DINING PHILOSOPHERS AND GRAPH COVERING PROBLEMS

Edward T. Ordman
Department of Mathematical Sciences
Memphis State University
Memphis, TN 38152

ABSTRACT. Let the vertices of a graph denote processes in a distributed or time-shared computer system; let two vertices be connected by an edge if the two processes cannot proceed at the same time (they mutually exclude one another). Managing mutual exclusion and related scheduling problems has given rise to substantial literature in computer science. Some methods of attack include covering or partitioning the graph with cliques or threshold graphs. Here I survey some recent graph-theoretic results and examples motivated by this approach.

Partially supported by National Science Foundation Grant Number DCR-8503922

1. Introduction.

I am a pure mathematician by training and a computer scientist by choice and employment. I work in a department that is strong in graph theory. It is thus fairly natural for me to keep an eye out for computer science problems that have graphs in them, and to see if I can't attack them or generalize them by asking my graph theorist friends for the tools that I need. I'm happy to report that it sometimes works. It also has the happy result that sometimes I ask what appears to be a new question to my colleagues in graph theory, or find that an example or approach which seems natural in the computer science context helps answer a preexisting graph theory question.

The uses of graph theory in computer science are so great that I'm certainly not capable of a large survey. I'm going to pick two areas that have been productive for me in the last year or two. In this paper I'll talk a little

about mutual exclusion problems and a lot about graph coverings; in the next I'm going to talk a lot about fault-tolerant networks and a little about multiple connectivity in graphs.

In an example due to Dijkstra [D], five philosophers are gathered around a table. In the center is a large platter of spaghetti. In front of each philosopher is a plate; between each two philosophers is a fork [Figure 1]. If a philosopher becomes hungry, she attempts to pick up

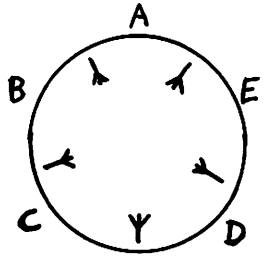

Figure 1.

the fork on either side of her; if she gets them both, she eats and puts them back down. If she gets only one fork, she waits patiently for the other; if she doesn't get both for a long enough time, she starves to death. The philosophers don't speak, and don't pass forks; in particular, if each one grabs her right-hand fork, they all starve.

The problem computer scientists see in this is writing programs (regarding each philosopher as a computer process) that will

(1) Prevent adjoining processes from eating at once (mutual exclusion);

(2) Prevent the system from halting with no-one able to eat (deadlock prevention);

(3) Let everyone eat eventually(starvation avoidance) or even promptly (fairness, bounded waiting).

When I first saw the problem, the graph jumped out at me. Each process is represented by a vertex; two processes that cannot proceed at the same time have their vertices connected by an edge. And I had the following dreadful thought[O1]: the waiter, seeing the problem, removes the

five forks and places in the center of the table, where anyone can reach them, 2 spoons, two knives, and two forks. Unfortunately, Philosopher A wants 2 spoons; B wants a spoon and a knife; C wants two knives and a fork; D wants two forks; and E wants a fork and a spoon. It is easy to see that the conflict pattern is exactly the same. The waiter takes those utensils away and brings a different set. Regrettably, the wants lists of the philosophers change again, and cause the same conflict pattern. I leave to you the following nasty exercise: What is the smallest number of utensils, and the smallest number of types of utensils, that could cause this conflict pattern? (Note that the original five forks were five different "kinds" of utensils, in that they were noninterchangable).

2. Synchronization Primitives.

One of the kinds of tools computer scientists use in modelling mutual exclusion problems is a device called a synchronization primitive. A simple example is called a semaphore. A simple semaphore is rather like one of the original philosophers' forks; it is a variable that can be grabbed by the first process to get to it. Suppose the processes P1, P2, P3 need to execute a certain piece of code (in each) called a critical section in such a way that only one at a time is executing a critical section. Let the variable X have the value 1 at start-up time. In each process put the following code:

```
P(X);
critical section ...
V(X);
```

The operation P(X) works as follows: if X = 1 then set X to 0 and proceed; if X = 0 then WAIT until X becomes 1. The operation V(X) changes X from 0 back to 1. (I think that P and V stand for Dutch words for "take" and "free").

The simple semaphore or PV operation will control only a set of processes such that any one excludes all the others: in the sense of our conflict graphs, a clique. To model the five dining philosophers with such operations, we would need 5 semaphores X1 through X5.

There are many other conflict patterns that arise naturally and that don't lead to cliques. One of the most important in practice is the so-called "Readers and Writers" problem[C]. Suppose we have some information in the computer, and several computer terminals that want to access it; some want to read the data, others want to change it (write or rewrite it). In some sense, it is all right if several processes read it at once; however, if someone is writing it, all other readers and writers must be excluded or they risk recieving (or creating) inconsistent information. Assuming 3 writers and 5 readers, the conflict graph looks like Figure 2.

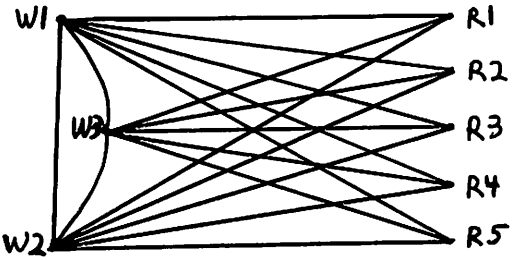

Figure 2.

How many simple semaphores would it take to model this? This amounts to asking for a set of cliques that edge-cover the graph. Note that each reader lies in a unique clique K_4 and that those 5 cliques cover the graph.

That isn't a very desirable solution. For one thing, W1 and W2 seem to be competing for a single resource and yet they need to grab 5 semaphores. We can't associate the edge between them with a single semaphore. So we already have a suggestion that it might be nice to consider edge-partitions as well as edge-coverings of the graph. Another complaint is that this way of controlling things needs 5 separate semaphores and thus 5 bits of shared memory.

We can solve the second complaint by using a slightly more powerful operation called a PV-chunk operation[HZ]. Here we allow the shared variable X to be larger than one and we allow it to be decremented or incremented by any positive integer. For example, start X at 5; each reader proceeds only if it can decrement X by 1, while a writer proceeds only if it can decrement X by 5. This operation is

actually present in current systems, like UNIX and C; it also seems feasible to implement in special-purpose hardware if needed in the future. Our programs are now much simpler, and we've achieved the needed mutual exclusion with only 3 bits of memory.

3. Threshold graphs.

The reader-writer graph is an example of a threshold graph. These were introduced first by Chvatal and Hammer [CH] but then turned up independently later by Henderson and Zalcstein [HZ] who were simply identifying the conflict graphs controllable by the PV-chunk synchronization primitive. Label the readers in our graph each by a 1 and the writers each by a 5. Let $t = 5$ be the threshold associated with the graph. The important property is that a set of vertices induces at least one edge if and only if the sum of its labels exceeds the threshold. A graph is a threshold graph if and only if it can be so labelled.

Here are a few equivalent definitions: a graph is a threshold graph if and only if it has no induced subgraph which is $2K_2$ (two disjoint edges), C_4 (a square), or P_4 (a path on 4 vertices); a graph is a threshold graph if and only if every induced subgraph has at least one vertex which is dominating, that is, a vertex which is connected to every non-isolated vertex in the graph.

Given a graph, one can quickly test whether it is a threshold graph. A more interesting question is to know the minimum required value of the threshold t, since this tells us among other things how much shared memory we need to run our synchronization primitive. This was calculated in an algorithmic way by Orlin [O]; a more recent calculation [O2] based on the "normal form" of Henderson and Zalcstein relates it more closely to the clique covering of the threshold graph.

Suppose the graph has d_0 isolated points; label them 0 and omit them from the graph. Now there are some vertices connected to all the vertices ("dominating vertices"); delete them from the graph. What is left has d_1 isolated vertices; label them 1 and delete them. Repeat the

deletion of dominating vertices; then label the d_2 new isolated vertices each with the integer d_1+1. At the next stage label the d_3 isolated vertices each with the integer $(d_1+1)(d_2+1)$. In stage j+1 label the d_j isolated vertices each with $(d_1+1)(d_2+1)\ldots(d_{j-1}+1)$. At the end one is left either with an isolated set of d_k points or with a clique. In the former case label the isolated vertices $(d_1+1)\ldots(d_{k-1}+1)$ and let t be that times (d_k+1) minus 1. In the latter case set $d_k = 1$, label all the vertices of the clique with the appropriate product of earlier (d_i+1) and let t be twice that minus 1. I haven't told how to label the "dominating" vertices but those can now be filled in fairly easily.

4. Shared Memory and covering theshold graphs with threshold graphs.

The value t can be motivated as a shared memory measurement; we can control a threshold graph having separator t by using PV-chunk operations on a shared variable capable of having the values 0 to t. The computer science theorem that motivated me to do this is the following: We cannot do mutual exclusion for a threshold graph with any synchronization primitives using shared variables capable of being in less than a total of t+1 states[02]. The proof is not too hard. It hinges on the fact that any union of subsets of the various "isolated sets" of d_1, d_2,..., d_k vertices represents a collection of processes capable of running at one time; there are t+1 such essentially different unions of subsets (two of them are not essentially different, in this sense, if they have the same number of processes from each set.) Since any two such essentially different collections of running processes can be followed by different sequences of processes starting and stopping, they must be represented by different states of the shared variables. This argument works even if we use much stronger primitives than PV-chunk (for instance, the strong test-and-set of Lynch and Fischer[LF]); it shows that PV-chunk is in some sense a "best possible" primitive for doing mutual exclusion on

threshold graphs.

This theorem has a surprising consequence for graph coverings. Let the threshold graph G be edge-covered by a collection $\{G_i\}$ of threshold graphs, where G_i has threshold separator t_i. Then G can be controlled by several PV-chunk variables, one for each G_i, with total number of states the product of the (t_i+1). Hence that product can be no less than t+1. My original proof of this involved an extremely complex argument on clique coverings of the various graphs and subgraphs; it now falls easily out as a corollary of a theorem on mutual exclusion.

5. Clique coverings and clique partitions of threshold graphs.

The clique covering number of the graph is now just $d_1 + \ldots + d_k$, since no two of the "isolated" vertices can lie in the same clique. What can we say about the clique partition number? That is, what if we attempt to choose cliques that partition the edges rather than just cover them? In the "normal form" of the graph, we draw a version of the graph that has all vertices that became "dominating vertices" on the right and all that became "isolated vertices" on the left. One possible clique partition comes to mind: cover the big right-hand clique with one clique, and all the crosswise edges with single edges. Under what circumstances is this best possible? Let's look at a concrete case, to see that it may be pretty far from the best possible partition.

Suppose that we have 2t vertices arranged as follows: t form a clique A, written on the right; t form a discrete set B, written on the left. I'll denote the crosswise edges -- in this case they form $K_{t,t}$ -- by C (Figure 3). The partition we suggested above needs 1 clique for A and t^2 for the edges in C, a total of $t^2 + 1$. But we can do much better. Partition A into perfect matchings (let t be even) and associate each of the t-1 matchings with a point of B. Now the edges from that point of B together with the edges

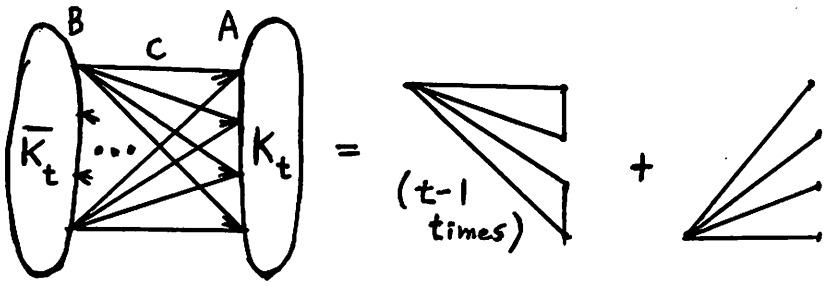

Figure 3.

of the matching form t/2 disjoint triangles. Altogether we have (t-1)t/2 triangles plus t edges left over from the remaining point of B. Thus we have a partition into (t-1)t/2 + t = (t+1)t/2 disjoint triangles and edges, just about half as many cliques as before.

Is this best possible? Yes. We have the t^2 edges in C; every time we combine 2 into a clique, we use at least one edge from A. If we try to combine more than two edges from C into one clique, we use edges from A at a faster rate.

6. Clique partitions of other graphs.

All this suggests an approach to clique partitions that has been pursued in [CEOP], [EFO], [EOZ], Here are a few of the results:

Suppose we consider a graph G on n vertices that divides into three parts: A and B containing vertices and edges, C with just connecting edges from A to B. The following results are from [EFO] except as noted:

Lemma. If G has a edges in side A, no edges in side B, and c connecting edges, then the clique partition number cp(G) is at least $c^2/(2a + c)$.

Corollary. If $1/2 < a < 1$ and $m = cn^a$, then for n large enough, $cp(K_n - K_m)$ is approximately c^2n^{2a}.

Lemma. Let G have a edges in side A, b edges in side B, and c connecting edges. Then cp(G) is at least $c - a - b - \min(a,b)$.

Example. [CEOP] Let cc(G) be the clique covering number of G. Then cp(G) - cc(G) can be at least $n^2/4 - n^{3/2}/2 + n/4$. The trick is to let A be the union of k/2 cliques on k

vertices each, where k is the square root of n, and B be a discrete set on n/2 vertices; C has all the edges from A to B.

Example. [EFO] cp(G)/cc(G) can grow as fast as cn^2 for some constant. This time we want A to be a single clique on n/2 vertices, and B the union of 4 cliques each on n/8 vertices; C has all the edges from A to B. (The constant c we obtain is very low, about 1/64. We know that it cannot exceed 1/12 for large n.)

Lemma. Let a clique K_r have r vertices partitioned into two sets A and B. Suppose A has a vertices and B has b vertices with $a + b = r$ and $a \geq m$. Then

$$((m-1)/m)ab \leq a(a-1)/2 + b(b-1)/2$$

Example. Let G be a graph with A a clique on n/3 vertices, B the union of two cliques each on n/3 vertices, and all cross edges. Then cp(G)/cc(G) grows proportionally to n^2. (This answers a question of Dom DeCaen).

More recently [EOZ] we have been looking at coverings and partitions by and of threshold graphs and chordal graphs (a chordal graph is a graph in which every cycle larger than the triangle C_3 has a chord). Actually, we've been looking at all pairs from the set (clique, threshold graph, chordal graph, arbitrary graph) and attempting to ask such questions as

(1) If a graph of type Y contains many edges, how big a subgraph of type X must it contain?

(2) How many graphs of type X does it take to cover a graph of type Y?

(3) How many graphs of type X does it take to partition a graph of type Y?

Some of these questions are already answered in the literature, but quite a few are not, and the above methods of attack have been quite productive. For example, consider an arbitrary chordal graph and ask how many cliques it may take to partition it. It is easy to see as above that $K_n - K_{2n/3}$ requires roughly $n^2/6$ cliques to partition it, and that any graph can be partitioned with

$n^2/4$ cliques; we don't see anything in the literature that would tend to close the gap between 1/4 and 1/6 as a coefficient. Using techniques very much like those above, we ([EOZ]) can show that there is a constant c (our proof only yields about .01) such that any chordal graph can be partitioned with no more than $(1-c)n^2/4$ cliques.

References

[C] P.J. Courtois, F. Heymans, and D.L. Parnas, Concurrent control with "readers" and "writers", Comm. Assn. Computing Machinery 14(1971), 667-668.

[CEOP] L. Caccetta, P. Erdős, E.T. Ordman, and N.J. Pullman, On the difference between clique numbers of a graph, Ars Combinatoria 19A(1985), 97-106.

[CH] V. Chvátal and P. Hammer, Aggregation of inequalities in integer programming, Ann. Discrete Math. 1(1977), 145-162.

[D] E.W. Dijkstra, "Hierarchical Ordering of Sequential Processes," in C.A.R. Hoare and R.H. Perrott (eds.), *Operating Systems Techniques*, Academic Press, New York (1972), pp. 72-93.

[EFO] P. Erdős, R. Faudree and E.T. Ordman, Clique partitions and Clique Coverings, Proc. First Japan Internat. Conf. on Graph Theory and Applications, Hakone, June, 1986(to appear).

[EOZ] P. Erdős, E.T. Ordman, and Y. Zalcstein, Graph coverings and partitions with chordal threshold graphs, in rough draft.

[HZ] P.B. Henderson and Y. Zalcstein, A graph-theoretic characterization of the POV-chunk class of synchronization primitives, SIAM J. Comp. 6(1977), 88-108.

[LF] N.A. Lynch and M.J. Fischer, On describing the behavior and implementation of distributed systems, Theoret. Comp. Sci. 13(1981), 17-43.

[O] J. Orlin, The minimal integral separator of a threshold graph, Ann. Discrete Math. 1(1977), 415-419.

[O1] E.T. Ordman, Threshold coverings and resource allocation, Proc. 16th S.E. Conf. on Graph Theory, Combinatorics, and Computing, Congr. Numer. 49(1985), 99-113.

[O2] E.T. Ordman, Minimal threshold separators and memory requirements for synchronization, SIAM J. on Computing (submitted).