Maximal Clique

Contents

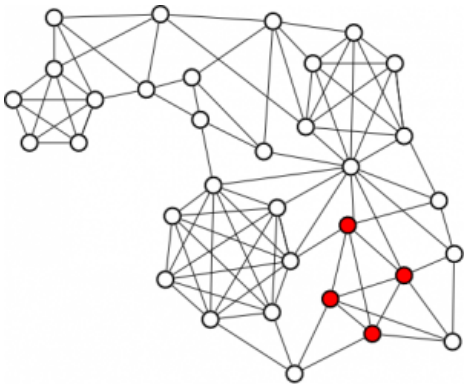
1 Generating all cliques (Kreher and Stinson section 4.3)

1.1 What is a clique?

Given a graph $G = (V, E)$ recall that a clique is subset C of the vertex set V such that every edge $\{x, y\}$ such that $x, y \in C$ is contained in E. Otherwise said, it is a subgraph which is also a complete graph. A maximal clique of a graph is a clique which cannot be extended.

To understand the choice of terminology (clique), imagine that each edge denotes friendship amongst a group. A clique is a group which are all mutually friends, much like the cliques of highschool....

Example. Maximal clique. The red vertices in the graph below are a maximal clique. There is no vertex in the rest of the graph which is adjacent to all four elements of the graph, hence this clique is *maximal*. We can clearly see a K_5 else where in the graph, so it is not the *maximum* clique. In general, 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 cliques. Figure 1 gives an example.

1.3 Finding 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 configuration, and what is a feasible partial solution. Then, we give a method to describe the *choice sets*. Recall, in the backpack example if were processing a partial configuration and determined that we were already at the maximum allowable weight we did not consider configurations which added additional objects to the backpack.

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.

Here let us label the vertices $1 \dots n$, and consider a sequence of vertices $[v_1, v_2, \dots, v_m]$ to be a partial solution if and only if it forms a clique. Given this partial solution, we consider the set of all vertices $C_m = \{v : v > v_m$ and $\{v_i, v\} \in E, i = 1 \dots m\}$.

Let us be efficient in the computation of C_k. We will keep track of the neighbour of each vertex $neighbour[v] := \{x : \{v, x\} \in E\}$ and $big[v] := \{x \in V : x > v\}$, the set of vertices that are 'larger' than v in the vertex ordering. These can be computed before the algorithm runs. Thus,

$$C_m = neighbour(x_m) \cap bigger(x_m) \cap C_{m-1}.$$

Essentially, we know all of the neighbours of the vertices of x_i $i = 1..m-1$! So we use this. We compute which neighbours of x_m are larger valued, and are also adjacent to x_i for $i = 1..m-1$.

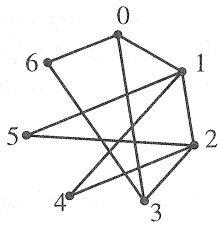
Finally, we keep track of $N[m] := N_{[m-1]} \cap neighbour[x_{m-1}]$. We set $N[1] = V$. This is the set of vertices that still need to be considered. Once we have $N[m] = \emptyset$, we are done.

All Cliques

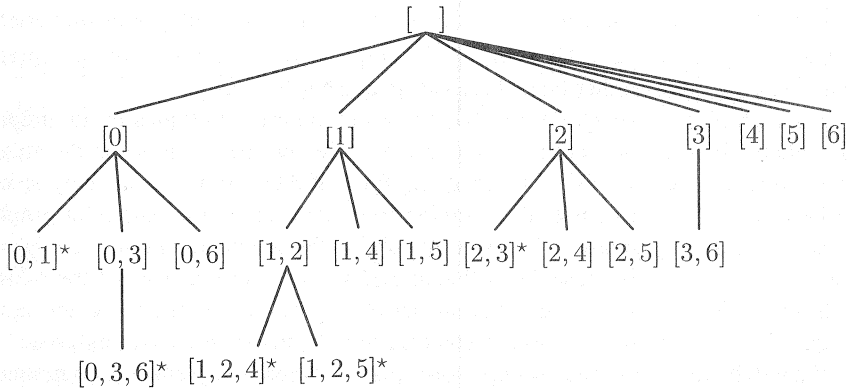
```
AllCliques:= proc(m) // m=1,2, ..., n
global X   // current feasible solution built up one vertex at a time.
                // It is alway a clique (but not necc. maximal)
          C // Choice set
local N // the set of vertices still to consider
         neighbour
         bigger

if m=1 then []
else output [x1,..., xm] // output one clique, and continue on
fi

if m=1 then N[m] := V // we are just getting started!
else              N[m]: = neighbour[xm] intersect N[m-1]
                      // all the neighbours will be added to the
                      // choice set, and hence be recursively
                      // considered. N[m] is what is leftover.
fi

if N[m] is empty, then X is a maximal clique fi

if m=0, then C[m]:=V
else C[m]:= intersect( neighbour[xm], bigger[xm], C[m-1])
fi

for each v in C[m] do
    xm:= v
    AllCliques(m+1)
od;

end proc;
```

Example. Kreher and Stinson work through the following Example 4.1. They name neighbour "A" and big "B".

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 because of pruning.

Exercise. Explain how to modify the algorithm to solve the problem of maximal independant set of a graph.

2 Average case analysis

The state tree tells you the run-time of the algorithm. Each clique appears exacty once. Thus, the run-time of the algorithm on G is proportional to $c(G) * n$, the number of cliques in the graph. It is easy to compute the average value of this by considering *all* labelled graphs.

The total number of labelled graphs on n vertices is $2^{\binom{n}{2}}$.

Define

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

We can do a sneaky computation to compute this sum, and hence the average number of cliques.

We break this apart, and it is the sum over every subset $W \subseteq V$ of the probability that W is a clique. This is straightforward to compute. The number of graphs where W is a clique are computed

by considering all possible choices for the edges that are not within W. There are $\binom{n}{2}$ total edges and we have already accounted for $\binom{|W|}{2}$ of them, hence the number of edges not in W is $\binom{n}{2} - \binom{|W|}{2}$. By alternating all possibilities of these edges to be in the graph or not we compute that the number of graphs in which W is a clique is $2^{\binom{n}{2}-\binom{|W|}{2}}$. This depends only on the size of W hence

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

Some estimations show that $\overline{c}(n) = O(n^{\log_2(n)+1})$ which is considered quasipolynomial time. Thus the average run time is $O(n^{\log_2(n)+2})$.