Maximal Clique

Contents

1 Generating all cliques (Kreher and Stinson section 4.3) 1
1.1 Graphs and Cliques 1
1.2 Application 1
1.3 Finding all maximal cliques 1

2 Average case analysis 4

1 Generating all cliques (Kreher and Stinson section 4.3)

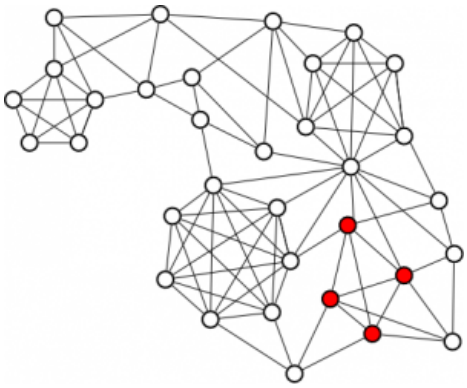
1.1 Graphs and Cliques

A graph of order n is an ordered pair $G = (V, E)$, where V is a set of n vertices (also called nodes and E is a set of unordered pairs of vertices called the edges of G. Two vertices x, y are adjacent in G if $\{x, y\}$ is an edge of G. A complete graph of order m is a graph denoted by K_m with m vertices and each of the $\binom{m}{2}$ vertex pairs is an edge. A clique in G is a subset $C \subseteq V$ such that every pair $\{x, y\}$ with $x, y \in C$ is an edge of G. In other words, C is a clique of order m in G if and only if C induces a complete subgraph isomorphic K_m in G. A clique C in G is maximal if C is not a subset of a larger clique in G, and C is maximum if no clique in G has order greater than $|C|$. Every maximum clique in G is a maximal clique, but not vice versa.

To understand the choice of terminology (clique), imagine that V is a set of people, and that each edge indicates a friendship between two people in V. A clique is a set C of people which are all mutual friends.

Example. We can represent a graph with a little circle for each vertex and an arc for each edge. The red vertices in the graph below form a maximal clique, since no vertex of G is adjacent to all four red vertices. The red vertices do not form a the *maximum* clique because G contains a larger clique (indeed G contains a clique of order 7).

If you know come Complexity Theory, you should take note that the problem of finding the *maximum clique* for a given graph is NP-hard (Karp, 1972).



1.2 Application

Though stated abstractly, there are a vast collection of applications for an algorithm that generates all maximal cliques. Figure 1 gives an example.

BIOINFORMATICS
Vol. 23 ECCB 2006, pages e184–e190
doi:10.1093/bioinformatics/btl308

Similarities and differences of gene expression in yeast stress conditions

Oleg Rokhlenko[1,*], Ydo Wexler[1] and Zohar Yakhini[1,2]
[1]Technion-Israel Institute of Technology, Department of Computer Science, Haifa 32000, Israel and
[2]Agilent Laboratories, Palo Alto, CA, USA

ABSTRACT

Motivation and Methods: All living organisms and the survival of all cells critically depend on their ability to sense and quickly adapt to changes in the environment and to other stress conditions. We study stress response mechanisms in *Saccharomyces cerevisiae* by identifying genes that, according to very stringent criteria, have persistent co-expression under a variety of stress conditions. This is enabled through a fast clique search method applied to the intersection of several co-expression graphs calculated over the data of Gasch *et al.* This method exploits the topological characteristics of these graphs.

Results: We observe cliques in the intersection graphs that are much larger than expected under a null model of changing gene identities for different stress conditions but maintaining the co-expression topology within each one. Persistent cliques are analyzed to identify enriched function as well as enriched regulation by a small number of TFs. These TFs, therefore, characterize a universal and persistent reaction to stress response. We further demonstrate that the vertices (genes) of many cliques in the intersection graphs are co-localized in the yeast genome, to a degree far beyond the random expectation. Co-localization can hypothetically contribute to a quick co-ordinated response. We propose the use of persistent cliques in further study of properties of co-regulation.

Supplementary information: http://www.cs.technion.ac.il/~olegro/stress.html

Contact: olegro@cs.technion.ac.il

three mechanisms of stress response in *Saccharomyces cerevisiae*—the positive transcriptional control activated by heat shock elements, stress response elements and AP-1 responsive elements. They identify yeast genes with a universal stress response as well as genes with a more specific reaction profile. In a breakthrough application of a high-throughput approach, Gasch *et al.* (2000) use expression profiling with microarrays to measure the changes, as a function of time, of almost all yeast genes, as a result of the exposure to a variety of stress conditions. They observe that a large set of genes (~900) show drastic response to most of the studied conditions. They also study the correlation between the response patterns of genes in single stress conditions by using clustering techniques. In this article we study the sets of genes that seem to be persistently and strongly co-ordinated as part of the stress response mechanism, not restricted to a single specific condition.

For every stress condition we define the co-expression graph to be an undirected graph whose vertices correspond to genes, and the vertices of two genes are connected by an edge if their expression profiles are sufficiently correlated. Namely, the p-value of the Pearson correlation between the expression patterns of the two genes is statistically significant (p-value <0.01). Two genes are said to be co-co-expressed in stress conditions A and B if their expression patterns in both time-courses correlate; alternatively—if they have an edge connecting them in both co-expression graphs. The k-stress persistence graphs (k-pers) are the intersection graphs of sets of k co-expression graphs. By studying cliques in k-pers

Figure 1: An example of an application of maximal clique to generating and confirming biological hyptotheses.

1.3 Finding all maximal cliques

Done naively, one could simply test all 2^n subsets, but this is a very poor solution. Instead we will describe a pruning algorithm that is quasi-linear in the size of the graph.

In order to proceed by backtracking, we need to define what is a feasible partial configuration and describe the choice sets. Recall, in the backpack example we pruned any children that went beyond the maximum allowable weight. Here we will prune any children whose addition does not result in a clique.

Let us label the vertices of the input graph with $V = \{1, \ldots, n\}$. Each node at level m of the search tree is an m-subset of vertices recorded as a list $X = [v_1, v_2, \ldots, v_m]$ where $v_1 < v_2 < \cdots < v_m$. For each procedure call, X will be a clique. The list $[v_1, v_2, \ldots, v_m, v]$ is a child of node X, if and only if $v_m < v \leq n$, and this child is a clique if and only if v is a neighbour of v_i for $i = 1, 2, \ldots, m$. We prune every child of X that is not a clique by defining its choice set to be

$$C_m = \{v \in V : v > v_m \text{ and } \{v_i, v\} \in E \text{ for } i = 1, 2, \ldots, m\}.$$

We can derive C_m efficiently by using the choice set C_{m-1} of its parent node $[v_1, v_2, \ldots, v_{m-1}]$, by using the two vertex sets $nbr(v_m) := \{x \in V : \{v_m, x\} \in E\}$ and $big(v_m) := \{x \in V : x > v_m\}$.

$$C_0 = V \qquad \text{and} \qquad C_m = C_{m-1} \cap nbr(x_m) \cap big(x_m),\ m = 1, 2, \ldots, n.$$

We can compute and store the sets $nbr[v]$ and $big[v]$ for every $v \in V$ before the algorithm starts.

How can we tell whether a node $X = [v_1, v_2, \ldots, v_m]$ is a maximal clique? Certainly if X is a maximal clique, then it is a leaf of the pruned search tree (so C_m is empty). But not every leaf of the pruned search tree is a maximal clique. We must also keep track of the set N_m of *all* vertices v which do not the belong to X (including those with $v < v_m$) and which are neighbours of every vertex in X. A clique X at level m is a maximal clique if and only if $N_m = \emptyset$. We can update N_m efficiently with

$$N_0 = V \qquad \text{and} \qquad N_m = N_{m-1} \cap nbr(x_m),\ m = 1, 2, \ldots, n.$$

The following recursive backtrack algorithm, when called with `AllCliques(0)`, finds all cliques and reports which ones are maximal.

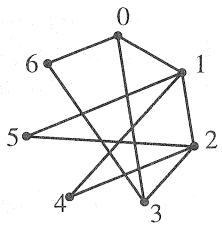
All Cliques

```
AllCliques := proc(m)
arguments m              // Level of recursion
global nbr           // nbr[v] is the set of neighbours of vertex v (precomputed)
       big           // big[v] = { v+1, v+2, ..., n }
       x[1],...,x[n] // The node X = [x[1],x[2],...,x[m]] is a clique in G
       C[0],...,C[n] // C[k] is the choice set for clique [x1,...,xk], k=0,1, ...,m
       N[0],...,N[n] // N[k] lists all vertices adjacent to x[1],...,x[k], k=0,...,m

output [x[1],...,x[m]] // output the clique X (including the empty clique, if m=0)

if m=0
    N[m] := V
    C[m] := V
else
    N[m] := intersect( N[m-1], nbr[x[m]] )
    C[m] := intersect( C[m-1], nbr[x[m]], big[x[m]] )

if N[m] is empty
    output "X is a maximal clique"

for x[m+1] in C[m] do
    AllCliques(m+1)
```

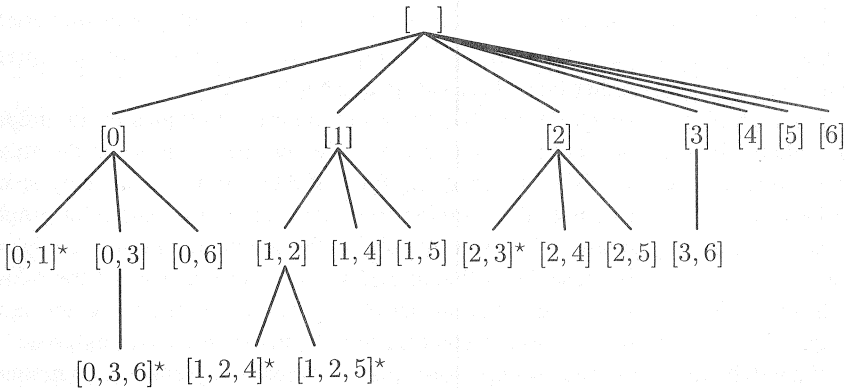
With an easy modification we can find and output a maximum clique of G by using a global variable `optX` recording the largest clique found so far, similarly to `oknapsac` in Lecture 18.

Example. Kreher and Stinson (Example 4.1) work the following example. They use A for nbr and B for big.

Example 4.1 *Finding all the cliques in a graph*



v	A_v	B_v
0	1, 3, 6	1, 2, 3, 4, 5, 6
1	0, 2, 4, 5	2, 3, 4, 5, 6
2	1, 3, 4, 5	3, 4, 5, 6
3	0, 2, 6	4, 5, 6
4	1, 2	5, 6
5	1, 2	6
6	0, 3	



Maximal cliques are indicated with a $\star$. □

The nodes of the state tree are all of the cliques, each presented once. The maximal cliques arise in the leaves, although not every leaf is a maximal clique.

2 Average case analysis

Each clique appears exactly once in the state tree. And each node takes $O(n)$ time to pricess. Thus, the run-time of the algorithm on G is proportional to $n\,c(G)$, where $c(G)$ is the number of cliques in the graph.

It is possible to compute the average runtime over the set $\mathcal{G}_n$ of labelled graphs with vertex set $V = \{1, 2, \ldots, n\}$. The number of graphs in $\mathcal{G}_n$ is $2^{\binom{n}{2}}$, so the average run-time for graphs with n vertices is $n\,\overline{c}(n)$ where

$$\overline{c}(n) := \frac{1}{2^{\binom{n}{2}}} \sum_{G \in \mathcal{G}_n} c(G).$$

is the average number of cliques of a graph in $\mathcal{G}_n$. We can find this sum, and hence the average number of cliques, with a sneaky computation that involves counting something in two different ways.

For any graph $G \in \mathcal{G}_n$ and any subset of vertices $X \subseteq V$, let

$$\chi(G, X) = \begin{cases} 1 & \text{if } X \text{ is a clique in } G \\ 0 & \text{otherwise} \end{cases}$$

Then $c(G) = \sum_{X \subseteq V} \chi(G, X)$. We can derive the sum we seek by changing the order of summation as follows.

$$\sum_{G \in \mathcal{G}_n} c(G) = \sum_{G \in \mathcal{G}_n} \sum_{X \subseteq V} \chi(G, X) = \sum_{X \subseteq V} \sum_{G \in \mathcal{G}_n} \chi(G, X)$$

For any $X \subseteq V$, the inner sum $\sum_{G \in \mathcal{G}_n} \chi(G, X)$ equals the number graphs with vertex set V in which X is a clique. This equals the number of possible choices for the edges G that are *not* joining two vertices in X. There are $\binom{n}{2}$ possible edges for G, and $\binom{|S|}{2}$ have both of its ends in X. Hence there are $\binom{n}{2} - \binom{|X|}{2}$ places to decide whether to include an edge in the graph. Thus the number of graphs in which X is a clique is $2^{\binom{n}{2} - \binom{|X|}{2}}$. This depends only on the size of X hence

$$\begin{aligned} \overline{c}(n) &= \frac{1}{2^{\binom{n}{2}}} \sum_{G \in G(n)} c(G) && (1) \\ &= \frac{1}{2^{\binom{n}{2}}} \sum_{X \subseteq V} 2^{\binom{n}{2} - \binom{|X|}{2}} && (2) \\ &= \sum_{X \subseteq V} 2^{-\binom{|X|}{2}} && (3) \\ &= \sum_{k=0}^{n} \binom{n}{k} 2^{-\binom{k}{2}}. && (4) \end{aligned}$$

Some estimations (not shown here) show that $\overline{c}(n) = O(n^{\log_2(n)+1})$ Thus the average run time of `AllCliques` is $n\,\overline{c}(n) = O(n^{\log_2(n)+2})$, which is considered quasipolynomial time.