Discovering Locally Dense Groups

What is Groups, What is Locally Dense Groups?

Finding groups in a network is of interest, and we need to know what characteristics do the groups we intent to find have. The most common and necessary property a group should have is cohesiveness and here the group we talk about is cohesive group. There is four characteristics of cohesiveness:

1. Mutuality.
2. Density.
3. Compactness.
4. Separation.

They are all representing the cohesiveness of a network with different emphasis, and here we will focus on the density, with is the fundamental and important. First, it is observed that the assortative mixing is a common property in a social network group, which means that the members intend to have the properties that their neighbors have. When the property is the degree, then, the density matters (we use degree to compute the density, so as we can use density to imply the degree). Second, it is mathematically proved that high density implies other cohesiveness characteristics.

And the term "locally" means the invariant under network changes outside the group. A local property is definable over all the vertexes inside the group.

And we usually have a maximality condition where we want to discover some groups. That is to find the maximum group that satisfied the query condition.

Without specially specification, we only consider the undirected unweighted graph here.

The Prototype of other kind of Group : Clique

Definitions :

Clique : a complete subgraph of a graph G.

Maximum Clique : a clique with maximum cardinality.

Maximal Clique : a clique with maximal vertexes.

N-Clique : a subgraph whose vertexes' distances are smaller than N.

N-Club : a subgraph with a diameter smaller than N.

N-Clan : a subgraph is both a N-Clique and N-Club.

The last three definition are not important.

Comparing maximum clique and maximal clique:

maximal_vs_maximum_clique

Properties of Clique

1. Perfectly dense.
2. Perfectly compact.
3. Perfectly connected.
4. Closed under exclusion.
5. Nested.

Theorems

The existence of cliques of size k :

turan

The unbounded number of maximal cliques :

moom_and_moser

Problems

1. We can know the existence of a cliques of size k, but we have difficulty to locate them.
2. We have enormous maximal cliques and they may overlap each others, and we have troubles to count them and rank them.
3. Brute-force algorithm is computational unacceptable

Finding the Size of Maximum Clique is an NP-problem

We translate the problem of computing the size of maximum clique into determining whether a conjunctive normal form H is satisfiable by constructing a K-partite graph such that:

$$V_H = \{(L,i) | L \; is \; a \; literal \; of \; clause \; i \; in \; H\}$$

$$E_H = \{\{(L,i),(L',j)\} | L \neq \neg L' \; and \; i \neq j\}$$

So, an edge exist if and only if two conjunctive clauses that are composed of disjunctive literals are satisfiable, then an an clique of size k exist if and only if H of k clauses is satisfiable.

Then, we gonna give an upper bound of this question/decision.

If the minimum degree of a graph is greater than n-4, than we can compute the maximum clique easily which is a bottom condition. Let's assume the minimum degree is not greater than n-4 and the vertex with the min degree is noted *v*, then the maximum clique either contain *v* or not contain *v*, and since clique is closed under exclusion then we get a recursive inequality that the running time is T(n) :

$$T(n) \leq T(n-4) + T(n-1) + c \cdot (n+m)$$

Solve it, then we get T(n) = O($n^2 1.3803^n$) . But the minimum upper bound remind unknown and the known fast and"space-friendly" algorithm has a running time T(n) = O($n^2 1.2227^n$) .

Then a natural idea comes out : can we get an approximate result with justified running time?

How far can we go using approximation technique or heuristic technique? Well, there is a theorem, delightful and exhausted.

t060107

Exhaustive Searching to Find the Maximum Clique

To determent whether a give set of vertexes is a clique, the simplest way is to examine whether there is an edge between each pair. Clearly, the worst case is O(n^2).

And to exhaustively search, we need to examine every possible pair, that is to say, the total number of vertexes subsets we need to examine is the sum over $\binom{k}{n}$, which is exactly $(1+1)^n$.

Thus the worst running time is O($n^2 2^n$), which in shorthand is $O*(2^n)$.

Finding Fixed Size Clique

It should be better than finding all cliques exhaustively and then pick up the clique of certain size. And we get started from the basic problem: finding all triangles in a graph, since finding the clique of size 2 is trivial.

even the exhaustive searching can be improve when the question comes down. We just examine every subset of size k. Which takes O($k^2 n^3$).

Remark the multiplication of adjacency matrix of a graph, the result is the counting of the routes of 2 distances between each pair. Remark again that the graph is simple, undirected and unweighted and loop free.

Then, we quickly compare the result of multiplication and the original matrix, then if an index (i,j) has positive integer in both matrix, then there is a triangle. And it is faster than the exhaustive approach for the reason that the multiplication is smaller than O(n^3)

The same idea can be extended further with divide-and-conquer method and it cannot simply extended due the some facts such and the graph would not be a "loop-free" graph if a triangle exist (the same positive integer on the same position).

The algorithm to find the k-size cliques is simply described as follow:

1. Find the cliques of size $\frac{k}{3}-1, \frac{k}{3}, \frac{k}{3}+1$.
2. Divided the graph into a tripartite, with three set of vertex that have all the cliques of $\frac{k}{3}-1, \frac{k}{3}, \frac{k}{3}+1$ size.
3. Create a tripartite auxiliary graph.
4. Determine whether two cliques of different partites compose a bigger clique. If so, create a two vertexes linked by a edge, and put the vertex into it corresponding partite in the auxiliary graph. All work can be done in time smaller than O($n^{\frac{2k}{3}}$)).
5. Find the triangles in the auxiliary graph, it a triangle exist, then there is a clique with size k. Finding the triangle takes three matrix multiplications, and the multiplication can be fasten to O($n^{1+\frac{2k}{3}}$} by the fast rectangular matrix multiplication. (Is it the running time is only dependent on the size of the matrix? The balanced rectangle size, we get n/(2k/3), so the running time is simple (3n/2k)^3, how can it be $O(n^{max\{k_1+k2, k_1+k_3, k_2+k_3\}})$ as the textbook described? I don't known the fast rectangular matrix multiplication, but I don't think there is a relationship between k and the multiplication running time).

Then recursive expression roughly:

T(n,k) = 3T(n ,k/3) + O($n^{\frac{2k}{3}+1}$).

So it, it is O($n^{\frac{2k}{3}+1}$). More precisely, it is $O(n^{\alpha})$, where α is the upper floor of (2k/3).

Let's compare:

find_fixed_size_clique_comple

Enumerating Maximal Cliques

Let's introduce the algorithm given by the textbook, which is easier to understand and a better instruction to human, comparing to the algorithm given in Wikipedia:

> 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* \ *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* \ *v*, so they eliminate duplicates by outputting a clique in G only when its parent in *G* \ *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.

The textbook gives a iterative adaption of the algorithm above, it enumerates the maximal cliques by construction, ,more precisely, the construction of a binary tree. The key feature of the tree is that on every level of the tree, the node of the tree is the maximal clique at that time, at that level, at the set of the chosen vertexes from the graph.

1. Select an arbitrary vertex as its root. And this vertex form a vertex set U. the current vertex is the root.
2. Select another vertex v and then test that if the neighbors of v contains U.
    1. If so, add v to U, and make a left child on the current node.
    2. If not, there is two choices: a) move downward with v without the some vertexes in U that are not willing to go downward with v being a clique together; b) move downward with U without v, friendship left forever. The key idea is to form a clique, and to form a maximal clique. If a) works, the we make a right child on the current node, we have a new maximal clique to move forward. And we will also consider b), if b) works, old friendship goes on. Remark that, the children are the maximal cliques at the next level, only considering the set of selected vertexes.
3. repeat 2, until no vertex left.

One more thing that may be improved in the textbook is the delay between output. The book say the delay between output is $O(n^3)$, but I think it is $O((2n-1)(n+m))$ that is $O(n^2+nm)$, but the Wikipedia tells it should be $O(mn)$:

> 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.

There exists an algorithm to exploit the property that we can imply the second lexicographically smartest clique from the first lexicographically smartest clique, as long as we know the lexicographically smartest clique of the graph.

Here I briefly mention the idea of the exploitation and the proof. We extract the lexicographically first clique U_0 from a minimum priority queue Q. Then find out a vertex v_j which are not adjacent to some vertex v_i which is the lexicographically preceding vertexes. Then we take the vertex v_i to form a clique T and insert T into Q. After finding out all the v_j based on U_0, then we extract a clique from Q and replace U with it and loop again.

The proof is inductive, and it means to proof that the lexicographically next graph must be inserted into Q. The idea is in the reversed order of the algorithm. First, we have a clique U that is greater than U_0. Remark that the U is can be any clique that satisfied this condition. Then we can see that U will finally bring up a clique T that is smaller than U and T will be inserted into Q, which ensures the lexicographically first clique must be in Q. Second, we find out the largest index j, that makes $U_j = \{v_1, ..., v_j\} \cup U$ is not a maximal clique in $\{v_1, ..., v_j\}$. Then, based on the construction (the maximality of j and the maximality of maximal clique), we have $v_{j+1} \in U$, and a non-empty vertex set S, such that $S \cup U_j$ is a maximal clique in $\{v_1, ..., v_j\}$, and $N(v_{j+1}) \cap S \neq S$. Let's note that $S \cup U_j$ is contained in a clique on V and is denoted as U_0, then U_0 is smaller than U. Then, reverse the process, let us have known U_0 first, then we get the algorithm mentioned above that produce a larger clique T that contains $U \cap \{v_1, ..., v_{j+1}\}$, and which is smaller than U, otherwise we would have a U_k with $j + 1 < k$. Because all the generated T are agree with U_0 on $T \cap \{v_1, ..., vj\}$. If all v_j are considered, then the lexicographically next clique T must be in Q.