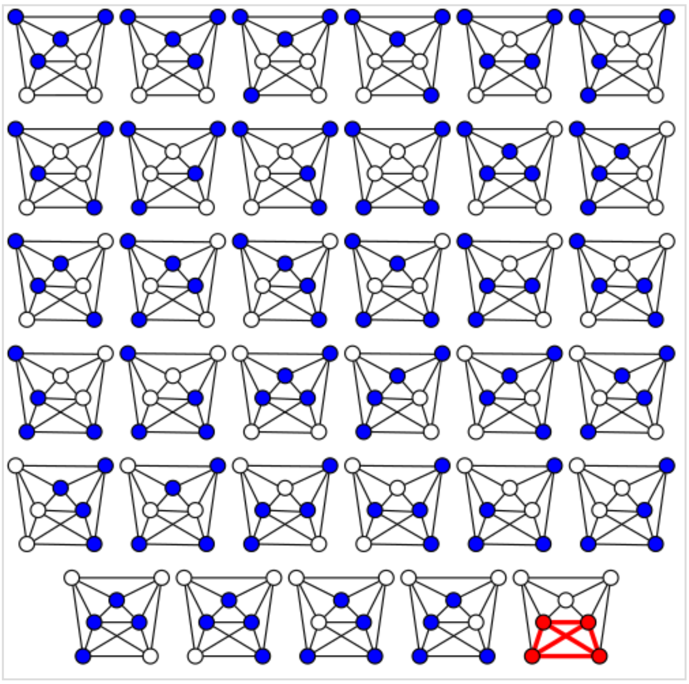
WIKIPEDIA
The Free Encyclopedia

Clique problem

In computer science, the **clique problem** is the computational problem of finding cliques (subsets of vertices, all adjacent to each other, also called complete subgraphs) in a graph. It has several different formulations depending on which cliques, and what information about the cliques, should be found. Common formulations of the clique problem include finding a maximum clique (a clique with the largest possible number of vertices), finding a maximum weight clique in a weighted graph, listing all maximal cliques (cliques that cannot be enlarged), and solving the decision problem of testing whether a graph contains a clique larger than a given size.


The brute force algorithm finds a 4-clique in this 7-vertex graph (the complement of the 7-vertex path graph) by systematically checking all C(7,4) = 35 4-vertex subgraphs for completeness.

The clique problem arises in the following real-world setting. Consider a social network, where the graph's vertices represent people, and the graph's edges represent mutual acquaintance. Then a clique represents a subset of people who all know each other, and algorithms for finding cliques can be used to discover these groups of mutual friends. Along with its applications in social networks, the clique problem also has many applications in bioinformatics, and computational chemistry.

Most versions of the clique problem are hard. The clique decision problem is NP-complete (one of Karp's 21 NP-complete problems). The problem of finding the maximum clique is both fixed-parameter intractable and hard to approximate. And, listing all maximal cliques may require exponential time as there exist graphs with exponentially many maximal cliques. Therefore, much of the theory about the clique problem is devoted to identifying special types of graphs that admit more efficient algorithms, or to establishing the computational difficulty of the general problem in various models of computation.

To find a maximum clique, one can systematically inspect all subsets, but this sort of brute-force search is too time-consuming to be practical for networks comprising more than a few dozen vertices. Although no polynomial time algorithm is known for this problem, more efficient algorithms than the brute-force search are known. For instance, the Bron–Kerbosch algorithm can be used to list all maximal cliques in worst-case optimal time, and it is also possible to list them in polynomial time per clique.

History and applications

The study of complete subgraphs in mathematics predates the "clique" terminology. For instance, complete subgraphs make an early appearance in the mathematical literature in the graph-theoretic reformulation of Ramsey theory by Erdős & Szekeres (1935). But the term "clique" and the problem of algorithmically listing cliques both come from the social sciences, where complete subgraphs are used to model social cliques, groups of people who all know each other. Luce & Perry (1949) used graphs to model social networks, and adapted the social science terminology to graph theory. They were the first to call complete subgraphs "cliques". The first algorithm for solving the clique problem is that of Harary & Ross (1957),[1] who were motivated by the sociological application. Social science researchers have also defined various other types of cliques and maximal cliques in social network, "cohesive subgroups" of people or actors in the network all of whom share one of several different kinds of connectivity relation. Many of these generalized notions of cliques can also be found by constructing an undirected graph whose edges represent related pairs of actors from the social network, and then applying an algorithm for the clique problem to this graph.[2]

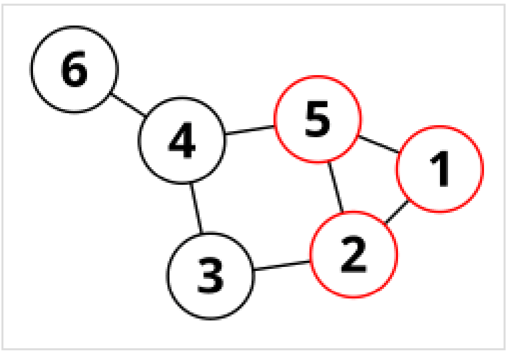
Since the work of Harary and Ross, many others have devised algorithms for various versions of the clique problem.[1] In the 1970s, researchers began studying these algorithms from the point of view of worst-case analysis. See, for instance, Tarjan & Trojanowski (1977), an early work on the worst-case complexity of the maximum clique problem. Also in the 1970s, beginning with the work of Cook (1971) and Karp (1972), researchers began using the theory of NP-completeness and related intractability results to provide a mathematical explanation for the perceived difficulty of the clique problem. In the 1990s, a breakthrough series of papers beginning with Feige et al. (1991) showed that (assuming P ≠ NP) it is not even possible to approximate the problem accurately and efficiently.

Clique-finding algorithms have been used in chemistry, to find chemicals that match a target structure[3] and to model molecular docking and the binding sites of chemical reactions.[4] They can also be used to find similar structures within different molecules.[5] In these applications, one forms a graph in which each vertex represents a matched pair of atoms, one from each of two molecules. Two vertices are connected by an edge if the matches that they represent are compatible with each other. Being compatible may mean, for instance, that the distances between the atoms within the two molecules are approximately equal, to within some given tolerance. A clique in this graph represents a set of matched pairs of atoms in which all the matches are compatible with each other.[6] A special case of this method is the use of the modular product of graphs to reduce the problem of finding the maximum common induced subgraph of two graphs to the problem of finding a maximum clique in their product.[7]

In automatic test pattern generation, finding cliques can help to bound the size of a test set.[8] In bioinformatics, clique-finding algorithms have been used to infer evolutionary trees,[9] predict protein structures,[10] and find closely interacting clusters of proteins.[11] Listing the cliques in a dependency graph is an important step in the analysis of certain random processes.[12] In mathematics, Keller's conjecture on face-to-face tiling of hypercubes was disproved by Lagarias & Shor (1992), who used a clique-finding algorithm on an associated graph to find a counterexample.[13]

Definitions

An undirected graph is formed by a finite set of vertices and a set of unordered pairs of vertices, which are called edges. By convention, in algorithm analysis, the number of vertices in the graph is denoted by n and the number of edges is denoted by m. A clique in a graph G is a complete subgraph of G. That is, it is a subset K of the vertices such that every two vertices in K are the two endpoints of an edge in G. A maximal clique is a clique to which no more vertices can be added. For each vertex v that is not part of a maximal clique, there must be another vertex w that is in the clique and non-adjacent to v, preventing v from being added to the clique. A maximum clique is a clique that includes the largest possible number of vertices. The clique number $\omega(G)$ is the number of vertices in a maximum clique of G.[1]



The graph shown has one maximum clique, the triangle {1,2,5}, and four more maximal cliques, the pairs {2,3}, {3,4}, {4,5}, and {4,6}.

Several closely related clique-finding problems have been studied.[14]

- In the maximum clique problem, the input is an undirected graph, and the output is a maximum clique in the graph. If there are multiple maximum cliques, one of them may be chosen arbitrarily.[14]
- In the weighted maximum clique problem, the input is an undirected graph with weights on its vertices (or, less frequently, edges) and the output is a clique with maximum total weight. The maximum clique problem is the special case in which all weights are equal.[15] As well as the problem of optimizing the sum of weights, other more complicated bicriterion optimization problems have also been studied.[16]
- In the maximal clique listing problem, the input is an undirected graph, and the output is a list of all its maximal cliques. The maximum clique problem may be solved using as a subroutine an algorithm for the maximal clique listing problem, because the maximum clique must be included among all the maximal cliques.[17]
- In the k-clique problem, the input is an undirected graph and a number k. The output is a clique with k vertices, if one exists, or a special value indicating that there is no k-clique otherwise. In some variations of this problem, the output should list all cliques of size k.[18]
- In the clique decision problem, the input is an undirected graph and a number k, and the output is a Boolean value: true if the graph contains a k-clique, and false otherwise.[19]

The first four of these problems are all important in practical applications. The clique decision problem is not of practical importance; it is formulated in this way in order to apply the theory of NP-completeness to clique-finding problems.[19]

The clique problem and the independent set problem are complementary: a clique in G is an independent set in the complement graph of G and vice versa.[20] Therefore, many computational results may be applied equally well to either problem, and some research papers do not clearly distinguish between the two problems. However, the two problems have different properties when applied to restricted families of graphs. For instance, the clique problem may be solved in polynomial time for planar graphs[21] while the independent set problem remains NP-hard on planar graphs.[22]

Algorithms

Finding a single maximal clique

A maximal clique, sometimes called inclusion-maximal, is a clique that is not included in a larger clique. Therefore, every clique is contained in a maximal clique.[23] Maximal cliques can be very small. A graph may contain a non-maximal clique with many vertices and a separate clique of size 2 which is maximal. While a maximum (i.e., largest) clique is necessarily maximal, the converse does not hold. There are some types of graphs in which every maximal clique is maximum; these are the complements of the well-covered graphs, in which every maximal independent set is maximum.[24] However, other graphs have maximal cliques that are not maximum.

A single maximal clique can be found by a straightforward greedy algorithm. Starting with an arbitrary clique (for instance, any single vertex or even the empty set), grow the current clique one vertex at a time by looping through the graph's remaining vertices. For each vertex v that this loop examines, add v to the clique if it is adjacent to every vertex that is already in the clique, and discard v otherwise. This algorithm runs in linear time.[25] Because of the ease of finding maximal cliques, and their potential small size, more attention has been given to the much harder algorithmic problem of finding a maximum or otherwise large clique. However, some research in parallel algorithms has studied the problem of finding a maximal clique. In particular, the problem of finding the lexicographically first maximal clique (the one found by the algorithm above) has been shown to be complete for the class of polynomial-time functions. This result implies that the problem is unlikely to be solvable within the parallel complexity class NC.[26]

Cliques of fixed size

One can test whether a graph G contains a k-vertex clique, and find any such clique that it contains, using a brute force algorithm. This algorithm examines each subgraph with k vertices and checks to see whether it forms a clique. It takes time $O(n^k k^2)$, as expressed using big O notation. This is because there are $O(n^k)$ subgraphs to check, each of which has $O(k^2)$ edges whose presence in G needs to be checked. Thus, the problem may be solved in polynomial time whenever k is a fixed constant. However, when k does not have a fixed value, but instead may vary as part of the input to the problem, the time is exponential.[27]

The simplest nontrivial case of the clique-finding problem is finding a triangle in a graph, or equivalently determining whether the graph is triangle-free. In a graph G with m edges, there may be at most $\Theta(m^{3/2})$ triangles (using big theta notation to indicate that this bound is tight). The worst case for this formula occurs when G is itself a clique. Therefore, algorithms for listing all triangles must take at least $\Omega(m^{3/2})$ time in the worst case (using big omega notation), and algorithms are known that match this time bound.[28] For instance, Chiba & Nishizeki (1985) describe an algorithm that sorts the vertices in order from highest degree to lowest and then iterates through each vertex v in the sorted list, looking for triangles that include v and do not include any previous vertex in the list. To do so the algorithm marks all neighbors of v, searches through all edges incident to a neighbor of v outputting a triangle for every edge that has two marked endpoints, and then removes the marks and deletes v from the graph. As the authors show, the time for this algorithm is proportional to the arboricity of the graph (denoted $a(G)$) multiplied by the number of edges, which is $O(m\ a(G))$. Since the arboricity is at most $O(m^{1/2})$, this algorithm runs in time $O(m^{3/2})$. More generally, all k-vertex cliques can be listed by a similar algorithm that takes time proportional to the number of edges multiplied by the arboricity to the power $(k-2)$. For graphs of constant arboricity, such as planar graphs (or in general graphs from any non-trivial minor-closed graph family), this algorithm takes $O(m)$ time, which is optimal since it is linear in the size of the input.[18]

If one desires only a single triangle, or an assurance that the graph is triangle-free, faster algorithms are possible. As Itai & Rodeh (1978) observe, the graph contains a triangle if and only if its adjacency matrix and the square of the adjacency matrix contain nonzero entries in the same cell. Therefore, fast matrix multiplication techniques can be applied to find triangles in time $O(n^{2.376})$. Alon, Yuster & Zwick (1994) used fast matrix multiplication to improve the $O(m^{3/2})$ algorithm for finding triangles to $O(m^{1.41})$. These algorithms based on fast matrix multiplication have also been extended to problems of finding k-cliques for larger values of k.[29]

Listing all maximal cliques

By a result of Moon & Moser (1965), every n-vertex graph has at most $3^{n/3}$ maximal cliques. They can be listed by the Bron–Kerbosch algorithm, a recursive backtracking procedure of Bron & Kerbosch (1973). The main recursive subroutine of this procedure has three arguments: a partially constructed (non-maximal) clique, a set of candidate vertices that could be added to the clique, and another set of vertices that should not be added (because doing so would lead to a clique that has already been found). The algorithm tries adding the candidate vertices one by one to the partial clique, making a recursive call for each one. After trying each of these vertices, it moves it to the set of vertices that should not be added again. Variants of this algorithm can be shown to have worst-case running time $O(3^{n/3})$, matching the number of cliques that might need to be listed.[30] Therefore, this provides a worst-case-optimal solution to the problem of listing all maximal cliques. Further, the Bron–Kerbosch algorithm has been widely reported as being faster in practice than its alternatives.[31]

However, when the number of cliques is significantly smaller than its worst case, other algorithms might be preferable. As Tsukiyama et al. (1977) showed, it is also possible to list all maximal cliques in a graph in an amount of time that is polynomial per generated clique. An algorithm such as theirs in which the running time depends on the output size is known as an output-sensitive algorithm. Their algorithm is based on the following two observations, relating the maximal cliques of the given graph G to the maximal cliques of a graph $G \setminus v$ formed by removing an arbitrary vertex v from G:

- For every maximal clique K of $G \setminus v$, either K continues to form a maximal clique in G, or $K \cup \{v\}$ forms a maximal clique in G. Therefore, G has at least as many maximal cliques as $G \setminus v$ does.
- Each maximal clique in G that does not contain v is a maximal clique in $G \setminus v$, and each maximal clique in G that does contain v can be formed from a maximal clique K in $G \setminus v$ by adding v and removing the non-neighbors of v from K.

Using these observations they can generate all maximal cliques in G by a recursive algorithm that chooses a vertex v arbitrarily and then, for each maximal clique K in $G \setminus v$, outputs both K and the clique formed by adding v to K and removing the non-neighbors of v. However, some cliques of G may be generated in this way from more than one parent clique of $G \setminus v$, so they eliminate duplicates by outputting a clique in G only when its parent in $G \setminus v$ is lexicographically maximum among all possible parent cliques. On the basis of this principle, they show that all maximal cliques in G may be generated in time $O(mn)$ per clique, where m is the number of edges in G and n is the number of vertices. Chiba & Nishizeki (1985) improve this to O(ma) per clique, where a is the arboricity of the given graph. Makino & Uno (2004) provide an alternative output-sensitive algorithm based on fast matrix multiplication. Johnson & Yannakakis (1988) show that it is even possible to list all maximal cliques in lexicographic order with polynomial delay per clique. However, the choice of ordering is important for the efficiency of this algorithm: for the reverse of this order, there is no polynomial-delay algorithm unless P = NP.

On the basis of this result, it is possible to list all maximal cliques in polynomial time, for families of graphs in which the number of cliques is polynomially bounded. These families include chordal graphs, complete graphs, triangle-free graphs, interval graphs, graphs of bounded boxicity, and planar graphs.[32] In particular, the planar graphs have $O(n)$ cliques, of at most constant size, that can be listed in linear time. The same is true for any family of graphs that is both sparse (having a number of edges at most a constant times the number of vertices) and closed under the operation of taking subgraphs.[18][33]

Finding maximum cliques in arbitrary graphs

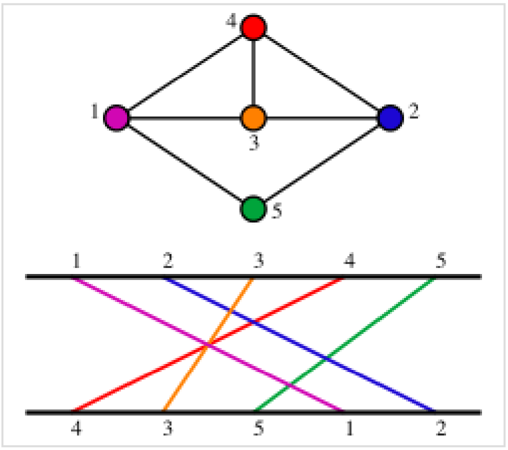
It is possible to find the maximum clique, or the clique number, of an arbitrary n-vertex graph in time $O(3^{n/3}) = O(1.4422^n)$ by using one of the algorithms described above to list all maximal cliques in the graph and returning the largest one. However, for this variant of the clique problem better worst-case time bounds are possible. The algorithm of Tarjan & Trojanowski (1977) solves this problem in time $O(2^{n/3}) = O(1.2599^n)$. It is a recursive backtracking scheme similar to that of the Bron–Kerbosch algorithm, but is able to eliminate some recursive calls when it can be shown that the cliques found within the call will be suboptimal. Jian (1986) improved the time to $O(2^{0.304n}) = O(1.2346^n)$, and Robson (1986) improved it to $O(2^{0.276n}) = O(1.2108^n)$ time, at the expense of greater space usage. Robson's algorithm combines a similar backtracking scheme (with a more complicated case analysis) and a dynamic programming technique in which the optimal solution is precomputed for all small connected subgraphs of the complement graph. These partial solutions are used to shortcut the backtracking recursion. The fastest algorithm known today is a refined version of this method by Robson (2001) which runs in time $O(2^{0.249n}) = O(1.1888^n)$.[34]

There has also been extensive research on heuristic algorithms for solving maximum clique problems without worst-case runtime guarantees, based on methods including branch and bound,[35] local search,[36] greedy algorithms,[37] and constraint programming.[38] Non-standard computing methodologies that have been suggested for finding cliques include DNA computing[39] and adiabatic quantum computation.[40] The maximum clique problem was the subject of an implementation challenge sponsored by DIMACS in 1992–1993,[41] and a collection of graphs used as benchmarks for the challenge, which is publicly available.[42]

Special classes of graphs

Planar graphs, and other families of sparse graphs, have been discussed above: they have linearly many maximal cliques, of bounded size, that can be listed in linear time.[18] In particular, for planar graphs, any clique can have at most four vertices, by Kuratowski's theorem.[21]

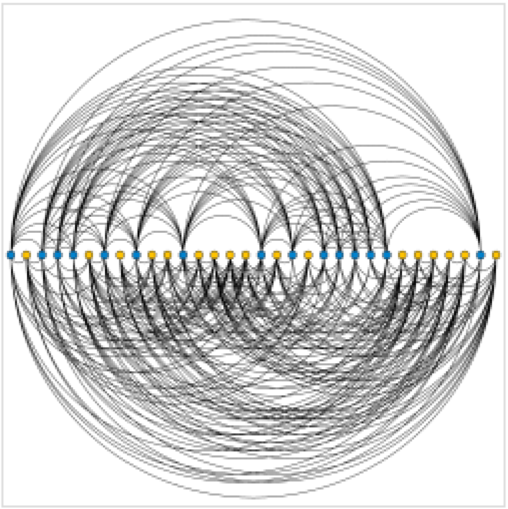
Perfect graphs are defined by the properties that their clique number equals their chromatic number, and that this equality holds also in each of their induced subgraphs. For perfect graphs, it is possible to find a maximum clique in polynomial time, using an algorithm based on semidefinite programming.[43] However, this method is complex and non-combinatorial, and specialized clique-finding algorithms have been developed for many subclasses of perfect graphs.[44] In the complement graphs of bipartite graphs, Kőnig's theorem allows the maximum clique problem to be solved using techniques for matching. In another class of perfect graphs, the permutation graphs, a maximum clique is a longest decreasing subsequence of the permutation defining the graph and can be found using known algorithms for the longest decreasing subsequence problem. Conversely, every instance of the longest decreasing subsequence problem can be described equivalently as a problem of finding a maximum clique in a permutation graph.[45] Even, Pnueli & Lempel (1972) provide an alternative quadratic-time algorithm for maximum cliques in comparability graphs, a broader class of perfect graphs that includes the permutation graphs as a special case.[46] In chordal graphs, the maximal cliques can be found by listing the vertices in an elimination ordering, and checking the clique neighborhoods of each vertex in this ordering.[47]



In this permutation graph, the maximum cliques correspond to the longest decreasing subsequences (4,3,1) and (4,3,2) of the defining permutation.

In some cases, these algorithms can be extended to other, non-perfect, classes of graphs as well. For instance, in a circle graph, the neighborhood of each vertex is a permutation graph, so a maximum clique in a circle graph can be found by applying the permutation graph algorithm to each neighborhood.[48] Similarly, in a unit disk graph (with a known geometric representation), there is a polynomial time algorithm for maximum cliques based on applying the algorithm for complements of bipartite graphs to shared neighborhoods of pairs of vertices.[49]

The algorithmic problem of finding a maximum clique in a random graph drawn from the Erdős–Rényi model (in which each edge appears with probability 1/2, independently from the other edges) was suggested by Karp (1976). Because the maximum clique in a random graph has logarithmic size with high probability, it can be found by a brute force search in expected time $2^{O(\log^2 n)}$. This is a quasi-polynomial time bound.[50] Although the clique number of such graphs is usually very close to $2\log_2 n$, simple greedy algorithms as well as more sophisticated randomized approximation techniques only find cliques with size $\log_2 n$, half as big. The number of maximal cliques in such graphs is with high probability exponential in $\log^2 n$, which prevents methods that list all maximal cliques from running in polynomial time.[51] Because of the difficulty of this problem, several authors have investigated the planted clique problem, the clique problem on random


A random graph with a planted clique

graphs that have been augmented by adding large cliques.[52] While spectral methods[53] and semidefinite programming[54] can detect hidden cliques of size $\Omega(\sqrt{n})$, no polynomial-time algorithms are currently known to detect those of size $o(\sqrt{n})$ (expressed using little-o notation).[55]

Approximation algorithms

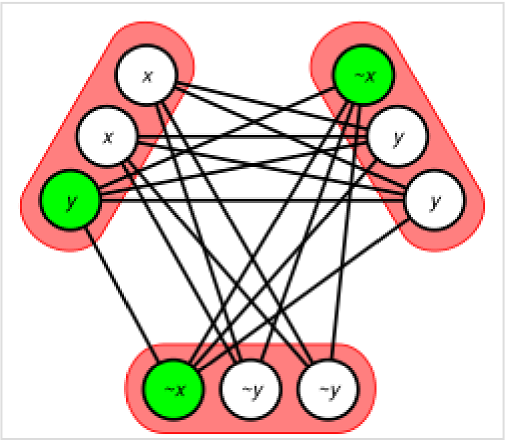
Several authors have considered approximation algorithms that attempt to find a clique or independent set that, although not maximum, has size as close to the maximum as can be found in polynomial time. Although much of this work has focused on independent sets in sparse graphs, a case that does not make sense for the complementary clique problem, there has also been work on approximation algorithms that do not use such sparsity assumptions.[56]

Feige (2004) describes a polynomial time algorithm that finds a clique of size $\Omega((\log n/\log\log n)^2)$ in any graph that has clique number $\Omega(n/\log^k n)$ for any constant k. By using this algorithm when the clique number of a given input graph is between $n/\log n$ and $n/\log^3 n$, switching to a different algorithm of Boppana & Halldórsson (1992) for graphs with higher clique numbers, and choosing a two-vertex clique if both algorithms fail to find anything, Feige provides an approximation algorithm that finds a clique with a number of vertices within a factor of $O(n(\log\log n)^2/\log^3 n)$ of the maximum. Although the approximation ratio of this algorithm is weak, it is the best known to date.[57] The results on hardness of approximation described below suggest that there can be no approximation algorithm with an approximation ratio significantly less than linear.

Lower bounds

NP-completeness

The clique decision problem is NP-complete. It was one of Richard Karp's original 21 problems shown NP-complete in his 1972 paper "Reducibility Among Combinatorial Problems".[59] This problem was also mentioned in Stephen Cook's paper introducing the theory of NP-complete problems.[60] Because of the hardness of the decision problem, the problem of finding a maximum clique is also NP-hard. If one could solve it, one could also solve the decision problem, by comparing the size of the maximum clique to the size parameter given as input in the decision problem.



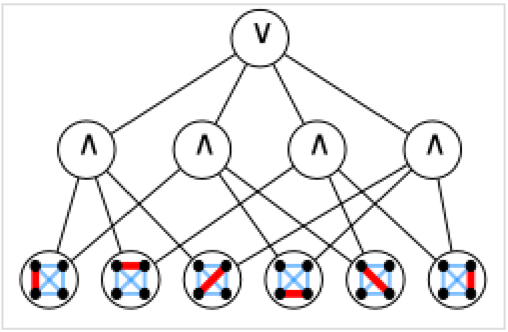
The 3-CNF Satisfiability instance $(x \vee x \vee y) \wedge (\sim x \vee \sim y \vee \sim y) \wedge (\sim x \vee y \vee y)$ reduced to Clique. The green vertices form a 3-clique and correspond to a satisfying assignment.[58]

Karp's NP-completeness proof is a many-one reduction from the Boolean satisfiability problem. It describes how to translate Boolean formulas in conjunctive normal form (CNF) into equivalent instances of the maximum clique problem.[61] Satisfiability, in turn, was proved NP-complete in the Cook–Levin theorem. From a given CNF formula, Karp forms a graph that has a vertex for every pair (v,c), where v is a variable or its negation and c is a clause in the formula that contains v. Two of these vertices are connected by an edge if they represent compatible variable assignments for different clauses. That is, there is an edge from (v,c) to (u,d) whenever $c \neq d$ and u and v are not each other's negations. If k denotes the number of clauses in the CNF formula, then the k-vertex cliques in this graph represent consistent ways of assigning truth values to some of its variables in order to satisfy the formula. Therefore, the formula is satisfiable if and only if a k-vertex clique exists.[59]

Some NP-complete problems (such as the travelling salesman problem in planar graphs) may be solved in time that is exponential in a sublinear function of the input size parameter n, significantly faster than a brute-force search.[62] However, it is unlikely that such a subexponential time bound is possible for the clique problem in arbitrary graphs, as it would imply similarly subexponential bounds for many other standard NP-complete problems.[63]

Circuit complexity

The computational difficulty of the clique problem has led it to be used to prove several lower bounds in circuit complexity. The existence of a clique of a given size is a monotone graph property, meaning that, if a clique exists in a given graph, it will exist in any supergraph. Because this property is monotone, there must exist a monotone circuit, using only and gates and or gates, to solve the clique decision problem for a given fixed clique size. However, the size of these circuits can be proven to be a super-polynomial function of the number of vertices and the clique size, exponential in the cube root of the number of vertices.[64] Even if a small number of NOT gates are allowed, the complexity remains superpolynomial.[65] Additionally, the depth of a monotone circuit for the clique problem using gates of bounded fan-in must be at least a polynomial in the clique size.[66]



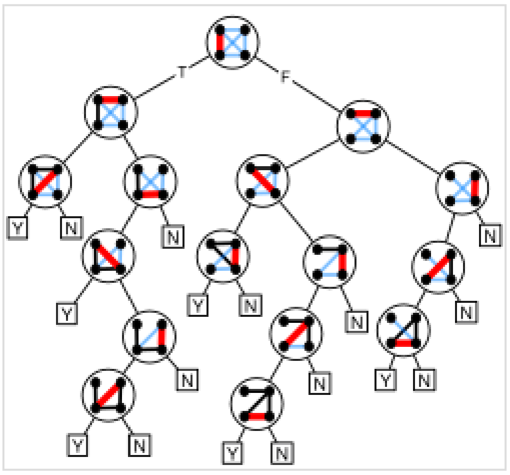
A monotone circuit to detect a k-clique in an n-vertex graph for $k = 3$ and $n = 4$. Each input to the circuit encodes the presence or absence of a particular (red) edge in the graph. The circuit uses one internal and-gate to detect each potential k-clique.

Decision tree complexity

The (deterministic) decision tree complexity of determining a graph property is the number of questions of the form "Is there an edge between vertex u and vertex v?" that have to be answered in the worst case to determine whether a graph has a particular property. That is, it is the minimum height of a Boolean decision tree for the problem. There are $n(n-1)/2$ possible questions to be asked. Therefore, any graph property can be determined with at most $n(n-1)/2$ questions. It is also possible to define random and quantum decision tree complexity of a property, the expected number of questions (for a worst case input) that a randomized or quantum algorithm needs to have answered in order to correctly determine whether the given graph has the property.[67]

Because the property of containing a clique is monotone, it is covered by the Aanderaa–Karp–Rosenberg conjecture, which states that the deterministic decision tree complexity of determining any non-trivial monotone graph property is exactly $n(n-1)/2$. For arbitrary monotone graph properties, this conjecture remains unproven. However, for deterministic decision trees, and for any k in the range $2 \leq k \leq n$, the property of containing a k-clique was shown to have decision tree complexity exactly $n(n-1)/2$ by Bollobás (1976). Deterministic decision trees also require exponential size to detect cliques, or large polynomial size to detect cliques of bounded size.[68]

The Aanderaa–Karp–Rosenberg conjecture also states that the randomized decision tree complexity of non-trivial monotone functions is $\Theta(n^2)$. The conjecture again remains unproven, but has been resolved for the property of containing a k clique for $2 \leq k \leq n$. This property is known to have randomized decision tree complexity $\Theta(n^2)$.[69] For quantum decision trees, the best known lower bound is $\Omega(n)$, but no matching algorithm is known for the case of $k \geq 3$.[70]



A simple decision tree to detect the presence of a 3-clique in a 4-vertex graph. It uses up to 6 questions of the form "Does the red edge exist?", matching the optimal bound $n(n-1)/2$.

Fixed-parameter intractability

Parameterized complexity is the complexity-theoretic study of problems that are naturally equipped with a small integer parameter k and for which the problem becomes more difficult as k increases, such as finding k-cliques in graphs. A problem is said to be fixed-parameter tractable if there is an algorithm for solving it on inputs of size n, and a function f, such that the algorithm runs in time $f(k)\, n^{O(1)}$. That is, it is fixed-parameter tractable if it can be solved in polynomial time for any fixed value of k and moreover if the exponent of the polynomial does not depend on k.[71]

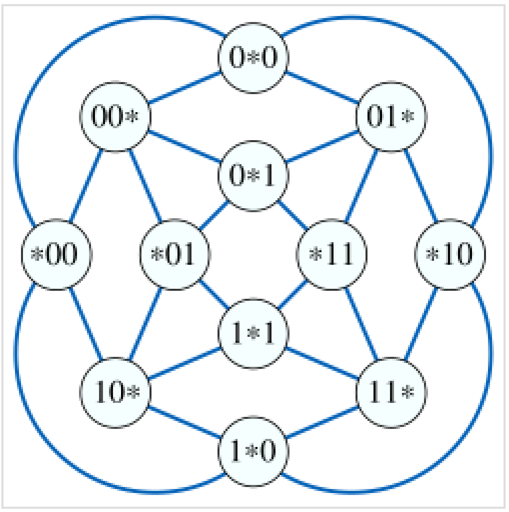
For finding k-vertex cliques, the brute force search algorithm has running time $O(n^k k^2)$. Because the exponent of n depends on k, this algorithm is not fixed-parameter tractable. Although it can be improved by fast matrix multiplication, the running time still has an exponent that is linear in k. Thus, although the running time of known algorithms for the clique problem is polynomial for any fixed k, these algorithms do not suffice for fixed-parameter tractability. Downey & Fellows (1995) defined a hierarchy of parametrized problems, the W hierarchy, that they conjectured did not have fixed-parameter tractable algorithms. They proved that independent set (or, equivalently, clique) is hard for the first level of this hierarchy, W[1]. Thus, according to their conjecture, clique has no fixed-parameter tractable algorithm. Moreover, this result provides the basis for proofs of W[1]-hardness of many other problems, and thus serves as an analogue of the Cook–Levin theorem for parameterized complexity.[72]

Chen et al. (2006) showed that finding k-vertex cliques cannot be done in time $n^{o(k)}$ unless the exponential time hypothesis fails. Again, this provides evidence that no fixed-parameter tractable algorithm is possible.[73]

Although the problems of listing maximal cliques or finding maximum cliques are unlikely to be fixed-parameter tractable with the parameter k, they may be fixed-parameter tractable for other parameters of instance complexity. For instance, both problems are known to be fixed-parameter tractable when parametrized by the degeneracy of the input graph.[33]

Hardness of approximation

Weak results hinting that the clique problem might be hard to approximate have been known for a long time. Garey & Johnson (1978) observed that, because the clique number takes on small integer values and is NP-hard to compute, it cannot have a fully polynomial-time approximation scheme, unless P = NP. If too accurate an approximation were available, rounding its value to an integer would give the exact clique number. However, little more was known until the early 1990s, when several authors began to make connections between the approximation of maximum cliques and probabilistically checkable proofs. They used these connections to prove hardness of approximation results for the maximum clique problem.[74] After many improvements to these results it is now known that, for every real number $\varepsilon > 0$, there can be no polynomial time algorithm that approximates the maximum clique to within a factor better than $O(n^{1-\varepsilon})$, unless P = NP.[75]



A graph of compatibility relations among 2-bit samples of 3-bit proof strings. Each maximal clique (triangle) in this graph represents all ways of sampling a single 3-bit string. The proof of inapproximability of the clique problem involves induced subgraphs of analogously defined graphs for larger numbers of bits.

The rough idea of these inapproximability results is to form a graph that represents a probabilistically checkable proof system for an NP-complete problem such as the Boolean satisfiability problem. In a probabilistically checkable proof system, a proof is represented as a sequence of bits. An instance of the satisfiability problem should have a valid proof if and only if it is satisfiable. The proof is checked by an algorithm that, after a polynomial-time computation on the input to the satisfiability problem, chooses to examine a small number of randomly chosen positions of the proof string. Depending on what values are found at that sample of bits, the checker will either accept or reject the proof, without looking at the rest of the bits. False negatives are not allowed: a valid proof must always be accepted. However, an invalid proof may sometimes mistakenly be accepted. For every invalid proof, the probability that the checker will accept it must be low.[76]

To transform a probabilistically checkable proof system of this type into a clique problem, one forms a graph with a vertex for each possible accepting run of the proof checker. That is, a vertex is defined by one of the possible random choices of sets of positions to examine, and by bit values for those positions that would cause the checker to accept the proof. It can be represented by a partial word with a 0 or 1 at each examined position and a wildcard character at each remaining position. Two vertices are adjacent, in this graph, if the corresponding two accepting runs see the same bit values at every position they both examine. Each (valid or invalid) proof string corresponds to a clique, the set of accepting runs that see that proof string, and all maximal cliques arise in this way. One of these cliques is large if and only if it corresponds to a proof string that many proof checkers accept. If the original satisfiability instance is satisfiable, it will have a valid proof string, one that is accepted by all runs of the checker, and this string will correspond to a large clique in the graph. However, if the original instance is not satisfiable, then all proof strings are invalid, each proof string has only a small number of checker runs that mistakenly accept it, and all cliques are small. Therefore, if one could distinguish in polynomial time between graphs that have large cliques and graphs in which all cliques are small, or if one could accurately approximate the clique problem, then applying this approximation to the graphs generated from satisfiability instances would allow satisfiable instances to be distinguished from unsatisfiable instances. However, this is not possible unless P = NP.[76]

Notes

1. Bomze et al. (1999); Gutin (2004).