CS 163 & CS 265: Graph Algorithms

Week 6: Cliques and coloring

Lecture 6b: Cliques and the Bron–Kerbosch algorithm

David Eppstein
University of California, Irvine

Winter Quarter, 2024

Cliques

What is a clique?

This is a normal English word meaning a group of close friends

(in US English, can be pronounced either “click” or “cleek”)

In graph theory, it means a set of vertices that are all adjacent to each other (a complete subgraph)

This word was introduced by Luce and Perry [1949] in the context of social network analysis; Duncan Luce was a sociologist who later became a professor at UC Irvine

Cliques in graphs have been studied (under other names) at least since Erdős and Szekeres [1935], but this is now the standard name

Using cliques to find mutually-compatible elements

Example from a recent YouTube video [Parker 2022]:

- Vertices = five-letter English words without repeated letters
- Edges = two words that do not share any letter
- Clique = multiple words that do not share any letter

Using cliques to find similar substructures

Given two large structures X and Y

- Make a graph whose vertices (x, y) match an an element from X and a possibly-matching element of Y
- Edges = "compatible" matches (x_1, y_1) and (x_2, y_2)
- Clique: system of mutually-compatible matches

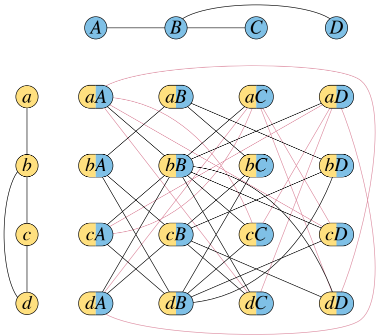
Example:

X and Y are graphs
element = vertex

"Compatible": both adjacent
or both non-adjacent

("modular product of graphs")

Cliques = equivalent subgraphs

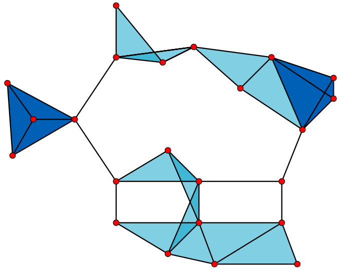


Maximum versus maximal

Maximum clique:

Has the biggest size of all cliques in the graph

(Only the dark blue 4-vertex cliques)



Maximal clique:

Cannot add more vertices to make a larger clique

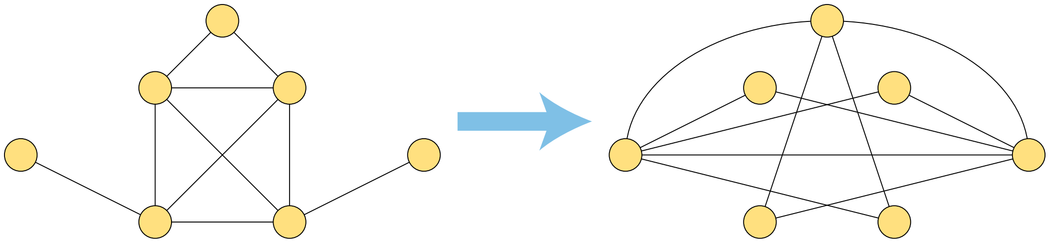
Two 4-vertex cliques, 11 light triangles, six edges

Cliques and independent sets

Clique: Subset of vertices with all pairs forming edges

Independent set: Subset of vertices with no edges

Can convert one problem to the other by complementing:
make new graph that has edges exactly when original graph doesn't



Both are NP-hard and hard to approximate

Finding a single maximal clique is easy

Linear-time greedy algorithm:

K = empty set
for each vertex v:
    if every vertex in K is a neighbor of v:
        add v to K
return K

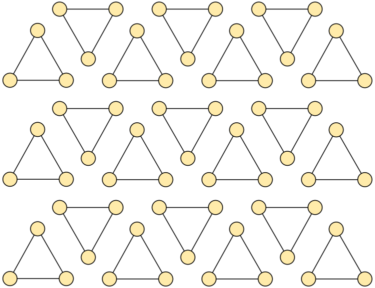
However, the resulting clique could be very small, and finding a maximum clique is NP-hard

We will look at algorithms for finding all maximal cliques, which could take exponential time.

The Moon–Moser theorem

$\#$ maximal cliques $\leq 3^{n/3}$ [Moon and Moser 1965]

Union of $n/3$ triangles has $3^{n/3}$ maximal independent sets



Its complement has $3^{n/3}$ maximal cliques

Main idea of proof that $\# \leq 3^{n/3}$: count max $\#$ independent sets $\mathcal{I}(n)$ by induction on n

Let v be a vertex with minimum possible degree d

Every maximal ind. set uses v or a neighbor, eliminating $\geq d$ other vertices $\Rightarrow$ recurrence $\mathcal{I}(n) \leq (d+1)\mathcal{I}(n-d-1)$

Worst case for recurrence is $d = 2$

The Bron–Kerbosch algorithm

The main idea

Recursive backtracking search with three sets of vertices R, P, X

Each level of recursion adds a vertex to the clique R we're building

P maintains the vertices that could potentially be added later

X maintains a set of vertices whose cliques we have already explored, and we should exclude from being searched again

In both P and X, the only useful vertices are the vertices adjacent to everything in R, so we remove all the others

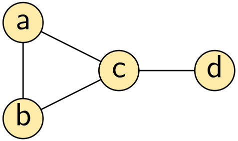
When the recursion runs out of vertices to add from P, it either has a maximal clique (if X is also empty) or a non-maximal clique that can only lead to already-generated cliques (if X is non-empty)

[Akkoyunlu 1973; Bron and Kerbosch 1973]

Basic version of BK algorithm

def $BK(R, P, X)$:
    if P is empty:
        if X is empty:
            output R
    else for v in P:
        $N =$ neighbors of v
        $BK(R + v, P \cap N, X \cap N)$
        move v from P to X

Then call $BK(\text{empty}, V(G), \text{empty})$



$BK(0, abcd, 0)$
  $BK(a, bc, 0)$
    $BK(ab, c, 0)$
      $BK(abc, 0, 0)$
        output abc
    move b to X
    $BK(ac, 0, b)$
  move a to X
  $BK(b, c, a)$
    $BK(bc, 0, a)$
  move b to X
  $BK(c, d, ab)$
    $BK(cd, 0)$
      output cd
  move c to X
  $BK(d, 0, c)$

Optimization 1: Pivoting

At each step of the algorithm, we're building a clique R by adding potential clique vertices from P and excluding vertices from the set X that we have already explored

The basic algorithm loops over all ways to add a vertex from P

Instead, first choose a "pivot" vertex $p \in P \cup X$

All maximal cliques in $R \cup P \cup X$ include a non-neighbor of p (either p, or another vertex that prevents p from being added)

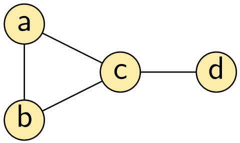
So we only need to loop over non-neighbors $\Rightarrow$ fewer recursive calls

Bron–Kerbosch with pivoting

def $BK(R, P, X)$:
    if P is empty:
        if X is empty:
            output R
    else:
        choose pivot p with
        smallest non-nbrs $\cap P$
        for v in non-nbrs $\cap P$:
            $N =$ neighbors of v
            $BK(R + v, P \cap N, X \cap N)$
            move v from P to X

Then call $BK(\text{empty}, V(G), \text{empty})$

$BK(0, abcd, 0)$
  $p = c$
  $BK(c, abd, 0)$
    $p = a$
    $BK(ac, b, 0)$
      $p = b$
      $BK(abc, 0, 0)$
        output abc
    move a to X
    $BK(cd, 0, 0)$
      output cd



Analysis of pivoting

We choose the pivot p to minimize the set of non-neighbors of p in P (the vertices that we add in recursive calls)

Let k denote the size of this set ($= \#$ recursive calls)

When a recursive call adds v, we remove from P the non-neighbors of v, another set of at least k vertices

$\#$ bottom-level recursive calls from a subproblem where $|P| = n$ can be analyzed using the recurrence

$$T(n) \leq \max_k k\, T(n-k)$$

Worst case $k = 3$ gives $L(n) \leq 3^{n/3}$, same as Moon–Moser bound
With some care we get time $O(3^{n/3})$ [Tomita et al. 2006]

Optimization 2: Degeneracy

At top level of the recursion, we are going to loop over all the vertices (or most of them, unless we can find a pivot that is adjacent to almost all other vertices)

We still have a lot of freedom to choose what order to do this loop in, so let's use the reverse of a degeneracy ordering: if graph has degeneracy d, every vertex has $\leq d$ later neighbors

Switch to pivoting at lower levels of recursion

Each second-level call has $P \subset$ later neighbors of added vertex, of size at most $d \Rightarrow \#$ recursive calls is $\leq n3^{d/3}$

We don't get $O(1)$ average time per call (like first optimization) because X can still be large, but each vertex v contributes to X for $\leq d$ later subproblems leading to total time $O(dn\,3^{d/3})$

[Eppstein et al. 2013]

Morals of the story

Definition of cliques

The difference between maximum and maximal cliques

There can be exponentially many but the Bron–Kerbosch algorithm can find them in time matching the worst-case Moon–Moser bound

Other variations can find them in polynomial time per output clique

We can take advantage of the structure of real-world networks (their degeneracy) to find maximal cliques much more quickly than in worst-case graphs

References I

E. A. Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM Journal on Computing*, 2:1–6, 1973. doi:10.1137/0202001.

Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973. doi:10.1145/362342.362367.

David Eppstein, Darren Strash, and Maarten Löffler. Listing all maximal cliques in large sparse real-world graphs in near-optimal time. *J. Experimental Algorithmics*, 18 (3):3.1, 2013. doi:10.1145/2543629.

Paul Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935. URL https://www.renyi.hu/~p_erdos/1935-01.pdf.

R. Duncan Luce and Albert D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949. doi:10.1007/BF02289146.

References II

J. W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3: 23–28, 1965. doi:10.1007/BF02760024.

Matt Parker. Can you find: five five-letter words with twenty-five unique letters? YouTube video, August 2022. URL https://www.youtube.com/watch?v=_-AfhLQfb6w.

Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006. doi:10.1016/j.tcs.2006.06.015.