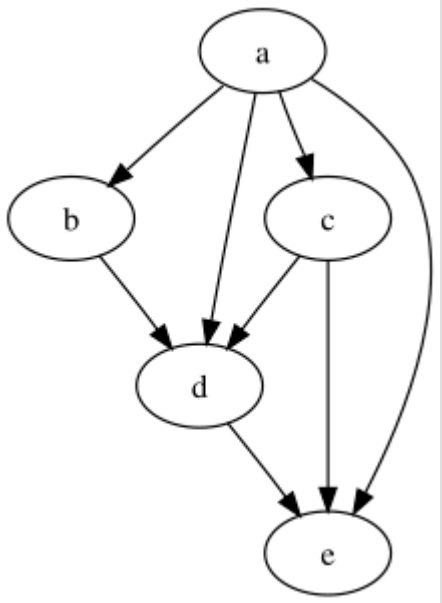
Directed acyclic graph

In mathematics, particularly graph theory, and computer science, a **directed acyclic graph** (**DAG**) is a directed graph with no directed cycles. That is, it consists of vertices and edges (also called *arcs*), with each edge directed from one vertex to another, such that following those directions will never form a closed loop. A directed graph is a DAG if and only if it can be topologically ordered, by arranging the vertices as a linear ordering that is consistent with all edge directions. DAGs have numerous scientific and computational applications, ranging from biology (evolution, family trees, epidemiology) to information science (citation networks) to computation (scheduling).

Directed acyclic graphs are sometimes instead called **acyclic directed graphs**[1] or **acyclic digraphs**.[2]



Example of a directed acyclic graph

Definitions

A graph is formed by vertices and by edges connecting pairs of vertices, where the vertices can be any kind of object that is connected in pairs by edges. In the case of a directed graph, each edge has an orientation, from one vertex to another vertex. A path in a directed graph is a sequence of edges having the property that the ending vertex of each edge in the sequence is the same as the starting vertex of the next edge in the sequence; a path forms a cycle if the starting vertex of its first edge equals the ending vertex of its last edge. A directed acyclic graph is a directed graph that has no cycles.[1][2][3]

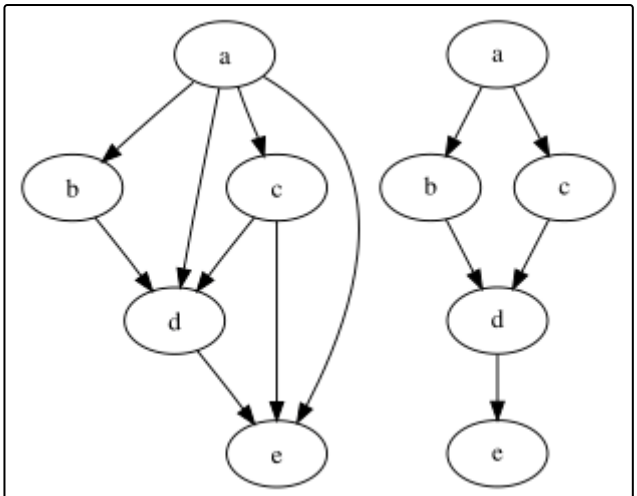
A vertex v of a directed graph is said to be reachable from another vertex u when there exists a path that starts at u and ends at v. As a special case, every vertex is considered to be reachable from itself (by a path with zero edges). If a vertex can reach itself via a nontrivial path (a path with one or more edges), then that path is a cycle, so another way to define directed acyclic graphs is that they are the graphs in which no vertex can reach itself via a nontrivial path.[4]

Mathematical properties

Reachability relation, transitive closure, and transitive reduction

The reachability relation of a DAG can be formalized as a partial order $\leq$ on the vertices of the DAG. In this partial order, two vertices u and v are ordered as $u \leq v$ exactly when there exists a directed path from u to v in the DAG; that is, when u can reach v (or v is reachable from u).[5] However, different DAGs may give rise to the same reachability relation and the same partial order.[6] For example, a DAG with

two edges $u \rightarrow v$ and $v \rightarrow w$ has the same reachability relation as the DAG with three edges $u \rightarrow v$, $v \rightarrow w$, and $u \rightarrow w$. Both of these DAGs produce the same partial order, in which the vertices are ordered as $u \leq v \leq w$.

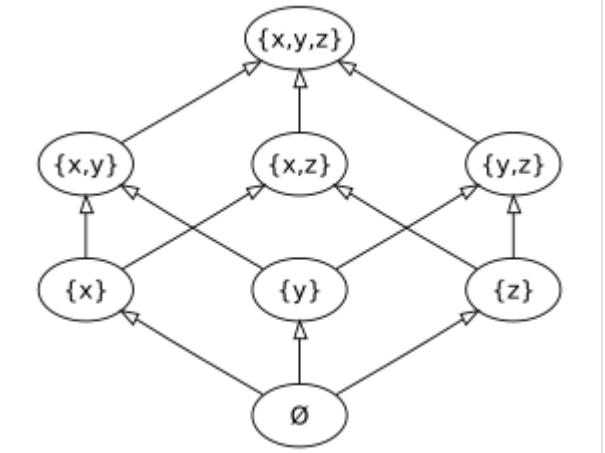


A DAG

Its transitive reduction

The transitive closure of a DAG is the graph with the most edges that has the same reachability relation as the DAG. It has an edge $u \rightarrow v$ for every pair of vertices (u, v) in the reachability relation $\leq$ of the DAG, and may therefore be thought of as a direct translation of the reachability relation $\leq$ into graph-theoretic terms. The same method of translating partial orders into DAGs works more generally: for every finite partially ordered set $(S, \leq)$, the graph that has a vertex for every element of S and an edge for every pair of elements in $\leq$ is automatically a transitively closed DAG, and has $(S, \leq)$ as its reachability relation. In this way, every finite partially ordered set can be represented as a DAG.

The transitive reduction of a DAG is the graph with the fewest edges that has the same reachability relation as the DAG. It has an edge $u \rightarrow v$ for every pair of vertices (u, v) in the covering relation of the reachability relation $\leq$ of the DAG. It is a subgraph of the DAG, formed by discarding the edges $u \rightarrow v$ for which the DAG also contains a longer directed path from u to v. Like the transitive closure, the transitive reduction is uniquely defined for DAGs. In contrast, for a directed graph that is not acyclic, there can be more than one minimal subgraph with the same reachability relation.[7] Transitive reductions are useful in visualizing the partial orders they represent, because they have fewer edges than other graphs representing the same orders and therefore lead to simpler graph drawings. A Hasse diagram of a partial order is a drawing of the transitive reduction in which the orientation of every edge is shown by placing the starting vertex of the edge in a lower position than its ending vertex.[8]



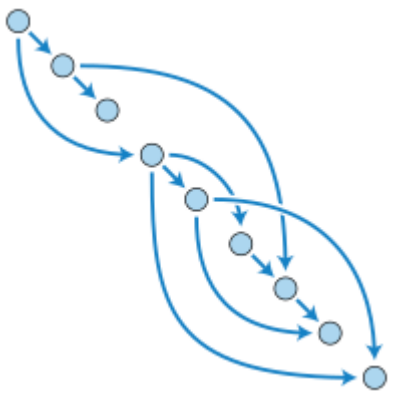
A Hasse diagram representing the partial order of set inclusion ($\subseteq$) among the subsets of a three-element set

Topological ordering

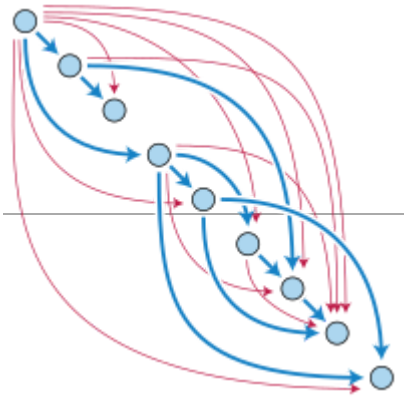
A topological ordering of a directed graph is an ordering of its vertices into a sequence, such that for every edge the start vertex of the edge occurs earlier in the sequence than the ending vertex of the edge. A graph that has a topological ordering cannot have any cycles, because the edge into the earliest vertex of a cycle would have to be oriented the wrong way. Therefore, every graph with a topological ordering is acyclic. Conversely, every directed acyclic graph has at least one topological ordering. The existence of a topological ordering can therefore be used as an equivalent definition of a directed acyclic graphs: they

are exactly the graphs that have topological orderings.[2] In general, this ordering is not unique; a DAG has a unique topological ordering if and only if it has a directed path containing all the vertices, in which case the ordering is the same as the order in which the vertices appear in the path.[9]

The family of topological orderings of a DAG is the same as the family of linear extensions of the reachability relation for the DAG,[10] so any two graphs representing the same partial order have the same set of topological orders.



A topological ordering of a directed acyclic graph: every edge goes from earlier in the ordering (upper left) to later in the ordering (lower right). A directed graph is acyclic if and only if it has a topological ordering.



Adding the red edges to the blue directed acyclic graph produces another DAG, the transitive closure of the blue graph. For each red or blue edge $u \to v$, v is reachable from u: there exists a blue path starting at u and ending at v.

Combinatorial enumeration

The graph enumeration problem of counting directed acyclic graphs was studied by Robinson (1973).[11] The number of DAGs on n labeled vertices, for $n = 0, 1, 2, 3, \ldots$ (without restrictions on the order in which these numbers appear in a topological ordering of the DAG) is

1, 1, 3, 25, 543, 29281, 3781503, … (sequence A003024 in the OEIS).

These numbers may be computed by the recurrence relation

$$a_n = \sum_{k=1}^{n} (-1)^{k-1} \binom{n}{k} 2^{k(n-k)} a_{n-k}.$$[11]

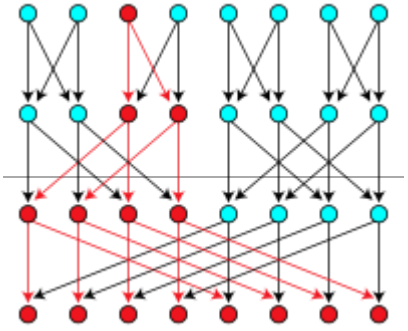
Eric W. Weisstein conjectured,[12] and McKay et al. (2004) proved, that the same numbers count the (0,1) matrices for which all eigenvalues are positive real numbers. The proof is bijective: a matrix A is an adjacency matrix of a DAG if and only if $A + I$ is a (0,1) matrix with all eigenvalues positive, where I denotes the identity matrix. Because a DAG cannot have self-loops, its adjacency matrix must have a zero diagonal, so adding I preserves the property that all matrix coefficients are 0 or 1.[13]

Related families of graphs

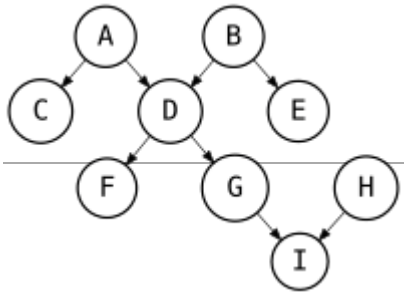
A *multitree* (also called a *strongly unambiguous graph* or a *mangrove*) is a DAG in which there is at most one directed path between any two vertices. Equivalently, it is a DAG in which the subgraph reachable from any vertex induces an undirected tree.[14]

A *polytree* (also called a *directed tree*) is a multitree formed by orienting the edges of an undirected tree.[15]

An *arborescence* is a polytree formed by orienting the edges of an undirected tree away from a particular vertex, called the *root* of the arborescence.



A multitree, a DAG in which the subgraph reachable from any vertex induces an undirected tree (e.g. in red)



A polytree, a DAG formed by orienting the edges of an undirected tree

Computational problems

Topological sorting and recognition

Topological sorting is the algorithmic problem of finding a topological ordering of a given DAG. It can be solved in linear time.[16] Kahn's algorithm for topological sorting builds the vertex ordering directly. It maintains a list of vertices that have no incoming edges from other vertices that have not already been included in the partially constructed topological ordering; initially this list consists of the vertices with no incoming edges at all. Then, it repeatedly adds one vertex from this list to the end of the partially constructed topological ordering, and checks whether its neighbors should be added to the list. The algorithm terminates when all vertices have been processed in this way.[17] Alternatively, a topological ordering may be constructed by reversing a postorder numbering of a depth-first search graph traversal.[16]

It is also possible to check whether a given directed graph is a DAG in linear time, either by attempting to find a topological ordering and then testing for each edge whether the resulting ordering is valid[18] or alternatively, for some topological sorting algorithms, by verifying that the algorithm successfully orders all the vertices without meeting an error condition.[17]

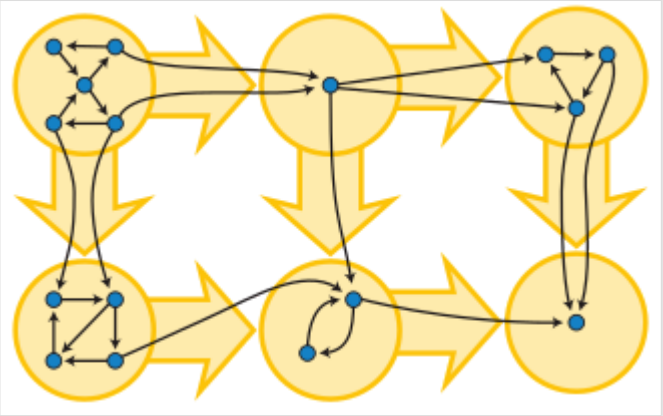
Construction from cyclic graphs

Any undirected graph may be made into a DAG by choosing a total order for its vertices and directing every edge from the earlier endpoint in the order to the later endpoint. The resulting orientation of the edges is called an acyclic orientation. Different total orders may lead to the same acyclic orientation, so an n-vertex graph can have fewer than $n!$ acyclic orientations. The number of acyclic orientations is equal to $|\chi(-1)|$, where χ is the chromatic polynomial of the given graph.[19]

Any directed graph may be made into a DAG by removing a feedback vertex set or a feedback arc set, a set of vertices or edges (respectively) that touches all cycles. However, the smallest such set is NP-hard to find.[20] An arbitrary directed graph may also be transformed into a DAG, called its condensation, by contracting each of its strongly connected components into a single supervertex.[21] When the graph is already acyclic, its smallest feedback vertex sets and feedback arc sets are empty, and its condensation is the graph itself.

Transitive closure and transitive reduction

The transitive closure of a given DAG, with n vertices and m edges, may be constructed in time $O(mn)$ by using either breadth-first search or depth-first search to test reachability from each vertex.[22] Alternatively, it can be solved in time $O(n^\omega)$ where $\omega < 2.373$ is the exponent for matrix multiplication algorithms; this is a theoretical improvement over the $O(mn)$ bound for dense graphs.[23]



The yellow directed acyclic graph is the condensation of the blue directed graph. It is formed by contracting each strongly connected component of the blue graph into a single yellow vertex.

In all of these transitive closure algorithms, it is possible to distinguish pairs of vertices that are reachable by at least one path of length two or more from pairs that can only be connected by a length-one path. The transitive reduction consists of the edges that form length-one paths that are the only paths connecting their endpoints. Therefore, the transitive reduction can be constructed in the same asymptotic time bounds as the transitive closure.[24]

Closure problem

The closure problem takes as input a vertex-weighted directed acyclic graph and seeks the minimum (or maximum) weight of a closure – a set of vertices C, such that no edges leave C. The problem may be formulated for directed graphs without the assumption of acyclicity, but with no greater generality, because in this case it is equivalent to the same problem on the condensation of the graph. It may be solved in polynomial time using a reduction to the maximum flow problem.[25]

Path algorithms

Some algorithms become simpler when used on DAGs instead of general graphs, based on the principle of topological ordering. For example, it is possible to find shortest paths and longest paths from a given starting vertex in DAGs in linear time by processing the vertices in a topological order, and calculating the path length for each vertex to be the minimum or maximum length obtained via any of its incoming edges.[26] In contrast, for arbitrary graphs the shortest path may require slower algorithms such as Dijkstra's algorithm or the Bellman–Ford algorithm,[27] and longest paths in arbitrary graphs are NP-hard to find.[28]

Applications

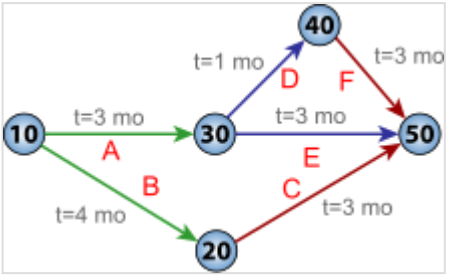
Scheduling

Directed acyclic graph representations of partial orderings have many applications in scheduling for systems of tasks with ordering constraints.[29] An important class of problems of this type concern collections of objects that need to be updated, such as the cells of a spreadsheet after one of the cells has been changed, or the object files of a piece of computer software after its source code has been changed.

In this context, a dependency graph is a graph that has a vertex for each object to be updated, and an edge connecting two objects whenever one of them needs to be updated earlier than the other. A cycle in this graph is called a circular dependency, and is generally not allowed, because there would be no way to consistently schedule the tasks involved in the cycle. Dependency graphs without circular dependencies form DAGs.[30]

For instance, when one cell of a spreadsheet changes, it is necessary to recalculate the values of other cells that depend directly or indirectly on the changed cell. For this problem, the tasks to be scheduled are the recalculations of the values of individual cells of the spreadsheet. Dependencies arise when an expression in one cell uses a value from another cell. In such a case, the value that is used must be recalculated earlier than the expression that uses it. Topologically ordering the dependency graph, and using this topological order to schedule the cell updates, allows the whole spreadsheet to be updated with only a single evaluation per cell.[31] Similar problems of task ordering arise in makefiles for program compilation[31] and instruction scheduling for low-level computer program optimization.[32]

A somewhat different DAG-based formulation of scheduling constraints is used by the program evaluation and review technique (PERT), a method for management of large human projects that was one of the first applications of DAGs. In this method, the vertices of a DAG represent milestones of a project rather than specific tasks to be performed. Instead, a task or activity is represented by an edge of a DAG, connecting two milestones that mark the beginning and completion of the task. Each such edge is labeled with an estimate for the amount of time that it will take a team of workers to perform the task. The longest path in this DAG represents the critical path of the project, the one that controls the total time for the project. Individual milestones can be scheduled according to the lengths of the longest paths ending at their vertices.[33]



PERT chart for a project with five milestones (labeled 10–50) and six tasks (labeled A–F). There are two critical paths, ADF and BC.

Data processing networks

A directed acyclic graph may be used to represent a network of processing elements. In this representation, data enters a processing element through its incoming edges and leaves the element through its outgoing edges.

For instance, in electronic circuit design, static combinational logic blocks can be represented as an acyclic system of logic gates that computes a function of an input, where the input and output of the function are represented as individual bits. In general, the output of these blocks cannot be used as the input unless it is captured by a register or state element which maintains its acyclic properties.[34] Electronic circuit schematics either on paper or in a database are a form of directed acyclic graphs using instances or components to form a directed reference to a lower level component. Electronic circuits themselves are not necessarily acyclic or directed.

Dataflow programming languages describe systems of operations on data streams, and the connections between the outputs of some operations and the inputs of others. These languages can be convenient for describing repetitive data processing tasks, in which the same acyclically-connected collection of operations is applied to many data items. They can be executed as a parallel algorithm in which each operation is performed by a parallel process as soon as another set of inputs becomes available to it.[35]

In compilers, straight line code (that is, sequences of statements without loops or conditional branches) may be represented by a DAG describing the inputs and outputs of each of the arithmetic operations performed within the code. This representation allows the compiler to perform common subexpression elimination efficiently.[36] At a higher level of code organization, the acyclic dependencies principle states that the dependencies between modules or components of a large software system should form a directed acyclic graph.[37]

Feedforward neural networks are another example.

Causal structures

Graphs in which vertices represent events occurring at a definite time, and where the edges always point from the early time vertex to a late time vertex of the edge, are necessarily directed and acyclic. The lack of a cycle follows because the time associated with a vertex always increases as you follow any path in the graph so you can never return to a vertex on a path. This reflects our natural intuition that causality means events can only affect the future, they never affect the past, and thus we have no causal loops. An example of this type of directed acyclic graph are those encountered in the causal set approach to quantum gravity though in this case the graphs considered are transitively complete. In the version history example below, each version of the software is associated with a unique time, typically the time the version was saved, committed or released. In the citation graph examples below, the documents are published at one time and can only refer to older documents.

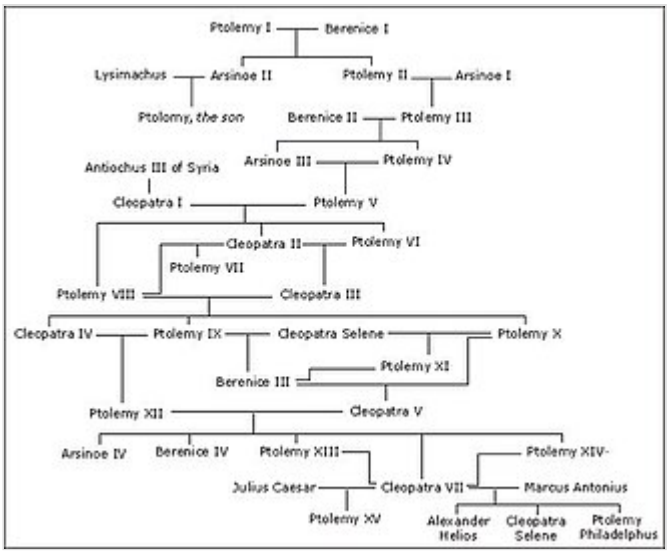
Sometimes events are not associated with a specific physical time. Provided that pairs of events have a purely causal relationship, that is edges represent causal relations between the events, we will have a directed acyclic graph.[38] For instance, a Bayesian network represents a system of probabilistic events as vertices in a directed acyclic graph, in which the likelihood of an event may be calculated from the likelihoods of its predecessors in the DAG.[39] In this context, the moral graph of a DAG is the undirected graph created by adding an (undirected) edge between all parents of the same vertex (sometimes called *marrying*), and then replacing all directed edges by undirected edges.[40] Another type of graph with a similar causal structure is an influence diagram, the vertices of which represent either decisions to be made or unknown information, and the edges of which represent causal influences from one vertex to another.[41] In epidemiology, for instance, these diagrams are often used to estimate the expected value of different choices for intervention.[42][43]

The converse is also true. That is in any application represented by a directed acyclic graph there is a causal structure, either an explicit order or time in the example or an order which can be derived from graph structure. This follows because all directed acyclic graphs have a topological ordering, i.e. there is at least one way to put the vertices in an order such that all edges point in the same direction along that order.

Genealogy and version history

Family trees may be seen as directed acyclic graphs, with a vertex for each family member and an edge for each parent-child relationship.[44] Despite the name, these graphs are not necessarily trees because of the possibility of marriages between relatives (so a child has a common ancestor on both the mother's and

father's side) causing pedigree collapse.[45] The graphs of matrilineal descent (mother-daughter relationships) and patrilineal descent (father-son relationships) are trees within this graph. Because no one can become their own ancestor, family trees are acyclic.[46]



Family tree of the Ptolemaic dynasty, with many marriages between close relatives causing pedigree collapse.

The version history of a distributed revision control system, such as Git, generally has the structure of a directed acyclic graph, in which there is a vertex for each revision and an edge connecting pairs of revisions that were directly derived from each other. These are not trees in general due to merges.[47]

In many randomized algorithms in computational geometry, the algorithm maintains a *history DAG* representing the version history of a geometric structure over the course of a sequence of changes to the structure. For instance in a randomized incremental algorithm for Delaunay triangulation, the triangulation changes by replacing one triangle by three smaller triangles when each point is added, and by "flip" operations that replace pairs of triangles by a different pair of triangles. The history DAG for this algorithm has a vertex for each triangle constructed as part of the algorithm, and edges from each triangle to the two or three other triangles that replace it. This structure allows point location queries to be answered efficiently: to find the location of a query point q in the Delaunay triangulation, follow a path in the history DAG, at each step moving to the replacement triangle that contains q. The final triangle reached in this path must be the Delaunay triangle that contains q.[48]

Citation graphs

In a citation graph the vertices are documents with a single publication date. The edges represent the citations from the bibliography of one document to other necessarily earlier documents. The classic example comes from the citations between academic papers as pointed out in the 1965 article "Networks of Scientific Papers"[49] by Derek J. de Solla Price who went on to produce the first model of a citation network, the Price model.[50] In this case the citation count of a paper is just the in-degree of the corresponding vertex of the citation network. This is an important measure in citation analysis. Court judgements provide another example as judges support their conclusions in one case by recalling other earlier decisions made in previous cases. A final example is provided by patents which must refer to earlier prior art, earlier patents which are relevant to the current patent claim. By taking the special properties of directed acyclic graphs into account, one can analyse citation networks with techniques not available when analysing the general graphs considered in many studies using network analysis. For instance transitive reduction gives new insights into the citation distributions found in different applications highlighting clear differences in the mechanisms creating citations networks in different contexts.[51] Another technique is main path analysis, which traces the citation links and suggests the most significant citation chains in a given citation graph.

The Price model is too simple to be a realistic model of a citation network but it is simple enough to allow for analytic solutions for some of its properties. Many of these can be found by using results derived from the undirected version of the Price model, the Barabási–Albert model. However, since Price's model gives a directed acyclic graph, it is a useful model when looking for analytic calculations of properties unique to directed acyclic graphs. For instance, the length of the longest path, from the n-th node added to the network to the first node in the network, scales as[52] $\ln(n)$.

Data compression

Directed acyclic graphs may also be used as a compact representation of a collection of sequences. In this type of application, one finds a DAG in which the paths form the given sequences. When many of the sequences share the same subsequences, these shared subsequences can be represented by a shared part of the DAG, allowing the representation to use less space than it would take to list out all of the sequences separately. For example, the directed acyclic word graph is a data structure in computer science formed by a directed acyclic graph with a single source and with edges labeled by letters or symbols; the paths from the source to the sinks in this graph represent a set of strings, such as English words.[53] Any set of sequences can be represented as paths in a tree, by forming a tree vertex for every prefix of a sequence and making the parent of one of these vertices represent the sequence with one fewer element; the tree formed in this way for a set of strings is called a trie. A directed acyclic word graph saves space over a trie by allowing paths to diverge and rejoin, so that a set of words with the same possible suffixes can be represented by a single tree vertex.[54]

The same idea of using a DAG to represent a family of paths occurs in the binary decision diagram,[55][56] a DAG-based data structure for representing binary functions. In a binary decision diagram, each non-sink vertex is labeled by the name of a binary variable, and each sink and each edge is labeled by a 0 or 1. The function value for any truth assignment to the variables is the value at the sink found by following a path, starting from the single source vertex, that at each non-sink vertex follows the outgoing edge labeled with the value of that vertex's variable. Just as directed acyclic word graphs can be viewed as a compressed form of tries, binary decision diagrams can be viewed as compressed forms of decision trees that save space by allowing paths to rejoin when they agree on the results of all remaining decisions.[57]

References

1. Thulasiraman, K.; Swamy, M. N. S. (1992), "5.7 Acyclic Directed Graphs", *Graphs: Theory and Algorithms*, John Wiley and Son, p. 118, ISBN 978-0-471-51356-8.
2. Bang-Jensen, Jørgen (2008), "2.1 Acyclic Digraphs", *Digraphs: Theory, Algorithms and Applications*, Springer Monographs in Mathematics (2nd ed.), Springer-Verlag, pp. 32–34, ISBN 978-1-84800-997-4.
3. Christofides, Nicos (1975), *Graph theory: an algorithmic approach*, Academic Press, pp. 170–174.
4. Mitrani, I. (1982), *Simulation Techniques for Discrete Event Systems* (https://books.google.com/books?id=CF04AAAAIAAJ&pg=PA27), Cambridge Computer Science Texts, vol. 14, Cambridge University Press, p. 27, ISBN 9780521282826.
5. Kozen, Dexter (1992), *The Design and Analysis of Algorithms* (https://books.google.com/books?id=L_AMnf9UF9QC&pg=PA9), Monographs in Computer Science, Springer, p. 9, ISBN 978-0-387-97687-7.

6. Banerjee, Utpal (1993), "Exercise 2(c)", *Loop Transformations for Restructuring Compilers: The Foundations* (https://books.google.com/books?id=Cog7zSSlqFwC&pg=PA19), Springer, p. 19, Bibcode:1993ltfr.book.....B (https://ui.adsabs.harvard.edu/abs/1993ltfr.book.....B), ISBN 978-0-7923-9318-4.
7. Bang-Jensen, Jørgen; Gutin, Gregory Z. (2008), "2.3 Transitive Digraphs, Transitive Closures and Reductions", *Digraphs: Theory, Algorithms and Applications* (https://books.google.com/books?id=4UY-ucucWucC&pg=PA36), Springer Monographs in Mathematics, Springer, pp. 36–39, ISBN 978-1-84800-998-1.
8. Jungnickel, Dieter (2012), *Graphs, Networks and Algorithms* (https://books.google.com/books?id=PrXxFHmchwcC&pg=PA92), Algorithms and Computation in Mathematics, vol. 5, Springer, pp. 92–93, ISBN 978-3-642-32278-5.
9. Sedgewick, Robert; Wayne, Kevin (2011), "4,2,25 Unique topological ordering", *Algorithms* (https://books.google.com/books?id=idUdqdDXqnAC&pg=PA598) (4th ed.), Addison-Wesley, pp. 598–599, ISBN 978-0-13-276256-4.
10. Bender, Edward A.; Williamson, S. Gill (2005), "Example 26 (Linear extensions – topological sorts)", *A Short Course in Discrete Mathematics* (https://books.google.com/books?id=iuEoAwAAQBAJ&pg=PA142), Dover Books on Computer Science, Courier Dover Publications, p. 142, ISBN 978-0-486-43946-4.
11. Robinson, R. W. (1973), "Counting labeled acyclic digraphs", in Harary, F. (ed.), *New Directions in the Theory of Graphs*, Academic Press, pp. 239–273. See also Harary, Frank; Palmer, Edgar M. (1973), *Graphical Enumeration*, Academic Press, p. 19, ISBN 978-0-12-324245-7.
12. Weisstein, Eric W., "Weisstein's Conjecture" (https://mathworld.wolfram.com/WeissteinsConjecture.html), *MathWorld*
13. McKay, B. D.; Royle, G. F.; Wanless, I. M.; Oggier, F. E.; Sloane, N. J. A.; Wilf, H. (2004), "Acyclic digraphs and eigenvalues of (0,1)-matrices" (http://www.cs.uwaterloo.ca/journals/JIS/VOL7/Sloane/sloane15.html), *Journal of Integer Sequences*, **7**: 33, arXiv:math/0310423 (https://arxiv.org/abs/math/0310423), Bibcode:2004JIntS...7...33M (https://ui.adsabs.harvard.edu/abs/2004JIntS...7...33M), Article 04.3.3.
14. Furnas, George W.; Zacks, Jeff (1994), "Multitrees: enriching and reusing hierarchical structure", *Proc. SIGCHI conference on Human Factors in Computing Systems (CHI '94)*, pp. 330–336, doi:10.1145/191666.191778 (https://doi.org/10.1145%2F191666.191778), ISBN 978-0897916509, S2CID 18710118 (https://api.semanticscholar.org/CorpusID:18710118).
15. Rebane, George; Pearl, Judea (1987), "The recovery of causal poly-trees from statistical data", *in Proc. 3rd Annual Conference on Uncertainty in Artificial Intelligence (UAI 1987), Seattle, WA, USA, July 1987* (http://ftp.cs.ucla.edu/tech-report/198_-reports/870031.pdf) (PDF), pp. 222–228.
16. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001) [1990], *Introduction to Algorithms* (2nd ed.), MIT Press and McGraw-Hill, ISBN 0-262-03293-7 Section 22.4, Topological sort, pp. 549–552.
17. Jungnickel (2012), pp. 50–51.
18. For depth-first search based topological sorting algorithm, this validity check can be interleaved with the topological sorting algorithm itself; see e.g. Skiena, Steven S. (2009), *The Algorithm Design Manual* (https://books.google.com/books?id=7XUSn0IKQEgC&pg=PA179), Springer, pp. 179–181, ISBN 978-1-84800-070-4.
19. Stanley, Richard P. (1973), "Acyclic orientations of graphs" (http://math.mit.edu/~rstan/pubs/pubfiles/18.pdf) (PDF), *Discrete Mathematics*, **5** (2): 171–178, doi:10.1016/0012-365X(73)90108-8 (https://doi.org/10.1016%2F0012-365X%2873%2990108-8).

20. Garey, Michael R.; Johnson, David S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences (1st ed.). New York: W. H. Freeman and Company. ISBN 9780716710455. MR 0519066 (https://mathscinet.ams.org/mathscinet-getitem?mr=0519066). OCLC 247570676 (https://search.worldcat.org/oclc/247570676)., Problems GT7 and GT8, pp. 191–192.
21. Harary, Frank; Norman, Robert Z.; Cartwright, Dorwin (1965), *Structural Models: An Introduction to the Theory of Directed Graphs*, John Wiley & Sons, p. 63.
22. Skiena (2009), p. 495.
23. Skiena (2009), p. 496.
24. Bang-Jensen & Gutin (2008), p. 38.
25. Picard, Jean-Claude (1976), "Maximal closure of a graph and applications to combinatorial problems", *Management Science*, **22** (11): 1268–1272, doi:10.1287/mnsc.22.11.1268 (https://doi.org/10.1287%2Fmnsc.22.11.1268), MR 0403596 (https://mathscinet.ams.org/mathscinet-getitem?mr=0403596).
26. Cormen et al. 2001, Section 24.2, Single-source shortest paths in directed acyclic graphs, pp. 592–595.
27. Cormen et al. 2001, Sections 24.1, The Bellman–Ford algorithm, pp. 588–592, and 24.3, Dijkstra's algorithm, pp. 595–601.
28. Cormen et al. 2001, p. 966.
29. Skiena (2009), p. 469.
30. Al-Mutawa, H. A.; Dietrich, J.; Marsland, S.; McCartin, C. (2014), "On the shape of circular dependencies in Java programs", *23rd Australian Software Engineering Conference*, IEEE, pp. 48–57, doi:10.1109/ASWEC.2014.15 (https://doi.org/10.1109%2FASWEC.2014.15), ISBN 978-1-4799-3149-1, S2CID 17570052 (https://api.semanticscholar.org/CorpusID:17570052).
31. Gross, Jonathan L.; Yellen, Jay; Zhang, Ping (2013), *Handbook of Graph Theory* (https://books.google.com/books?id=cntcAgAAQBAJ&pg=PA1181) (2nd ed.), CRC Press, p. 1181, ISBN 978-1-4398-8018-0.
32. Srikant, Y. N.; Shankar, Priti (2007), *The Compiler Design Handbook: Optimizations and Machine Code Generation* (https://books.google.com/books?id=1kqAv-uDEPEC&pg=SA19-PA39) (2nd ed.), CRC Press, pp. 19–39, ISBN 978-1-4200-4383-9.
33. Wang, John X. (2002), *What Every Engineer Should Know About Decision Making Under Uncertainty* (https://books.google.com/books?id=C3yKML0dUVIC&pg=PA160), CRC Press, p. 160, ISBN 978-0-8247-4373-4.
34. Sapatnekar, Sachin (2004), *Timing* (https://books.google.com/books?id=fL9k-VkZVr0C&pg=PA133), Springer, p. 133, ISBN 978-1-4020-7671-8.
35. Dennis, Jack B. (1974), "First version of a data flow procedure language", *Programming Symposium*, Lecture Notes in Computer Science, vol. 19, pp. 362–376, doi:10.1007/3-540-06859-7_145 (https://doi.org/10.1007%2F3-540-06859-7_145), ISBN 978-3-540-06859-4.
36. Touati, Sid; de Dinechin, Benoit (2014), *Advanced Backend Optimization* (https://books.google.com/books?id=nO2-AwAAQBAJ&pg=PA123), John Wiley & Sons, p. 123, ISBN 978-1-118-64894-0.
37. Garland, Jeff; Anthony, Richard (2003), *Large-Scale Software Architecture: A Practical Guide using UML* (https://books.google.com/books?id=_2oQLLSqZ88C&pg=PA215), John Wiley & Sons, p. 215, ISBN 9780470856383.
38. Gopnik, Alison; Schulz, Laura (2007), *Causal Learning* (https://books.google.com/books?id=35MKXlKoXIUC&pg=PA4), Oxford University Press, p. 4, ISBN 978-0-19-803928-0.
39. Shmulevich, Ilya; Dougherty, Edward R. (2010), *Probabilistic Boolean Networks: The Modeling and Control of Gene Regulatory Networks* (https://books.google.com/books?id=RfshqEgO7KgC&pg=PA58), Society for Industrial and Applied Mathematics, p. 58, ISBN 978-0-89871-692-4.

40. Cowell, Robert G.; Dawid, A. Philip; Lauritzen, Steffen L.; Spiegelhalter, David J. (1999), "3.2.1 Moralization", *Probabilistic Networks and Expert Systems*, Springer, pp. 31–33, ISBN 978-0-387-98767-5.
41. Dorf, Richard C. (1998), *The Technology Management Handbook* (https://books.google.com/books?id=C2u8I0DFo4IC&pg=SA9-PA7), CRC Press, p. 9-7, ISBN 978-0-8493-8577-3.
42. Boslaugh, Sarah (2008), *Encyclopedia of Epidemiology, Volume 1* (https://books.google.com/books?id=wObgnN3x14kC&pg=PA255), SAGE, p. 255, ISBN 978-1-4129-2816-8.
43. Pearl, Judea (1995), "Causal diagrams for empirical research" (https://escholarship.org/uc/item/6gv9n38c), *Biometrika*, **82** (4): 669–709, doi:10.1093/biomet/82.4.669 (https://doi.org/10.1093%2Fbiomet%2F82.4.669).
44. Kirkpatrick, Bonnie B. (April 2011), "Haplotypes versus genotypes on pedigrees", *Algorithms for Molecular Biology*, **6** (10): 10, doi:10.1186/1748-7188-6-10 (https://doi.org/10.1186%2F1748-7188-6-10), PMC 3102622 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3102622), PMID 21504603 (https://pubmed.ncbi.nlm.nih.gov/21504603).
45. McGuffin, M. J.; Balakrishnan, R. (2005), "Interactive visualization of genealogical graphs" (http://profs.etsmtl.ca/mMcGuffin/research/genealogyVis/genealogyVis.pdf) (PDF), *IEEE Symposium on Information Visualization (INFOVIS 2005)*, pp. 16–23, doi:10.1109/INFVIS.2005.1532124 (https://doi.org/10.1109%2FINFVIS.2005.1532124), ISBN 978-0-7803-9464-3, S2CID 15449409 (https://api.semanticscholar.org/CorpusID:15449409).
46. Bender, Michael A.; Pemmasani, Giridhar; Skiena, Steven; Sumazin, Pavel (2001), "Finding least common ancestors in directed acyclic graphs" (http://dl.acm.org/citation.cfm?id=365411.365795), *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '01)*, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, pp. 845–854, ISBN 978-0-89871-490-6.
47. Bartlang, Udo (2010), *Architecture and Methods for Flexible Content Management in Peer-to-Peer Systems* (https://books.google.com/books?id=vXdEAAAAQBAJ&pg=PA59), Springer, p. 59, Bibcode:2010aamf.book.....B (https://ui.adsabs.harvard.edu/abs/2010aamf.book.....B), ISBN 978-3-8348-9645-2.
48. Pach, János; Sharir, Micha, *Combinatorial Geometry and Its Algorithmic Applications: The Alcalá Lectures* (https://books.google.com/books?id=-fguzNaYoqcC&pg=PA93), Mathematical surveys and monographs, vol. 152, American Mathematical Society, pp. 93–94, ISBN 978-0-8218-7533-9.
49. Price, Derek J. de Solla (July 30, 1965), "Networks of Scientific Papers" (http://garfield.library.upenn.edu/papers/pricenetworks1965.pdf) (PDF), *Science*, **149** (3683): 510–515, Bibcode:1965Sci...149..510D (https://ui.adsabs.harvard.edu/abs/1965Sci...149..510D), doi:10.1126/science.149.3683.510 (https://doi.org/10.1126%2Fscience.149.3683.510), PMID 14325149 (https://pubmed.ncbi.nlm.nih.gov/14325149).
50. Price, Derek J. de Solla (1976), "A general theory of bibliometric and other cumulative advantage processes", *Journal of the American Society for Information Science*, **27** (5): 292–306, doi:10.1002/asi.4630270505 (https://doi.org/10.1002%2Fasi.4630270505), S2CID 8536863 (https://api.semanticscholar.org/CorpusID:8536863).
51. Clough, James R.; Gollings, Jamie; Loach, Tamar V.; Evans, Tim S. (2015), "Transitive reduction of citation networks", *Journal of Complex Networks*, **3** (2): 189–203, arXiv:1310.8224 (https://arxiv.org/abs/1310.8224), doi:10.1093/comnet/cnu039 (https://doi.org/10.1093%2Fcomnet%2Fcnu039), S2CID 10228152 (https://api.semanticscholar.org/CorpusID:10228152).

52. Evans, T.S.; Calmon, L.; Vasiliauskaite, V. (2020), "The Longest Path in the Price Model", *Scientific Reports*, **10** (1): 10503, arXiv:1903.03667 (https://arxiv.org/abs/1903.03667), Bibcode:2020NatSR..1010503E (https://ui.adsabs.harvard.edu/abs/2020NatSR..1010503E), doi:10.1038/s41598-020-67421-8 (https://doi.org/10.1038%2Fs41598-020-67421-8), PMC 7324613 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7324613), PMID 32601403 (https://pubmed.ncbi.nlm.nih.gov/32601403)
53. Crochemore, Maxime; Vérin, Renaud (1997), "Direct construction of compact directed acyclic word graphs", *Combinatorial Pattern Matching*, Lecture Notes in Computer Science, vol. 1264, Springer, pp. 116–129, CiteSeerX 10.1.1.53.6273 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.6273), doi:10.1007/3-540-63220-4_55 (https://doi.org/10.1007%2F3-540-63220-4_55), ISBN 978-3-540-63220-7, S2CID 17045308 (https://api.semanticscholar.org/CorpusID:17045308).
54. Lothaire, M. (2005), *Applied Combinatorics on Words* (https://books.google.com/books?id=fpLUNkj1T1EC&pg=PA18), Encyclopedia of Mathematics and its Applications, vol. 105, Cambridge University Press, p. 18, ISBN 9780521848022.
55. Lee, C. Y. (1959), "Representation of switching circuits by binary-decision programs", *Bell System Technical Journal*, **38** (4): 985–999, doi:10.1002/j.1538-7305.1959.tb01585.x (https://doi.org/10.1002%2Fj.1538-7305.1959.tb01585.x).
56. Akers, Sheldon B. (1978), "Binary decision diagrams", *IEEE Transactions on Computers*, **C-27** (6): 509–516, doi:10.1109/TC.1978.1675141 (https://doi.org/10.1109%2FTC.1978.1675141), S2CID 21028055 (https://api.semanticscholar.org/CorpusID:21028055).
57. Friedman, S. J.; Supowit, K. J. (1987), "Finding the optimal variable ordering for binary decision diagrams", *Proc. 24th ACM/IEEE Design Automation Conference (DAC '87)*, New York, NY, USA: ACM, pp. 348–356, doi:10.1145/37888.37941 (https://doi.org/10.1145%2F37888.37941), ISBN 978-0-8186-0781-3, S2CID 14796451 (https://api.semanticscholar.org/CorpusID:14796451).

External links

- Weisstein, Eric W., "Acyclic Digraph" (https://mathworld.wolfram.com/AcyclicDigraph.html), *MathWorld*
- DAGitty (http://www.dagitty.net/) – an online tool for creating DAGs