, **58** (4): 527–539, doi:10.3758/BF03213088.
- Mitchell, J. S. B. (1999), "Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, *k*-MST, and related problems", *SIAM Journal on Computing*, **28** (4): 1298–1309, doi:10.1137/S0097539796309764.
- Rao, S.; Smith, W. (1998), "Approximating geometrical graphs via 'spanners' and 'banyans'", *Proc. 30th Annual ACM Symposium on Theory of Computing*, pp. 540–550.
- Rosenkrantz, Daniel J.; Stearns, Richard E.; Lewis, Philip M., II (1977), "An Analysis of Several Heuristics for the Traveling Salesman Problem", *SIAM Journal on Computing*, **6** (5): 563–581, doi:10.1137/0206041.
- Vickers, D.; Butavicius, M.; Lee, M.; Medvedev, A. (2001), "Human performance on visually presented traveling salesman problems", *Psychological Research*, **65** (1): 34–45, doi:10.1007/s004260000031, PMID 11505612.
- Walshaw, Chris (2000), *A Multilevel Approach to the Travelling Salesman Problem*, CMS Press.
- Walshaw, Chris (2001), *A Multilevel Lin-Kernighan-Helsgaun Algorithm for the Travelling Salesman Problem*, CMS Press.

6.1.15 External links

- Traveling Salesman Problem at University of Waterloo
- TSPLIB at the University of Heidelberg
- *Traveling Salesman Problem* by Jon McLoone at the Wolfram Demonstrations Project

6.2 Route inspection problem

In graph theory, a branch of mathematics and computer science, the **Chinese postman problem (CPP)**, **postman tour** or **route inspection problem** is to find a shortest closed path or circuit that visits every edge of a (connected) undirected graph. When the graph has an Eulerian circuit (a closed walk that covers every edge once), that circuit is an optimal solution. Otherwise, the optimization problem is to find the smallest number of graph edges to duplicate (or the subset of edges with the minimum possible total weight) so that the resulting multigraph does have an Eulerian circuit.[1] It may be solved in polynomial time.[2]

The problem was originally studied by the Chinese mathematician Kwan Mei-Ko in 1960, whose Chinese paper was translated into English in 1962.[3] The alternative name "Chinese postman problem" was coined in his honor; different sources credit the coinage either to Alan J. Goldman or Jack Edmonds, both of whom were at the U.S. National Bureau of Standards at the time.[4][5]

6.2.1 Undirected solution

The undirected route inspection problem may be solved in polynomial time by an algorithm based on the concept of a T-join. Let T be a subset of the vertex set of a graph. An edge set J is called a **T-join** if the collection of vertices that have an odd number of neighboring edges in J is exactly the set T. A T-join exists whenever every connected component of the graph contains an even number of vertices in T. The **T-join problem** is to find a T-join with the minimum possible number of edges or the minimum possible total weight.

For any T, a smallest T-join (when it exists) necessarily consists of $\frac{1}{2}|T|$ paths that join the vertices of T in pairs. The paths will be such that the total length or total weight of all of them is as small as possible. In an optimal solution, no two of these paths will share any edge, but they may have shared vertices. A minimum T-join can be obtained by constructing a complete graph on the vertices of T, with edges that represent shortest paths in the given input graph, and then finding a minimum weight perfect matching in this complete graph. The edges of this matching represent paths in the original graph, whose union forms the desired T-join. Both constructing the complete graph, and then finding a matching in it, can be done in $O(n^3)$ computational steps.

For the route inspection problem, T should be chosen as the set of all odd-degree vertices. By the assumptions of the problem, the whole graph is connected (otherwise no tour exists), and by the handshaking lemma it has an even number of odd vertices, so a T-join always exists. Doubling the edges of a T-join causes the given graph to become an Eulerian multigraph (a connected graph in which every vertex has even degree), from which it follows that it has an Euler tour, a tour that visits each edge of the multigraph exactly once. This tour will be an optimal solution to the route inspection problem.[6][2]

6.2.2 Directed solution

On a directed graph, the same general ideas apply, but different techniques must be used. If the graph is Eulerian, one need only find an Euler cycle. If it is not,

one must find T-joins, which in this case entails finding paths from vertices with an in-degree greater than their out-degree to those with an out-degree greater than their in-degree such that they would make in-degree of every vertex equal to its out-degree. This can be solved as an instance of the minimum-cost flow problem in which there is one unit of supply for every unit of excess in-degree, and one unit of demand for every unit of excess out-degree. As such it is solvable in $O(|V|^2|E|)$ time. A solution exists if and only if the given graph is strongly connected.[2][7]

6.2.3 Windy postman problem

The **windy postman problem** is a variant of the route inspection problem in which the input is an undirected graph, but where each edge may have a different cost for traversing it in one direction than for traversing it in the other direction. In contrast to the solutions for directed and undirected graphs, it is NP-complete.[8][9]

6.2.4 Applications

Various combinatorial problems are reduced to the Chinese Postman Problem, including finding a maximum cut in a planar graph and a minimum-mean length circuit in an undirected graph.[10]

6.2.5 Variants

A few variants of the Chinese Postman Problem have been studied and shown to be NP-complete.[11]

- Minimize the Chinese postman problem for mixed graphs: for this problem, some of the edges may be directed and can therefore only be visited from one direction. When the problem calls for a minimal traversal of a digraph (or multidigraph) it is known as the "New York Street Sweeper problem."[12]
- Minimize the k-Chinese postman problem: find k cycles all starting at a designated location such that each edge is traversed by at least one cycle. The goal is to minimize the cost of the most expensive cycle.
- Minimize the "Rural Postman Problem": solve the problem with some edges not required. [9]

6.2.6 See also

- Travelling salesman problem

6.2.7 References

[1] Roberts, Fred S.; Tesman, Barry (2009), *Applied Combinatorics* (2nd ed.), CRC Press, pp. 640–642, ISBN 9781420099829

[2] Edmonds, J.; Johnson, E.L. (1973), "Matching Euler tours and the Chinese postman problem", *Mathematical Programming*, **5**: 88–124, doi:10.1007/bf01580113

[3] Kwan, Mei-ko (1960), "Graphic programming using odd or even points", *Acta Mathematica Sinica* (in Chinese), **10**: 263–266, MR 0162630. Translated in *Chinese Mathematics* **1**: 273–277, 1962.

[4] Pieterse, Vreda; Black, Paul E., eds. (September 2, 2014), "Chinese postman problem", *Dictionary of Algorithms and Data Structures*, National Institute of Standards and Technology, retrieved 2016-04-26

[5] Grötschel, Martin; Yuan, Ya-xiang (2012), "Euler, Mei-Ko Kwan, Königsberg, and a Chinese postman" (PDF), Optimization stories: 21st International Symposium on Mathematical Programming, Berlin, August 19–24, 2012, *Documenta Mathematica*, Extra: 43–50, MR 2991468.

[6] Lawler, E.L. (1976), *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston

[7] Eiselt, H. A.; Gendeaeu, Michel; Laporte, Gilbert. "Arc Routing Problems, Part 1: The Chinese Postman Problem". *Operations Research*. INFORMS. pp. 231–242. doi:10.1287/opre.43.2.231. Retrieved 2 December 2014.

[8] Guan, Meigu (1984), "On the windy postman problem", *Discrete Applied Mathematics*, **9** (1): 41–46, doi:10.1016/0166-218X(84)90089-1, MR 754427.

[9] Lenstra, J.K.; Rinnooy Kan, A.H.G. (1981), "Complexity of vehicle routing and scheduling problems", *Networks*, **11** (2): 221–227, doi:10.1002/net.3230110211

[10] A. Schrijver, Combinatorial Optimization, Polyhedra and Efficiency, Volume A, Springer. (2002).

[11] Crescenzi, P.; Kann, V.; Halldórsson, M.; Karpinski, M.; Woeginger, G. "A compendium of NP optimization problems". KTH NADA, Stockholm. Retrieved 2008-10-22. Cite uses deprecated parameter |coauthors= (help)

[12] Roberts, Fred S.; Tesman, Barry (2009), *Applied Combinatorics* (2nd ed.), CRC Press, pp. 642–645, ISBN 9781420099829

6.2.8 External links

- Weisstein, Eric W. "Chinese Postman Problem". *MathWorld*.

6.3 Hamiltonian path problem

This article is about the specific problem of determining whether a Hamiltonian path or cycle exists in a given graph. For the general graph theory concepts, see Hamiltonian path.

In the mathematical field of graph theory the **Hamiltonian path problem** and the **Hamiltonian cycle problem** are problems of determining whether a Hamiltonian path (a path in an undirected or directed graph that visits each vertex exactly once) or a Hamiltonian cycle exists in a given graph (whether directed or undirected). Both problems are NP-complete.[1]

There is a simple relation between the problems of finding a Hamiltonian path and a Hamiltonian cycle. In one direction, the Hamiltonian path problem for graph G is equivalent to the Hamiltonian cycle problem in a graph H obtained from G by adding a new vertex and connecting it to all vertices of G. Thus, finding a Hamiltonian path cannot be significantly slower (in the worst case, as a function of the number of vertices) than finding a Hamiltonian cycle. In the other direction, the Hamiltonian cycle problem for a graph G is equivalent to the Hamiltonian path problem in the graph H obtained by copying one vertex v of G, v', that is, letting v' have the same neighbourhood as v, and by adding two dummy vertices of degree one, and connecting them with v and v', respectively.[2] The Hamiltonian cycle problem is also a special case of the travelling salesman problem, obtained by setting the distance between two cities to one if they are adjacent and two otherwise, and verifying that the total distance travelled is equal to n (if so, the route is a Hamiltonian circuit; if there is no Hamiltonian circuit then the shortest route will be longer).

6.3.1 Algorithms

The first algorithm for finding an Hamiltonian cycle on a directed graph was the enumerative algorithm of Martello.[3] There are $n!$ different sequences of vertices that *might* be Hamiltonian paths in a given n-vertex graph (and are, in a complete graph), so a brute force search algorithm that tests all possible sequences would be very slow. There are several faster approaches. A search procedure by Frank Rubin[4] divides the edges of the graph into three classes: those that must be in the path, those that cannot be in the path, and undecided. As the search proceeds, a set of decision rules classifies the undecided edges, and determines whether to halt or continue the search. The algorithm divides the graph into components that can be solved separately. Also, a dynamic programming algorithm of Bellman, Held, and Karp can be used to solve the problem in time $O(n^2\,2^n)$. In this method, one determines, for each set S of vertices and each vertex v in S, whether there is a path that covers exactly the vertices in S and ends at v. For each choice of S and v, a path exists for (S,v) if and only if v has a neighbor w such that a path exists for $(S-v,w)$, which can be looked up from already-computed information in the dynamic program.[5][6]

Andreas Björklund provided an alternative approach using the inclusion–exclusion principle to reduce the problem of counting the number of Hamiltonian cycles to a simpler counting problem, of counting cycle covers, which can be solved by computing certain matrix determinants. Using this method, he showed how to solve the Hamiltonian cycle problem in arbitrary n-vertex graphs by a Monte Carlo algorithm in time $O(1.657^n)$; for bipartite graphs this algorithm can be further improved to time $o(1.415^n)$.[7]

For graphs of maximum degree three, a careful backtracking search can find a Hamiltonian cycle (if one exists) in time $O(1.251^n)$.[8]

Because of the difficulty of solving the Hamiltonian path and cycle problems on conventional computers, they have also been studied in unconventional models of computing. For instance, Leonard Adleman showed that the Hamiltonian path problem may be solved using a DNA computer. Exploiting the parallelism inherent in chemical reactions, the problem may be solved using a number of chemical reaction steps linear in the number of vertices of the graph; however, it requires a factorial number of DNA molecules to participate in the reaction.[9]

6.3.2 Complexity

The problem of finding a Hamiltonian cycle or path is in FNP; the analogous decision problem is to test whether a Hamiltonian cycle or path exists. The directed and undirected Hamiltonian cycle problems were two of Karp's 21 NP-complete problems. They remain NP-complete even for undirected planar graphs of maximum degree three,[10] for directed planar graphs with in-degree and outdegree at most two,[11] for bridgeless undirected planar 3-regular bipartite graphs, for 3-connected 3-regular bipartite graphs,[12] subgraphs of the square grid graph,[13] and cubic subgraphs of the square grid graph.[14]

However, putting all of these conditions together, it remains open whether 3-connected 3-regular bipartite planar graphs must always contain a Hamiltonian cycle, in which case the problem restricted to those graphs could not be NP-complete; see Barnette's conjecture.

In graphs in which all vertices have odd degree, an argument related to the handshaking lemma shows that the number of Hamiltonian cycles through any fixed edge is always even, so if one Hamiltonian cycle is given, then a second one must also exist.[15] However, finding this second cycle does not seem to be an easy computational task. Papadimitriou defined the complexity class PPA to encapsulate problems such as this one.[16]

6.3.3 References

Media related to Hamiltonian path problem at Wikimedia Commons

[1] Michael R. Garey and David S. Johnson (1979), *Computers and Intractability: A Guide to the Theory of*

NP-Completeness, W.H. Freeman, ISBN 0-7167-1045-5 A1.3: GT37–39, pp. 199–200.

[2] http://math.stackexchange.com/questions/7130/reduction-from-hamiltonian-cycle-to-hamiltonian-path/1290804#1290804

[3] Martello, Silvano (1983), "An Enumerative Algorithm for Finding Hamiltonian Circuits in a Directed Graph", *ACM Transactions on Mathematical Software*, **9** (1): 131–138, doi:10.1145/356022.356030

[4] Rubin, Frank (1974), "A Search Procedure for Hamilton Paths and Circuits", *Journal of the ACM*, **21** (4): 576–80, doi:10.1145/321850.321854.

[5] Bellman, R. (1962), "Dynamic programming treatment of the travelling salesman problem", *Journal of the ACM*, **9**: 61–63, doi:10.1145/321105.321111.

[6] Held, M.; Karp, R. M. (1962), "A dynamic programming approach to sequencing problems", *J. SIAM*, **10** (1): 196–210, doi:10.1137/0110015.

[7] Björklund, Andreas (2010), "Determinant sums for undirected Hamiltonicity", *Proc. 51st IEEE Symposium on Foundations of Computer Science (FOCS '10)*, pp. 173–182, arXiv:1008.0541, doi:10.1109/FOCS.2010.24, ISBN 978-1-4244-8525-3.

[8] Iwama, Kazuo; Nakashima, Takuya (2007), "An improved exact algorithm for cubic graph TSP", *Proc. 13th Annual International Conference on Computing and Combinatorics (COCOON 2007)*, Lecture Notes in Computer Science, **4598**, pp. 108–117, doi:10.1007/978-3-540-73545-8_13, ISBN 978-3-540-73544-1.

[9] Adleman, Leonard (November 1994), "Molecular computation of solutions to combinatorial problems", *Science*, **266** (5187): 1021–1024, doi:10.1126/science.7973651, JSTOR 2885489, PMID 7973651.

[10] Garey, M. R.; Johnson, D. S.; Stockmeyer, L. (1974), "Some simplified NP-complete problems", *Proc. 6th ACM Symposium on Theory of Computing (STOC '74)*, pp. 47–63, doi:10.1145/800119.803884.

[11] Plesńik, J. (1979), "The NP-completeness of the Hamiltonian cycle problem in planar digraphs with degree bound two" (PDF), *Information Processing Letters*, **8** (4): 199–201, doi:10.1016/0020-0190(79)90023-1.

[12] Akiyama, Takanori; Nishizeki, Takao; Saito, Nobuji (1980–1981), "NP-completeness of the Hamiltonian cycle problem for bipartite graphs", *Journal of Information Processing*, **3** (2): 73–76, MR 596313.

[13] Itai, Alon; Papadimitriou, Christos; Szwarcfiter, Jayme (1982), "Hamilton Paths in Grid Graphs", *SIAM Journal on Computing*, **4** (11): 676–686, doi:10.1137/0211056.

[14] Buro, Michael (2000), "Simple Amazons endgames and their connection to Hamilton circuits in cubic subgrid graphs" (PDF), *Conference on Computers and Games*, doi:10.1007/3-540-45579-5_17.

[15] Thomason, A. G. (1978), "Hamiltonian cycles and uniquely edge colourable graphs", *Advances in Graph Theory (Cambridge Combinatorial Conf., Trinity College, Cambridge, 1977)*, Annals of Discrete Mathematics, **3**, pp. 259–268, doi:10.1016/S0167-5060(08)70511-9, ISBN 9780720408430, MR 499124.

[16] Papadimitriou, Christos H. (1994), "On the complexity of the parity argument and other inefficient proofs of existence", *Journal of Computer and System Sciences*, **48** (3): 498–532, doi:10.1016/S0022-0000(05)80063-7, MR 1279412.

Chapter 7

Text and image sources, contributors, and licenses

7.1 Text

- **Seven Bridges of Königsberg** *Source:* https://en.wikipedia.org/wiki/Seven_Bridges_of_K%C3%B6nigsberg?oldid=759230125 *Contributors:* Zundark, Eclecticology, Deb, D, Michael Hardy, Chris-martin, Gabbe, Seav, Den fjättrade ankan~enwiki, Mark Foskey, Bogdangiusca, Berteun, Ed Cormany, Reddi, Dysprosia, Wik, Shizhao, AnonMoos, Jerzy, Robbot, Murray Langton, Fredrik, Donreed, Altenmann, Kneiphof, Bkell, Matt Gies, Giftlite, JamesMLane, Harp, MSGJ, Dratman, Finn-Zoltan, Macrakis, Matthead, Gadfium, Antandrus, DRE, Icairns, Ukexpat, Clubjuggle, Deadlock, Wikiacc, Ascánder, Bender235, Shanes, C S, Blotwell, La goutte de pluie, AllTom, Arthena, Keenan Pepper, Cdc, Americanadian, Oghmoir, Gene Nygaard, Alai, Ghirlandajo, Hq3473, Jftsang, Apokrif, Tabletop, Cbdorsett, Audiovideo, Xiong, Marudubshinki, StefanFuhrmann~enwiki, Graham87, BD2412, Qwertyus, Salix alba, Mkehrt, FlaBot, Gurch, Bgwhite, YurikBot, Wavelength, Hairy Dude, Snillet, Michael Slone, Sikon, Stallions2010, CptnMisc, Arthur Rubin, Cmglee, DVD R W, SmackBot, McGeddon, Stegano~enwiki, Wzhao553, Betacommand, Anachronist, Bird of paradox, Thumperward, DHN-bot~enwiki, Scray, John Reid, LtPowers, John, JLeander, NongBot~enwiki, DaBjork, TheFarix, BranStark, Dilip rajeev, Eyefragment, Courcelles, Stuart Wimbush, CmdrObot, Ivan Pozdeev, Phauly, Nalpdii~enwiki, Nczempin, WLior, Iempleh, Thijs!bot, Headbomb, Lethargy, Gswitz, Seaphoto, Smith2006, Hurmari, JAnDbot, MER-C, CheMechanical, The Anomebot2, David Eppstein, WPaulB, DerHexer, Gwern, LapisQuem, R'n'B, CommonsDelinker, Nev1, Maproom, Smitty, Independentdependent, TXiKiBoT, David Condrey, LFStokols, Falcon8765, Dusti, YonaBot, Hertz1888, Triwbe, Smsarmad, Foljiny, Ctxppc, Ken123BOT, Nic bor, Mikeharris111, Pnijssen, ClueBot, Kl4m, CounterVandalismBot, Piledhigheranddeeper, Excirial, Steveheric, Manu-ve Pro Ski, Jth1994, Addbot, Andunie, Godwin100, Fottry55i6, LinkFABot, Numbo3-bot, Komischn, PV=nRT, Teles, Zorrobot, Luckas-bot, Yobot, AnakngAraw, Ciphers, Materialscientist, Smmlit, ArthurBot, Obersachsebot, Xqbot, Zevyefa, RibotBOT, Tmgreen, Zmorell, Chenopodiaceous, AstaBOTh15, Winterst, MarcelB612, Bmclaughlin9, Serols, Christopher1968, MFrawn, Nascar1996, Deadlyops, WikitanvirBot, TuHan-Bot, Thecheesykid, Cobaltcigs, Donaldm314, Donner60, Scientific29, Orange Suede Sofa, Haythamdouaihy, ClueBot NG, Lord Chamberlain, the Renowned, Vacation9, Santacloud, Ianr790, Athos, MusikAnimal, Cyberbot II, Dexbot, Hmainsbot1, Christallkeks, RockvilleRideOn, Ynaamad, MasterTriangle12, Monkbot, Acekqj, Blois2014, Imdifferentyo123456789, Poppy sheppard and Anonymous: 170
- **Glossary of graph theory** *Source:* https://en.wikipedia.org/wiki/Glossary_of_graph_theory_terms?oldid=754386735 *Contributors:* Damian Yerrick, XJaM, Nonenmac, Tomo, Edward, Patrick, Michael Hardy, Wshun, Chris-martin, Dcljr, TakuyaMurata, GTBacchus, Eric119, Charles Matthews, Dcoetzee, Dysprosia, Doradus, Reina riemann, Markhurd, Maximus Rex, Hyacinth, Populus, McKay, GPHemsley, Altenmann, MathMartin, Bkell, Giftlite, Dbenbenn, Brona, Sundar, GGordonWorleyIII, HorsePunchKid, Peter Kwok, D6, Rich Farmbrough, ArnoldReinhold, Paul August, Bender235, Zaslav, Kjoonlee, Elwikipedista~enwiki, El C, Yitzhak, TheSolomon, A1kmm, 3mta3, Jérôme, Ricky81682, Rdvdijk, Oleg Alexandrov, Joriki, Linas, MattGiuca, Ruud Koot, Jwanders, Xiong, BD2412, Lasunncty, SixWingedSeraph, Grammarbot, Tizio, Salix alba, Mathbot, Margosbot~enwiki, Sunayana, Pojo, Quuxplusone, Vonkje, N8wilson, Chobot, Bgwhite, Algebraist, YurikBot, Me and, Altoid, Grubber, Archelon, Gaius Cornelius, Rick Norwood, Joshlk, Ott2, Closedmouth, SmackBot, Stux, Achab, Brick Thrower, Mgreenbe, Mcld, Michael@nosivad.com, Lansey, Thechao, JLeander, DVanDyck, Quaeler, RekishiEJ, CmdrObot, Csaracho, Citrus538, Jokes Free4Me, Cydebot, Starcrab, Quintopia, Ferris37, Scarpy, Headbomb, Salgueiro~enwiki, Spanningtree, A3nm, David Eppstein, JoergenB, Kope, AlexShkotin, CopyToWiktionaryBot, R'n'B, Leyo, Mikhail Dvorkin, The Transliterator, Ratfox, MentorMentorum, Skaraoke, Dggreen, SanitySolipsism, Anonymous Dissident, PaulTanenbaum, Ivan Štambuk, WereSpielChequers, Whorush, Eggwadi, Thehotelambush, Doc honcho, Anchor Link Bot, Rsdetsch, Denisarona, Justin W Smith, Unbuttered Parsnip, Happynomad, Alexey Muranov, Addbot, Aarond144, Jfitzell, Nate Wessel, Yobot, Jalal0, Ian Kelling, Jim1138, Citation bot, Buenasdiaz, Twri, LilHelpa, Kinewma, Miym, Prunesqualer, Mzamora2, JZacharyG, Pmq20, Shadowjams, Hobsonlane, DixonDBot, Reaper Eternal, EmausBot, John of Reading, Wikipelli, Bethnim, Mastergreg82, ClueBot NG, EmanueleMinotto, Warumwarum, DavidRideout, BG19bot, Andrey.gric, Szebenisz, BattyBot, Millennium bug, ChrisGualtieri, Deltahedron, Jw489kent, Jmerm, Morgoth106, Magriteappleface, SofjaKovalevskaja, Baking Soda, Fmadd, FuzzyMz, Khalpink and Anonymous: 143
- **Graph theory** *Source:* https://en.wikipedia.org/wiki/Graph_theory?oldid=751253810 *Contributors:* AxelBoldt, Kpjas, LC~enwiki, Robert Merkel, Zundark, Taw, Jeronimo, BlckKnght, Dze27, Oskar Flordal, Andre Engels, Karl E. V. Palmen, Shd~enwiki, XJaM, JeLuF, Arvindn, Gianfranco, Matusz, PierreAbbat, Miguel~enwiki, Boleslav Bobcik, FvdP, Camembert, Hirzel, Tomo, Patrick, Chas zzz brown, Michael Hardy, Wshun, Chris-martin, Glinos, Meekohi, Jakob Voss, TakuyaMurata, GTBacchus, Grog~enwiki, Pcb21, Dgrant, CesarB, Looxix~enwiki, Ellywa, Ams80, Ronz, Nanshu, Gyan, Nichtich~enwiki, Mark Foskey, Александър, Poor Yorick, Caramdir~enwiki, Mxn, Charles Matthews, Berteun, Almi, Hbruhn, Dysprosia, Daniel Quinlan, Gutza, Doradus, Zoicon5, Roachmeister, Populus, Zero0000, Doctorbozzball, McKay, Shizhao, Optim, Robbot, Brent Gulanowski, Fredrik, Altenmann, Dittaeva, Gandalf61, MathMartin, Sverdrup, Puckly, KellyCoinGuy, Thesilverbail, Bkell, Paul Murray, Fuelbottle, ElBenevolente, Aknxy, Dina, Tea2min, Giftlite, Dbenbenn, Thv,

The Cave Troll, Elf, Lupin, Brona, Pashute, Duncharris, Andris, Jorge Stolfi, Tyir, Sundar, GGordonWorleyIII, Alan Au, Bact, Knutux, APH, Tomruen, Tyler McHenry, Naerbnic, Peter Kwok, Robin klein, Ratiocinate, Andreas Kaufmann, Chmod007, Madewokherd, Discospinster, Solitude, Guanabot, Qutezuce, Mani1, Paul August, Bender235, Zaslav, Tompw, Diego UFCG~enwiki, Chalst, Shanes, Renice, C S, Csl77, Jojit fb, Photonique, Jonsafari, Obradovic Goran, Tsirel, Jumbuck, Msh210, Alansohn, Liao, Mailer diablo, Marianocecowski, Aquae, Blair Azzopardi, Oleg Alexandrov, Youngster68, Linas, LOL, Ruud Koot, Tckma, Astrophil, Davidfstr, GregorB, SCEhardt, Stochata, Xiong, Graham87, Magister Mathematicae, BD2412, SixWingedSeraph, Rjwilmsi, Gmelli, George Burgess, Eugeneiiim, Arbor, Kalogeropoulos, Fred Bradstadt, FayssalF, FlaBot, PaulHoadley, RexNL, Vonkje, Chobot, Jinma, YurikBot, Wavelength, Michael Slone, Gaius Cornelius, Alex Bakharev, Morphh, SEWilcoBot, Jaxl, Ino5hiro, Xdenizen, Daniel Mietchen, Shepazu, Voidxor, Rev3nant, Lt-wiki-bot, Jwissick, Arthur Rubin, Netrapt, LeonardoRob0t, Agro1986, Eric.weigle, Allens, Sardanaphalus, Melchoir, Brick Thrower, Ohnoitsjamie, Oli Filth, OrangeDog, Taxipom, DHN-bot~enwiki, Tsca.bot, Onorem, GraphTheoryPwns, Lpgeffen, Jon Awbrey, Henning Makholm, Mlpkr, SashatoBot, Whyfish, Disavian, MynameisJayden, Idiosyncratic-bumblebee, Dicklyon, Quaeler, Lanem, Tawkerbot2, Ylloh, Mahlerite, CRGreathouse, Dycedarg, Requestion, Bumbulski, Myasuda, RUVARD, The Isiah, Ntsimp, Abeg92, Corpx, DumbBOT, Anthonynow12, Thijs!bot, Jheuristic, King Bee, Pstanton, Hazmat2, Headbomb, Marek69, Eleuther, AntiVandalBot, Whiteknox, Hannes Eder, Spacefarer, Myanw, JAnDbot, MER-C, The Transhumanist, Igodard, Restname, Sangak, Tmusgrove, Feeeshboy, Usien6, Ldecola, David Eppstein, Kope, DerHexer, Oroso, MartinBot, R'n'B, Uncle Dick, Joespiff, Ignatzmice, Shikhar1986, Tarotcards, Policron, XxjwuxX, Yecril, JohnBlackburne, Dggreen, Anonymous Dissident, Alcidesfonseca, Anna Lincoln, Ocolon, Magmi, PaulTanenbaum, Geometry guy, Fivelittlemonkeys, Sacredmint, Spitfire8520, Radagast3, SieBot, Dawn Bard, Toddst1, Jon har, Bananastalktome, Titanic4000, Beda42, Maxime.Debosschere, Aechase1, Damien Karras, ClueBot, DFRussia, PipepBot, Justin W Smith, Vacio, Wraithful, Garyzx, Mild Bill Hiccup, DragonBot, Fchristo, Hans Adler, Dafyddg, Razorflame, Rmiesen, Kruusamägi, Pugget, Darkicebot, BodhisattvaBot, Dekart, Tangi-tamma, Addbot, Dr.S.Ramachandran, Cerber, DOI bot, Ronhjones, Low-frequency internal, CanadianLinuxUser, Protonk, LaaknorBot, Smoke73, Delaszk, Favonian, Maurobio, Lightbot, Jarble, Ettrig, Luckas-bot, Yobot, Kilom691, Trinitrix, Jean.julius, AnomieBOT, Womiller99, Sonia, Jim1138, Piano non troppo, Gragragra, RandomAct, Citation bot, Ayda D, Xqbot, Jerome zhu, Capricorn42, Nasnema, Miym, GiveAFishABone, RibotBOT, Jalpar75, Aaditya 7, Ankitbhatt, MultiPoly, FrescoBot, Mark Renier, SlumdogAramis, Citation bot 1, Sibian, Maggyero, Pinethicket, RobinK, Wsu-dm-jb, D75304, Wsu-f, Xnn, Obankston, Andrea105, RjwilmsiBot, TjBot, Powerthirst123, Aaronzat, EmausBot, Domesticenginerd, EleferenBot, Jmencisom, Dcirovic, Slawekb, Akutagawa10, D.Lazard, Netha Hussain, Tolly4bolly, ChuispastonBot, EdoBot, ClueBot NG, Wcherowi, Watersmeetfreak, Matthiaspaul, MelbourneStar, Outraged duck, OMurgo, Bazuz, Aks1521, Masssly, Joel B. Lewis, Johnsopc, HMSSolent, 4368a, BG19bot, Ajweinstein, Канеюку, MusikAnimal, AvocatoBot, Bereziny, Brad7777, Sofia karampataki, ChrisGualtieri, GoShow, Dexbot, DaltonCastle, Cerabot~enwiki, Omgigotanaccount, Wikiisgreat123, Faizan, Maxwell bernard, Bg9989, Zsoftua, SakeUPenn, Yloreander, StaticElectricity, Gold4444, Cyborgbadger, Monkbot, Zachwaltman, Gr pbi, Jdcomix, KasparBot, Lr0^^k, Meachamus.Prime, Baking Soda, GreenC bot, Ywang416, Anish karimaloor, Jetroberts, Mitsou~dewiki and Anonymous: 403

- **Element (mathematics)** *Source:* https://en.wikipedia.org/wiki/Element_(mathematics)?oldid=751284446 *Contributors:* Michael Hardy, Charles Matthews, Hyacinth, Chuunen Baka, Tea2min, Giftlite, Dbenbenn, Maximaximax, Eep^2, Brianjd, Dissipate, Paul August, Elwikipedista~enwiki, Nickj, Mlm42, Oleg Alexandrov, Mutford, Rjwilmsi, Salix alba, Chobot, YurikBot, RobotE, Thane, Goffrie, InverseHypercube, Mhss, Octahedron80, Gracenotes, Khoikhoi, MichaelBillington, Wvbailey, Bjankuloski06en~enwiki, SpyMagician, 16@r, Lim Wei Quan, Mets501, JohnCD, Gregbard, Cydebot, Epbr123, CZeke, The Transhumanist, Enlil2, Zorro CX, KeypadSDM, Fruits Monster, Idioma-bot, Pleasantville, Meters, SieBot, Paolo.dL, Harry-, Gamall Wednesday Ida, Mx. Granger, ClueBot, Phileasson, Heckledpie, SoxBot III, RMFan1, Addbot, Professor Calculus, Betterusername, Binary TSO, Ben Ben, Yobot, TaBOT-zerem, KamikazeBot, Ciphers, Zach2231, Materialscientist, Citation bot, ArthurBot, DSisyphBot, FrescoBot, AllCluesKey, Spartan S58, Pinethicket, Jauhienij, Jophos, Rsagira58, EmausBot, Solarra, Traxs7, Ebrambot, ChuispastonBot, ClueBot NG, Wcherowi, SusikMkr, Solomon7968, Atomician, Wikih101, Funnydiamond99, Aciganj, YFdyh-bot, Saehry, Stephan Kulla, Melonkelon, Justin15w, KH-1, Jwall42, Sonicbethesame, Zacc woes, Jarred Roker, Wikishovel and Anonymous: 88
- **Path (graph theory)** *Source:* https://en.wikipedia.org/wiki/Path_(graph_theory)?oldid=762096259 *Contributors:* Chris-martin, Charles Matthews, Dcoetzee, Olego, Dysprosia, Doradus, Zero0000, GPHemsley, Altenmann, MathMartin, Tea2min, Giftlite, MSGJ, SWAdair, Neilc, Vadmium, Gdr, DRE, Chmod007, Rich Farmbrough, Paul August, Jonon, Derbeth, Oleg Alexandrov, Joriki, Linas, Daira Hopwood, BD2412, Adking80, Maxal, Jittat~enwiki, YurikBot, Bota47, Macskeeball, Modify, SmackBot, Gelingvistoj, Tsca.bot, ElommolE, Yaninass2, CBM, Flamholz, Thijs!bot, Djr36, Stannered, VictorAnyakin, JAnDbot, Cowrider, David Eppstein, TXiKiBoT, Ashesh0326, Jamelan, Svick, Justin W Smith, Ideal gas equation, Addbot, Allliam, Yobot, TaBOT-zerem, Vini 17bot5, Ian Kelling, Citation bot, Twri, Xqbot, Serkan Kenar, Miym, SassoBot, Eomund, FrescoBot, Alejandro Erickson, ClueBot NG, KLBot2, Paolo Lipparini, Solomon7968, BattyBot, DarafshBot, Mog333, David9550, Werddemer, Matroids, Npip99, Bender the Bot and Anonymous: 18
- **Graph (discrete mathematics)** *Source:* https://en.wikipedia.org/wiki/Graph_(discrete_mathematics)?oldid=761702964 *Contributors:* The Anome, Manning Bartlett, XJaM, Tomo, Stevertigo, Patrick, Michael Hardy, W~enwiki, Zocky, Wshun, Chris-martin, Karada, Ahoerstemeier, Den fjättrade ankan~enwiki, Jiang, Dcoetzee, Dysprosia, Doradus, Zero0000, McKay, BenRG, Robbot, LuckyWizard, Mountain, Altenmann, Mayooranathan, Gandalf61, MathMartin, Timrollpickering, Bkell, Tea2min, Tosha, Giftlite, Dbenbenn, Harp, Tom harrison, Chinasaur, Jason Quinn, Matt Crypto, Neilc, Erhudy, Knutux, Yath, Joeblakesley, Tomruen, Peter Kwok, Aknorals, Chmod007, Abdull, Corti, PhotoBox, Discospinster, Rich Farmbrough, Andros 1337, Paul August, Bender235, Zaslav, Gauge, Tompw, Crisófilax, Yitzhak, Kine, Bobo192, Jpiw~enwiki, Mdd, Jumbuck, Zachlipton, Sswn, Liao, Rgclegg, Paleorthid, Super-Magician, Mahanga, Joriki, Mindmatrix, Wesley Moy, Oliphaunt, Brentdax, Jwanders, Tbc2, Cbdorsett, Ch'marr, Davidfstr, Xiong, Marudubshinki, Tslocum, Magister Mathematicae, Ilya, BD2412, SixWingedSeraph, Sjö, Rjwilmsi, Salix alba, Bhadani, FlaBot, Nowhither, Mathbot, Gurch, MikeBorkowski~enwiki, Chronist~enwiki, Silversmith, Chobot, Peterl, Siddhant, Borgx, Karlscherer3, Hairy Dude, Gene.arboit, Michael Slone, Gaius Cornelius, Rsrikanth05, Shanel, Gwaihir, Dtrebbien, Dureo, Doetoe, Wknight94, Arthur Rubin, Netrapt, RobertBorgersen, Cjfsyntropy, RonnieBrown, Burnin1134, SmackBot, Nihonjoe, Stux, McGeddon, BiT, Algont, Ohnoitsjamie, Chris the speller, Bluebot, TimBentley, Theone256, Cornflake pirate, Zven, Anabus, Can't sleep, clown will eat me, Tamfang, Cybercobra, Jon Awbrey, Kuru, Nat2, Tomhubbard, RichMorin, Dicklyon, Cbuckley, Quaeler, BranStark, Wandrer2, George100, Ylloh, Avigreen18, Vaughan Pratt, Repied, CRGreathouse, Citrus538, Jokes Free4Me, Requestion, Myasuda, Danrah, Robertsteadman, Eric Lengyel, Headbomb, Urdutext, AntiVandalBot, Hannes Eder, JAnDbot, MER-C, Dreamster, Struthious Bandersnatch, JNW, Catgut, David Eppstein, JoergenB, MartinBot, Rettetast, R'n'B, J.delanoy, Hans Dunkelberg, Yecril, Pafcu, Ijdejter, Deor, ABF, Maghnus, TXiKiBoT, Sdrucker, Someguy1221, PaulTanenbaum, Lambyte, Ilia Kr., Jpeeling, Falcon8765, RaseaC, Insanity Incarnate, Zenek.k, Radagast3, Debamf, Debeolaurus, SieBot, Minder2k, Dawn Bard, Cwkmail, Jon har, SophomoricPedant, Oxymoron83, Henry Delforn (old), Ddxc, Svick, Phegyi81, Anchor Link Bot, Jarauh, ClueBot, Vacio, Nsk92, JuPitEer, Huynl, JP.Martin-Flatin, Xavexgoem, UKoch, Mitmaro, Editor70, Watchduck, Hans Adler, Suchap, Wikidsp, Muro Bot, 3ICE, Aitias, Versus22, Djk3, Kruusamägi, SoxBot III, XLinkBot, Marc van Leeuwen, Libcub, WikiDao, Tangi-tamma, Addbot, Gutin, Athenray, Willking1979, Royerloic, West.andrew.g, Tyw7, Zorrobot, LuK3, Luckas-bot, Yobot, TaBOT-zerem, THEN WHO

WAS PHONE?, E mraedarab, Tempodivalse, Пика Пика, Ulric1313, RandomAct, Materialscientist, Twri, Dockfish, Anand jeyahar, Miym, Prunesqualer, Andyman100, VictorPorton, JonDePlume, Shadowjams, A.amitkumar, Kracekumar, Edgars2007, Citation bot 1, Maggyero, DrilBot, Amintora, Pinethicket, Calmer Waters, RobinK, Barras, Tgv8925, DARTH SIDIOUS 2, Powerthirst123, DRAGON BOOSTER, Mymyhoward16, Kerrick Staley, Ajraddatz, Wgunther, Dcirovic, Bethnim, Akutagawa10, White Trillium, Josve05a, D.Lazard, L Kensington, Maschen, Inka 888, Chewings72, ClueBot NG, Wcherowi, MelbourneStar, Kingmash, O.Koslowski, Joel B. Lewis, Andrewsky00, Timflutre, Helpful Pixie Bot, HMSSolent, BG19bot, Grolmusz, John Cummings, Stevetihi, Канеюку, Void-995, MRG90, Vanischenu, Tman159, BattyBot, Ekren, Lugia2453, Jeff Erickson, CentroBabbage, Nina Cerutti, Chip Wildon Forster, Yloreander, Manul, JaconaFrere, Monkbot, Hou710, Anon124, Aryan5496, Dr.basheer09, Jean Raimbault, SlvrKy, Fmadd and Anonymous: 365

- **Directed graph** *Source:* https://en.wikipedia.org/wiki/Directed_graph?oldid=761330360 *Contributors:* Michael Hardy, Chris-martin, Furykef, Altenmann, Bkell, Giftlite, Vadmium, Andreas Kaufmann, Zaslav, Rgdboer, Grue, Linas, BD2412, SixWingedSeraph, Chobot, Wavelength, Stepa, WookieInHeat, BiT, Nbarth, Delcypher, Meno25, Was a bee, Kozuch, Headbomb, Hamaryns, JAnDbot, Catgut, David Eppstein, R'n'B, Marcuse7~enwiki, M-le-mot-dit, Llorenzi, VolkovBot, HughD, Constant314, TXiKiBoT, Shauncutts, PaulTanenbaum, Rinix, Henry Delforn (old), Justin W Smith, JP.Martin-Flatin, Brews ohare, NuclearWarfare, MystBot, Addbot, Jarble, Luckas-bot, Ptbotgourou, Pcap, Calle, Dzied Bulbash, Bryan.burgers, Twri, Ms.wiki.us, Xqbot, آرمان, Anne Bauval, Miym, Corruptcopper, Einkil, Mark Renier, Ricardo Ferreira de Oliveira, Maggyero, Pierre5018, EmausBot, WikitanvirBot, Dcirovic, Sinuhe20, ChuispastonBot, Umesh.wankhede, Joerg Bader, Helpful Pixie Bot, Alesiom, Waflix, Mervat Salman, Mark viking, Saranavan, Werddemer, SofjaKovalevskaja, Matiia, Esponenziale, Boomer Vial, RobbieIanMorrison, Queroliita, Elirankoren and Anonymous: 42
- **Complete graph** *Source:* https://en.wikipedia.org/wiki/Complete_graph?oldid=752046343 *Contributors:* AxelBoldt, Andre Engels, XJaM, PierreAbbat, Imran, Tomo, Michael Hardy, Chris-martin, Dominus, Gabbe, Ejrh, Dcoetzee, Dysprosia, McKay, AnonMoos, MathMartin, DHN, HaeB, Giftlite, Dbenbenn, Lupin, Bact, Knutux, Tomruen, Kelson, Zaslav, Obradovic Goran, Mailer diablo, WojciechSwiderski~enwiki, Mindmatrix, Isnow, Salix alba, Fred Bradstadt, FlaBot, DVdm, Alpt, Gustavb, Bota47, Arthur Rubin, SmackBot, Maksim-e~enwiki, Incnis Mrsi, Mgreenbe, BiT, Cool3, Kostmo, 16@r, CRGreathouse, N2e, Coe McSweet, Thijs!bot, Gcm, Magioladitis, Nyttend, David Eppstein, GermanX, Anton Khorev, Koko90, Fridemar, Yecril, VolkovBot, Broadbot, Radagast3, Da Joe, KoenDelaere, Thehotelambush, ClueBot, Bender2k14, Addbot, EdPeggJr, Luckas-bot, TaBOT-zerem, AnomieBOT, Twri, DSisyphBot, Omnipaedista, Howard McCay, DoostdarWKP, MastiBot, Professor Fiendish, 4, R. J. Mathar, Wild Lion, ClueBot NG, El Roih, ChrisGualtieri, JYBot, நடோஞ்சழியன், InternetArchiveBot, Bender the Bot and Anonymous: 35
- **Tree (graph theory)** *Source:* https://en.wikipedia.org/wiki/Tree_(graph_theory)?oldid=755598455 *Contributors:* AxelBoldt, Bryan Derksen, XJaM, Miguel~enwiki, Edemaine, Patrick, Michael Hardy, Chris-martin, Dominus, Rp, Poor Yorick, Cherkash, Charles Matthews, Timwi, Andrevan, Dysprosia, Jitse Niesen, Doradus, Zero0000, McKay, Josh Cherry, Jaredwf, Altenmann, Kuszi, MathMartin, Tea2min, Tosha, Giftlite, Dbenbenn, Smjg, MSGJ, Neilc, Knutux, Zarvok, Almit39, Karl-Henner, Naerbnic, Andreas Kaufmann, PhotoBox, Gadykozma, Paul August, Bender235, Zaslav, Kundor, Obradovic Goran, Mdd, Hippophaë~enwiki, Lssilva, Angr, Linas, Kzollman, OdedSchramm, Adking80, Tizio, Salix alba, Arunkumar, Nguyen Thanh Quang, Maxal, Gurch, Algebraist, Debivort, Xcelerate, Michael Slone, Hyad, Bhny, Trovatore, Lt-wiki-bot, SmackBot, Sori, Bluebot, Nbarth, Vanished User 0001, Battamer, Sms97, Jon Awbrey, Lambiam, Breno, Wrstark, Physis, George100, Ylloh, Mudd1, Ezrakilty, Yaris678, Pjvpjv, Mdotley, Salgueiro~enwiki, Hermel, JAnDbot, Uv1234, 3D-Grabber, David Eppstein, Inhumandecency, Airbornemihir, RaitisMath, MartinBot, R'n'B, N4nojohn, Mikhail Dvorkin, Policron, Sternkampf, Anonymous Dissident, Anna Lincoln, PaulTanenbaum, Spinningspark, SieBot, Cwkmail, Philly jawn, Justin W Smith, JP.Martin-Flatin, Dagh, Philippe Giabbanelli, Baudway, Libcub, Addbot, Topology Expert, Marsupilcoalt, NjardarBot, سعی, Yobot, AnomieBOT, Citation bot, Twri, ArthurBot, Miym, GrouchoBot, Nicolas Perrault III, StaticVision, Citation bot 1, Maggyero, Ioakar, RedBot, DixonDBot, Xnn, EmausBot, ClueBot NG, Edelac, Helpful Pixie Bot, BG19bot, TrickSterWolf, AvocatoBot, Solomon7968, PsichoPuzo, StarryGrandma, Victor Yus, Purdygb, Illia Connell, Frosty, Telfordbuck, Garfield Garfield, JMP EAX, JQTriple7, Kelvin Santarita, Zhirose, SentientSeaTurtle, CINNAMONSWEETZ and Anonymous: 94
- **Multigraph** *Source:* https://en.wikipedia.org/wiki/Multigraph?oldid=756726665 *Contributors:* Silverfish, McKay, Jerzy, Altenmann, Giftlite, Michael Devore, Robert Weemeyer, Paul August, Kwamikagami, EmilJ, Arthena, Shreevatsa, Bluemoose, BD2412, SixWingedSeraph, Rjwilmsi, Mike Segal, Nihiltres, Batztown, YurikBot, Gaius Cornelius, Modify, SmackBot, Vaughan Pratt, Myasuda, Sopoforic, Colin Rowat, Headbomb, Futurebird, Guy Macon, David Eppstein, R'n'B, Roly Perera, PaulTanenbaum, JP.Martin-Flatin, WestwoodMatt, Farisori, Hans Adler, Hasteur, Dwiddows, Qwfp, Webonfim, Anticipation of a New Lover's Arrival, The, Addbot, مانی, Yobot, Danno uk, Citation bot, Twri, Shadowjams, Citation bot 1, Kiefer.Wolfowitz, Redrobsche, 0x24a537r9, Chaotic iak, Accelerometer, MerlIwBot, Helpful Pixie Bot, Alistairwindsor, Lathsim, Solomon7968, Eternalko, Stigmatella aurantiaca, Nicholasbarry, Jerodlycett, Bender the Bot and Anonymous: 20
- **Extremal graph theory** *Source:* https://en.wikipedia.org/wiki/Extremal_graph_theory?oldid=702634168 *Contributors:* William Avery, Andris, BD2412, Quuxplusone, Nethgirb, Davepape, CRGreathouse, Hardmath, Hermel, David Eppstein, JoergenB, Gwern, STBot, JohnBlackburne, Leen Droogendijk, Addbot, Xqbot, RobinK, ExtremalGraphTheory, WikitanvirBot, Brad7777, Brirush and Anonymous: 15
- **Minimum spanning tree** *Source:* https://en.wikipedia.org/wiki/Minimum_spanning_tree?oldid=758735872 *Contributors:* Magnus Manske, LC~enwiki, Timo Honkasalo, Boleslav Bobcik, Michael Hardy, Chris-martin, David Martland, Nixdorf, Shyamal, Julesd, Jeff abrahamson, Poor Yorick, Hike395, Dcoetzee, Dysprosia, Jogloran, Itai, McKay, Pakaran, Robbot, Altenmann, Romanm, MathMartin, DHN, LX, Giftlite, WiseWoman, Levin, Pashute, Andris, Andreas Kaufmann, Superninja, Daf, Andrewbadr, Haham hanuka, Eric Kvaalen, Keenan Pepper, Aranae, Wtmitchell, K3rb, Zzen, Kenyon, Oleg Alexandrov, Mindmatrix, LOL, 25or6to4, Waldir, Leemeng, BD2412, Qwertyus, Rjwilmsi, Mavroprovato, B6s~enwiki, Miserlou, GeorgeBills, DevastatorIIC, King of Hearts, Chobot, YurikBot, Michael Slone, Cedar101, Cojoco, Mebden, SmackBot, RDBury, Wzhao553, Chris the speller, Bluebot, Mbclimber, Tomekpe, Wladston, Christian Thiemann, Hiiiiiiiiiiiiiiiiiii, Duckax, Pqrstuv, Evoluzion~enwiki, CRGreathouse, CmdrObot, Ladida, Kallerdis, Sytelus, Www.cgal.org, Headbomb, PeterDz, Escarbot, TuvicBot, Vitalyr, Rami R, Americanhero, David Eppstein, Lyle lee, Ftiercel, R'n'B, J.delanoy, Blueharmony, Musically ut, Michael Angelkovich, Asymmetric, Philip Trueman, TXiKiBoT, Orie0505, Jgarrett, Alangowitz, Coverback, King rhoton, Aednichols, Nicecupotea, Juniuswikia, Oxymoron83, Lourakis, Hariva, ClueBot, Obelix83, Bender2k14, Sun Creator, Thehelpfulone, Drsadeq, KKoolstra, Addbot, Alex.mccarthy, Download, Amit kumar22, Turinghasaposse, Lightbot, Yobot, Vevek, Erel Segal, Lynxoid84, AdjustShift, Citation bot, XViStA, Addihockey10, Miym, George94, Shmomuffin, HsuanHan, Luisbona, LSG1, LSG1-Bot, X7q, Akashssp, Sindhu sundar, IRudyak, Mohitsinghr, Jan Wassenberg, DARTH SIDIOUS 2, RjwilmsiBot, EmausBot, Q309185, Perfectionatic, K6ka, Osman-pasha, Eda eng, Swfung8, Voomoo, ClueBot NG, Ejarendt, Movses-bot, Frietjes, Matpiw, Tricorne~enwiki, GKFX, DLUrner, Hamish59, Kyle.cackett, Jerry Hintze, Dexbot, Akerbos, Me, Myself, and I are Here, 7804j, Tylercusack, Vivek pandita, Yuxwiki, Monkbot, Zackchase, Bobintornado, HelpUsStopSpam, Marshalf, JueLinLi and Anonymous: 166
- **Steiner tree problem** *Source:* https://en.wikipedia.org/wiki/Steiner_tree_problem?oldid=749150235 *Contributors:* Bryan Derksen, Timo Honkasalo, The Anome, Michael Hardy, W~enwiki, Dcoetzee, Dysprosia, Moncrief, Altenmann, Giftlite, Mellum, Andris, Amoss, Andreas

Kaufmann, Bender235, Mindmatrix, BD2412, Rjwilmsi, Maxal, Banazir, Sbrools, Rbarreira, AndrewWTaylor, SmackBot, RDBury, Davepape, Melchoir, Eskimbot, Commander Keane bot, Bluebot, GoodDay, Ohconfucius, J. Finkelstein, Gandalfxviv, Beetstra, NaBUru38, Miyomiyo1050, David Eppstein, WhyTanFox, Krishnachandranvn, VolkovBot, Moltean, WereSpielChequers, Justin W Smith, Daveagp, C. lorenz, Addbot, Lightbot, يمانم, Ben Ben, Yobot, TaBOT-zerem, Citation bot, Twri, Xqbot, Miym, Citation bot 1, RobinK, Gshaham, RjwilmsiBot, EmausBot, WikitanvirBot, Dewritech, Klbrain, ZéroBot, TomYHChan, Eda eng, Alexeytuzhilin, Arthur L Edwards, Sambayless, Helpful Pixie Bot, Emanuele.paolini, StarryGrandma, Pintoch, Monkbot, Zackchase, Gronk Oz, Aoiva, Wiki randomhuman, Danielrehfeldt and Anonymous: 63

- **Shortest path problem** *Source:* https://en.wikipedia.org/wiki/Shortest_path_problem?oldid=761330091 *Contributors:* AxelBoldt, General Wesc, LC~enwiki, The Anome, Shd~enwiki, JeLuF, B4hand, Michael Hardy, Chris-martin, Ijon, Dcoetzee, Dysprosia, Furrykef, Pigorsch, Altenmann, Lzur, Sho Uemura, Giftlite, Wolfkeeper, BenFrantzDale, Andris, Lqs, OverlordQ, Andreas Kaufmann, Rich Farmbrough, Rasmusdf, DcoetzeeBot~enwiki, ESkog, TerraFrost, Brian0918, MisterSheik, Caesura, Oleg Alexandrov, Nuno Tavares, Mindmatrix, Camw, Oliphaunt, Ruud Koot, BD2412, Qwertyus, Rjwilmsi, Xperimental~enwiki, Mathbot, Mathiastck, Choess, Kri, Cthe, Chobot, Phelanpt, Chris Capoccia, Gaius Cornelius, Anomie, Nethgirb, Lt-wiki-bot, Cedar101, Treesmill, SmackBot, Mcld, Ohnoitsjamie, DHN-bot~enwiki, Tommyjb, Lpgeffen, RomanSpa, MadScientistVX, Optakeover, Graph Theory page blanker, Devis, Headbomb, Magioladitis, Squire55, David Eppstein, Kope, Glrx, R'n'B, Edepot, Essess, Dmforcier, Alcidesfonseca, Aaron Rotenberg, Alfredo J. Herrera Lago, Kbrose, SieBot, Jan Winnicki, Taemyr, Lourakis, Jdaloner, Hariva, Denisarona, Justin W Smith, Metaprimer, Cairomax, Daveagp, Johnuniq, Marc van Leeuwen, C. lorenz, Addbot, Fgnievinski, Jason.surratt, Download, Delaszk, Cipher1024, WikiDreamer Bot, Jarble, Luckas-bot, Yobot, AnomieBOT, Erel Segal, Materialscientist, Citation bot, Xqbot, Alexander Anoprienko, Pmlineditor, Suzhouwuyue, Captain-n00dle, Deanphd, Citation bot 1, RedBot, Serols, Robert Geisberger, RobinK, Mjs1991, Trappist the monk, MoreNet, Horcrux92, ToneDaBass, John of Reading, WikitanvirBot, Super48paul, RA0808, Hari6389, Fæ, Mkroeger, Templatetypedef, ClueBot NG, MiroBrada, Nullzero, BG19bot, Marcelkcs, Happyuk, BattyBot, Jerry Hintze, Jochen Burghardt, Simonpratt, Amine.marref, HoboMcJoe, Ginsuloft, Robmccoll, Artyom Kalinin, Yacs, Monkbot, JMP EAX, Boky90, HelpUsStopSpam, KasparBot, CAPTAIN RAJU, Danieloliveira56, Mikkel2thorup, Bahaabadi, Bender the Bot and Anonymous: 128
- **Dijkstra's algorithm** *Source:* https://en.wikipedia.org/wiki/Dijkstra'{}s_algorithm?oldid=760769431 *Contributors:* AxelBoldt, LC~enwiki, Css, Shd~enwiki, Matusz, Edemaine, Ezubaric, Someone else, Michael Hardy, Nixdorf, Kku, Cyde, Julesd, Aragorn2, Cema, Hashar, Charles Matthews, Timwi, Dcoetzee, Dysprosia, Gutza, Hao2lian, Itai, Csurguine, Shizhao, Owen, Quidquam, Jaredwf, Altenmann, MathMartin, Bkell, Hadal, Wildcat dunny, Clementi, Decrypt3, Giftlite, Christopher Parham, BenFrantzDale, Tesse, Brona, Robert Southworth, Leonard G., AJim, Guanaco, Sundar, Esrogs, MarkSweep, Watcher, RISHARTHA, Gerrit, MementoVivere, Kooo, Kndiaye, ZeroOne, BACbKA, Diego UFCG~enwiki, Nicholasbishop, Vecrumba, RoyBoy, Aydee, Ewedistrict, Foobaz, Jellyworld, Quill18, Obradovic Goran, Haham hanuka, 4v4l0n42, HasharBot~enwiki, Lawpjc, Terrycojones, Jeltz, B3virq3b, Velella, Mikeo, K3rb, LunaticFringe, Yurivict, Oleg Alexandrov, Mahanga, ProBoj!, Shreevatsa, LOL, Dandv, Oliphaunt, Danmaz74, Jacobolus, MattGiuca, Drostie, Pol098, Ruud Koot, Noogz, GregorB, Dionyziz, Agthorr, Kesla, Graham87, Qwertyus, Laurinkus, Grammarbot, Rjwilmsi, Assimil8or~enwiki, Dosman, B6s~enwiki, TheRingess, Eric Burnett, JanSuchy, Grantstevens, Mathiastck, Jorvis, Choess, Fresheneesz, King of Hearts, Chobot, Bgwhite, FrankTobia, YurikBot, Wavelength, Borgx, Angus Lepper, Hairy Dude, Michael Slone, Wierdy1024, CambridgeBayWeather, Pseudomonas, Zhaladshar, Anog, Shizny, RFBailey, Nethgirb, Tomisti, Sarkar112, Abu adam~enwiki, Zr2d2, GraemeL, Alanb, HereToHelp, DoriSmith, Sidonath~enwiki, NetRollller 3D, Thijswijs, Dudzcom, SmackBot, McGeddon, KocjoBot~enwiki, Mgreenbe, Gilliam, Optikos, Gaiacarra, Oli Filth, MalafayaBot, Kostmo, DHN-bot~enwiki, Frap, GRuban, B^4, Ryan Roos, Illnab1024, Daniel.Cardenas, Ycl6, Tobei, Slakr, SQGibbon, Scorintha, MTSbot~enwiki, RamiWissa, Norm mit, BranStark, Iridescent, Paul Koning, Lavaka, Joniscool98, JForget, Ahy1, Ezrakilty, VTBassMatt, Arrenlex, Huazheng, T0ljan~enwiki, Sytelus, Boemanneke, ThomasGHenry, Thijs!bot, RodrigoCamargo, Crazy george, Sprhodes, Williamyf, WikiSlasher, AntiVandalBot, Behco, Mccraig, Spencer, Dougher, Deflective, Harish victory, Gordonnovak, Jheiv, Mwarren us, Radim Baca, JBocco, SiobhanHansa, Rami R, Stdazi, B3N, David Eppstein, Martynas Patasius, PoliticalJunkie, Piojo, Obscurans, Yonidebot, Ryanli, AlcoholVat, Turketwh, Joelimlimit, Bcnof, Sk2613, BernardZ, PesoSwe, JohnBlackburne, Andreasneumann, Soytuny, TXiKiBoT, Kjmitch, MusicScience, Dmforcier, Mcculley, Milcke~enwiki, Mkw813, Andy Dingley, Kbkorb, AlleborgoBot, Mameisam, Davekaminski, Rhanekom, Subh83, SieBot, Wphamilton, Adamarnesen, Sephiroth storm, Keilana, Digwuren, Quest for Truth, Beatle Fab Four, Gerel, Ctxppc, Svick, AlanUS, Sokari, ClueBot, Justin W Smith, Foxj, CGamesPlay, Pskjs~enwiki, Cicconetti, Adamianash, Nanobear~enwiki, Daveagp, Vermooten, Coralmizu, ElonNarai, Dr.Koljan, Peatar, Peasaep, Sniedo, DumZiBoT, XLinkBot, SilvonenBot, Hell112342, Apalamarchuk, SteveJothen, Addbot, DarrylNester, Alquantor, Alex.mccarthy, Jason.Rafe.Miller, KorinoChikara, MrOllie, Herry12, Torla42, AgadaUrbanit, Vwm, Matěj Grabovský, Jarble, Luckas-bot, ZX81, Yobot, Vevek, AnomieBOT, Erel Segal, Arjun G. Menon, Rubinbot, Jim1138, Galoubet, Materialscientist, 90 Auto, Citation bot, ArthurBot, Amenel, Airalcorn2, Quintus314, LordArtemis, Crefrog, Davub, Jongman.koo, Thayts, Geron82, X7q, Recognizance, Ibmua, D'ohBot, Jewillco, Shuroo, I dream of horses, Frankrod44, Jonesey95, The Arbiter, Skyerise, MondalorBot, Yutsi, Mikrosam Akademija 2, Merlion444, Dmitri666, Cincoutprabu, Faure.thomas~enwiki, EmausBot, Dreske, Pixelu, Kh naba, Blueshifting, Wikipelli, Pshanka, Pxtreme75, Woshiqiqiye, Allan speck, Sheepeatgrass, Venkatarun95, ClueBot NG, Muon, Sambayless, Helpful Pixie Bot, BG19bot, Mr.TAMER.Shlash, AdamTReineke, Happyuk, Pilode, Arsstyleh, BattyBot, Xerox 5B, IkamusumeFan, Trunks175, Tehwikipwnerer, IgushevEdward, Thom2729, Megharajv, Lone boatman, MindAfterMath, Jochen Burghardt, Aladdin.chettouh, Kyousuke.k, Seanhalle, Ryangerard, I am One of Many, Olivernina, Chehabz, Gauravxpress, Quenhitran, Juliusz Gonera, Alyssaq, Yujianzhao, Monkbot, Thegreekgonzo, KH-1, WillemienH, Kanargias, Ldthai, Zairwolf, Giovatardu, Prakashmeansvictory, Z5eacom, Kupferhirn, Nbro, Mark Schröder, Jackson tale, Sro23, Mikymaione, Bo.chen.cool, Trianam, Drishti Wali, Tzanger, Dirkjhogan, Marble machine, Shiyu Ji, Culturefanatic12, Vampamp, Tusharsoni099, Karspider and Anonymous: 542
- **Bellman–Ford algorithm** *Source:* https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm?oldid=759185442 *Contributors:* Tomo, Michael Hardy, Phoe6, Docu, Ciphergoth, Poor Yorick, Charles Matthews, Dcoetzee, Itai, Fvw, Mazin07, Jaredwf, Fredrik, Altenmann, Bkell, Enochlau, Giftlite, BenFrantzDale, Brona, Stern~enwiki, Andris, Wmahan, Gadfium, Sam Hocevar, Rspeer, Orbst, Jellyworld, Helix84, HasharBot~enwiki, B3virq3b, Pion, Oleg Alexandrov, Brookie, Stderr.dk, LOL, BlankVerse, Ruud Koot, GregorB, Waldir, Agthorr, Sigkill, Qwertyus, Rjwilmsi, Salix alba, Ucucha, FlaBot, Ecb29, Mathbot, Nihiltres, Jftuga, AlexCovarrubias, Quuxplusone, Istanton, CiaPan, Chobot, FrankTobia, Roboto de Ajvol, YurikBot, Wavelength, Taejo, Rsrikanth05, Josteinaj, Nils Grimsmo, BOT-Superzerocool, Bota47, Cedar101, SmackBot, Posix4e, McGeddon, P b1999, Mcld, Skizzik, DHN-bot~enwiki, Konstable, Anabus, Mathmike, N Shar, Solon.KR, SpyMagician, Drdevil44, Pjrm, CBM, Myasuda, Thijs!bot, Epbr123, Headbomb, Williamyf, Heineman, Lavv17, JAnDbot, Harish victory, Magioladitis, Tonyfaull, Abednigo, Stdazi, David Eppstein, Gwern, J.delanoy, LordAnubisBOT, Monsday, VolkovBot, Zholdas, JohnBlackburne, TXiKiBoT, Ferengi, Aaron Rotenberg, Jamelan, SQL, AlleborgoBot, Aednichols, YonaBot, ToePeu.bot, VVVBot, Tvidas, Naroza, Arlekean, PipepBot, Justin W Smith, Pskjs~enwiki, Alexbot, PixelBot, Arjayay, Aene, DumZiBoT, Writer130, Addbot, Tsunanet, Iceblock, Protonk, Jasper Deng, Luckas-bot, Yobot, Ptbotgourou, The Earwig, Linket, Gmile~enwiki, Backslash Forwardslash, PanLevan, ArthurBot, Str82no1, Miym, Mario777Zelda, Shuroo, RobinK, Dinamik-bot, ToneDaBass, Nostalgius, EmausBot, Wikipelli,

Bjozen, Guahnala, AManWithNoPlan, ClueBot NG, Nullzero, Happyuk, Cyberbot II, Lone boatman, Aladdin.chettouh, Carlwitt, Gauravxpress, Jianhui67, Jonchen42, Amortias, AHusain3141, Subshiri, Shelke.disha, Georgiraichovgeorgiev, Ericpony, Adreno, Tusharsoni099, Karaminchan, Plurmiscuous and Anonymous: 183

- **A* search algorithm** *Source:* https://en.wikipedia.org/wiki/A*_search_algorithm?oldid=761920249 *Contributors:* Damian Yerrick, AxelBoldt, Mav, PierreAbbat, Mrwojo, Frecklefoot, Bdesham, Kku, Ahoerstemeier, Docu, Julesd, Trisweb, Jll, BenKovitz, IMSoP, Cherkash, Charles Matthews, Timwi, Dcoetzee, Nohat, JCarriker, Dysprosia, Furrykef, GPHemsley, Phil Boswell, Catskul, Robbot, Fredrik, Altenmann, Auric, Hadal, Tea2min, Giftlite, Mat-C, Mintleaf~enwiki, Laurens~enwiki, Lee J Haywood, Brona, Mellum, Siroxo, Neilc, Rdsmith4, Beau~enwiki, Discospinster, Rich Farmbrough, Rspeer, Yknott, Talldean, Kndiaye, Bender235, ZeroOne, Blogjack, BACbKA, Hart~enwiki, Whosyourjudas, Eje211, Keenan Pepper, RoySmith, Runtime, Xnk, MIT Trekkie, Joelpt, TheGoblin, MattGiuca, JonH, Ruud Koot, GregorB, Dionyziz, TrentonLipscomb, Rufous, Kesla, Ashmoo, KaisaL, Qwertyus, Jacob Finn, AllanBz, Rjwilmsi, Grantstevens, Jameshfisher, Wctaiwan, Quuxplusone, Fresheneesz, Kri, Samkass, Chobot, DVdm, YurikBot, Wavelength, Michael Slone, Taejo, KamuiShirou, Stephenb, Ritchy, Thsgrn, Nick, Neil.steiner, Orca456, Syrthiss, BOT-Superzerocool, BIS Suma, Eyal0, Regnaron~enwiki, Cedar101, Gulliveig, Kevin, JLaTondre, Katieh5584, Tomp, SmackBot, Mcld, DHN-bot~enwiki, Tekhnofiend, Nick Levine, Trudslev, Acdx, Remko~enwiki, Disavian, Jrouquie, Ripe, Pjrm, SkyWalker, Ale jrb, Szabolcs Nagy, Tac-Tics, Simeon, DumbBOT, Keosan, Martyr2566, AlexAlex, Headbomb, Markulf, Braphael, GiM, Malcolm, Falsedef, Kainino, Gökhan, IanOsgood, Geoffadams, SiobhanHansa, Benstown, Magioladitis, VoABot II, Hornbydd, CountingPine, David Chouinard, David Eppstein, HebrewHammerTime, Justinhj, Piojo, FelipeVargasRigo, Hodja Nasreddin, Arronax50, Sigmundur, Joshua Issac, JoshReeves2, Aninhumer, Thomas.W, Chaos5023, JohnBlackburne, TXiKiBoT, Oshwah, Mcculley, WillUther, Alexander Shekhovtsov, Subh83, Yintan, Henke37, Xprycker, JackSchmidt, Abraham, B.S., Svick, Fiarr, WikiLaurent, Denisarona, Muhends, Justin W Smith, Kotniski, Nanobear~enwiki, Daveagp, Mohit05011992, Excirial, Alexbot, Dr.Koljan, Dmyersturnbull, Peatar, Gmentat, 1ForTheMoney, Pzoxicuvybtnrm, Ddccc, Darkicebot, Alex Krainov, XLinkBot, Rankiri, C. A. Russell, Maco1421, SteveJothen, Addbot, Mortense, Crazy2be, DOI bot, Alex.mccarthy, LinkFA-Bot, Tassedethe, Glenstamp, HerculeBot, Luckas-bot, Yobot, DavidHarkness, AnomieBOT, DemocraticLuntz, Piano non troppo, Silnarm, Kingpin13, Hiihammuk, Materialscientist, Cycling-professor, Vanished user 04, Citation bot, Xqbot, Leirbag.arc, Naugtur, Boom1234567, FrescoBot, C7protal, Age Happens, Albertzeyer, Sanderd17, George126~enwiki, MastiBot, Yuval Baror, Trappist the monk, Electro, Lotje, Antonbharkamsan, Faure.thomas~enwiki, RjwilmsiBot, Millerdl, VernoWhitney, Timotei21, EmausBot, Dreske, WikitanvirBot, Ghodsnia, Srossd, Gaganbansal123, K6ka, Jiri Pavelka, Johnjianfang, Dr0b3rts, Josve05a, MithrandirAgain, Chire, Bamyers99, Sylverone, HandsomeFella, Alejandro.isaza, ClueBot NG, Andrewrosenberg, Khanser, MelbourneStar, Jiyeyuran, Keithphw, Korrawit, Arrandale, Masssly, CaroleHenson, Widr, Salzahrani, 316 student, HMSSolent, BG19bot, Hallows AG, Aedieder, Frasmog, Axule, Pratyya Ghosh, Dexbot, JingguoYao, Bjorn Reese, Mogism, Frosty, Sriharsh1234, JimDeLaHunt, Theemathas, Cesarramsan, Bojannestorovic, Manish181192, Lyn240690234, Treaster, Mhavard999, Amcshane, Opencooper, Raaghu03, HXZBZSHDHDHED, Kanargias, BringsVictory, Johanna, Rivascalps2, Mutantoe, Nbro, HelpUsStopSpam, Gdamyanov, CAPTAIN RAJU, Jfraumen, PeizonChen, MacShrike, Wiki2016edit, Ppery, Markjin1990, Mehrotraparth and Anonymous: 410
- **Bipartite graph** *Source:* https://en.wikipedia.org/wiki/Bipartite_graph?oldid=742021681 *Contributors:* Jdpipe, Nonenmac, Michael Hardy, Chris-martin, Manojmp, TakuyaMurata, Delirium, Eric119, Altenmann, Netpilot43556, MathMartin, Giftlite, Tom harrison, Tobo~enwiki, Tomruen, Corti, Shahab, Paul August, Rgdboer, Bobo192, Mdd, Jérôme, Burn, Bkkbrad, BD2412, Brighterorange, FlaBot, Kri, YurikBot, Freiberg, Catgofire, Jpbowen, Nethgirb, Bota47, H@r@ld, Melchoir, BiT, DHN-bot~enwiki, Tsca.bot, Gassa, Jon Awbrey, Michael Dinolfo, Ojan, CRGreathouse, Flamholz, Cydebot, Kozuch, Thijs!bot, Jdudar, MistWiz, AndreasWittenstein, Kevinmon, David Eppstein, JoergenB, Robin S, Hpfister, Robert Illes, Policron, AlnoktaBOT, PaulTanenbaum, Jason Klaus, Charliearcuri, Mild Bill Hiccup, Alexbot, Addbot, Luckas-bot, Yobot, Rubinbot, Twri, Miym, Shmomuffin, Locobot, Howard McCay, LucienBOT, D'ohBot, Doostdar-WKP, Pinethicket, RjwilmsiBot, Ripchip Bot, Ebrambot, Pgdx, ClueBot NG, Wcherowi, Hoorayforturtles, 149AFK, Nullzero, Helpful Pixie Bot, Solomon7968, AndiPersti, GeoffreyT2000, SofjaKovalevskaja, Bender the Bot and Anonymous: 68
- **Complete bipartite graph** *Source:* https://en.wikipedia.org/wiki/Complete_bipartite_graph?oldid=743601363 *Contributors:* PierreAbbat, Nonenmac, Chris-martin, Yaronf, Dcoetzee, Adoarns, McKay, Jaredwf, MathMartin, Giftlite, Dbenbenn, Tom harrison, JeffBobFrank, Andris, Tomruen, Shahab, Paul August, Keenan Pepper, FlaBot, Quuxplusone, Chobot, Michael Slone, Evilbu, J. Finkelstein, NongBot~enwiki, David Eppstein, Koko90, Robert Illes, VolkovBot, Jamelan, Robert Samal, Justin W Smith, Alexbot, Bender2k14, Addbot, Numbo3-bot, PV=nRT, Calle, Twri, DSisyphBot, Miym, Erik9bot, LucienBOT, X7q, RobinK, Solomon7968, □□□□□, Bender the Bot and Anonymous: 33
- **Petri net** *Source:* https://en.wikipedia.org/wiki/Petri_net?oldid=760944973 *Contributors:* Zundark, Ap, Andre Engels, LionKimbro, Mark Durst, Michael Hardy, JakeVortex, Rp, Charles Matthews, Samsara, Phil Boswell, Robbot, Giftlite, Thv, Pretzelpaws, Michael Devore, Khalid hassani, Neilc, Gadfium, RedCrystal, Andreas Kaufmann, Ericbodden, Leibniz, Ascánder, Mani1, Mkegelmann, Bobo192, Sam Korn, Mdd, Ramaz, Kotasik, Kelly Martin, Linas, Ruud Koot, Josh Parris, Rjwilmsi, Ligulem, John Deas, Gurch, Vonkje, Gpig2013, Anrie Nord, Roboto de Ajvol, YurikBot, Arado, Gaius Cornelius, CarlHewitt, Lpedro, Msoos, Gareth Jones, ArmadniGeneral, Jpbowen, El Pollo Diablo, K.Nevelsteen, Cbogart2, Cedar101, Xaxafrad, Esprit15d, ArielGold, That Guy, From That Show!, Jsnx, SmackBot, Eskimbot, Theone256, Dfletter, AndrewChinery, JonHarder, Allan McInnes, SQB, Clicketyclack, Spiritia, JorisvS, Loadmaster, Mets501, Phuzion, Randomity, Vocaro, Spdegabrielle, CRGreathouse, CmdrObot, Outriggr (2006-2009), Neonleonb, Cydebot, Sam Staton, Patrick O'Leary, Smiteri, Malleus Fatuorum, Thijs!bot, Torben.green, Headbomb, Polymorph self, Gioto, Heysan, Remaire, JAnDbot, Fbahr, MattBan, Nouiz, Creacon, Magioladitis, MyNameIsNeo, Wvdaalst, A3nm, Gwern, Glrx, R'n'B, Theups, Alessiobissoli, Helios369, Outmind~enwiki, Yoichi3, Marashie, SieBot, OKBot, ClueBot, Alksentrs, Polyamorph, Niceguyedc, Vivio Testarossa, Singularity42, Un11imig, Wikiplant, Good Olfactory, Addbot, DOI bot, IXavier, Jklowden, MrOllie, WikiDreamer Bot, Janrou, Favrin, Luckas-bot, Yobot, Plasticbot, DemocraticLuntz, Thachx, Citation bot, Pelerin2, False vacuum, WaysToEscape, FrescoBot, Ariek, Patrick J. May, Citation bot 1, Arthur MILCHIOR, DrilBot, Vasywriter, RedBot, Lotje, Clemens2000, Amkilpatrick, Pyschobbens, Bethnim, Ш, Yagalone, Runestone1, Nerlnsqy, Robbiemorrison, ClueBot NG, Widr, Helpful Pixie Bot, BG19bot, Manu31415, Lmkaff, David Maung, Adɤijɛ̃, Igor potapov, Mark viking, Lenaherscheid, Zsoftua, MichaelBlondin, Monkbot, U2fanboi, DraXus, GFavrin, Gfavrin1, Le blue, HelpUsStopSpam, TimmKam, Bender the Bot, Wilvdaalst and Anonymous: 176
- **Adjacency matrix** *Source:* https://en.wikipedia.org/wiki/Adjacency_matrix?oldid=751840428 *Contributors:* AxelBoldt, Tbackstr, Tomo, Michael Hardy, Chris-martin, TakuyaMurata, Mbogelund, Schneelocke, Dcoetzee, Dysprosia, Reina riemann, Aleph4, Fredrik, MathMartin, Bkell, ElBenevolente, Giftlite, BenFrantzDale, Arthouse~enwiki, MarkSweep, Tomruen, Abdull, Rich Farmbrough, Bender235, Zaslav, Gauge, Rgdboer, Phils, Burn, Cburnett, Oleg Alexandrov, Timendum, BD2412, Rjwilmsi, YurikBot, Jpbowen, Jogers, Netrapt, Sneftel, SmackBot, Ttzz, Kneufeld, Senfo, Abha Jain, Tamfang, Juffi, EVula, Dreadstar, Mdrine, Paulish, Tim Q. Wells, Beetstra, Hu12, Rhetth, CmdrObot, Only2sea, Thijs!bot, Headbomb, Olenielsen, Salgueiro~enwiki, Natelewis, Xeno, JamesBWatson, David Eppstein, Yonidebot, Squids and Chips, VolkovBot, LokiClock, YuryKirienko, Morre~enwiki, Periergeia, JackSchmidt, Garyzx, Watchduck, Bender2k14, Yoav HaCohen, Addbot, Fgnievinski, LaaknorBot, Debresser, Matěj Grabovský, سعی, Luckas-bot, Calle, Milk's Favorite Bot

II, AnomieBOT, Twri, ArthurBot, SPTWriter, Miym, Bitsianshash, X7q, Jean Honorio, Sławomir Biały, Maggyero, Chenopodiaceous, Tom.Reding, RobinK, TobeBot, Patmorin, John of Reading, WikitanvirBot, Felix Hoffmann, ZéroBot, ChuispastonBot, ClueBot NG, Wcherowi, Helpful Pixie Bot, Shilpi4560, Snowcream, DarafshBot, Minidiable, Kompot 3, Horseless Headman, Shamus03, Rmatych and Anonymous: 86

- **Travelling salesman problem** *Source:* https://en.wikipedia.org/wiki/Travelling_salesman_problem?oldid=760378776 *Contributors:* Marj Tiefert, The Anome, AstroNomer, Aldie, ChangChienFu, FvdP, B4hand, Stevertigo, Edward, Michael Hardy, Wwwwolf, Ixfd64, Zeno Gantner, Karada, Ellywa, Ahoerstemeier, Kyokpae~enwiki, Hike395, Charles Matthews, Dcoetzee, Dino, Dysprosia, Doradus, Raul654, Johnleach, David.Monniaux, JackH, Donarreiskoffer, Fredrik, R3m0t, Altenmann, MathMartin, Brw12, Bkell, Chris-gore, Ruakh, Mattflaschen, Tea2min, Giftlite, Laudaka, MSGJ, Herbee, Qaramazov, Mellum, Andris, Prosfilaes, Boothinator, Vanished user wdjklasdjskla, Rheun, PeterC, Gdr, Quadell, Beland, DragonflySixtyseven, Naff89, Hyperneural, Andreas Kaufmann, The stuart, Mormegil, Tinman, Ronaldo~enwiki, Random contributor, Sladen, ArnoldReinhold, Smyth, Zamfi, Bender235, ZeroOne, Petrus~enwiki, STGM, El C, Superninja, Gnomz007, Aaronbrick, Shoujun, Touriste, Tomgally, Stochastix, Obradovic Goran, Mdd, Musiphil, Jhertel, Kotasik, Ynhockey, Zyqqh, Theodore Kloba, DreamGuy, MoraSique, Isaac, LFaraone, Pauli133, Kenyon, Stemonitis, Simetrical, LOL, StradivariusTV, Ruud Koot, DavidBiesack, Jok2000, Smmurphy, GregorB, Male1979, Nobbie, Graham87, BD2412, Kane5187, Ketiltrout, Rjwilmsi, MarSch, Seraphimblade, Salix alba, SpNeo, Bubba73, Bensin, Nguyen Thanh Quang, Tedder, Kri, JonathanFreed, Chobot, DVdm, Siddhant, YurikBot, Wavelength, Angus Lepper, Hairy Dude, RedLyons, Michael Slone, Jeffhoy, SpuriousQ, Jugander, Pmdboi, Irrevenant, Astral, Dbfirs, Brucevdk, Nethgirb, Geofftech, Sarkar112, Fmccown, Tom Duff, Xiaojeng~enwiki, Bo Jacoby, Kf4bdy, That Guy, From That Show!, David.hillshafer, SmackBot, RDBury, Apanag, Akokskis, McGeddon, David.Mestel, KocjoBot~enwiki, Eskimbot, Adammathias, Stimpy, IstvanWolf, Mcld, Richmeister, Gilliam, Bluebot, Colonies Chris, Jasonb05, Can't sleep, clown will eat me, Tamfang, Thisisbossi, Downtown dan seattle, Nr9, Ryan Roos, Smremde, Lambiam, ArglebargleIV, Zaphraud, Ninjagecko, Vgy7ujm, Tomhubbard, Disavian, Brian Gunderson, Beetstra, Sharcho, H, Alan.ca, Wizard191, Dansiman, Twas Now, Sahuagin, Azotlichid, Chris55, ChrisCork, Ludwig Boltzmann, CRGreathouse, CmdrObot, Devis, ShelfSkewed, Requestion, Pgr94, Honnza, Gogo Dodo, Ferris37, Lady-shirakawa, Kvamsi82, Klausikm~enwiki, Davidhorman, Trlkly, Alphachimpbot, Gaeddal, Dricherby, Johngouf85, Xypron, Seet82, Magioladitis, Bongwarrior, Nyq, CountingPine, Ajihood, 28421u2232nfenfcenc, David Eppstein, Martynas Patasius, IvR, ANONYMOUS COWARD0xC0DE, Jamesd9007, Anaxial, R'n'B, J.delanoy, Wumpus3000, Yonidebot, Gfox88, Dispenser, Jsamarziya, Kenneth M Burke, Ross Fraser, Ratfox, KGV, Paul Silverman, Jim.Callahan,Orlando, Pleasantville, TXiKiBoT, Mantipula, Kjells, Lingwanjae, Melsaran, Dwhdwh, Tsplog, Jackbars, Phcho8, Jasonyo, Hawk777, DivineBurner, AlanUS, Mojoworker, Cngoulimis, Classicalecon, MagicMatt1021, Justin W Smith, The Thing That Should Not Be, Mild Bill Hiccup, Ottoshmidt, DragonBot, Greenmatter, Bender2k14, TedDunning, Spock of Vulcan, Flapitrr, 7&6=thirteen, Hans Adler, Ksana, Rocarvaj, Orfest, XLinkBot, Rror, Gerhardvalentin, Saeed.Veradi, C. lorenz, Addbot, Fmorstatter, Grayfell, DOI bot, Mhahsler, Ironholds, Download, Tigerqin, SpBot, SamatBot, Jasper Deng, Yixin.cao, Lightbot, Vasiľ, Legobot, PierreSelim, Yobot, Coconut7594, Mikeedla, Bathysphere, AnomieBOT, 1exec1, Galoubet, Soupz, Piano non troppo, Dzyadyk, Sparshong, Bjornson81, Materialscientist, Monstergurkan, Citation bot, Benhen1997, ArthurBot, Qorilla, Quebec99, Capricorn42, Aftermath1983, Miym, Thore Husfeldt, Mzamora2, Fanis84, Shadowjams, Some standardized rigour, Shubhrasankar, WhatisFeelings?, FrescoBot, Pegasusbupt, Pschaus, Haeinous, Daniel Karapetyan, Citation bot 1, Tom3118, Kiefer.Wolfowitz, Jonesey95, RobinK, Reconsider the static, Horst-schlaemma, Laleppa, Sergey539, Fox Wilson, Duoduoduo, JumpDiscont, 4ndyD, Xnn, Teamtheo, Ripchip Bot, Shafaet, EmausBot, John of Reading, Bernard Teo, Geomatique, Andreasr2d2, Scravy~enwiki, Tommy2010, Dcirovic, Haterade111, Jasmjc2, South Texas Waterboy, Isomorphismus, Alexander Misel, Uziel302, ErnestSDavis, ClueBot NG, Fioravante Patrone, Slartibartfastibast, Lanthanum-138, Widr, Helpful Pixie Bot, BG19bot, Qx2020, Aronisstav, Happyuk, Snow Blizzard, Bmears11, Gdessy, Shakir28, ChrisGualtieri, Ozziev, Dexbot, Lone boatman, Beatkist, Dishantpandya777, EzequielBalmori, Saurabh.harsh, Lesser Cartographies, AustinBuchanan, Anonymous Random Person, SJ Defender, Jonhkr, Samusoft1994, AkselA, Ilya.gazman, Ricksmt, Ju12358, Monkbot, Andy19601, BethNaught, Gronk Oz, Niraj Aher, Dralea, Signedzzz, Bammie73, Magicalbendini, Boky90, Phi1618pi314, Nbro, Cryptomaniac2, Voltran46, Vueza, Yaron1m, Marianna251, Cnoonphd, J20160628, Acopyeditor, Tusharsoni099, JacobIsrael18, Vaidersith and Anonymous: 480

- **Route inspection problem** *Source:* https://en.wikipedia.org/wiki/Route_inspection_problem?oldid=747462033 *Contributors:* Bryan Derksen, FvdP, Michael Hardy, Dysprosia, Wik, Robbot, Bkell, Giftlite, Stern~enwiki, Sam Hocevar, Bender235, Zaslav, Robotje, PolyGnome, Oleg Alexandrov, Rjwilmsi, Mathbot, YurikBot, Taejo, Arthur Rubin, Bluebot, Mwtoews, Honnza, Thijs!bot, WhiteCrane, Headbomb, AbcXyz, VictorAnyakin, JAnDbot, David Eppstein, Mikhail Dvorkin, VolkovBot, TXiKiBoT, Softtest123, Hagman, Da Joe, Rockstone35, Mangledorf, Petermgiles, Justin W Smith, Bender2k14, Phantom xxiii, C. lorenz, Addbot, Mariozzyzx, PV=nRT, Aviados, Luckas-bot, ArthurBot, Xqbot, GrouchoBot, SD5, Shuroo, RedBot, RobinK, ErikvanB, Kedwar5, Wcherowi, Gjoret, FritzFasbinder, Pintoch, Bergil VI, Wan fokkink and Anonymous: 33

- **Hamiltonian path problem** *Source:* https://en.wikipedia.org/wiki/Hamiltonian_path_problem?oldid=757811880 *Contributors:* Arvindn, Dominus, Ixfd64, Dcoetzee, Taxman, Altenmann, MathMartin, Giftlite, CryptoDerk, Andreas Kaufmann, Brian0918, Obradovic Goran, Oleg Alexandrov, Shreevatsa, BD2412, Rjwilmsi, Matt Cook, Ecb29, Ewlyahoocom, Vecter, Michael Slone, Antidrugue, Rupert Clayton, SmackBot, Davepape, Anachronist, Nr9, Sbluen, Ligulembot, Graph Theory page blanker, Headbomb, Widefox, David Eppstein, Henrylaxen, Mariehuynh~enwiki, Laurusnobilis, Silas S. Brown, MKoltnow, Pahari Sahib, Hqb, Wicher Minnaard, Luuva, Jamelan, Jochgem, Smaug123, Contestcen, Callinus, Addbot, AnomieBOT, Citation bot, Miym, Thore Husfeldt, MathsPoetry, Sachmo2, Joostik, I dream of horses, RobinK, Ginopinoshow, Miracle Pen, JumpDiscont, RjwilmsiBot, Dwc89, BG19bot, CitationCleanerBot, Araujoraphael, Jochen Burghardt, Alabarre, Cosmia Nebula, Pkaftan, E. Rotenberg, Ziaozi and Anonymous: 40

7.2 Images

- **File:10-simplex_graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/8/86/10-simplex_graph.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Koko90
- **File:11-simplex_graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/9/9b/11-simplex_graph.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Koko90
- **File:3-simplex_graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/b/be/3-simplex_graph.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Koko90
- **File:4-simplex_graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/2d/4-simplex_graph.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Koko90
- **File:4-tournament.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/8/89/4-tournament.svg *License:* Public domain *Contributors:* Own work *Original artist:* Booyabazooka

- **File:5-simplex_graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/e/e9/5-simplex_graph.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Koko90
- **File:6-simplex_graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/c/c8/6-simplex_graph.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Koko90
- **File:6n-graf.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/5/5b/6n-graf.svg *License:* Public domain *Contributors:* Image:6n-graf.png simlar input data *Original artist:* User:AzaToth
- **File:6n-graph2.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/28/6n-graph2.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:7-simplex_graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/c/cb/7-simplex_graph.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Koko90
- **File:7_bridges.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/9/91/7_bridges.svg *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?
- **File:7_bridgesID.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/b/b3/7_bridgesID.png *License:* CC-BY-SA-3.0 *Contributors:* Transferred from en.wikipedia to Commons. *Original artist:* The original uploader was Xiong at English Wikipedia
- **File:7b-graph09.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/6/69/7b-graph09.png *License:* CC-BY-SA-3.0 *Contributors:* Transferred from en.wikipedia to Commons. *Original artist:* Xiong at English Wikipedia
- **File:8-simplex_graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/2c/8-simplex_graph.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Koko90
- **File:9-simplex_graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/b/bb/9-simplex_graph.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Koko90
- **File:A*_Search_Example_on_North_American_Freight_Train_Network.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/6/60/A%2A_Search_Example_on_North_American_Freight_Train_Network.gif *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Srossd
- **File:Aco_TSP.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/2a/Aco_TSP.svg *License:* CC-BY-SA-3.0 *Contributors:* Own work *Original artist:* Nojhan
- **File:Animated_Petri_net_commons.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d7/Animated_Petri_net_commons.gif *License:* CC BY-SA 2.5 *Contributors:* ? *Original artist:* ?
- **File:AntColony.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/8/8c/AntColony.gif *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Saurabh.harsh
- **File:AstarExampleEn.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/9/98/AstarExampleEn.gif *License:* CC0 *Contributors:* Own work *Original artist:* CountingPine
- **File:Astar_progress_animation.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/5/5d/Astar_progress_animation.gif *License:* CC BY 3.0 *Contributors:* Own work *Original artist:* Subh83
- **File:Bellman-Ford_worst-case_example.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/8/85/Bellman-Ford_worst-case_example.svg *License:* CC0 *Contributors:* Own work *Original artist:* User:Dcoetzee
- **File:Biclique_K_3_5.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d6/Biclique_K_3_5.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Koko90
- **File:Branchbound.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/3/3c/Branchbound.gif *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Saurabh.harsh
- **File:Bruteforce.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/2b/Bruteforce.gif *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Saurabh.harsh
- **File:Commons-logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/4/4a/Commons-logo.svg *License:* PD *Contributors:* ? *Original artist:* ?
- **File:Comparison_7_bridges_of_Konigsberg_5_room_puzzle_graphs.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/f/fb/Comparison_7_bridges_of_Konigsberg_5_room_puzzle_graphs.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Cmglee
- **File:Complete_graph_K1.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/ad/Complete_graph_K1.svg *License:* Public domain *Contributors:* Own work *Original artist:* David Benbennick wrote this file.
- **File:Complete_graph_K2.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/9/96/Complete_graph_K2.svg *License:* Public domain *Contributors:* Own work *Original artist:* David Benbennick wrote this file.
- **File:Complete_graph_K3.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/5/5a/Complete_graph_K3.svg *License:* Public domain *Contributors:* Own work *Original artist:* David Benbennick wrote this file.
- **File:Complete_graph_K5.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/c/cf/Complete_graph_K5.svg *License:* Public domain *Contributors:* Own work *Original artist:* David Benbennick wrote this file.
- **File:Complete_graph_K7.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/9/9e/Complete_graph_K7.svg *License:* Public domain *Contributors:* Own work *Original artist:* David Benbennick
- **File:Complex_polygon_2-4-3-bipartite_graph.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/7/70/Complex_polygon_2-4-3-bipartite_graph.png *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Tomruen
- **File:Complex_polygon_2-4-3.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/7/7e/Complex_polygon_2-4-3.png *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Tomruen
- **File:Complex_polygon_2-4-4.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/5/59/Complex_polygon_2-4-4.png *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Tomruen

- **File:Complex_polygon_2-4-4_bipartite_graph.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/e/e3/Complex_polygon_2-4-4_bipartite_graph.png *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Tomruen
- **File:Complex_polygon_2-4-5-bipartite_graph.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/9/92/Complex_polygon_2-4-5-bipartite_graph.png *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Tomruen
- **File:Complex_polygon_2-4-5.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/ac/Complex_polygon_2-4-5.png *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Tomruen
- **File:Creating_a_matching.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/f/ff/Creating_a_matching.png *License:* Public domain *Contributors:* own creation *Original artist:* own creation
- **File:Detailed_petri_net.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/f/fe/Detailed_petri_net.png *License:* CC-BY-SA-3.0 *Contributors:* No machine-readable source provided. Own work assumed (based on copyright claims). *Original artist:* No machine-readable author provided. Msoos assumed (based on copyright claims).
- **File:DijkstraDemo.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/e/e4/DijkstraDemo.gif *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Shiyu Ji
- **File:Dijkstra_Animation.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/5/57/Dijkstra_Animation.gif *License:* Public domain *Contributors:* Work by uploader. *Original artist:* Ibmua
- **File:Dijkstras_progress_animation.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/23/Dijkstras_progress_animation.gif *License:* CC BY 3.0 *Contributors:* Own work *Original artist:* Subh83
- **File:Directed.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/a2/Directed.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:DirectedDegrees.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/7/77/DirectedDegrees.svg *License:* GFDL *Contributors:* Own work *Original artist:* Melchoir
- **File:Directed_acyclic_graph_2.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/0/03/Directed_acyclic_graph_2.svg *License:* Public domain *Contributors:* Own work *Original artist:* Johannes Rössel (talk)
- **File:First_usage_of_the_symbol_∈.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/f/f4/First_usage_of_the_symbol_%E2%88%88.png *License:* Public domain *Contributors:* Arithmetices principia nova methodo exposita. (page X) *Original artist:* Giuseppe Peano
- **File:Folder_Hexagonal_Icon.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/4/48/Folder_Hexagonal_Icon.svg *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:GLPK_solution_of_a_travelling_salesman_problem.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/1/11/GLPK_solution_of_a_travelling_salesman_problem.svg *License:* Public domain *Contributors:* Own work *Original artist:* Xypron
- **File:Incidence_matrix_-_directed_graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/5/50/Incidence_matrix_-_directed_graph.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Pistekjakub
- **File:Internet_map_1024.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d2/Internet_map_1024.jpg *License:* CC BY 2.5 *Contributors:* Originally from the English Wikipedia; description page is/was here. *Original artist:* The Opte Project
- **File:Koenigsberg_Bridges_Variations_Graph10.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/b/b7/Koenigsberg_Bridges_Variations_Graph10.png *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?
- **File:Koenigsberg_Bridges_Variations_Graph7.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/af/Koenigsberg_Bridges_Variations_Graph7.png *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?
- **File:Koenigsberg_Bridges_Variations_Graph8.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/9/90/Koenigsberg_Bridges_Variations_Graph8.png *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?
- **File:Koenigsberg_Bridges_Variations_Problem.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/6/66/Koenigsberg_Bridges_Variations_Problem.png *License:* CC-BY-SA-3.0 *Contributors:* Transferred from en.wikipedia to Commons by Legoktm using CommonsHelper. *Original artist:* Xiong at English Wikipedia
- **File:Konigsberg_bridges.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/5/5d/Konigsberg_bridges.png *License:* CC-BY-SA-3.0 *Contributors:* Public domain (PD), based on the image
    -

  Original artist: Bogdan Giuşcă
- **File:Königsberg_graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/9/96/K%C3%B6nigsberg_graph.svg *License:* CC-BY-SA-3.0 *Contributors:* ? *Original artist:* ?
- **File:Liveness-levels.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/7/75/Liveness-levels.gif *License:* CC BY-SA 3.0 *Contributors:* Own work by uploader. The example it shows is due to T. Murata. *Original artist:* Rp22
- **File:Lock-green.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/6/65/Lock-green.svg *License:* CC0 *Contributors:* en:File:Free-to-read_lock_75.svg *Original artist:* User:Trappist the monk
- **File:Mergefrom.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/0/0f/Mergefrom.svg *License:* Public domain *Contributors:* ? *Original artist:* ?

- **File:Minimum_spanning_tree.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d2/Minimum_spanning_tree.svg *License:* Public domain *Contributors:* No machine-readable source provided. Own work assumed (based on copyright claims). *Original artist:* No machine-readable author provided. Dcoetzee assumed (based on copyright claims).
- **File:Moreno_Sociogram_2nd_Grade.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/b/b6/Moreno_Sociogram_2nd_Grade.png *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Martin Grandjean
- **File:Msp-the-cut-correct.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/f/f7/Msp-the-cut-correct.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* King rhoton
- **File:Multi-pseudograph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/c/c9/Multi-pseudograph.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* 0x24a537r9
- **File:Multiple_minimum_spanning_trees.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/c/c9/Multiple_minimum_spanning_trees.svg *License:* CC BY-SA 3.0 *Contributors:* File:Msp1.jpg *Original artist:* Ftiercel
- **File:Nearestneighbor.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/23/Nearestneighbor.gif *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Saurabh.harsh
- **File:Nuvola_apps_atlantik.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/7/77/Nuvola_apps_atlantik.png *License:* LGPL *Contributors:* http://icon-king.com *Original artist:* David Vignoni / ICON KING
- **File:Odd_Cycle_Transversal_of_size_2.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d4/Odd_Cycle_Transversal_of_size_2.png *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* PG Drange
- **File:Petri_Net_A.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/6/6a/Petri_Net_A.jpg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Runestone
- **File:Petri_Net_B.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/e/e2/Petri_Net_B.jpg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Runestone
- **File:Petri_net_types.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/e/e1/Petri_net_types.svg *License:* Public domain *Contributors:* Own work *Original artist:* User:Msoos
- **File:Portal-puzzle.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/f/fd/Portal-puzzle.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Present_state_of_the_Seven_Bridges_of_Königsberg.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/e/e8/Present_state_of_the_Seven_Bridges_of_K%C3%B6nigsberg.png *License:* CC BY-SA 2.5 *Contributors:* http://openstreetmap.ru/#map=15/54.7044/20.5175&layer=S *Original artist:* Map data by OpenStreetMap contributors; rendering by GIScience Research Group @ Heidelberg University; produced work by Santacloud
- **File:Question_book-new.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/9/99/Question_book-new.svg *License:* Cc-by-sa-3.0 *Contributors:*
  Created from scratch in Adobe Illustrator. Based on Image:Question book.png created by User:Equazcion *Original artist:*
  Tkgd2007
- **File:Reachability_graph_for_petri_net.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/f/ff/Reachability_graph_for_petri_net.png *License:* CC-BY-SA-3.0 *Contributors:* No machine-readable source provided. Own work assumed (based on copyright claims). *Original artist:* No machine-readable author provided. Msoos assumed (based on copyright claims).
- **File:SRI_Shakey_with_callouts.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/0/0c/SRI_Shakey_with_callouts.jpg *License:* CC BY-SA 3.0 *Contributors:* SRI International *Original artist:* SRI International
- **File:Shortest_path_with_direct_weights.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/3/3b/Shortest_path_with_direct_weights.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Artyom Kalinin
- **File:Showing_a_step_of_the_two-opt_heuristic.png** *Source:* https://upload.wikimedia.org/wikipedia/en/f/f2/Showing_a_step_of_the_two-opt_heuristic.png *License:* CC-BY-SA-3.0 *Contributors:*
  created on gliffy
  Original artist:
  Lady-shirakawa
- **File:Simple-bipartite-graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/e/e8/Simple-bipartite-graph.svg *License:* Public domain *Contributors:* Own work *Original artist:* MistWiz
- **File:Snake-in-the-box_and_Hamiltonian_path.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/7/70/Snake-in-the-box_and_Hamiltonian_path.svg *License:* Public domain *Contributors:* https://en.wikipedia.org/wiki/File:Snakeinthebox.svg and own work. *Original artist:* Alejandro Erickson
- **File:Speaker_Icon.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/21/Speaker_Icon.svg *License:* Public domain *Contributors:* No machine-readable source provided. Own work assumed (based on copyright claims). *Original artist:* No machine-readable author provided. Mobius assumed (based on copyright claims).
- **File:Star_graphs.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/7/7d/Star_graphs.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Koko90
- **File:Steiner_3_points.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/3/3f/Steiner_3_points.svg *License:* Public domain *Contributors:* Own work *Original artist:* User:Bryan Derksen
- **File:Steiner_4_points.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/e/e4/Steiner_4_points.svg *License:* Public domain *Contributors:* Own work *Original artist:* User:Bryan Derksen
- **File:Symmetric_group_4;_Cayley_graph_1,5,21_(Nauru_Petersen);_numbers.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/27/Symmetric_group_4%3B_Cayley_graph_1%2C5%2C21_%28Nauru_Petersen%29%3B_numbers.svg *License:* Public domain *Contributors:* Own work *Original artist:* Watchduck (a.k.a. Tilman Piesk)

- **File:Symmetric_group_4;_Cayley_graph_1,5,21_(adjacency_matrix).svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/1/17/Symmetric_group_4%3B_Cayley_graph_1%2C5%2C21_%28adjacency_matrix%29.svg *License:* Public domain *Contributors:* Own work *Original artist:* Lipedia
- **File:Symmetric_group_4;_Cayley_graph_4,9;_numbers.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/8/86/Symmetric_group_4%3B_Cayley_graph_4%2C9%3B_numbers.svg *License:* Public domain *Contributors:*
- GrapheCayley-S4-Plan.svg *Original artist:* GrapheCayley-S4-Plan.svg: Fool (talk)
- **File:Symmetric_group_4;_Cayley_graph_4,9_(adjacency_matrix).svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/22/Symmetric_group_4%3B_Cayley_graph_4%2C9_%28adjacency_matrix%29.svg *License:* Public domain *Contributors:* Own work *Original artist:* Lipedia
- **File:Text_document_with_red_question_mark.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/a4/Text_document_with_red_question_mark.svg *License:* Public domain *Contributors:* Created by bdesham with Inkscape; based upon Text-x-generic.svg from the Tango project. *Original artist:* Benjamin D. Esham (bdesham)
- **File:Tree_graph.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/24/Tree_graph.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Turan_13-4.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/1/1b/Turan_13-4.svg *License:* Public domain *Contributors:* Transferred from en.wikipedia to Commons. *Original artist:* SVG image created by Braindrain0000 at en.wikipedia, based on previous public-domain PNG image en::Image:Turan-13-4.png by David Eppstein
- **File:Two-boundedness-cb.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/4/4d/Two-boundedness-cb.png *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Rp22
- **File:Two-boundedness-ub.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/7/7f/Two-boundedness-ub.png *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Rp22
- **File:UbMjAyAmQrSwtP0gdeKe_matchingshortcut.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/0/0f/UbMjAyAmQrSwtP0gdeKe_matchingshortcut.png *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Lady-shirakawa
- **File:Undirected.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/b/bf/Undirected.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Venn_A_intersect_B.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/6/6d/Venn_A_intersect_B.svg *License:* Public domain *Contributors:* Own work *Original artist:* Cepheus
- **File:Weighted_A_star_with_eps_5.gif** *Source:* https://upload.wikimedia.org/wikipedia/commons/8/85/Weighted_A_star_with_eps_5.gif *License:* CC BY 3.0 *Contributors:* Own work *Original artist:* Subh83
- **File:Weighted_K4.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/3/30/Weighted_K4.svg *License:* CC BY-SA 2.5 *Contributors:* self-made using xfig *Original artist:* Sdo
- **File:Wiki_letter_w_cropped.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/1/1c/Wiki_letter_w_cropped.svg *License:* CC-BY-SA-3.0 *Contributors:* This file was derived from Wiki letter w.svg:
Original artist: Derivative work by Thumperward
- **File:Wikipedia_multilingual_network_graph_July_2013.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/5/5b/Wikipedia_multilingual_network_graph_July_2013.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Computermacgyver
- **File:Wiktionary-logo-v2.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/0/06/Wiktionary-logo-v2.svg *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* Dan Polansky based on work currently attributed to Wikimedia Foundation but originally created by Smurrayinchester
- **File:William_Rowan_Hamilton_painting.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/1/15/William_Rowan_Hamilton_painting.jpg *License:* Public domain *Contributors:* http://www.askaboutireland.ie/search.xml?query=William+Rowan+Hamilton&radio_filter=images&type=and *Original artist:* Unknown

7.3 Content license

- Creative Commons Attribution-Share Alike 3.0