



WIKIPEDIA
The Free Encyclopedia

[Main page](#)
[Contents](#)
[Featured content](#)
[Current events](#)
[Random article](#)
[Donate to Wikipedia](#)
[Wikipedia store](#)

Interaction
[Help](#)
[About Wikipedia](#)
[Community portal](#)
[Recent changes](#)
[Contact page](#)

Tools
[What links here](#)
[Related changes](#)
[Upload file](#)
[Special pages](#)
[Permanent link](#)
[Page information](#)
[Wikidata item](#)
[Cite this page](#)

Print/export
[Create a book](#)
[Download as PDF](#)
[Printable version](#)

Languages
[Русский](#)
[Српски / srpski](#)
[ᲛᲠᲣ](#)
[Українська](#)
[Edit links](#)

[Create account](#) [Log in](#)

Article [Talk](#)

[Read](#) [Edit](#) [View history](#)

Bron–Kerbosch algorithm

From Wikipedia, the free encyclopedia

In [computer science](#), the **Bron–Kerbosch algorithm** is an [algorithm](#) for finding maximal [cliques](#) in an undirected [graph](#). That is, it lists all subsets of vertices with the two properties that each pair of vertices in one of the listed subsets is connected by an edge, and no listed subset can have any additional vertices added to it while preserving its [complete connectivity](#). The Bron–Kerbosch algorithm was designed by [Dutch](#) scientists [Joep Kerbosch](#) and [Coenraad Bron](#), who published its description in 1973. Although other algorithms for solving the [clique problem](#) have running times that are, in theory, better on inputs that have few maximal independent sets, the Bron–Kerbosch algorithm and subsequent improvements to it are frequently reported as being more efficient in practice than the alternatives.^[1] It is well-known and widely used in application areas of graph algorithms such as [computational chemistry](#).^[2]

A contemporaneous algorithm of [Akkoyunlu \(1973\)](#), although presented in different terms, can be viewed as being the same as the Bron–Kerbosch algorithm, as it generates the same recursive search tree.^[3]

Contents [hide]

- [1 Without pivoting](#)
- [2 With pivoting](#)
- [3 With vertex ordering](#)
- [4 Example](#)
- [5 Worst-case analysis](#)
- [6 Notes](#)
- [7 References](#)
- [8 External links](#)

Without pivoting [\[edit\]](#)

The basic form of the Bron–Kerbosch algorithm is a [recursive backtracking](#) algorithm that searches for all maximal cliques in a given graph *G*. More generally, given three sets *R*, *P*, and *X*, it finds the maximal cliques that include all of the vertices in *R*, some of the vertices in *P*, and none of the vertices in *X*. In each call to the algorithm, *P* and *X* are disjoint sets whose union consists of those vertices that form cliques when added to *R*. In other words, *P* ∪ *X* is the set of vertices which are joined to every element of *R*. When *P* and *X* are both empty there are no further elements that can be added to *R*, so *R* is a maximal clique and the algorithm outputs *R*.

The recursion is initiated by setting *R* and *X* to be the [empty set](#) and *P* to be the vertex set of the graph. Within each recursive call, the algorithm considers the vertices in *P* in turn; if there are no such vertices, it either reports *R* as a maximal clique (if *X* is empty), or backtracks. For each vertex *v* chosen from *P*, it makes a recursive call in which *v* is added to *R* and in which *P* and *X* are restricted to the neighbor set *N*(*v*) of *v*, which finds and reports all clique extensions of *R* that contain *v*. Then, it moves *v* from *P* to *X* to exclude it from consideration in future cliques and continues with the next vertex in *P*.

That is, in pseudocode, the algorithm performs the following steps:

```
BronKerbosch1(R, P, X) :  
    if P and X are both empty:  
        report R as a maximal clique  
    for each vertex v in P:  
        BronKerbosch1(R ∪ {v}, P ∩ N(v), X ∩ N(v))  
    P := P \ {v}  
    X := X ∪ {v}
```

With pivoting [\[edit\]](#)

The basic form of the algorithm, described above, is inefficient in the case of graphs with many non-maximal cliques: it makes a recursive call for every clique, maximal or not. To save time and allow the algorithm to backtrack more quickly in branches of the search that contain no maximal cliques, Bron and Kerbosch

introduced a variant of the algorithm involving a "pivot vertex" u , chosen from P (or more generally, as later investigators realized,^[4] from $P \cup X$). Any maximal clique must include either u or one of its non-neighbors, for otherwise the clique could be augmented by adding u to it. Therefore, only u and its non-neighbors need to be tested as the choices for the vertex v that is added to R in each recursive call to the algorithm. In pseudocode:

```
BronKerbosch2(R, P, X) :
  if P and X are both empty:
    report R as a maximal clique
  choose a pivot vertex u in P ∪ X
  for each vertex v in P \ N(u) :
    BronKerbosch2(R ∪ {v}, P ∩ N(v), X ∩ N(v))
  P := P \ {v}
  X := X ∪ {v}
```

If the pivot is chosen to minimize the number of recursive calls made by the algorithm, the savings in running time compared to the non-pivoting version of the algorithm can be significant.^[5]

With vertex ordering [\[edit\]](#)

An alternative method for improving the basic form of the Bron–Kerbosch algorithm involves forgoing pivoting at the outermost level of recursion, and instead choosing the ordering of the recursive calls carefully in order to minimize the sizes of the sets P of candidate vertices within each recursive call.

The **degeneracy** of a graph G is the smallest number d such that every **subgraph** of G has a vertex with **degree** d or less. Every graph has a *degeneracy ordering*, an ordering of the vertices such that each vertex has d or fewer **neighbors** that come later in the ordering; a degeneracy ordering may be found in **linear time** by repeatedly selecting the vertex of minimum degree among the remaining vertices. If the order of the vertices v that the Bron–Kerbosch algorithm loops through is a degeneracy ordering, then the set P of candidate vertices in each call (the neighbors of v that are later in the ordering) will be guaranteed to have size at most d . The set X of excluded vertices will consist of all earlier neighbors of v , and may be much larger than d . In recursive calls to the algorithm below the topmost level of the recursion, the pivoting version can still be used.^{[6][7]}

In pseudocode, the algorithm performs the following steps:

```
BronKerbosch3(G) :
  P = V(G)
  R = X = empty
  for each vertex v in a degeneracy ordering of G:
    BronKerbosch2(R ∪ {v}, P ∩ N(v), X ∩ N(v))
  P := P \ {v}
  X := X ∪ {v}
```

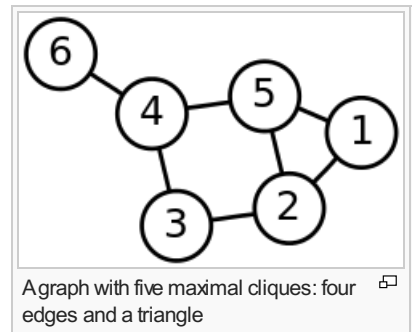
This variant of the algorithm can be proven to be efficient for graphs of small degeneracy,^[6] and experiments show that it also works well in practice for large sparse **social networks** and other real-world graphs.^[7]

Example [\[edit\]](#)

In the example graph shown, the algorithm is initially called with $R = \emptyset$, $P = \{1, 2, 3, 4, 5, 6\}$, and $X = \emptyset$. The pivot u should be chosen as one of the degree-three vertices, to minimize the number of recursive calls; for instance, suppose that u is chosen to be vertex 2. Then there are three remaining vertices in $P \setminus N(u)$: vertices 2, 4, and 6.

The iteration of the inner loop of the algorithm for $v = 2$ makes a recursive call to the algorithm with $R = \{2\}$, $P = \{1, 3, 5\}$, and $X = \emptyset$. Within this recursive call, one of 1 or 5 will be chosen as a pivot, and there will be two second-level recursive calls, one for vertex 3 and the other for whichever vertex was not chosen as pivot. These two calls will eventually report the two cliques $\{1, 2, 5\}$ and $\{2, 3\}$. After returning from these recursive calls, vertex 2 is added to X and removed from P .

The iteration of the inner loop of the algorithm for $v = 4$ makes a recursive call to the algorithm with $R = \{4\}$, $P = \{3, 5, 6\}$, and $X = \emptyset$ (although vertex 2 belongs to the set X in the outer call to the algorithm, it is not a



neighbor of v and is excluded from the subset of X passed to the recursive call). This recursive call will end up making three second-level recursive calls to the algorithm that report the three cliques $\{3,4\}$, $\{4,5\}$, and $\{4,6\}$. Then, vertex 4 is added to X and removed from P .

In the third and final iteration of the inner loop of the algorithm, for $v = 6$, there is a recursive call to the algorithm with $R = \{6\}$, $P = \emptyset$, and $X = \{4\}$. Because this recursive call has P empty and X non-empty, it immediately backtracks without reporting any more cliques, as there can be no maximal clique that includes vertex 6 and excludes vertex 4.

The call tree for the algorithm, therefore, looks like:

```
BronKerbosch2( $\emptyset$ , {1,2,3,4,5,6},  $\emptyset$ )
  BronKerbosch2({2}, {1,3,5},  $\emptyset$ )
    BronKerbosch2({2,3},  $\emptyset$ ,  $\emptyset$ ): output {2, 3}
    BronKerbosch2({2,5}, {1},  $\emptyset$ )
      BronKerbosch2({1,2,5},  $\emptyset$ ,  $\emptyset$ ): output {1,2,5}
  BronKerbosch2({4}, {3,5,6},  $\emptyset$ )
    BronKerbosch2({3,4},  $\emptyset$ ,  $\emptyset$ ): output {3,4}
    BronKerbosch2({4,5},  $\emptyset$ ,  $\emptyset$ ): output {4,5}
    BronKerbosch2({4,6},  $\emptyset$ ,  $\emptyset$ ): output {4,6}
  BronKerbosch2({6},  $\emptyset$ , {4}): no output
```

The graph in the example has degeneracy two; one possible degeneracy ordering is 6,4,3,1,2,5. If the vertex-ordering version of the Bron–Kerbosch algorithm is applied to the vertices, in this order, the call tree looks like

```
BronKerbosch3(G)
  BronKerbosch2({6}, {4},  $\emptyset$ )
    BronKerbosch2({6,4},  $\emptyset$ ,  $\emptyset$ ): output {6,4}
  BronKerbosch2({4}, {3,5}, {6})
    BronKerbosch2({4,3},  $\emptyset$ ,  $\emptyset$ ): output {4,3}
    BronKerbosch2({4,5},  $\emptyset$ ,  $\emptyset$ ): output {4,5}
  BronKerbosch2({3}, {2}, {4})
    BronKerbosch2({3,2},  $\emptyset$ ,  $\emptyset$ ): output {3,2}
  BronKerbosch2({1}, {2,5},  $\emptyset$ )
    BronKerbosch2({1,2}, {5},  $\emptyset$ )
      BronKerbosch2({1,2,5},  $\emptyset$ ,  $\emptyset$ ): output {1,2,5}
  BronKerbosch2({2}, {5}, {1,3}): no output
  BronKerbosch2({5},  $\emptyset$ , {1,2,4}): no output
```

Worst-case analysis [\[edit\]](#)

The Bron–Kerbosch algorithm is not an [output-sensitive algorithm](#): unlike some other algorithms for the clique problem, it does not run in [polynomial time](#) per maximal clique generated. However, it is efficient in a worst-case sense: by a result of [Moon & Moser \(1965\)](#), any n -vertex graph has at most $3^{n/3}$ maximal cliques, and the worst-case running time of the Bron–Kerbosch algorithm (with a pivot strategy that minimizes the number of recursive calls made at each step) is $O(3^{n/3})$, matching this bound.^[8]

For [sparse graphs](#), tighter bounds are possible. In particular the vertex-ordering version of the Bron–Kerbosch algorithm can be made to run in time $O(dn3^{d/3})$, where d is the [degeneracy](#) of the graph, a measure of its sparseness. There exist d -degenerate graphs for which the total number of maximal cliques is $(n - d)3^{d/3}$, so this bound is close to tight.^[6]












Notes [\[edit\]](#)

- ^a [Cazals & Karande \(2008\)](#).
- ^a [Chen \(2004\)](#).
- ^a [Johnston \(1976\)](#).
- ^a [Tomita, Tanaka & Takahashi \(2006\)](#); [Cazals & Karande \(2008\)](#).
- ^a [Johnston \(1976\)](#); [Koch \(2001\)](#); [Cazals & Karande \(2008\)](#).
- ^{a b c} [Eppstein, Löffler & Strash \(2010\)](#).
- ^{a b} [Eppstein & Strash \(2011\)](#).
- ^a [Tomita, Tanaka & Takahashi \(2006\)](#).





References [\[edit\]](#)

- Akkoyunlu, E. A. (1973), "The enumeration of maximal cliques of large graphs", *SIAM Journal on Computing* **2**: 1–6,

[doi:10.1137/0202001](#) .

- Chen, Lingran (2004), "Substructure and maximal common substructure searching", in Bultinck, Patrick, *Computational Medicinal Chemistry for Drug Discovery*, CRC Press, pp. 483–514, [ISBN 978-0-8247-4774-9](#).
- Bron, Coen; Kerbosch, Joep (1973), "Algorithm 457: finding all cliques of an undirected graph", *Commun. ACM* (ACM) **16** (9): 575–577, [doi:10.1145/362342.362367](#) .
- Cazals, F.; Karande, C. (2008), "A note on the problem of reporting maximal cliques"  (PDF), *Theoretical Computer Science* **407** (1): 564–568, [doi:10.1016/j.tcs.2008.05.010](#) .
- Eppstein, David; Löffler, Maarten; Strash, Darren (2010), "Listing all maximal cliques in sparse graphs in near-optimal time", in Cheong, Otfried; Chwa, Kyung-Yong; Park, Kunsoo, *21st International Symposium on Algorithms and Computation (ISAAC 2010), Jeju, Korea*, Lecture Notes in Computer Science **6506**, Springer-Verlag, pp. 403–414, [arXiv:1006.5440](#) , [doi:10.1007/978-3-642-17517-6_36](#) .
- Eppstein, David; Strash, Darren (2011), "Listing all maximal cliques in large sparse real-world graphs", *10th International Symposium on Experimental Algorithms*, [arXiv:1103.0318](#) .
- Johnston, H. C. (1976), "Cliques of a graph—variations on the Bron–Kerbosch algorithm", *International Journal of Parallel Programming* **5** (3): 209–238, [doi:10.1007/BF00991836](#) .
- Koch, Ina (2001), "Enumerating all connected maximal common subgraphs in two graphs", *Theoretical Computer Science* **250** (1–2): 1–30, [doi:10.1016/S0304-3975\(00\)00286-3](#) .
- Moon, J. W.; Moser, L. (1965), "On cliques in graphs", *Israel J. Math.* **3**: 23–28, [doi:10.1007/BF02760024](#) , [MR 0182577](#) .
- Tomita, Etsuji; Tanaka, Akira; Takahashi, Haruhisa (2006), "The worst-case time complexity for generating all maximal cliques and computational experiments", *Theoretical Computer Science* **363** (1): 28–42, [doi:10.1016/j.tcs.2006.06.015](#) .

External links [\[edit\]](#)

- [Review of the Bron-Kerbosch algorithm and variations](#)  by Alessio Conte
- [Bron-Kerbosch algorithm implementation in Python](#) .
- [Bron-Kerbosch algorithm with vertex ordering implementation in Python](#) .
- [Finding all cliques of an undirected graph](#) . Seminar notes by Michaela Regneri, January 11, 2007.

Categories: [Graph algorithms](#)

This page was last modified on 16 April 2015, at 20:46.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

