Threshold coverings and resource allocation

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

ABSTRACT

In the Generalized Dining Philosophers problem, a collection of processes (in a distributed computer system) cannot all proceed at once due to conflicts over shared resources. The set of conflicts gives rise to a hypergraph, or, if minimal conflict sets are just pairs, a graph. We determine the minimal number of units and of types of shared resources giving rise to an arbitrary graph; these turn out to be the clique number and threshold dimension of the graph. We also examine the amount of storage (in bits) required to save usage information about the shared resources; this has implications for distributed process synchronization algorithms and for the strength of synchronization primitives such as PV-chunk.

1. Introduction.

This paper concerns certain graph theory problems suggested by the literature on synchronizing primitives, and particularly by the generalized dining philosophers problem. Loosely, any graph can be viewed as arising from a scheduling conflict problem, with the nodes representing processes and the edges representing pairs of processes that cannot run simultaneously. The conflicts may arise since the processes demand access to resources not available in enough quantity for both, since they change the same data set, because they have different security classifications, or for other reasons; but we are motivated primarily by the first reason, resource limitation. A main problem we consider is this: given a conflict graph, arising perhaps from some other circumstance, what resource

CONGRESSUS NUMERANTIUM, **49** (1985), pp.99-113

allocation problem(s) could have led to this graph?

In particular, a given graph (conflict pattern) may arise from a fairly small number k of separately labelled resources, or a typically larger number n of resources in one or more pools of interchangable resources. How large is n compared to k? If we use k distinctly labelled resources, each with in-use or not-in-use status recorded by a one-bit flag, will this take more or less space than using n interchangable resource units with the number in use stored in a shared variable using $\lfloor \log(n) \rfloor + 1$ bits?

Threshold graphs were introduced in [1] and [6]; the connection with the present work stems from [6], where it is shown that a threshold graph is precisely a graph for which one pool of interchangable resources suffices. The correspondence between multiple resources and threshold coverings is suggested by [3]. If none of the resources are interchangable, we find ourselves with a covering by cliques: similar situations arise in [4].

2. A simple example.

A classical resource conflict problem, known as the dining philosophers problem, was introduced in [2] and has been studied extensively. Suppose n philosophers are seated around a table; between each pair of philosophers is a single chopstick. The philosophers sit and think. Occasionally a philosopher gets hungry. When one gets hungry, she attempts to pick up the two chopsticks adjoining her. If she succeeds, she eats and puts the chopsticks back down. If she cannot get both, she puts down one (if she picked it up) and resumes thinking, trying again later. If she fails to eat too many times, she starves to death.

In this problem, we have n processes (the philosophers), and n units of resources (the chopsticks), one unit each of n types of resource (the chopsticks cannot be passed around). To store the state of the system in enough detail that we can tell whether any given

philosopher can eat, we must store n one-bit flags, or n bits total. A set of philosophers can all eat at once provided that no two are adjacent; phrased differently, each philosopher conflicts only with those immediately adjacent to the right and left. If we let n points denote the philosophers and draw lines between those pairs in conflict, the resulting graph is C_n, the n-cycle or n-sided polygon.

It is possible that some other resource allocation problem might lead to the same graph. For example, for n = 4, place two forks and two spoons in the center of the table. Let philosopher 1 require two spoons to eat, philosopher 3 (opposite) require two forks, and the other two each require a fork and a spoon. This leads to the same conflict pattern as the original problem, but requires only two resource types instead of four. It did still require a total of four units of resources, (two each of two types) and if we store the state of resource utilization in two counters, each with value zero to two, four bits of storage are still required.

Now consider an arbitrary resource allocation problem on n processes, and suppose that the conflict graph is C_n, that is, we have the same conflict graph as the dining philosophers. What is the minimum number of resource units present? The minimum number of distinct resource types? The minimum number of bits needed to record resource usage? The theorems later in this paper will make it easy to see that we need at least n units of resources, at least n/2 types, and at least n bits of storage.

3. Definitions and preliminaries.

An n-resource allocation problem (we usually omit the n) is a set of k processes numbered 1 to k; an available resources vector $(t_1, t_2, \ldots, t_n)$ where each t_j is a non-negative integer denoting the number of available resource units of type j; and k resource requirements vectors $(r_{m,1}, r_{m,2}, \ldots, r_{m,n})$ for m = 1 to k where $r_{m,j}$ denotes the

number of resources of type j that are required by process m. A subset S of {1, ..., n} is called admissible if the sum of the $r_{m,j}$ for m in S is less than or equal to t_j for all j; that is, all the processes in S can run at once.

Two resource allocation problems, both on k processes, are called equivalent if their collections of admissible sets are the same.

An (n-)resource allocation graph is a graph for which there is an n-resource allocation problem such that this graph has one node for each process and a set S of processes is admissible if and only if the induced subgraph on the set S of nodes has no edges. That is, each node is labelled by an n-tuple of non-negative integers and a set of nodes induces an edge if the sum of the n-tuples associated with them exceeds in at least one component the corresponding component of the "threshold vector" (t_1, ..., t_n).

Example 1. Not every resource allocation problem corresponds to a resource allocation graph. For example, if a computer has two tape drives and jobs 1, 2, 3, and 4 which require respectively 2, 1, 1, and 1 tape drives, a graph representing this contains edges from node 1 to nodes 2, 3, and 4, but now nodes 2, 3, and 4 form an inadmissible set yet induce no edges. Thus no resource allocation graph represents this problem. One would have to go to hypergraphs (in this case, one with a hyperedge {2, 3, 4}) to model this situation.

Proposition 1. A resource allocation problem is represented by a resource allocation graph if and only if every minimal inadmissible set is of size two.

Proof. Note that we do not consider problems where there are processes that cannot run at all, i.e. inadmissible sets of size one. If there is an inadmissible set S of size greater than two in a resource allocation graph, the induced subgraph on S contains an edge and hence an inadmissible set of size two. Conversely, if every minimal inadmissible set for a problem is of size two, then

the graph with edges defined by those inadmissible sets represents the given problem. //

We saw in section 2 that distinct resource allocation problems could lead to the same graph (e.g. the four-chopsticks and two-spoons, two-forks problems for the graph C_4). It is easy to see that by contrast, two graphs representing the same n-resource allocation problem are isomorphic: for any corresponding pairs of nodes have an edge between them (in both graphs) if and only if they form an inadmissible set.

Proposition 2. Every graph is the graph of an n-resource allocation problem, for some n.

Proof. Given a graph, consider each edge in the graph as a resource type, with one unit available; make that resource be required by each of the two nodes at the ends of that edge. Clearly this problem induces precisely the edges we need. //

A one-resource allocation graph (for a one-resource allocation problem) is the special case when n = 1; in this graph a set S of nodes is admissible if and only if the induced subgraph on S is totally disconnected. A graph is the graph of some one-resource allocation problem precisely if there is a labelling of the nodes of the graph by integers (the $r_{1,i}$) and a threshold t_1 such that a set of nodes is stable (induces no edges) if and only if the total of the labels does not exceed t_1. This is exactly equivalent to the definition of PV-chunk definable graphs in [6] and also the definition of a threshold graph [1]. For further background on threshold graphs, see [7]. The techniques exploited in this paper have developed in [5], [8], and [9].

We restate our problems in the light of the notation we have now established. Given an arbitrary graph G, it corresponds to (possibly many) n-resource allocation problems, each of which has a vector (t_1, ..., t_n). We would like to find the minimum value of n, that is the minimum number of resource types associated with G; call

this T(G). We want to find the minimum total number of resource units associated with G, that is, minimize $t_1 + \ldots + t_n$; we call this U(G). Finally, letting #(x) denote the number of binary digits needed to store x, we would like to minimize the storage needed to keep the present state of resources-in-use, that is, minimize $\#(t_1) + \ldots + \#(t_n)$, which we call B(G).

4. Threshold Dimension Arguments.

Lemma 1. Let G be an n-resource allocation graph. Then two nodes j and k lie on an edge (j,k) if and only if for some i from 1 to n, $r_{j,i} + r_{k,i} > t_i$.

The proof is obvious. We say that this edge is induced by resource i.

Lemma 2. Let G be an n-resource allocation graph. Then G can be covered (edge-covered) by (no more than) n threshold graphs.

Proof. For each i, i = 1, ..., n, let G_i be the subgraph with all the nodes of G and only those edges of G induced by resource i. By Lemma 1, every edge is in some G_i; each G_i is a one-resource allocation graph and thus a threshold graph. //

The threshold dimension of a graph G is the smallest integer t(G) such that G can be covered by t(G) threshold graphs. Lemma 2 now states that if G is an n-resource allocation graph, t(G) cannot exceed n. The smallest possible n for a graph G was called T(G) above; this was deliberate.

Theorem 1. T(G) = t(G). That is, for any graph G, the smallest n for which G is an n-resource allocation graph is the threshold dimension of G.

Proof. Suppose G = (N,E) is covered by n threshold graphs $G_i = (N_i,E_i)$ (where n may be taken as t(G)). Each G_i may be taken as a one-resource allocation graph with a corresponding one-resource allocation problem. Adding any extra nodes to bring N_i up to N (each new node needs no units of the resource) keeps G_i a threshold graph. We now

know how many units of each of these n types of resources is required by each node of G, so we have specified an n-resource allocation problem for G. This shows T(G) cannot exceed t(G); the converse was given by Lemma 2. //

Example 2. We already can solve the questions about the dining philosophers raised earlier. C_n has threshold dimension n/2 for n>3, since the only nontrivial threshold graphs contained in C_n are K_2 and P_3 (No threshold graph can have as an induced subgraph $2K_2$, P_4, or C_4). To characterize P_3 as a threshold graph, we must have t = 2 (minimum) with the center node labelled 2 and each end node labelled 1. It is now easy to see that our n philosphers will in fact require n utensils, of at least n/2 types. One edge requires one utensil (and one bit of storage); two adjoining edges can be covered by two distinct utensils, one bit of storage each, or one pair of utensils (the middle philosopher wants both, each end one would settle for one) so this pattern also requires two bits to store the number of utensils in use and in all, n bits of storage are needed. Our solution with two spoons and two forks for n = 4 does in fact minimize all three parameters we are considering.

5. Clique Coverings.

By a clique K_i in a graph G we mean a set of i nodes all connected (pairwise) by edges. A clique covering of G is a collection of cliques in G which cover (the edges of) G. The clique number c(G) of G is the smallest number of cliques in a clique covering of G.

Lemma 3. U(G), the minimum number of resource units in a resource problem for a graph G, cannot exceed c(G).

Proof. Let G be covered by c = c(G) cliques G_1, ..., G_c. For each clique, introduce one resource unit of type i, i = 1, ..., c; and let each node require the resource unit associated with each clique that that node belongs to. Now G is the resource allocation graph for this c-resource allocation problem with a total of c units of resources. //

In fact, U(G) = c(G); but to see this easily seems to require some additional machinery about threshold graphs. Let G be a threshold graph. Each node x has a label r_x associated with it, and a set of nodes induces an edge if their labels total more than a "threshold separator" t. Given a graph, what can we say about the size of the smallest possible t, which is called the "minimal separator" of G? In particular, how does t relate to c(G)? Orlin [10] gives an algorithm for determining the minimum possible t; Ordman [8] uses methods of Henderson and Zalcstein [6] to express this in a closed form.

By a dominating node of a graph G [9] we mean a node that is connected by an edge to each non-isolated node of G. A graph is a threshold graph if and only if every induced subgraph of G (including G itself) has a dominating node [8]. We can put G into normal form [6] as follows: place the isolated nodes of G (if any) in a set named D_0 and delete them from G; then put all dominating nodes of G into a set named C_1 and delete them (and their incident edges) from G. Continue inductively, isolated nodes going in D_i and dominating nodes in C_{i+1}, until G is exhausted. Let j be the last subscript for which C_j is nonempty. At the end make a minor modification: if C_j is nonempty and D_j is empty, move 1 node from C_j to D_j. This decomposition of G is essentially unique; in particular the orders c_i of the C_i and d_i of the D_i are uniquely determined. Each node in each D_i has the following nice properties: (1) It is connected to each point of each C_k for k not exceeding i, and to no other point of G. (2) It is contained in a unique maximal clique (whose points are itself and the points listed in (1)). Further, the maximal cliques obtained in this way cover G. Hence c(G) is equal to the total number of nodes contained in the D_i. Finally, it is shown in [8] that the minimal separator t of G, satisfies

$$t + 1 = \prod (d_i + 1), \quad (i = 1,\ldots,n).$$

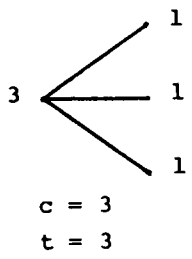
We are now in a position to prove:

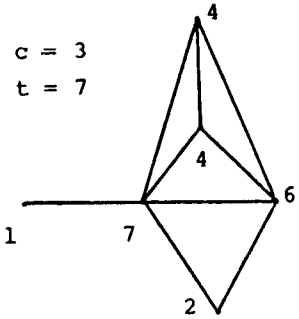
Theorem 2. Let G be a threshold graph with minimal threshold separator t and clique number c. Then

$$c \leq t \leq 2^c - 1$$

Proof. Observe that the sum of the d_i must be c, and that t is the product of the $d_i + 1$. We give the extreme cases here; that these are extreme may be shown formally using Lemma 5 below if desired. Fixing c, we minimize t by putting all the elements of the D_i into D_1. With c elements in D_1 and 1 in C_1, each element of D_1 gets the label 1 and the element of C_1 gets label c; t = c also. We maximize t by placing 1 node in each of D_1, D_2, ..., D_c as well as in each corresponding C_i; the node in each D_i is then labelled $2^i - 1$, and $t = 2^c - 1$ as required. //

For further details of the calculating methods, see [8]. Here are two examples of the extreme cases:





Theorem 3. U(G) = c(G); that is, the minimum total units of resources for a resource allocation graph equals its clique covering number.

Proof. We have already seen that U(G) cannot exceed c(G). Conversely, suppose we are given a graph G and a corresponding n-resource allocation problem; we must show that c(G) does not exceed the total resource usage $t_1 + \ldots + t_n$. First cover G by the n threshold graphs induced by the n resource types; the i-th threshold graph has threshold separator t_i. But then this i-th graph can be covered by no more than t_i cliques by Theorem 2 (the clique number of a threshold graph cannot exceed its lowest possible threshold separator). The union of these clique

coverings for the n threshold graphs is a clique covering for G, so G can be covered by no more than $t_1 + \ldots + t_n$ cliques as desired. //

6. Storage requirements for threshold graphs.

We have now characterized T(G), the number of resource types, and U(G), resource units usage, for arbitrary graphs. The remaining problem is storage requirements, in binary digits. This appears to be considerably harder; we resolve it in this section for threshold graphs and show examples for the general case in the next section.

Theorem 2 gives us a first step for threshold graphs. There are at least two resource allocation problems associated with a threshold graph that is not just a clique: treat it as a conflict over t copies of an interchangable resource, or as a conflict over c noninterchangable resources, one for each clique of the graph. Theorem 2 tells us that we cannot waste storage, and may save storage, by using the t interchangable units.

We want to conclude that a similar result holds for any resource allocation problem represented by the threshold graph G, that is, that B(G) = #(t), the number of binary digits in t.

Most of the rest of this section is devoted to the proof of the following theorem.

Theorem 4. Let G be a threshold graph with minimal separator t. Let G be covered by a collection of threshold graphs G_j each with separator t_j, $j = 1,\ldots,r$. Then

(i) the number of binary digits in t is no greater than the sum of the numbers of binary digits in each t_j; and

(ii) $t + 1 \leq \prod (t_j + 1)$, $(j = 1,\ldots,r)$.

Algebraic preliminaries.

Notation. #(x) denotes the number of binary digits in a positive integer x. $\lfloor y \rfloor$ is the greatest integer not exceeding y. log z denotes logarithm to the base 2.

Lemma 4. $\#(x) = \lfloor \log x \rfloor + 1 \geq \log(x+1) > \log x \geq \lfloor \log x \rfloor$

Lemma 5. Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_r)$ be lists of nonegative integers such that it is possible to partition B into disjoint subsets B_i, $i = 1, \ldots, n$, so that for each i the sum of the elements of B_i is at least a_i. Then $\prod_j (b_j + 1) \geq \prod_i (a_i + 1)$.

Proof. This follows from the fact that if $s + t \geq r$, then $(s+1)(t+1) = rs + s + t + 1 \geq (s+t) + 1 \geq r + 1$ //

Graph preliminaries.

Definitions. The subgraph of G over D_i is the subgraph of G induced by the union of D_i and all C_j with $j \leq i$. An edge f is an essential edge of this subgraph if it has one end in D_i.

The graph G has threshold height 1 if C_i and D_i are empty for $i > 1$. (That is, after the edges starting in C_1 are deleted, all nodes are isolated).

Lemma 6. G is covered by the subgraphs over the various D_i.

Lemma 7. Every maximal clique in the subgraph of G over D_i has exactly $1 + \sum c_j$, $(j = 1,\ldots,i)$, nodes.

Lemma 8. The subgraph of G over D_i has threshold height 1.

Proof. All nodes of each C_j for $j \leq i$ are dominating. If they are deleted, the remaining nodes (of D_i) are isolated. //

Lemma 9. Let f be an essential edge of the subgraph of G over D_i. Let H be a subgraph of G such that H is a threshold graph and let D'_k be one of the "D" sets in the threshold decomposition of H. Let K be the subgraph of H over D'_k. If f is an edge of K, then K contains no edge from C_r to D_s for $s \geq r > i$.

Proof. If K has an edge from C_r to D_s as well as the edge f from D_i to C_m, $m \leq i < r \leq s$, then K is not of threshold height 1 since the points in C_r and D_s on that edge are not dominating (there is no edge from C_r to D_i or from D_s to D_i) and also not isolated when dominating points are deleted (the edge between them remains). But K must be of threshold height 1 by Lemma 6, a contradiction. //

We can now manipulate the cliques of G and its

covering threshold graphs to allow us to use Lemma 5.

Proposition 3. Let G be a threshold graph with the usual notation for C_i, D_i, and t. Let G be covered by a collection of threshold graphs G_j, with the respective notations $C_{j,k}$, $D_{j,k}$, and t_j. (Recall that each G_j may have a different number of $D_{j,k}$'s). We can partition the collection of numbers $d_{j,k}$ into disjoint sets P_i such that for each d_i, the sum of the $d_{j,k}$ in P_i is at least d_i.

Proof. For each node in D_1, pick an edge f from it to C_1. Each such edge f is in some G_j and thus in the subgraph of G_j over some $D_{j,k}$. (It may be in many such; pick one). Let P_1 be the collection of $d_{j,k}$ corresponding to the selected $D_{j,k}$'s. Note that if m edges f lie in the same subgraph of G_j over $D_{j,k}$, we only place $d_{j,k}$ in P_1 once but claim that $d_{j,k}$ is at least m. This last claim follows since no two f's are in the same clique in G, hence they are in distinct cliques in the subgraph of G_j over $D_{j,k}$ and $d_{j,k}$ is simply the number of cliques in that subgraph. Clearly the sum of the $d_{j,k}$ chosen for P_1 in this way is at least d_1.

Having selected P_k for $k < i$, we select P_i as follows: for each node of D_i, pick an edge f from D_i to C_i. None of these edges are covered by any of the previously selected subgraphs since each subgraph selected so far was chosen to contain an essential edge from some D_i for an $i < k$ and hence by Lemma 9 cannot contain any edge from C_k to D_k. Thus we can select new $D_{k,j}$'s and continue as above.

Once we have selected sets P_i for all the d_i in this way, there may be some $d_{j,k}$ left unassigned. Assign them to the various P_i arbitrarily. This completes the proof of the proposition. //

Proof of Theorem 4.

(a) $t + 1 = \prod_i (d_i + 1)$ $(i=1,\ldots,n)$

(b) $\leq \prod_{j,k} (d_{j,k} + 1)$ $(j=1,\ldots,r;\ k=1,\ldots,n_r)$

(c) $= \prod_j \prod_k (d_{j,k} + 1)$ $(j=1,\ldots,r)(k=1,\ldots,n_r)$

(d) $= \prod_j (t_j + 1)$ $(j=1,\ldots,r)$

Here (a) and (d) are the known expression of t in terms of the d_i, (b) is by Proposition 3 and Lemma 5, and (c) is a simple rewriting of (b).

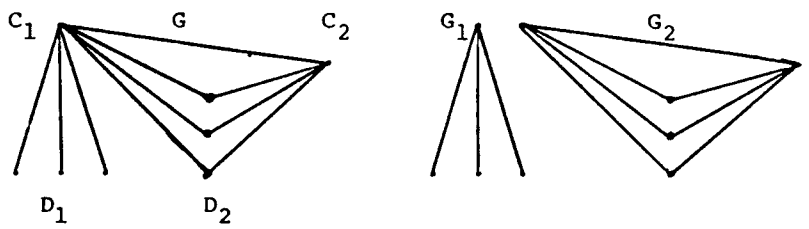
Now by lemma 4 and the above we see that

$$\sum \#(t_j) \geq \sum \log(t_j + 1) =$$
$$\log \prod (t_j + 1) \geq \log(t + 1) > \lfloor \log t \rfloor = \#(t) - 1$$

Since this is a strict inequality and the expressions on both ends are integers we can conclude that $\sum \#(t_j)$ is at least as large as #(t), as desired. This completes the proof of theorem 4. //

Example 3. Theorem 4 is tight in several ways. It is not necessarily true that the sum or the product of the t_i is at least t. To see this, consider G to be the graph with 1 element in each of C_1 and C_2, and 3 elements each in D_1 and D_2. Clearly t = 15. Now let G_1 be the graph induced by C_1 and D_1; clearly t_1 =3. Similarly let G_2 be induced by C_1, C_2, and D_2. (This is not a correct normal form for G_2 since C_1 and C_2 both dominate G_2). It is easy to see that t_2 is 3 and that G_1 and G_2 together cover G.

Note #(15) = 4 = 2 + 2 = #(3) + #(3)



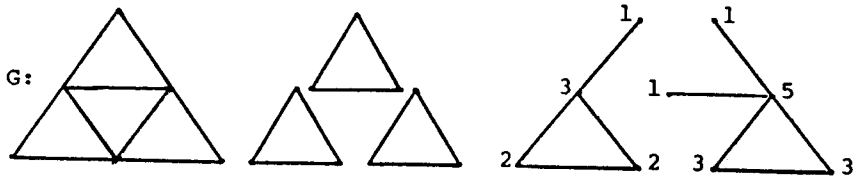
Remark. It would be possible to give a quite different proof of this theorem using results about distributed algorithms, in the spirit of [8]. [8] shows that to manage conflict avoidance by a distributed algorithm using PV-chunk as the synchronizing primitive, if the threshold graph G represents the conflicts between processes, one needs a shared memory capable of assuming at least t+1 values. If G could be covered by threshold graphs G_j, j = 1,...,r, then one could use that covering to construct a distributed algorithm that avoided conflicts

using r shared variables with value ranges 0 to t_j respectively. The shared memory would then be able to assume exactly $\prod(t_j + 1)$ values, completing the proof. The effect would be to prove a graph-theoretic result by proving correctness of the specified distributed computer program. While this would be an interesting application of a proof of program correctness, I suspect it would be less enlightening than the proof here.

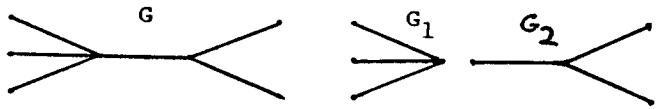
7. Storage requirements for general graphs.

It appears very hard, in the general case, to decide how to cover a graph G with threshold graphs G_i so as to minimize the total number of bits in the t_i. In particular, it is not true that one should use a covering by as few threshold graphs as possible.

Example 4. Consider the graph G below. c(G) = 3 by using the three "exterior" triangles as the cliques. G is not a threshold graph, but it can be covered with two threshold graphs, in for instance the way on the right. However, for this method, $t_1 = 3$ and $t_2 = 5$, so we would need 2 bits to store the status of the first threshold graph and 3 bits for the latter; hence this threshold covering requires 5 bits to store the status of the resources. It is not hard to see that any covering by two threshold graphs requires at least 5 bits storage; the 3-bit solution with three triangles is minimal for B(G).



Example 5. The figure shown below has c(G) = 6 and would require 6 bits using distinct resource units. It can



easily be covered by two threshold graphs, each with t = 3,

so that the state of the system can be stored in 4 bits.

Acknowledgements.

This paper would not have been possible without extensive discussions with Y. Zalcstein and with Ralph Faudree.

References

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

[2] E. W. Dijkstra, Hierarchical ordering of sequential processes, Operating Systems Techniques, Academic Press 1972.

[3] K. Ecker and S. Zaks, On a graph labelling problem, Bericht Nr. 99, Gesellschraft fur Mathematik und Datenverarbeitung MBH Bonn, 1977.

[4] P. Erdos, A. W. Goodman, and Louis Posa, The representation of a graph by set intersections, Canad. J. Math. 18(1967), 106-112.

[5] P. Erdos, E. T. Ordman, and Y. Zalcstein, Bounds on threshold dimension and disjoint threshold coverings, submitted.

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

[7] M. C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.

[8] E. T. Ordman, Minimal threshold separators and memory requirements for synchronization, submitted.

[9] E. T. Ordman and Y. Zalcstein, Nested interval graphs, submitted.

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