Optimal Process Structuring

Gary Klein
Southern Methodist University

Jay E. Aronson
The University of Georgia

Philip O. Beck
American Airlines

Benn R. Konsynski
Harvard Business School

Follow this and additional works at: https://scholar.smu.edu/business_workingpapers

Part of the Business Commons

This document is brought to you for free and open access by the Cox School of Business at SMU Scholar. It has been accepted for inclusion in Historical Working Papers by an authorized administrator of SMU Scholar. For more information, please visit http://digitalrepository.smu.edu.
OPTIMAL PROCESS STRUCTURING

Working Paper 88-051*

by

Gary Klein
Jay E. Aronson
Philip O. Beck
Benn R. Konsynski

Gary Klein
Assistant Professor of Management Information Sciences
Edwin L. Cox School of Business
Southern Methodist University
Dallas, Texas 75275

Jay E. Aronson
The University of Georgia

Philip O. Beck
American Airlines

Benn R. Konsynski
Harvard Business School

* This paper represents a draft of work in progress by the authors and is being sent to you for information and review. Responsibility for the contents rests solely with the authors and may not be reproduced or distributed without their written consent. Please address all correspondence to Gary Klein.
OPTIMAL PROCESS STRUCTURING

ABSTRACT

Systems of increasing complexity have led to an apparent need for computer aided techniques in software design. It is the ultimate task of software engineering to develop tools and procedures which reduce the effort involved in the production of effective software. In effective software we look for the characteristics of correctness, reliability, efficiency, flexibility, and maintainability. Measures of these characteristics have been proposed in the literature in terms of relationships between modules. These measures have been given the names coupling, cohesion, and transport volume, among others. In this paper, we review a procedure to develop a single measure for inter-module connections based on a combination of the more common measures. The measures utilized are from common attributes found in a complete system dictionary. The availability of the measures yields a metric that can be derived in a completely automated fashion.

The derived metric is used in a mathematical model that describes the total interactions among system or program modules. This mathematical model is in turn used as an integer program to derive a hierarchy chart of the identified processes in a top-down, optimal fashion. The integer program allows for module limits on time and size, precedence relationships, and specification of mutually exclusive and mutually inclusive processes. Since the model is a mathematical program, other constraints may be added giving the method a great deal of flexibility. An efficient algorithm for solving the integer programming problem is presented along with implementation issues. The algorithm is illustrated with an example. Computational results of the code show that the method is viable for problems of realistic size.
1. Introduction

Computer aided design environments are evolving to facilitate the develop­ment of information systems (Shackelton 1982). These approaches, and the use of structured techniques (Duran and McReady 1982), are directed at helping the information system manager control the massive development efforts now under­way. Tools that provide better control over the information system project and aid in the effective development of the system reduce the burden of the manager. The addition of a computer aid for the design process will achieve several benefits (Heninger and Shore 1982). System design effort is lessened resulting in shorter time horizons for the projects and lower costs. Stan­dardization is increased across projects where the automated methodology is employed and documentation can be programmed into the system insuring com­pleteness of the required documentation.

Several design methodologies have been proposed in the literature and applied in actual development. The more common are functional decomposition (Bergland 1981), data flow design (Yourdon and Constantine 1979), data struc­ture design (Warnier 1975 and Jackson 1975), and processing systems optimiza­tion (Nunamaker, et. al. 1974). These methods all use techniques to reduce a system to the representation of a structure chart or hierarchy chart. The hierarchy chart has come to be accepted as an effective representation that helps to minimize complexity (Kottemann and Konsynski 1983).

Design optimization methods based on the processing system optimization approach have been explored by Karimi (1986-87) and Klein, Beck and Konsynski (1988). The approaches both acquire all the required information to design a hierarchy chart for a system from existing documentation and apply mathemati­cal tools in constructing a design. The model proposed by Karimi utilizes standard statistical clustering methodology. This results in a hierarchy chart that is designed from the bottom up using an objective that only
considers the interaction of tasks within a module to a single median. Standard clustering also uses suboptimal heuristics which allow constraint violations. The model proposed by Klein, Beck and Konsynski incorporates measures that allow for a design of the hierarchy based upon the total interaction among tasks within a module. The approach is to optimize successive levels of the hierarchy chart in top-down fashion constrained by physical factors, such as task precedence requirements, that standard clustering methodologies cannot handle.

The mathematical program provided by Klein, Beck and Konsynski is an integer programming cluster analysis model that considers total group interaction. Previously, only a few integer programming models have been applied to clustering (Mulvey and Crowder 1979) because of the complexity of the problem. The need for a more complex model was handled by Klein and Aronson (1988) who developed an algorithm for solving integer programming clustering models. In this paper, we review the development of the mathematical program for automated system design and extend the clustering algorithm to be applicable to the particular problem. Computational results are reported for the new algorithm and a brief example is reviewed.

2. Decomposition and Modularization

2.1 Rationale

An information system is directed at a defined purpose. In order to accomplish the objective, interacting processes must be connected into a comprehensive whole. Each process is a logical unit that transports or manipulates data. Thus, a process can be a simple operation or an entire system. If the scope of the process is too large, the benefit of any analysis is lost. If the scope of the process is too small, the complexity of the system is increased. When a system is decomposed into modules, subsets of the processes are determined that satisfy a criterion such as the minimization of data
transference between modules. The subsets may be arranged hierarchically and may overlap.

Certain observations have been repeated in the literature regarding present practices in modularization. (see for example Jackson 1975, Konsynski 1984, Bergland 1981, and Yourdon and Constantine 1979). These observations include: 1) the number of feasible modularizations is very large and different designs are favorable to different criteria; 2) more than one criteria should be used in the design of the modules and structure/hierarchy chart; 3) the same modularization may result from different methods; and 4) the complexity of large systems suggests the use of automated tools.

2.2 Measures

In general, the descriptions given to desired designs are of a qualitative nature. The words understandability, flexibility, maintainability, testability, reliability, and efficiency are often used to define desirable traits of an information system design. In order to quantify these traits, guideline measures have been proposed. Myers (1975) and Yourdon and Constantine (1979) propose the use of coupling to measure the strength of interconnection between modules and cohesion to measure internal strength. Transport volume is a measure of efficiency that defines the amount of data passed form one module to another.

When these measures, and any other desired measures, are quantified, we can begin to automate the design process through the use of mathematics and/or rules. Attempts have been made to quantify these measures and to arrive at a composite measure for representing the relationships between modules. The development of a metric is now reviewed before the mathematical programming model is presented.
In order to systematize this phase of the design process, a process structure has been developed to organize the activities in the detailed design stage of software life cycle. Information on process attributes and inter-process relations are made available from earlier design stages (Konsynski 1981) or from source libraries and data dictionaries. Graphs are derived representing the network of processes within the system where each node represents a separate transformation on data. Decisions have to be made with regard to grouping of these processes to form separate modules.

In order to represent transport volume, matrices are generated to represent the relationships between processes. The incidence matrix (Nunamaker et al. 1974) shows the relationships of the processes and files.

Let

\[ k \text{ a file number index,} \]
\[ K \text{ number of files} \]
\[ e_{ik} = 1 \text{ if file } k (f_k) \text{ is input to process } i (P_i), \]
\[ e_{ik} = -1 \text{ if } f_k \text{ is output of } P_i, \]
\[ e_{ik} = 0 \text{ if there is no incidence between } f_k \text{ and } P_i. \]

The incidence matrix is useful in the process of finding the total transport volume of data between processes and files within the system. Let

\[ V_k \text{ volume of } f_k, \text{ and} \]
\[ TV_{ij} \text{ transport volume of files between process } i \text{ and } j. \]

Transport volume is determined to be:

\[ TV_{ij} = \sum_{k=1}^{K} V_k \cdot |e_{ik}| \cdot |e_{jk}|. \]

With this representation, the value \( TV_{ij} \) is present for the joint process \( i \) and \( j \) when they share the same files.
Cohesion indicates the degree of association or closeness with respect to data or the functional or logical relationship of the processes within a module. Attempts have been made, (Myer 1975 and Yourdon and Constantine 1979) to assign a relative weight or "cohesion factor" to each level to show the extent of the difference between levels by the cohesion factor rather than simple ranking. The same principle is used here to assign interdependency weights. The weights are chosen in the [0,1] range for normalization and decomposition purposes. The following two relational matrices are developed:

(1) **Precedence matrix** -- the matrix shows if a particular process is a direct precedent to any other process. Namely;

\[ P_{ij} = 1 \text{ if } P_i \text{ is a direct precedent to } P_j, \text{ and } \]
\[ P_{ij} = 0 \text{ otherwise.} \]

(2) **The Matrix of timing relationship** (Marimont 1959) is determined by the earliest time and the latest time of execution of each process. Using the procedure, the matrix \( T \) is defined in the following manner:

\[ T_{ij} = 1 \text{ if process } P_i \text{ is invoked at the same time as process } P_j, \]
\[ \text{and} \]
\[ T_{ij} = 0 \text{ otherwise.} \]

A close look at cohesion and the above matrices, which identify the process relationships, suggests a cohesion weighting scheme \( W_{ij} \) to be used when automating the design. When processing elements having no logical or data relationships are grouped in a poor design just to avoid repeating a segment of code, the resulting module will have coincidental cohesion. Therefore,

\[ \text{if } P_{ij} = 0, \sum_{k=1}^{K} |e_{ik}| \cdot |e_{jk}| = 0 \text{ and } T_{ij} = 0, \text{ then set } W_{ij} = 0. \]
In other words, if two processes have no direct precedence relationships, share no data, and they are not required to be invoked at the same time, then a zero weight would be assigned to the link joining the two processes, indicating coincidental cohesion as a result of their grouping.

When two processes have no data relationship but they are invoked at the same time interval, grouping of them would result in a module with temporal cohesion. A weight of 0.3 is assigned to the link joining them according to:

$$\text{if } \sum_{k=1}^{K} |e_{ik}| \cdot |e_{jk}| = 0, P_{ij} = 0, \text{ and } T_{ij} = 1, \text{ then set } W_{ij} = 0.3.$$ 

Two processes have procedural cohesion if they are activated by the same process, but they do not necessarily use the same data set as input:

$$\text{if } P_{ij} = 1 \text{ and } \sum_{k=1}^{K} |e_{ik}| \cdot |e_{jk}| = 0, \text{ then set } W_{ij} = 0.5.$$ 

Communicational cohesion results when the processing elements within the modules use the same input data sets or they produce the same output data sets. Therefore:

$$\text{if } P_{ij} = 0 \text{ and } \sum_{k=1}^{K} |e_{ik}| \cdot |e_{jk}| = 1, \text{ then set } W_{ij} = 0.7.$$ 

Sequential cohesion between processes is easily recognizable from the data flow graph and related matrices. In terms of a data flow graph, sequential association results from a linear chain of successive transformation of data. Since sequential cohesion produces fewer intermodule communications:

$$\text{if } P_{ij} = 1 \text{ and } \sum_{k=1}^{K} |e_{ik}| \cdot |e_{jk}| = 1, \text{ then set } W_{ij} = 0.9.$$ 

Two processes may have a functional relationship by being part of a single operation. For example, processing elements which perform the edit function...
within a module exhibit **functional cohesion**. In such cases, the designer would be asked to identify the processes. A weight of (1.0) is assigned to the link joining them.

A measure for potential coupling may be based upon the percentage of data sets used entirely within groupings of modules. Specific levels of coupling are determined by the code generated, but a measure to consider the potential for poor coupling may be derived from the percentage of data items shared by two modules, or:

\[
C_{ij} = \frac{\text{number of shared data items for } i \text{ and } j}{\text{number of total data items}}
\]

This information would be available from a comprehensive system dictionary. As \(C_{ij}\) decreases, the desire is to group process \(i\) with process \(j\) to avoid coupling problems.

These and other desired complex criteria, may be used to derive a single measure for evaluating a design. This can be accomplished with the following:

\[(1) \quad \rho_{ij} = w_1 T V_{ij} + w_2 W_{ij} + w_3 C_{ij}\]

where the \(w_i\)'s represent importance weights on each of the criterion. The significance is that we can arrive at a measure or figure of merit for an organization of processes into program modules. Thus, by quantifying the influence of the selected criteria on each property we have a scoring procedure to measure the success of each design.

The selection of the weights and the scaling of the individual criterion would be critical to the success of the proposed metric. Scaling should first be conducted to provide a single scale for each criterion. Approaches utilizing a strict 0 to 1 scale for each criterion, where 0 represents the most preferred value and 1 the least preferred, are common (Keeney and Raiffa
Informal or formal weighting methods can then be applied to finalize the metric.

2.3 Optimum Decomposition

The single metric defined in (1) must then be used to decompose the process graph into a hierarchy chart. Treating the metric as a composite measure, however, an objective function can be written as:

\[
\text{(2) } \text{MIN } \sum_i \sum_j \rho_{ij}
\]

where the \( \rho_{ij} \) are weights that represent conflict between modules. This objective function now strives to group modules that are closely related and separate modules performing different functions. Thus we more accurately reflect the goals of decomposition and are able to formulate a single mixed integer program that may be used to decompose the totality of processes in a system into successive groupings of modules on a hierarchy chart. Subsequent applications of the integer program would create each lower level on the hierarchy.

The advantage of having a mixed integer program include 1) the ability to use standard solution methods, 2) the ability to add precedence constraints that enforce any sequential module requirements, 3) top-down decomposition through successive application, 4) the ability to place restrictions on the number of modules in each group, and 5) the ability to set the number of groups. Let

- \( X_{ij} \) be a zero-one variable representing whether process \( i \) is in module \( \ell \) (=1) or not (=0),
- \( Y_{ij} \) be a zero-one variable representing if process \( i \) and process \( j \) are in the same module (=1) or not (=0),
- \( L \) be the number of desired modules,
be the number of processes,
be an upper bound on the number of processes in module \( l \),
be a time (or memory, etc.) unit (weight) for process \( i \), and
be an upper limit on the resource defined by \( t_i \) for module \( l \).

The mixed integer program (MIP) may be stated as:

\[
\text{(3)} \quad \text{MIN} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} [B_{ij}Y_{ij}]
\]

subject to:

\[
\text{(4)} \quad Y_{ij} \geq x_{i\ell} + x_{j\ell} - 1 \quad \text{for each } i,j \text{ pair in each module } \ell,
\]

\[
\text{(5)} \quad \sum_{\ell=1}^{L} x_{i\ell} \leq \sum_{\ell=1}^{L} x_{j\ell} \quad \text{for each } i,j \text{ precedence, where } i \text{ precedes } j,
\]

\[
\text{(6)} \quad \sum_{\ell=1}^{L} x_{i\ell} = 1 \quad \text{for each process } i,
\]

\[
\text{(7)} \quad \sum_{i=1}^{n} x_{i\ell} \leq b_{\ell} \quad \text{for each module } \ell,
\]

\[
\text{(8)} \quad x_{i\ell} + x_{j\ell} \leq 1 \quad \text{for each } i,j \text{ of mutual exclusion},
\]

\[
\text{(9)} \quad \sum_{i=1}^{n} t_i x_{i\ell} \leq C_{\ell} \quad \text{for each module } \ell,
\]

\[
\text{(10)} \quad x_{i\ell} = 0,1 \quad \text{for } i=1, \ldots, n; \ell=1, \ldots, L, \text{ and }
\]

Constraint (4) relates group membership to the objective function. The integrality restriction on \( Y_{ij} \) may be dropped, because each \( Y_{ij} \) will be driven to zero by the objective function unless forced to be one by constraint set (4). This greatly reduces the number of integer variables. Constraint (5) represents the precedence relations and (6) ensures that each process is assigned
to one and only one module. Constraint (7) allows an upper bound be placed on module membership and (8) is used if two processes must not appear in the same module. Constraint (9) allows for a limit on module capacity (time, memory, etc.) and (10) enforces the 0,1 integrality.

3. Optimization

3.1 The Algorithm

Our algorithm is an extension of Balas' Implicit Enumeration (see Balas (1965)) to handle multiple branches from each node in the tree. Though not required, a depth-first search strategy is used. The depth of the tree represents the process under consideration; the branches represents module membership, as shown in the example problem of Figure 1. The algorithm assumes positive distances for the objective function and no side constraints for the initial presentation, though these restrictions may be dropped with modifications to the algorithm. We need the following additional notation to state the algorithm.

Additional Notation:

\[ p \quad \text{- a pointer to the current depth in the tree,} \]
\[ m \quad \text{- module membership indicator (from 1 to K),} \]
\[ B_p \quad \text{- Bound on unassigned processes at depth } p, \text{ (Its computation is presented in the next section. A value of zero is assumed for the initial presentation of the algorithm),} \]
\[ n(m) \quad \text{- the number of processes in module } m, \]
\[ q \quad \text{- the number of modules with no membership,} \]
\[ Z \quad \text{- the value of the objective function of the incumbent, and} \]
\[ X_{IL} \quad \text{- the incumbent values of all } X_{iL}. \]

The algorithm consists of the following, detailed steps:

**STEP 0:** (Initialize.) \( p = 0. \) \( n(m) = 0 \text{ for } m = 1 \text{ to } L. \) \( q = L. \) \( Z = -. \)

Go to **STEP 1.**
STEP 1: (Increment search depth.) \( p = p + 1 \), \( m = 1 \), \( n(m) = n(m) + 1 \). If \( n(m) = 1 \) then \( q = q - 1 \). Go to STEP 2.

STEP 2: (Feasibility.) If \( (n-p) < q \) then go to STEP 5, else go to STEP 3.

STEP 3: (Suboptimality.) If \( Z \leq \sum_{i=j}^{p-1} \sum_{j=i+1}^{p} d_{ij}y_{ij} + b_{ij} \) and go to STEP 5, else go to STEP 4.

STEP 4: (Update Incumbent.) If \( p = n \) then \( X_{IL} = X_{ig} \) for all \( i, l \) and \( \sum_{i=j}^{p-1} \sum_{j=i+1}^{p} d_{ij}y_{ij} \), and go to STEP 5, else go to STEP 1.

STEP 5: (Determine Action After Fathom.) If \( m = L \) or \( (n(m) = 1 \) and \( n(m+1) = 0 \) then go to STEP 7, else go to STEP 6.

STEP 6: (Fathom: Branch Right on Module.) \( n(m) = n(m) - 1 \), \( m = m + 1 \), \( n(m) = n(m) + 1 \). If \( n(m) = 1 \) then \( q = q - 1 \). Go to STEP 2.

STEP 7: (Fathom: Depth Retraction.) \( n(m) = n(m) - 1 \) and \( p = p - 1 \). If \( n(m) = 0 \) the \( q = q + 1 \). If \( P > 0 \) then go to STEP 5, else STOP.

3.2 Implementation Issues

The convergence properties of the algorithm are of great concern because the algorithm could evaluate the maximum number of solution nodes in the tree, \( \sum_{i=0}^{n} L_i \). Convergence is dramatically improved through the obtainment of tight bounds and the use of a heuristic procedure to obtain an initial solution.

The bounding routines used for the basic problem involve the use of upper and lower bounds on the module assignment based on the precedence relations and limitations. These bounds were introduced by Patterson and Albracht (1975). Intuitively, if a process is involved in a precedence structure then there are a certain number of processes that must be completed prior to the beginning of the process and a certain set of processes that must be accomplished after the process. In Figure 1, process 2 must be preceded by process 1. Assume that this required preprocess takes a total of 7 time units and process 2 takes 3 units. Thus, if a desired time limit in each module is 9 time units, process 2 cannot possibly be placed before module 2. A similar argument holds for the
upper bound on the module assignment by considering the time of the processes that must follow a particular process. Using the module membership restrictions, we can describe a similar type of limitation on the earliest (leftmost) and latest (rightmost) modules into which a process may be placed. The formal definition of these bounds may be found in Patterson and Albracht (1975).

Lower bounds on the interactions of the remaining processes to be classified, $B_p$, are computed in Step 3 and added to the computed objective function value at each iteration. These bounds are composed of a portion for the interactions of the unassigned processes to the assigned processes and a portion associated strictly within the unassigned processes. The bound on the unassigned processes to the assigned processes is computed in a straightforward manner. Each unassigned process is tested in turn for assignment to each module. The minimum interactions across all modules are used for the lower bound. These are then summed over all unassigned processes. The bound on the interactions within the unassigned processes is more involved. First the minimum number of interactions at a particular level in the tree are computed using the notion that a balanced tree provides the fewest number of interactions. Thus, the minimum number of interactions is computed assuming that, in the worst bounding case, the processes are as evenly distributed as possible. The number of interactions within each module of unassigned processes is the sum of the integers less one process. These are summed across all modules to give the remaining minimum number of interactions, $T_p$. Finally, sum the $T_p$ lowest distance entries in the bottom right hand part of the metric matrix of size $r \times r$, where $r=n-p$ (the number of unassigned processes). The determination of this tight bound is a relatively fast computational step that directly fathoms a significant portion of the branch and bound tree. More details of this bound can be found in Klein and Aronson (1988).
Feasibility tests are made in Step 2. Module membership constraints (7) are tested by verifying that the module to which the addition of an additional process is being attempted has enough capacity to accept it. Weight capacity constraint (9) tests are made by comparing the value of the weight in the current module to the specified limit for the module. Precedence constraints (4) are checked by verifying that the assignment of process p to module m does not violate any precedence relations. Any violation leads to fathoming the current trial solution.

Rather than starting the implicit enumeration algorithm with no incumbent, i.e., having an infinite objective value, it became clear that the development and implementation of a fast heuristic methodology was in order to speed up convergence of the implicit enumeration algorithm. Because the network structure of (3) through (10) is essentially that of an assembly line balancing problem with a different objective function, an initial incumbent is found by a variation of heuristics used for solving line balancing problems. Refer to Aronson and Klein (1987) for details.

4.0 Example

A small example will be solved to illustrate the algorithm. In Figure 1, we present the distance matrix and branch and bound tree for a problem with \( n=3 \) and \( L=2 \). Exterior to each solution node are the variable assignments that represent the node and the value of its objective (underlined). The values at the interior of each node represent: the node number; the value of the depth pointer p; and the module membership of process p in the node.

**STEP 0:** The values of the algorithm are initialized as represented by node 0. The incumbent solution may be initialized by using a known clustering
heuristic (see Zupan 1982), or by the heuristic in Aronson and Klein (1987) if desired.

**STEP 1:** Step 1 increments the depth of the search. A depth first search strategy is used. Initially, process 1 is placed in module 1 at node 1 in the tree. The counter for the number of processes in the module is incremented.

**STEP 2:** Feasibility tests are conducted here. If any branch proves to be infeasible it is fathomed and the algorithm proceeds to Step 5. If \((n-p)\) is less than the number of empty modules, the branch will be suboptimal since all costs are positive. This test fails on the first pass.

**STEP 3:** Compute the objective function value at the current depth. If this is greater than or equal to the current incumbent value, the node of the branch can only be suboptimal. In the first pass the cost is zero because no two processes are in the same module. The lower bound on the remainder of the processes is determined by computing the minimum number of interactions among the remaining processes and selecting the lowest values in the distance matrix as discussed in the subsection on bounds. The bound is then added to the value of the objective at the current node and used in the determination of suboptimality. Suboptimality is not found in the first pass.

**STEP 4:** Step 4 determines that all processes have not been assigned to modules and therefore an incumbent has not been found on the first pass. Control passes back to Step 1.

The algorithm makes one more pass to reach node 3. Another pass is started by adding the third process to the first module in node 3. This pass is aborted in Step 2 when it is determined that node 3 is
unreasonable because at least one module will be empty. Thus control is passed to Step 5 for the first time.

**STEP 5:** Node 3 is fathomed. This can be done either by moving up a level or by branching to the right to try the current process in the next module. Two conditions can indicate a depth retraction: 1) the current process is already in the final module; or 2) there is only one entry in the current module and no entries in the next module, i.e., to the right. It is determined at node 3 that the process must still be tested in the second module. This is done by passing control to Step 6.

**STEP 6:** The pointer for module membership for the current process, 3, is incremented. The appropriate counters are maintained. This is now represented by node 4 in that tree. Control returns to Step 2 for further processing.

Node 4 proves feasible in Step 2, is not suboptimal in Step 3, and as a result, the incumbent is updated in Step 4 before being fathomed. The algorithm proceeds in this fashion until all nodes are explicitly or implicitly enumerated. The optimal solution is preserved as the final incumbent. The values shown in the tree provide enough data to follow the rest of the algorithm to completion. The resulting hierarchy chart would be as shown in Figure 2.

5.0 *Computational Experience*

The clustering algorithm was implemented in the FORTRAN code GROUPS2, on the Southern Methodist University IBM 3081-D24. The FORTVS2 compiler at optimization level 3 was used. Four problems with several variations were solved. The variations, for different maximum group sizes, include: 1) the problem with only precedence constraints (3-6, 10); and 2) the problem with precedence
1.64%. Problems having both precedence and group constraints are much more tightly constrained than those not having such constraints. Therefore, many solution nodes are fathomed based on feasibility.

The last two problem sets are more representative of large-scale cluster analysis problems. Currently there are no real-world CAPO problems of this size available in the literature. However, this may be due to the fact that until now it was not possible to solve such problems to optimality. As can be seen in Table 1, the larger problems show the same characteristic trends as the smaller ones.

6. Conclusions

A computer aided system and program design method is proposed that helps speed the design process and reduce the need for resources in the design phase of a new system’s development. The objectives of the method are to develop a top down modular system design based on accepted measures as coupling, cohesion, and transport volume. Once a measure is determined, the method decomposes a graph of directed procedures into a hierarchy chart through the successive application of a mixed integer programming model.

An efficient, implicit enumeration algorithm for optimizing the clustering criteria along with specialized bounding techniques is developed. The development and implementation of specialized bounding rules derived from the precedence and module capacity constraints dramatically decrease the number of subproblem nodes that need to be evaluated by the branch and bound algorithm. This method obtains better solutions to the design problems than applied clustering heuristics with the constraints relaxed.
References


Table 1

Representative Computational Results of Testing GROUPS2

<table>
<thead>
<tr>
<th>Code Number</th>
<th>Code</th>
<th>OPT./ HEUR.</th>
<th>CPU TIME</th>
<th>TOTAL EVALUATED</th>
<th>MAXIMUM NUMBER</th>
<th>PERCENT EVALUATED</th>
</tr>
</thead>
<tbody>
<tr>
<td>K13_2P</td>
<td>98.15%</td>
<td>0.01</td>
<td>269</td>
<td>16383</td>
<td>1.64%</td>
<td></td>
</tr>
<tr>
<td>K13_3P</td>
<td>86.34%</td>
<td>0.14</td>
<td>4279</td>
<td>2391484</td>
<td>0.18%</td>
<td></td>
</tr>
<tr>
<td>K13_4P</td>
<td>63.86%</td>
<td>1.13</td>
<td>35789</td>
<td>89478485</td>
<td>0.04%</td>
<td></td>
</tr>
<tr>
<td>K13_5P</td>
<td>49.06%</td>
<td>7.14</td>
<td>234047</td>
<td>1.53E+09</td>
<td>0.02%</td>
<td></td>
</tr>
<tr>
<td>K13_6P</td>
<td>33.96%</td>
<td>21.89</td>
<td>657950</td>
<td>1.57E+10</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>K13_7P</td>
<td>25.79%</td>
<td>74.45</td>
<td>2278215</td>
<td>1.13E+11</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>K13_8P</td>
<td>20.75%</td>
<td>222.30</td>
<td>6994178</td>
<td>6.28E+11</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>K13_2PG</td>
<td>98.51%</td>
<td>0.01</td>
<td>103</td>
<td>16383</td>
<td>0.63%</td>
<td></td>
</tr>
<tr>
<td>K13_3PG</td>
<td>92.94%</td>
<td>0.03</td>
<td>866</td>
<td>2391484</td>
<td>0.04%</td>
<td></td>
</tr>
<tr>
<td>K13_4PG</td>
<td>98.15%</td>
<td>0.37</td>
<td>11302</td>
<td>89478485</td>
<td>0.01%</td>
<td></td>
</tr>
<tr>
<td>K13_5PG</td>
<td>96.30%</td>
<td>0.59</td>
<td>19463</td>
<td>1.53E+09</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>K13_6PG</td>
<td>87.10%</td>
<td>1.36</td>
<td>43651</td>
<td>1.57E+10</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>K13_7PG</td>
<td>100.00%</td>
<td>10.44</td>
<td>333226</td>
<td>1.13E+11</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>K13_8PG</td>
<td>94.29%</td>
<td>58.78</td>
<td>1874763</td>
<td>6.28E+11</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>G23_3P</td>
<td>77.56%</td>
<td>0.75</td>
<td>10267</td>
<td>2391484</td>
<td>0.43%</td>
<td></td>
</tr>
<tr>
<td>G23_4P</td>
<td>48.21%</td>
<td>4.58</td>
<td>65529</td>
<td>89478485</td>
<td>0.07%</td>
<td></td>
</tr>
<tr>
<td>G23_5P</td>
<td>36.16%</td>
<td>36.95</td>
<td>560999</td>
<td>1.53E+09</td>
<td>0.04%</td>
<td></td>
</tr>
<tr>
<td>G23_6P</td>
<td>27.23%</td>
<td>214.59</td>
<td>3303572</td>
<td>1.57E+10</td>
<td>0.02%</td>
<td></td>
</tr>
<tr>
<td>G23_3PG</td>
<td>72.77%</td>
<td>0.96</td>
<td>13608</td>
<td>89478485</td>
<td>0.02%</td>
<td></td>
</tr>
<tr>
<td>G23_4PG</td>
<td>59.86%</td>
<td>12.29</td>
<td>166784</td>
<td>1.53E+09</td>
<td>0.01%</td>
<td></td>
</tr>
<tr>
<td>G23_5PG</td>
<td>54.80%</td>
<td>26.79</td>
<td>481053</td>
<td>1.57E+10</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>G23_6PG</td>
<td>54.80%</td>
<td>55.67</td>
<td>1060935</td>
<td>1.13E+11</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>T70_3PG</td>
<td>56.37%</td>
<td>164.34</td>
<td>227073</td>
<td>3.75E+33</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>T70_5PG</td>
<td>7.42%</td>
<td>7107.30</td>
<td>37762883</td>
<td>1.66E+49</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>T70-30PG</td>
<td>96.01%</td>
<td>5411.32</td>
<td>28147712</td>
<td>**</td>
<td>0.00%</td>
<td></td>
</tr>
<tr>
<td>D140-3PG</td>
<td>99.97%</td>
<td>633.21</td>
<td>3281917</td>
<td>9.40E+66</td>
<td>0.00%</td>
<td></td>
</tr>
</tbody>
</table>

* No Feasible Solution Found by the Heuristic.
** Beyond the Precision of the IBM 3081-D24.
Figure 2. The resulting hierarchy chart for the example.
Figure 1: Example Problem with Precedence Solved by the Branch and Bound Algorithm. In each node are the node number, its depth in the tree (= process number), and the module membership assignment. Outside each node are the variable assignment and objective value. Large nodes are explicitly enumerated. Small nodes are implicitly enumerated. The optimum is found at node 6 (indicated by a *) with three processes grouped as 1 in 1, 2 in 2, and 3 in 1. There are no module membership or capacity limits.
NOTE: The following is a partial list of papers that are currently available in the Edwin L. Cox School of Business Working Paper Series. When requesting a paper, please include the Working Paper number as well as the title and author(s), and enclose payment of $2.50 per copy made payable to SMU. A complete list is available upon request from:

Director, Research and Development
Edwin L. Cox School of Business
Southern Methodist University
Dallas, Texas 75275
<table>
<thead>
<tr>
<th>Volume</th>
<th>Title</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>86-084</td>
<td>&quot;A Simple Algebraic Estimation Procedure for Innovation Diffusion Models of New Product Acceptance,&quot; by Vijay Mahajan and Subhash Sharma</td>
<td></td>
</tr>
<tr>
<td>86-091</td>
<td>&quot;Some Probabilities Associated with the Ordering of Unknown Multinomial Cell Probabilities,&quot; by S. Y. Dennis</td>
<td></td>
</tr>
<tr>
<td>86-094</td>
<td>&quot;Pricing and Diffusion of Primary and Contingent Products,&quot; by Vijay Mahajan and Eitan Muller</td>
<td></td>
</tr>
<tr>
<td>86-101</td>
<td>&quot;Determination of Adopter Categories Using Innovation Diffusion Models,&quot; by Vijay Mahajan and Eitan Muller</td>
<td></td>
</tr>
<tr>
<td>86-111</td>
<td>&quot;A Probabilistic Analysis of the Eigenvector Problem for Dominance Matrices of Unit Rank,&quot; by S. Y. Dennis</td>
<td></td>
</tr>
<tr>
<td>86-121</td>
<td>&quot;Optimal Clustering: A Model and Method,&quot; by Gary Klein and Jay E. Aronson</td>
<td></td>
</tr>
<tr>
<td>86-123</td>
<td>&quot;Question Effects on Information Processing: An Alternative Paradigm,&quot; by Daniel J. Howard</td>
<td></td>
</tr>
<tr>
<td>87-011</td>
<td>&quot;Criteria for Selecting Joint Venture Partners,&quot; by J. Michael Geringer</td>
<td></td>
</tr>
<tr>
<td>87-012</td>
<td>&quot;A Model of the Joint Venture Partner Selection Process,&quot; by J. Michael Geringer</td>
<td></td>
</tr>
<tr>
<td>87-013</td>
<td>&quot;Selection of Partners for International Joint Ventures,&quot; by J. Michael Geringer</td>
<td></td>
</tr>
<tr>
<td>87-021</td>
<td>&quot;Metamorphosis in Strategic Market Planning,&quot; by Vijay Mahajan, P. &quot;Rajan&quot; Varadarajan, and Roger A. Kerin</td>
<td></td>
</tr>
<tr>
<td>87-022</td>
<td>&quot;CEO Roles Across Cultures,&quot; by Ellen Jackofsky and John W. Slocum, Jr.</td>
<td></td>
</tr>
<tr>
<td>87-023</td>
<td>&quot;Strategy Formulation Processes: Differences in Perceptions of Strength and Weaknesses Indicators and Environmental Uncertainty by Managerial Level,&quot; by R. Duane Ireland, Michael A. Hitt, Richard A. Bettis, and Deborah Auld De Porras</td>
<td></td>
</tr>
</tbody>
</table>
"Financial Returns and Strategic Interaction: The Case of Instant Photography," by Richard A. Bettis and David Weeks

"Interactive Multiobjective Optimization Under Uncertainty," by G. Klein, H. Moskowitz, and A. Ravindran


"Stages in the Evolution of Managerial Interpretation: A Study of Interpreting Key Organizational Events," by Lynn A. Isabella

"An Agent for Intelligent Model Management," by John I. C. Liu, David Y. Y. Yun, and Gary Klein


"Home Ownership Rates of Married Couples: An Econometric Investigation," by Donald R. Haurin, Patrie H. Hendershott, David C. Ling

"A Longitudinal Study of Climates," by Ellen F. Jackofsky and John W. Slocum, Jr.

"Hidden Messages in Corporate Relocation," by Lynn A. Isabella and Suzyn Ornstein

"Cultural Values and the CEO: Alluring Companions?" by Ellen F. Jackofsky, John W. Slocum, Jr., and Sara J. McQuaid
