Ordering problems approximated: single-processor scheduling and interval graph completion

R. Ravi* Ajit Agrawal* Philip Klein*

Brown University

Abstract

In this paper, we give the first polynomial time approximation algorithms for two problems in combinatorial optimization. The first problem is single-processor scheduling to minimize weighted sum of completion times, subject to precedence constraints. The second problem, interval graph completion, is finding a minimum-size interval graph containing the input graph as a subgraph. Both problems are NP-complete; our algorithms output solutions that are within a polylogarithmic factor of optimal. To achieve these bounds, we make use of a technique developed and first applied by Leighton and Rao [12], together with a technique of Hansen [5].

1 Introduction

1.1 Graph ordering problems

The single-processor scheduling problem, scheduling to minimize weighted sum of completion times, arises in a manufacturing process or computational process when we want to get everything done in a hurry, and it is more important that some tasks get done quickly than others (thus the weights). Numerous papers have been written on the solution of special cases and approaches to this problem [1, 2, 6] and Lawler [9] showed that it is NP-complete. We give an approximation algorithm for a more general problem,

*Research supported by NSF grant CCR-9012357, NSF grant CDA 8722809, ONR and DARPA contract N00014-83-K-0146 and ARPA Order No. 6320, Amendment 1.

in which the goal is to minimize the storage-time product for the process. This problem is intended to model a situation arising in manufacturing or computing in which storage is an expensive resource, and its use over time must be minimized.

The other problem we consider is that of finding a smallest interval graph that includes the input graph. This problem arises, e.g. in archeology [7] in finding a consistent chronological model for tool use while making as few assumptions as possible. Unfortunately, the interval graph completion problem is NP-complete [3]. Moreover, no algorithm is known for approximately minimizing the number of edges whose addition yields an interval graph. As a first step towards finding such an algorithm, we have developed an algorithm that approximately minimizes (to within a log-squared factor) the number of edges in the completed graph.

1.2 Techniques

Both of the problems we consider are closely related to the problem of finding an optimal linear arrangement. This is the problem of ordering a graph's nodes from 1 to n so as to minimize the sum over all edges $\{v, w\}$ of the number of nodes between v and w. Optimal linear arrangement was shown to be NP-complete by Garey, Johnson, and Stockmeyer [4]. Hansen has shown [5] how to approximately solve a more general problem, embedding the node-set of a graph in a d-dimensional grid so as to approximately minimize the weighted sum of edge-distances. He obtains this result by proving a lower bound on the cost of the best embedding in terms of the minimum size of a separator in the graph. He then combines this with a recently developed algorithm of Leighton and Rao [12] for finding an approximately minimum balanced separator. This paper consists in further applying these two techniques.

2 Separators in Graphs

In this section, we define the notion of a separator which we use in this paper. The basic idea is to remove nodes or edges in order to split a graph into pieces, each of which is "small" with respect to the original graph in that its *node weight*—the sum of the weights of its nodes—is at most a fraction of the node weight of the original graph. Let G be a graph with node weights. If G is undirected, we define a b-bounded *node separator* of G to be a set of nodes of G whose removal separates G into pieces, where the node weight of each piece is at most a fraction b of the total node weight of G. The cost

of the separator is the total weight of the removed nodes. A method for finding small node separators[1] follows from the work of Leighton and Rao. The following lemma, which is due to Leighton, Makedon, and Tragoudas [11], is used in our algorithm for interval graph completion.

Lemma 2.1 *There is a polynomial-time algorithm to find a $\frac{3}{4}$-bounded node-separator with cost within a $O(\log n)$ factor of the optimal $\frac{2}{3}$-bounded node-separator.*

For a directed acyclic graph G, we define a *DAG edge-separator* to be a partition of the nodes into two sets, A and B, such that all edges between A and B go from A to B. The cost of the separator is the total weight of edges from A to B. The separator is said to be *b-balanced* if A and B each have node weight at least a fraction b of G's node weight. We show a reduction from finding a small separator of this form to finding a small separator of another form, defined by Leighton and Rao. Our reduction is closely related to one we presented in a recent paper [8].

Definition [12]: For a strongly connected graph G', the *edge separator* determined by a three-way node partition $(S, T, \overline{S \cup T})$ is the set of edges that leave S or enter T. The cost of the separator is defined to be the sum of the weights of the edges comprising it. Such a partition is said to be *b-balanced* if the node weight of $S \cup T$ and the node weight of $\overline{S \cup T}$ are each *at least* a fraction b of the total node weight of G'.

Leighton and Rao prove the following lemma.

Lemma 2.2 *There is a polynomial-time algorithm to find a $\frac{1}{4}$-balanced edge-separator with cost within a $O(\log n)$ factor of cost of the minimum-cost $\frac{1}{3}$-balanced edge-separator.*

We now derive from Lemma 2.2 a result about finding edge-separators in DAGs.

Lemma 2.3 *Given a DAG G with weights on the nodes and edges, we can find a $\frac{1}{8}$-balanced DAG edge-separator of G whose cost is within a factor of $O(\log n)$ of the minimum-cost $\frac{1}{3}$-balanced DAG edge-separator of G where n is the total number of nodes in G.*

Proof: We transform the DAG G to a strongly connected graph G' as follows. The nodes of G' are the same as the nodes of G. An edge (u, v) in G of weight w is replaced in G' by a pair of edges, an *original* edge (u, v) of weight w and a

[1]Here and henceforth, a "small" separator is one whose size is nearly minimum.

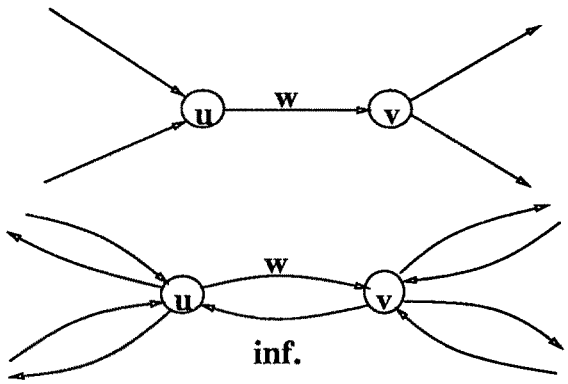

Figure 1: Augmentation of the DAG G to a strongly connected graph G' by adding a reverse edge of infinite cost for every original edge (u, v) of cost w in G.

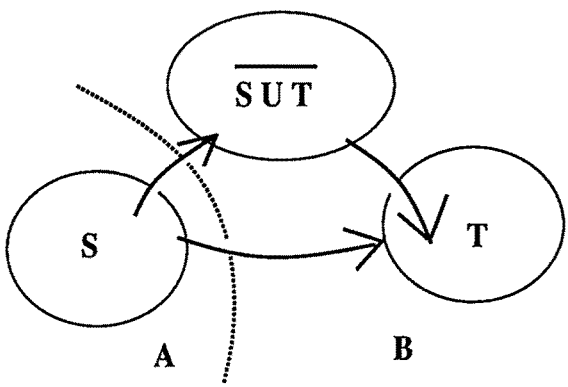

Figure 2: The augmented graph G' is separated by the algorithm of Leighton and Rao [12] into three pieces S, T and $\overline{S \cup T}$, where the edges of the separator are those leaving S or entering T as shown above. Since both pieces $S \cup T$ and $\overline{S \cup T}$ are $\frac{1}{4}$-balanced, assuming that S has more weight than T, then the partition A and B is $\frac{1}{8}$-balanced and forms a DAG separator of G.

reverse edge (v, u) of infinite weight (see Fig. 1). We now find an approximate $\frac{1}{4}$-balanced edge-separator $(S, T, \overline{S \cup T})$ in G' using the algorithm of Lemma 2.2. Note that no reverse edges can be in this separator since they have infinite weight. Hence, the edges in the separator are all original edges (Fig. 2). Without loss of generality, suppose S has at least as much node weight as T. Hence, its weight is at least a fraction $1/8$ of the total node weight of G. In this case, we choose A to be S, and B to be $T \cup (\overline{S \cup T})$. Then the partition (A, B) in the original graph G is a $\frac{1}{8}$-balanced directed edge separator. The cost of the directed edge separator in G is the same as the cost of the $\frac{1}{4}$-balanced edge-separator in G', which in turn is at most $O(\log n)$ times the minimum cost of a $\frac{1}{3}$-balanced edge-separator in G. It remains only to point out that for any $\frac{1}{3}$-balanced DAG edge-separator (A, B) in G, there corresponds a $\frac{1}{3}$-balanced edge-separator in G, namely $(A, \emptyset, B)$. □

3 Approximating scheduling problems

In this section, we present approximation algorithms for two single-processor scheduling problems. The performance bounds are polylogarithmic in the total weight of the input graph. We define each of these problems and present the approximation algorithms below.

3.1 Minimizing storage-time

Let $G = (V, E)$ be a DAG. The nodes represent tasks to be scheduled on a single processor. The time required to execute the task t is denoted by $\ell(t)$. Each edge $e = (u, v)$ has a weight $w(e)$ associated with it. This weight represents the number of units of storage required to save the intermediate results generated by task u until they are consumed at task v. Let $\tau = v_1, \ldots, v_n$ be a topological ordering of G. We denote the storage-time cost for this ordering by C_τ. We define C_τ as $C_\tau = \sum_{\text{all edges } e} s(e)w(e)$ where $s(e)$ for an edge $e = (v_i, v_j)$ is exactly the sum of the execution times of all tasks ordered between tasks v_i and v_j, inclusive. In other words, $s(e) = \sum_{k=i}^{j} \ell(v_k)$ The problem is to find the topological ordering τ_{opt} that minimizes the storage-time cost over all orderings. When $w(e)$ equals 1 for all edges e, then this problem reduces to the NP-complete problem of directed optimal linear arrangement [3]. We call $s(e)$ and $s(e)w(e)$ respectively the *stretch* and the *weighted stretch* of the edge e. The main result of this section is as follows.

Theorem 3.1 *There is a polynomial-time algorithm to minimize the total storage-time cost to within a factor of $O(\log n \log L)$, where L is the sum of execution times, and n is the number of nodes.*

3.1.1 A lower bound

We first derive a lower bound for the cost of a topological ordering using a technique analogous to that of Hansen[5]. Consider a topological ordering $\tau = v_1, \ldots, v_n$. We define the cut S_i to be the set of edges going from nodes v_1 through v_i to the rest of the nodes. We define the cost C^i of this cut to be the sum of the edge weights of all the edges in the cut, multiplied by the weight of the vertex v_i. We use the following observation to obtain the lower bound on the cost of the minimum storage-time schedule.

Observation 3.1 *For any topological ordering τ, the sum of the weighted stretches of all the edges, and hence the storage-time cost of τ, is at least $\sum_{i=1}^{n} C^i$.*

Lemma 3.1 *For a computation DAG G with total completion time L such that the length of each task is at most $L/6$, the optimal storage-time cost is at least $\Omega(LB)$ where B is the weight of the minimum $\frac{1}{3}$-balanced DAG edge-separator of G.*

Proof: Consider the topological ordering τ_{opt} that minimizes the storage-time product in G. Let i be the minimum index such that $\sum_{m=1}^{i-1} \ell(v_m)$ is at least $\frac{1}{3}$ of L, and let j be the maximum index such that $\sum_{m=j}^{n} \ell(v_m)$ is at least $\frac{1}{3}$ of L. For any k between i and $j-1$ inclusive, the cut S_k is a $\frac{1}{3}$-balanced DAG edge-separator, and hence has weight at least B, where B is the weight of the minimum $\frac{1}{3}$-balanced DAG edge-separator. Since we assumed that each node has weight at most $L/6$, it follows that $\sum_{k=j}^{n} \ell(v_k) \leq L/3 + L/6$, so

$$C_{opt} \geq \sum_{k=1}^{n} C^k \ell(v_k) \geq B \sum_{k=i}^{j-1} \ell(v_k) \geq BL/6$$

□

Now, let v be any node in G, let P be the set of ancestors of v (nodes that can reach v), and let Q be the set of descendents of v. Let $\hat{G}_v$ be the graph obtained from G by deleting v, contracting P to a single node p, and contracting Q to a single node q. We find a minimum DAG edge-separator $(\hat{A}, \hat{B})$ where $p \in \hat{A}$ and $q \in \hat{B}$. Such a separator can be found using a min-cut computation in $\hat{G}_v$. Let A be the set of nodes of G in $\hat{A}$, together with P, and let B be the set of nodes of G in $\hat{B}$, together with Q. We call

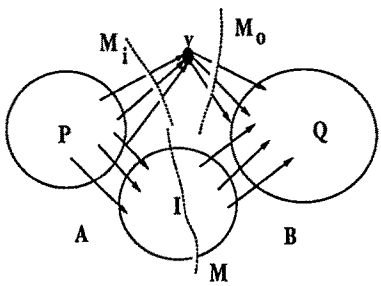

Figure 3: If there is a node of weight more than a sixth of the total node weight in the graph G, then we partition the nodes of $G - \{v\}$ into pieces P, Q and I that are respectively before, after, and incomparable to v in the partial order represented by the DAG G. We then contract P and Q to single nodes p and q respectively and find a minimum weighted DAG edge-separator $(\hat{A}, \hat{B})$ of weight M in $\hat{G}_v$. We use M_i and M_o to denote the weights of the incoming and outgoing edges of v respectively. We can then derive a minimum v-DAG edge-separator (A, B). Namely, we let A equal $\hat{A} \cup P - \{p\}$ and B equal $\hat{B} \cup Q - \{q\}$.

(A, B) the *minimum v-DAG edge-separator.* Intuitively, (A, B) is a *DAG* edge-separator in $G - \{v\}$. In the next lemma, we give a simple lower bound on the storage-time cost of G. This lower bound is particularly useful in the case where there is a node of large weight. The lemma is illustrated in Fig. 3.

Lemma 3.2 *Let M be the minimum v-DAG edge-separator in $\hat{G}_v$. Let M_i and M_o be the total weight of the incoming edges and outgoing edges of v, respectively. Then $(M + M_i + M_o)$ times the weight of v is a lower bound on the storage-time cost of G.*

3.1.2 The algorithm

We now give the algorithm for finding a topological ordering of a DAG G that approximately minimizes the storage-time cost. The algorithm is as follows. If G consists of a single node, the algorithm is trivial. Otherwise, we proceed recursively as follows.

If every node is of weight at most a factor $\frac{1}{6}$ of the total node weight of the graph L, then we find an approximately minimum $\frac{1}{8}$-balanced DAG edge-separator (A, B) in G as outlined in Lemma 2.3. The topological ordering of G consists of the nodes of A, recursively ordered, followed by those in B, also

recursively ordered. If there is a vertex v in G of weight greater than $L/6$, then we make use of Lemma 3.2 and find a minimum v-DAG edge-separator (A, B). The topological ordering of G consists of the recursive topological ordering of A, the node v, and the recursive topological ordering of B.

3.1.3 Performance Guarantee

Let us introduce some notation for convenience. We define a stage of the algorithm as follows. Let $G_1, \ldots, G_k$ be the subgraphs of G at the beginning of stage i. At the first stage there is only one such graph and that is G itself. The subgraphs at the beginning of the stage $i+1$ are the subgraphs $A_1, B_1, A_2, B_2, \ldots, A_k, B_k$, where A_j and B_j are the graphs found by decomposing G_j for recursively ordering its nodes. For each such graph G_j, some of its edges are removed during the decomposition. These edges go between the node sets A_j and B_j if G_j has no large weight node. If G_j has a large weight node v, then the edges either go between the node sets A_j and B_j, between the nodes in A_j and the node v, or between the node v and the nodes in B_j.

Observation 3.2 *Each edge in G is removed once and exactly once by the algorithm.*

Lemma 3.3 *The sum of the weighted stretches of the edges removed at a stage is at most $O(C_{\tau_{opt}} \log n)$, where $C_{\tau_{opt}}$ is the storage time product for the optimal topological ordering τ_{opt}, and n is the number of nodes in the graph.*

Proof: Let $G_1, \ldots, G_k$ be the subgraphs at stage i. Consider a subgraph G_j with node weight L_j.

If G_j does not have a high-weight node, then by lemma 3.1 the optimal storage time product C^j_{opt} for ordering its nodes is at least $B_j L_j/6$, where B_j is the weight of the minimum $\frac{1}{2}$-balanced edge separator of G_j. By lemma 2.3, the weight of the edges removed from G_j is at most $cB_j \log n_j$, where n_j is the number of nodes in G_j, and c is the constant for the separator algorithm. Hence the sum of their stretches in the ordering given by the algorithm can be at most $cL_jB_j \log n_j$, and hence at most $6cC^j_{opt} \log n$.

Now suppose G_j has a node of weight at least $\frac{L_j}{6}$. In this case, we use Lemma 3.2, which is illustrated in Fig. 3. By Lemma 3.2, C^j_{opt} is at least $(M_i + M_o + M)\frac{L_j}{6}$. The sum of the weighted stretches of the edges removed from G_j is at most $(M_i + M_o + M)L_j$ and hence is at most $6C^j_{opt}$.

Since $\sum_{j=1}^{k} C^j_{opt}$ is at most $C_{\tau_{opt}}$, it follows that the sum of stretches of the edges removed at a stage is at most $O\left(C_{\tau_{opt}} \log n\right)$. □

Since the node weights of each of the subgraphs at stage i is at most $(\frac{7}{8})^i L$, where L is the sum of the execution times of the tasks in G, it follows that the number of stages in the above algorithm is $O(\log L)$. Thus, Theorem 3.1 follows from Lemma 3.3.

3.2 Minimizing weighted completion time

We describe the problem of minimizing the weighted completion time of a DAG G. The nodes of G represent tasks. The execution time for the task t is denoted by $l(t)$, and the weight of the task is denoted by $w(t)$. For an ordering $\tau = \{v_1, \ldots, v_n\}$, the weighted completion time is given by $\sum_{i=1}^{n} w(v_i)\sigma(v_i)$, where $\sigma(v_k)$ denotes the time at which the task for the node v_k is completed in the given ordering. We assume that the execution of the first task begins at time 0, so $\sigma(v_k)$ is given by $\sum_{i=1}^{k} l(v_i)$. The objective is to find a topological ordering of G such that the weighted completion time is minimum over all such orderings. The problem is known to be NP-complete [3].

We reduce the above problem to an instance of minimizing the storage-time product. From the given DAG G representing the computation, we build an augmented DAG G' as follows. We add a new node s to G and add directed edges from s to each of the vertices of G. We then assign weights to the nodes and edges of G' as follows. Each original node v, is allotted a weight equal to its execution time $l(v)$. The added node s is assigned weight 0. All the original edges of G are weighted 0. For every node t, the added edge (s, t) is assigned weight $w(t)$. It can easily be checked that the value of the storage-time product for any ordering of G' is exactly the value of the weighted completion time of G for the same ordering of the nodes as in $G' - \{s\}$. This implies that approximating the minimum storage-time product for G' yields an approximation for the weighted completion time for G with the same performance guarantee. Using the results of the previous section, we can prove the following theorem.

Theorem 3.2 *There is a polynomial-time algorithm to minimize the weighted sum of completion times to within a factor of $O(\log n \log L)$, where L is the sum of execution times, and n is the number of nodes.*

4 Interval Graph Completion

The interval graph completion problem consists in finding a smallest interval graph that contains the input graph as a subgraph, and has the same nodes as the input graph. An interval graph is a graph whose vertices can be mapped

to distinct intervals in the real line such that two vertices in the graph have an edge between them iff their corresponding intervals overlap.

Fact 4.1 *[13] There exists an ordering of the nodes in any interval graph such that if a node u with number i has an edge to a node v with number j, for i less than j, then every node with number between i and j has an edge to v.*

We shall refer to such an ordering as the "interval graph ordering." An ordering of the nodes of the input graph induces an interval graph containing the input graph as a subgraph, namely the graph obtained by adding edges as needed until the condition of 4.1 holds. This augmentation can be done in time proportional to the number of edges in the augmented graph. Our algorithm for interval graph completion will output an ordering such that the total number of edges in the augmented graph is small with respect to the optimum. Let G be the input graph, and let G_{opt} be a smallest graph containing G. We shall start by giving a lower bound for the number of edges in G_{opt}. We then present an ordering of the nodes of G such that the number of edges in its augmented interval graph is no more than a polylog factor of the number of edges in G_{opt}.

4.1 Lower Bound

We again use the technique of Hansen to get a lower bound on the size of the minimum interval graph.

Lemma 4.1 *If B is the size of the optimal $\frac{2}{3}$-bounded node-separator for G, then G_{opt} has $\Omega(Bn)$ edges.*

Proof: Let interval graph ordering of G_{opt} be $\tau = v_1, \ldots, v_n$. For each i, consider the set of nodes $V_i = \{v_1, \ldots, v_i\}$. Let the set of neighbors of V_i not in V_i be N_i, and suppose it has size C_i. For i between $n/3$ and $2n/3$, each of the node sets N_i is a $\frac{2}{3}$-bounded node-separator and hence C_i must be at least B for each such cut. Moreover, every node in N_i is adjacent to some node in V_i. Hence by fact 4.1 it follows that the node v_i must have edges in G_{opt} to each of the nodes in N_i. Thus the total number of edges in G_{opt} must be at least $\sum_{i=1}^{n} C_i$ which is at least $\sum_{i=\frac{n}{3}}^{\frac{2n}{3}} C_i$ and hence at least $Bn/3$. □

4.2 Algorithm

We now give the algorithm for ordering the nodes of G. If G consists of at most two nodes, we use any ordering. Otherwise, we find a $\frac{3}{4}$-bounded

node separator of G using the algorithm of Lemma 2.1, order each of the pieces recursively, and pick any order between the pieces. The nodes in the separator are then ordered arbitrarily after all of the pieces. Let σ be the resulting ordering of the nodes of G. By adding edges to G, we can obtain a graph G^* for which σ is an interval graph ordering. Namely, if a node u numbered i has an edge to a node v numbered j, then for every k $(i < k < j)$, we add an edge between v and the node numbered k. We show in the next subsection that the number of edges in G^* is small.

4.3 Performance Guarantee

Lemma 4.2 *The number of edges in G^* is no more than a $O\left(\log^2 n\right)$ factor of C_{opt}, the number of edges in G_{opt}.*

Proof: Let us define the concept of stages as before. Let the ith stage consist of subgraphs $G_1, \ldots, G_k$. The first stage consists of the graph G. Let S_j be the the node separator for G_j found by the algorithm. Let the optimally augmented interval graph for G_j have C_j edges. By lemma 4.1 C_j is at least $B_j n_j/3$, where B_j is the optimal $\frac{2}{3}$-bounded node separator of G_j, and n_j is the total number of nodes in G_j. All the edges in the interval graph produced by the algorithm must go between the pieces of G_j and the node separator of G_j found by the algorithm. The total number of such edges is at most $x_j n_j$, where x_j is the size of the node separator of G_j found by the algorithm. By Lemma 2.1, x_j is at most $cB_j \log n_j$, where c is the constant in the guarantee of the algorithm. By Lemma 4.1, therefore, the total number of edges added between the pieces of G_j is at most $3c \log n$ times C_j. Since the value of $\sum_{j=1}^{k} C_j$ is at most the number of edges in G_{opt}, it follows that the total number of edges added to the interval graph at any stage is at most $3c \log n C_{opt}$. Moreover, since the sizes of the subgraphs go down by a factor of 3/4 at each stage, it follows that the total number of stages is $O(\log n)$. This implies the lemma. □

References

[1] D. Adolphson and T. C. Hu, "Optimal linear ordering", *SIAM J. Appl. Math. 25* (1973), pp. 403-423.

[2] R. W. Conway, W. L. Maxwell, and L. W. Miller, *Theory of Scheduling* (1967), Addison-Wesley, Reading, Massachusetts.

[3] M. R. Garey and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-completeness*, W. H. Freeman, San Francisco (1979).

[4] M. R. Garey., D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete graph problems," *Theor. Comput. Sci. 1* (1976), pp. 237-267.

[5] Mark D. Hansen, "Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems," *Proceedings, 30th Symposium on Foundations of Computer Science* (1989), pp. 604-609.

[6] W. A. Horn, "Single machine job sequencing with treelike precedence ordering and linear delay penalties," *SIAM J. Appl. Math.23* (1972), pp. 189-202.

[7] D. G. Kendall, "Incidence matrices, interval graphs, and seriation in archeology," *Pacific J. Math. 28* (1969), pp. 565-570.

[8] P. Klein, A. Agrawal, R. Ravi, and S. Rao, "Approximation through multicommodity flow", *Proceedings, 31st Annual Symp. on Foundations of Comp. Sci.,* (1990), pp. 726-737.

[9] E. L. Lawler, "Sequencing jobs to minimize total weighted completion time subject to precedence constraints," *Annals of Discrete Math. 2* (1978), pp. 75-90.

[10] F. T. Leighton, F. Makedon, S. Plotkin, C. Stein, E. Tardos, S. Tragoudas, "Fast approximation algorithms for multicommodity flow problems," *Proceedings, 23rd Annual ACM Symposium on Theory of Computing* (1991), to appear.

[11] F. T. Leighton, F. Makedon, and S. Tragoudas, personal communication, 1990

[12] F. T. Leighton and S. Rao, "An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms", *Proceedings, 29th Symposium on Foundations of Computer Science* (1988), pp. 422-431.

[13] G. Ramalingam, and C. Pandu Rangan, "A unified approach to domination problems in interval graphs", *Information Processing Letters, vol. 27* (1988), pp. 271-274.