The assignment of computational tasks among processors in a distributed system

by CAMILLE C. PRICE
Southern Methodist University
Dallas, Texas

ABSTRACT

The flexibility afforded by multiprocessor systems opens the question of how to assign computer program modules among functionally similar processors in a distributed computer network. In the model under consideration, the modules of a program are to be assigned among processors in such a way as to minimize interprocessor communication while taking advantage of affinities of certain modules to particular processors. The problem is formalized as a zero-one quadratic programming problem, and a solution is sought through an iterative technique that performs a series of transformations on an assignment matrix. Convergence to a locally optimum assignment is guaranteed, and an easily testable condition is given for which this local optimum is also a global optimum. An illustration of this algorithm is provided, results of performance experiments are reported, and suggestions are made for further study.

INTRODUCTION

A distributed computer network is considered to be a set of programmable processors interconnected to some extent by communication links.[14] Recent technological advances, such as the economical fabrication of processors and the development of broadband communication facilities, have contributed to the feasibility of distributed computing systems; and the trend toward large shared database systems promises an increased popularity for the use of distributed networks. It is important that cost-effective methods be developed for these systems to control the allocation of computing resources among the jobs introduced into the network.

Scheduling theory deals with the general problem of allocating limited resources among multiple tasks when choices exist in the allocation process.[7] The policies governing the apportionment of the resources are called scheduling rules or scheduling algorithms. Scheduling problems have demanded a great deal of attention since the development of digital computer systems, because scheduling algorithms are needed to assign a set of jobs to computer resources which are used in executing or servicing the jobs.[6]

The nature of job scheduling in a computer system depends on the functional similarity of the processing nodes and on the degree of communication available between processors. If the network consists of functionally different processors, then job scheduling is simple since each job would be designed for, and therefore assigned to, one particular specialized processor.

In a network of functionally similar processing nodes, it may be possible to assign the parts of a program freely among the processors; but in a practical sense, the communication links in a distributed network constitute inherent bottlenecks and therefore constrain the assignment of computational tasks. When high penalties are imposed for communication, the practical solution is to minimize the amount of communication between processors by assigning related tasks to the same processor. However, if the processors in the network were fully connected by high capacity data links, many feasible alternative assignments of computational tasks to processors would exist and should be evaluated by the job scheduler. In such cases, interprocessor communication would no longer be regarded as a serious constraint, but rather as a means of improving the overall efficiency of the system. The Cm* multimicroprocessor system[23] provides an example of precisely the kind of distributed system that will be considered in this paper. In the Cm* system, computational tasks, called *utilities,* may in general be executed by any microprocessor in the system.

The problem to be examined here is that of assigning computational tasks among processors in a distributed computer network having functionally similar nodes, but in which certain nodes have an advantage over others for particular jobs. The assignment is to be made in such a way as to take advantage of particular efficiencies of some processors for certain jobs while minimizing the costs of communication between jobs that are assigned to different processors.

In the next section, the problem is stated formally and formulated as a zero-one quadratic programming problem. The following section contains a description of an algorithm that can be applied to this scheduling problem. Conditions are given under which the local optimum achieved by this algorithm is also a global optimum. Results of performance experiments are reported. The final section contains a brief summary of work that has been done on this problem and gives suggestions for further study.

BASIC ASSUMPTIONS AND FORMULATION OF THE PROBLEM

The programs being executed within the distributed computer system are assumed to be partitioned into functional modules (containing executable code and/or data) which, in general, may reside on any processor in the system. There is no parallelism or multitasking of module execution within a program. Each processor may be multiprogrammed, and divide its time among several programs, but concurrent execution of the modules in one program is not considered. Thus the programs to be discussed here are serial programs, for which execution can shift from one processor to another.

Although the processors in the distributed system under discussion are functionally similar, they need not be identical. In fact, certain processors may have particular efficiencies for executing particular program modules. For example, some processors may have high speed arithmetic capabilities, access to a needed database, a large high-speed memory, access to certain peripheral devices, or other facilities associated with them that make them particularly well-suited for executing specific program modules.

The network is considered to be a fully-connected one, i.e., there is a direct communication link between every pair of processing nodes. It is also assumed that the communication paths between all processors are similar, that is, that the cost of sending a unit of data between any two processors is the same.

The modules of a modular program must be assigned among the processors in such a way as to minimize interprocessor communication while taking advantage of affinities of certain modules to particular processors. Therefore, there are two kinds of costs that must be considered in the search for a good assignment.

1. Each module has an execution cost that depends on the processor to which it is assigned. Let e_{ij} represent the cost of executing module i on processor j.
2. Any two modules that communicate during program execution incur a penalty if they are assigned to different processors. (It is assumed that the cost of such communication is zero when the reference is made between modules residing on the same processor.) Let the cost of communication between program modules i and k be denoted by c_{ik}.

An optimal assignment is one which minimizes the sum of the execution costs of the modules on the processors and the intermodule reference costs incurred when communicating modules reside on different processors.

It should be noted that the costs e_{ij} and c_{ik} must be measured in the same units of money or time. If these costs are measured in time, then the assignment minimizes the actual utilization of system resources.

Since the distributed program is to be executed in a serial fashion, and therefore all execution costs and communication costs are incurred in disjoint time intervals, the cost of the assignment is actually the minimal completion time of the program.

The problem can be formulated as a zero-one quadratic programming problem as follows:

minimize

$$\sum_{i=1}^{m}\sum_{j=1}^{n} e_{ij}x_{ij} + \sum_{i=1}^{m}\sum_{k=i+1}^{m} c_{ik} - \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=i+1}^{m} c_{ik}x_{ij}x_{kj}$$

subject to the constraints

$$x_{ij} = 0, 1 \qquad \text{for all } i, j \qquad \text{(c1)}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \text{for all } i \qquad \text{(c2)}$$

where m is the number of program modules and n is the number of processors.

Zero-one polynomial programs can be converted to linear programs with nonlinear secondary constraints,[12] but this problem is approached here with techniques which take advantage of the special structure of the problem.

The problem has been solved for $n = 2$ by Stone.[29] A model is developed that can be interpreted as a commodity flow network, and an assignment is made by applying a maximum flow algorithm.[11] Efforts to extend this method to the general n-processor case have not been completely successful.[30]

For n-processor problems in which the intermodule reference pattern is constrained to be a tree, an optimal assignment can be obtained by using a shortest path algorithm.[5,10,31] The graph model developed for this restricted problem has been extended to allow an arbitrary module intercommunication pattern, but a modified shortest path algorithm that has been developed is guaranteed to yield an optimal assignment only when the graph model exhibits certain identifiable structural properties.[25]

Assignment algorithms, such as the ones mentioned above and the one to be described in the following section, may be used to find a static assignment of modules to processors, but may also be applied repeatedly during the life of a program to reassign modules dynamically as the program's working set changes. (Models have been developed for special cases and an algorithm has been given to handle dynamic reassignment of modules.[4])

THE ASSIGNMENT ALGORITHM

Solutions to scheduling problems, assignment problems, and transportation problems have frequently been sought through iterative techniques. Examples are the simplex method for linear programming problems,[8] the "Hungarian" method[2,16,17] for assignment problems, and the "modified distribution" method[24] for transportation problems. Such techniques begin with an initial solution which is then augmented at each step of the procedure until an optimal feasible solution is obtained.

An iterative procedure is defined here, for the multiprocessor scheduling problem under consideration, that begins with an initial feasible assignment and repeatedly reassigns modules to processors until no further improvement is achievable by continuing the process. This reassignment of modules is accomplished by performing a transformation on the assignment matrix X.

An assignment X is an $m \times n$ matrix such that

$$x_{ij} = 0, 1 \quad \text{for all } i, j \text{ and}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad \text{for all } i.$$

The element $x_{ij} = 1$ if and only if module i is assigned to processor j. The set of all assignments X is called A. The cost of an assignment X is defined to be

$$c(X) = \sum_{i=1}^{m} \sum_{j=1}^{n} e_{ij}x_{ij} + \sum_{i=1}^{m} \sum_{k=i+1}^{m} c_{ik} - \sum_{i=1}^{m} \sum_{j=1}^{n} \sum_{k=i+1}^{m} c_{ik}x_{ij}x_{kj}$$

A transformation is described below that maps the set of all assignments into itself. The procedure determines whether reassignment is advisable and, if so, performs the most advantageous reassignment. The transformation T: $A \rightarrow A$ is defined as follows.

Transformation

1. For each element (i,j) in X, compute a "penalty" which is the cost of executing module i on processor j plus all communication costs for module i, given that all other modules (other than i) are assigned as indicated in matrix X. Thus the penalty matrix P is defined as

$$p_{ij} = e_{ij} + \sum_{k=1}^{m} c_{ik}\,(1 - x_{kj})$$

2. For each row i in X, compute the minimum penalty Θ_i as

$\Theta_i =$ greatest possible improvement in cost achievable in one step on row i; that is, by reassigning module i and leaving all other modules unchanged

$=$ penalty for current assignment of module i minus least penalty for any assignment of module i

$$= \sum_{j=1}^{n} p_{ij}x_{ij} - \min_{1 \leq j \leq n} \{p_{ij}\}$$

and let t be the value of j giving this minimum. Note all $\Theta_i \geq 0$.

3. Select the row that permits the most profitable reassignment by finding

$$\max_{1 \leq i \leq m} \{\Theta_i\}$$

and let s be the value of i giving this maximum.

4. Change the assignment matrix X by setting

$$x_{st} = 1$$

and

$$x_{sj} = 0 \qquad \text{for } j \neq t.$$

The transformation T is applied repeatedly until all $\Theta_i = 0$.

The transformation is illustrated by the following example. Let $m = 3$ and $n = 3$. The matrices E, C, and X represent execution costs, communication costs, and the assignment, respectively, and are initially defined as

$$E = \begin{bmatrix} 4 & 2 & 5 \\ 1 & 8 & 8 \\ 4 & 6 & 3 \end{bmatrix} \qquad C = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{bmatrix} \qquad X = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where X is obtained by assigning each module to the processor for which the execution cost is least (ignoring communication costs). The cost $c(X)$ is 12. In the first iteration, the penalty matrix P is computed as

$$P = \begin{bmatrix} 6 & 5 & 6 \\ 5 & 11 & 9 \\ 6 & 9 & 8 \end{bmatrix}$$

The theta values are

$$\Theta_1 = 5 - 5 = 0, \qquad \Theta_2 = 5 - 5 = 0, \qquad \Theta_3 = 8 - 6 = 2$$

and of these the maximum is Θ_3. Therefore $s = 3$ and $t = 1$ and the third row of X is changed to (1 0 0). The cost $c(X)$ now is 10. In the second iteration,

$$P = \begin{bmatrix} 4 & 5 & 8 \\ 2 & 11 & 12 \\ 6 & 9 & 8 \end{bmatrix}$$

The theta values are

$$\Theta_1 = 5 - 4 = 1, \qquad \Theta_2 = 2 - 2 = 0, \qquad \Theta_3 = 6 - 6 = 0$$

and $\Theta_1 = 1$ is selected as the maximum. Therefore $s = 1$ and $t = 1$ and the first row of X is changed to (1 0 0). The cost $c(X)$ is now 9. In the third iteration,

$$P = \begin{bmatrix} 4 & 5 & 8 \\ 1 & 12 & 12 \\ 4 & 11 & 8 \end{bmatrix}$$

and all Θ_i are zero:

$$\Theta_1 = 4 - 4 = 0, \qquad \Theta_2 = 1 - 1 = 0, \qquad \Theta_3 = 4 - 4 = 0.$$

Therefore the procedure terminates and the assignment matrix X is

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

The cost $c(X) = 9$, which happens to be the optimum cost for this problem.

It is important to ascertain that the iterative procedure described above does not "cycle" indefinitely, thereby generating assignments that have been previously generated. The following theorem states that the iterative procedure does converge, after finitely many applications of the transformation T, to a local optimum.

Theorem. Let A be the set of all feasible assignments for a particular assignment problem. Then the transformation T has a fixed point, that is, $T(X) = X$ for some X in A.
Proof: There are only finitely many assignment matrices X in A (in fact, $K = N^M$ of them, where M is the number of modules and N is the number of processors). The transformation T is monotone with respect to cost, that is,

$$c(X) \geqslant c(T(X)) \text{ for all } X.$$

Therefore $c(X_i) \geqslant c(X_j)$ for $i < j$.

Let X_0 be the starting feasible solution. Then

$$\begin{aligned} X_1 &= T(X_0) \\ X_2 &= T(X_1) \\ &\vdots \\ X_k &= T(X_{k-1}) \text{ and } k \leqslant K-1. \end{aligned}$$

It is always true that $T(X_i) = X_j \notin \{ X_l \mid l = 0,\ldots, i-1 \}$. Since the procedure continues only as long as an improvement can be made in one step, the transformation can be applied only finitely many times.

Optimal assignments are frequently obtained by assigning each module to the least expensive processor and using this as a starting feasible solution, but the following example provides a counter-example to guaranteed optimality.

Let $m = 4$ and $n = 2$. The matrices E, C, and X are initially as follows:

$$E = \begin{bmatrix} 6 & 10 \\ 8 & 6 \\ 8 & 6 \\ 8 & 10 \end{bmatrix} \quad C = \begin{bmatrix} 0 & 10 & 0 & 0 \\ 0 & 0 & 30 & 0 \\ 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

At the first iteration, $s = 4$ and $t = 2$, and row 4 of X becomes $(0 \;\; 1)$. At the second iteration, $s = 1$ and $t = 2$, and row 1 of X becomes $(0 \;\; 1)$. At the third iteration, all $\Theta_i = 0$, therefore the procedure terminates with the assignment

$$X = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

and $c(X) = 32$, whereas the optimal assignment is

$$X = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{bmatrix}$$

for which $c(X) = 30$.

The procedure terminates because no improvement is achievable in *one step*. In this example, the communication cost $c_{23} = 30$ is so high, relative to the difference in execution costs for modules 2 and 3, that no improvement is possible by temporarily assigning modules 2 and 3 to separate processors.

A slight modification may be made to the iterative procedure that would guarantee convergence to a global optimum, but at considerable computational expense. The modification consists of the following extensions. If $\Theta_i = 0$ for all i, then the procedure continues by considering all two step transformations, that is, all simultaneous reassignments of two modules. This requires that for each of the $\binom{M}{2}$ possible two-module reassignments, the penalty matrix P be computed. Values of Θ are then generated, the maximum is selected (unless all Θ are zero), and the appropriate two rows in the assignment matrix X are adjusted. If all Θ values are again zero, then no improvement is possible by reassigning only two modules. The procedure then considers all $\binom{M}{3}$ three-module reassignments, all $\binom{M}{4}$ four-module reassignments, and ultimately the simultaneous reassignment of all M modules. Clearly, generalization of the iterative procedure approaches an exhaustive search that requires a complete enumeration and evaluation of all reassignments in order to achieve a guaranteed global optimum.

Recall that, in the example shown just above, the difficulty arose because the communication costs were high relative to the differences in execution costs. By contrast, an optimal assignment can be achieved easily when communication costs satisfy the following condition

$$\sum_k c_{ik} \leq \min_{\text{all } j \neq l} (e_{ij} - e_{il}) \qquad \text{(c)}$$

for all i. If condition (c) is satisfied then the optimal assignment is that which assigns each module to the processor with least execution time. Intuitively, under condition (c), communication costs are sufficiently small that they can be ignored and the assignment can be made solely on the basis of execution costs (i.e., no significant communication cost penalty is paid for distributing the program modules to the processors best suited for them).

It is worth noting that the standard form for the objective function to be minimized in a quadratic programming problem with continuous decision variables is

$$Q(x) = \sum_{j=1}^{n} b_j x_j + \tfrac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_i x_j$$

When this quadratic function has the property that its quadratic part is non-negative for all x_i and x_j, then $Q(x)$ is a convex function. And in cases where the objective function is convex, a local optimum is a global optimum. In the problem under consideration in this paper, condition (c) guarantees that the quadratic terms in the objective function are negligible and thus provides a discrete approximation to the convexity condition in the general problem. Thus, under condition (c), a local optimum produced by the iterative transformation algorithm is also a global optimum.

A FORTRAN language implementation of the iterative algorithm has been developed for performance testing. Random test data for experimentation were systematically generated for hypothetical networks of 5, 10, 15, and 20 processors, and 5, 10, 15, and 20 program modules. Nine different networks were generated for each problem size (m,n).

Average computation times for the algorithm are reported in Table I. The computation time exhibited on these sample

TABLE I—Computation times

	$n = 5$	$n = 10$	$n = 15$	$n = 20$
$m = 5$	.0107	.023	.034	.043
$m = 10$	.077	.163	.283	.359
$m = 15$	.260	.557	.883	1.203
$m = 20$	.587	1.407	2.114	2.820

M = number of program modules.
N = number of processors.
Computation times in seconds.

test cases is $O(nm^3)$. As expected (based on the discussion just above), the algorithm's performance varied depending on the nature of the network. In networks having high execution costs and relatively low communication costs, the algorithm runs in approximately one-third the time required for networks having relatively high communication costs.

SUMMARY

The scheduling problem considered here has an efficient solution for $n = 2$. It has been shown that the problem is NP-hard for $n \geqslant 4$.[30] This property has not been established for the 3-processor case.

There are n^m possible assignments to be considered in this scheduling problem and, indeed, actual computational experiments using an enumerative algorithm require time that is $O(n^m)$. An optimal search procedure described in Price[26] requires $O(n^m)$ computation time in the worst case, but typically runs in polynomial time.[15, 20, 22] The shortest path and non-backtracking branch-and-bound algorithms[25] and the iterative algorithm (see above) are of low polynomial complexity but generally produce suboptimal solutions.

A variety of techniques have been applied to scheduling problems. For this particular scheduling problem, there remain several interesting alternative approaches, which to date have been explored with only limited success, but which probably deserve further study.

Stone's two-processor network flow approach[27] might be extended by using the multiterminal cut techniques of Gomory and Hu.[13] (The problem of processor load balancing in a two-processor system has also been studied[28] and it, too, may be extendable with multiterminal techniques.)

Spanning trees are of interest in various problems which can be studied through graph models.[9, 18] It may be possible to devise a graph model of a modular computer program in such a way that a minimal spanning tree can be interpreted as an optimal assignment of modules to processors.

This scheduling problem is formulated above as a quadratic programming problem. Perhaps algorithms, such as that of Balas, can be tailored to solve particular problems very efficiently.[1, 19, 32]

Clustering algorithms[33] might be applied to this problem to cluster program modules having high intercommunication costs together on the same processor.

It is clear that reasonable approaches to scheduling problems include (but are not limited to) techniques from the areas of mathematical programming, network analysis, and graph theory.[3, 9, 11, 21] Future practical developments will likely consist of heuristic methods and combinations of algorithms contributed from diverse fields.

ACKNOWLEDGMENTS

The first part of this paper is based on sections of a doctoral dissertation under the direction of Professor Udo W. Pooch at Texas A&M University. The author wishes to thank him, and Professors Don T. Phillips and A. P. Lucido, for their helpful suggestions and constructive review of the material.

REFERENCES

1. Balas, E. An additive algorithm for solving linear programs with zero-one variables. *Oper. Res. 13,* 4 (July-August 1965), 517-546.
2. Balinski, M. L., and Gomory, R. E. A primal method for the assignment and transportation problems. *Management Science 10,* 3 (April 1964), 578-593.
3. Bellman, R. E. *Dynamic Programming,* Princeton U. Press, Princeton, N.J., 1957.
4. Bokhari, S. H. Dual processor scheduling with dynamic reassignment. *IEEE Trans. Software Eng. SE-5,* 4 (July 1979), 341-349.
5. Bokhari, S. H. Multiprocessor scheduling with shortest path algorithms. Tech. Rep. ECE-CS-77-11, Dept. of Elec. and Computer Eng., University of Massachusetts, Amherst, Dec., 1977.
6. Coffman, E. G., Jr. et al. *Computer and Job-Shop Scheduling Theory,* John Wiley and Sons, New York, 1976.
7. Conway, R. W., Maxwell, W. L., and Miller, L. W. *Theory of Scheduling,* Addison-Wesley, Reading, Mass., 1967.
8. Dantzig, G. B. *Linear Programming and Extensions,* Princeton U. Press, Princeton, N.J., 1963.
9. Deo, N. *Graph Theory with Applications to Engineering and Computer Science,* Prentice-Hall, Englewood Cliffs, N.J., 1974.
10. Dijkstra, E. W. A note on two problems in connexion with graphs. *Numer. Math. 1* (1959), 269-271.
11. Ford, L. R., Jr., and Fulkerson, D. R. *Flows in Networks,* Princeton U. Press, Princeton, N.J., 1962.
12. Glover, F., and Woolsey, E. Converting a 0-1 polynomial programming problem to a 0-1 linear program. *Oper. Res. 22,* 1 (Jan.-Feb. 1974), 180-182.
13. Gomory, R. E., and Hu, T.C. Multiterminal network flows. *J. SIAM 9,* (Dec. 1961), 551-570.
14. Greene, W. H., and Pooch, U. W. A review of classification schemes for computer communication networks. *Computer 10,* 11 (Nov. 1977), 12-21.
15. Hart, P. E., Nilsson, N. J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. of Systems Science and Cybernetics, SSC-4,* (July 1968), 100-107.
16. Hu, T. C. *Integer Programming and Network Flows.* Addison-Wesley, Reading, Mass., 1969.
17. Klein, M. A primal method for minimal cost flows with applications to the assignment and transportation problems. *Management Science 14,* 3 (Nov. 1967), 205-220.
18. Kruskal, J. B. Jr. On the shortest spanning subtree of a graph and the travelling salesman problem. *Proc. Am. Math. Soc. 7,* (1956), 48-50.
19. Lawler, E. L. The quadratic assignment problem. *Management Science 9,* 4 (July 1963), 586-599.
20. Martelli, A., and Montanari, U. Optimizing decision trees through heuristically guided search. *Comm. ACM 21,* 12 (Dec. 1978), 1025-1039.
21. McMillan, C. *Mathematical Programming,* Wiley, New York, 1970.
22. Nilsson, N. J. *Problem-Solving Methods in Artificial Intelligence,* McGraw-Hill, New York, 1971.
23. Ousterhout, J. K., Scelza, D. A., and Sindhu, P. S. Medusa: an experiment in distributed operating system structure. *Comm. ACM 23,* 2 (February 1980), 92-105.
24. Phillips, D. T., Ravindran, A., and Solberg, J. J. *Operations Research: Principles and Practice,* John Wiley and Sons, New York, 1976.

25. Price, C. C. A Nonlinear Multiprocessor Scheduling Problem. Ph.D. Th., Texas A&M University, College Station, Texas, May 1979.
26. Price, C. C. Scheduling algorithms for a distributed computer system. University of Texas at Dallas Technical Report No. 65, September, 1979.
27. Rao, G. S., Stone, H. S., and Hu, T. C. Assignment of tasks in a distributed processor system with limited memory. *IEEE Trans. Computers C-28,* 4 (April 1979), 291-299.
28. Stone, H. S. Critical load factors in two-processor distributed systems. *IEEE Trans. Software Eng. SE-4,* 3 (May 1978), 254-258.
29. Stone, H. S. Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Trans. Software Eng. SE-3,* 1 (Jan. 1977), 85-93.
30. Stone, H. S. Private communication, Jan. 1979.
31. Stone, H. S., and Bokhari, S. H. Control of distributed processes. *Computer 11,* 7 (July 1978), 97-106.
32. Taha, H. A. A Balasian-based algorithm for zero-one polynomial programming. *Management Science 18,* 6 (Feb. 1972), B328-B343.
33. Zahn, C. T. Graph-theoretical methods for detecting and describing gestalt clusters. *IEEE Trans. Comp. C-20,* (Jan. 1971), 68-86.