INTERACTIVE DECISION SOFTWARE AND COMPUTER GRAPHICS FOR ARCHITECTURAL AND SPACE PLANNING

F. GLOVER and C. McMILLAN

Graduate School of Business Administration, University of Colorado, Campus Box 419, Boulder, Colorado 80309, USA

and

B. NOVICK

GSIA, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213, USA

Abstract

We describe the development and successful implementation of a decision support system now being used by several leading firms in the architecture and space planning industries. The system, which we call SPDS (*s*patial *p*rogramming *d*esign *s*ystem) has the following characteristics: (i) user-friendly convenience features permitting architects and space planners to operate the system without being experienced programmers; (ii) interactive capabilities allowing the user to control and to manipulate relevant parameters, orchestrating conditions to which his or her intuition provides valuable input; (iii) informative and understandable graphics, providing visual displays of interconnections that the computer itself treats in a more abstract methematical form; (iv) convenient ways to change configurations, and to carry out 'what if' analyses calling on the system's decision support capabilities; (v) a collection of new methods, invisible to the user, capable of generating good solutions to the mathematical programming problems that underlie each major design component. These new methods succeed in generating high quality solutions to a collection of complex discrete, highly nonlinear problems. While these problems could only be solved in hours, or not at all, with previously existing software, the new methods obtain answers in seconds to minutes on a minicomputer. Major users, including Dalton, Dalton, Newport, and Marshal Erdwin, report numerous advantages of the system over traditional architectural design methods.

Keywords and phrases

Modeling, micro-computers, integer programming, nonlinear programming, heuristics.

1. Introduction

This paper describes the development and successful implementation of a decision support system being used by several leading firms in the architecture and space planning industries. The system, which we call SPDS (*s*patial *p*rogramming *d*esign *s*ystem) helps to solve architectural and space planning problems that arise in the allocation of activities and functions in layout plans (i.e. rooms, departments, etc.). SPDS seeks to satisfy as closely as possible desired proximity, adjacency and separation relationships, providing a tool that usefully augments the design capabilities of an architect.

2. The problem

PROVIDING AN ARCHITECTURAL DESIGN TOOL

In order to appreciate the motivation for the SPDS system, it is important to understand the process used by an architect or space planner to design a building, a set of buildings, or part of an existing building. Over the years architecture, as a discipline, has grown to see this as the *design process*, and to recognize certain methods and certain sequences of methods as preferred over others.

From the intial site analysis for a building, through what are known as *preliminary design* and *design development*, to the final drafting of *working drawings*, the use of the computer is increasingly appropriate. A successful support system must be a convenient tool, respectfully enhancing the techniques and thought processes that have evolved among architects over centuries.

SPDS addresses the process defined above as *preliminary design*. For our purposes, preliminary design consists of determining the basic shape and orientation of the building and of allocating space for each room within the overall building layout.

Prior to preliminary design, the architect must define the constraints and requirements that his solution must ultimately satisfy by researching specific needs and desires of the client. In the design of a medical center, for example, an architectural team might meet with members of the hospital planning department to formulate a list of necessary rooms or *activity spaces* such as operating rooms, nurses quarters, etc. Square footage and proportional requirements would also be specified, together with a large set of pairwise relationships between activity spaces.

The pairwise relationships indicate, by means of numbers, the relative strangths of the desired proximity or adjacency of two activity spaces. For example, similar functions should be in closer proximity than dissimilar functions. Some relationships are so strong as to be virtually compulsory. For example, a scrub room must be directly adjacent to each operating room. There may also be desired adjacencies of certain rooms to specific exterior walls (e.g. the south side for sunlight).

There are also typically other spatial and positional considerations. The operating room may need to be 400 sq. ft. and approximately *square,* to assure proper housing of medical equipment. And usually a client will request that certain intangible requirements be met, such as the creation of a corporate image in an office lobby, or the establishment of a happy, reassuring feeling for the patient in the pre-op area. It is the purpose of SPDS to allow the user to easily input quantifiable constraints and requirements, to add to and to change this input, and to monitor satisfaction according to aesthetic, subjective criteria.

Once sufficient information has been gathered to define the *design problem,* it is handed to one or more architects who must then 'solve it', i.e. arrive at a satisfactory preliminary design. Clearly, the interacting requirements and the quantity of underlying data may often be too great to handle effectively without some means of identifying and integrating major components. Traditionally, architects have undertaken to do this by mentally dividing the problem into meaningful subsets. A designer may well know that various rooms in the surgical suite must all be relatively close together. He will investigate possible configurations for these specific rooms, and for other such subsets, later combining the solutions in the final plan.

SPDS, in its interactive solution of spatial allocation problems, seeks to extend the power and effectiveness of the 'subset creation' process by providing a decision support capability to help the designer divide and sort information in various meaningful ways – helping him to investigate many possibilities with the aid of informative graphic illustrations.

In the process of organizing and sorting information, in designing individual subsets and in combining them, the designer arrives at what architects term a *concept.* A concept for a medical center might be the means for handling the sterile areas for emergency and for surgery as two geometrically separate but equal configurations, tied perhaps by a planted atrium used for circulation and lighting. Such a concept would physically express the idea of those two areas being separate but equal, in function. A concept for a theatre may be that ancillary functions should be expressed as simple rectilinear shapes, while the theatre itself may take the form of a dramatic, visually exciting entity. A good designer will investigate several alternative concepts for a project.

3. The bubble diagram

A key tool in the investigation of concepts, which is a major system component in SPDS, is the *bubble diagram.* To understand this tool, consider that as a designer begins to map out rough spatial configurations for a hospital, he may see that it would work well to have the nurses' station centrally located in the sterile area of the emergency suite, and that the emergency entrance should be off the nurses' station, as should the public waiting area. The designer would not yet know, however, the exact configuration of the final layout.

The bubble diagram, as routinely used by architects, assists the rough spatial investigation process by representing rooms and activities as amorphous shapes – shapes whose respective proximity relationships are important, but whose exact location and relative configuration are not. In a challenging application of decision science techniques, SPDS utilizes and solves a nonlinear programming formulation of the problem of creating a bubble diagram. By further special processing of the solution, SPDS seeks to provide rapid, editable visual displays of diagrams similar in purpose and psychological impact to the traditional bubble diagram.

Finally, the designer begins to solidify his rough spatial concepts into a specific architectural solution. SPDS provides the user convenient interactive graphic tools to perform this step. The system provides the user convenient methods to change layout configurations and to carry out 'what if' analyses, allowing him to investigate various possibilities and helping to prevent him from becoming locked into a particular spatial arrangement while overlooking good alternatives.

In all, the SPDS system is a highly ambitious effort to integrate the best of management science formulations and solution techniques with user-friendly tools to analyze, interpret and modify the results. The use of graphics and visual displays at each step is absolutely essential due to the need to make the results useful and comprehensible for architectural designers. We believe SPDS is the first system of its kind to unify all these elements for architectural and space planning. (For surveys of prior attempts to handle restricted segments of space planning and layout design, see [4,5, 10].)

4. Specific formulations and methodologies

To provide the various capabilities discussed above, the part of SPDS that is invisible to the user had to be organized to solve three major types of problems. As a preliminary to characterizing and solving these problems, the system must accept inputs representing judgements of skilled planners as being the 'proximity strength' of pairs of activities, i.e. indicating the relative desired closeness or separation of these activities. Utilizing these initial inputs, it must then:

(1) group activities to optimize a function of the assigned proximity strengths, subject to targeted cardinality restrictions (a mixed integer goal programming problem);

(2) translate the proximity strengths within groups into spatial coordinates that agree as nearly as possible, within a scale factor, to the specified strengths (a nonlinear optimization problem);

(3) treat the spatial distance between point coordinates as desired distances between 'centers of gravity' for rooms of specified dimensions and shape restrictions, and locate rooms on a grid to meet the center-of-gravity distance relationships as nearly as possible (a combinatorial optimization problem).

These three problems are much too complex for a human to handle well. Moreover, they are beyond the state-of-the-art of mathematical optimization to achieve optimality without consuming many hours of computation time on today's most powerful computers. We faced the additional imperative of producing a system that was user-friendly in all the best senses, allowing interactive options for architects to utilize their own intuitive skills and to conduct 'what if' analyses. We required the development of a solution approach that would deliver useable answers within minutes on a minicomputer.

5. Overview of the system

We first describe the operation of the SPDS system from the standpoint of the user, and subsequently we will comment on the more technical, underlying algorithmic considerations.

A four-stage process for SPDS is used:

(1) The first involves the initial input of the results of the programming process. The user is prompted for the name of an activity, corresponding desired square footage, proportions, and proximity requirements. Six weights are available. SPDS produces a Relationship Chart as illustrated in fig. 1. The various shadings to

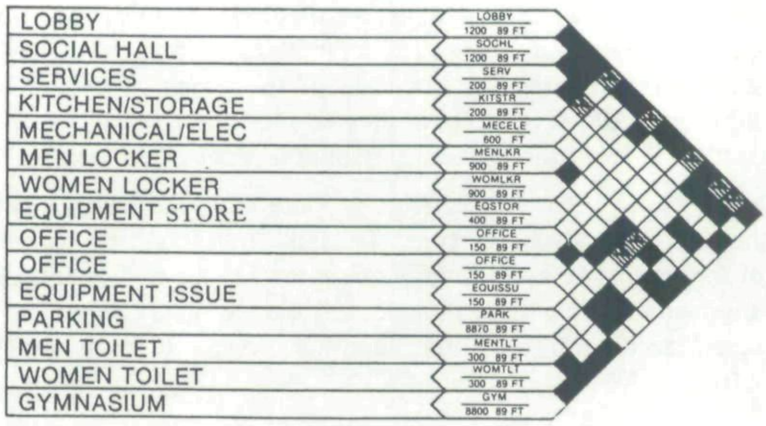

Fig. 1. Relationship chart.

the right represent relative desirability of proximity, ranging from absolutely necessary to unimportant. Many architects produce such charts manually for both design and presentation purposes. SPDS gives the user the option to edit or make additions to the chart at any stage. For a more complete description of this and other charts commonly used in architectural design, see Mather [8].

(2) A subset creation routine allows the user to break down the design problem into tightly related subsets of linked activities as in fig. 2. The set of all activities is

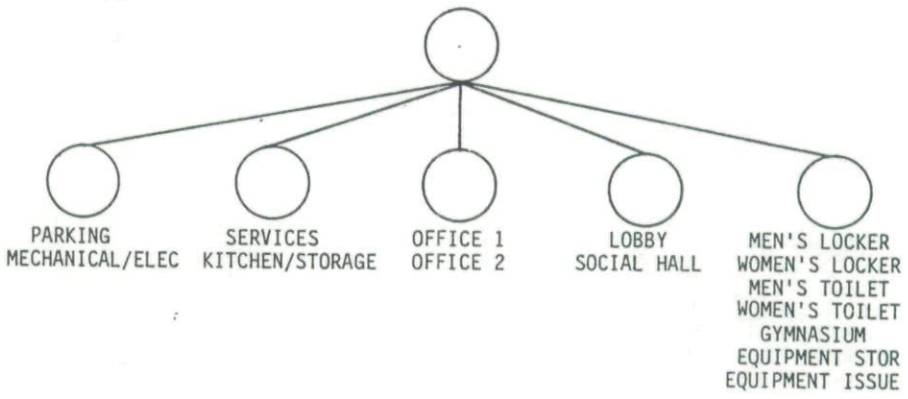

Fig. 2. Linked activity diagram.

represented by the node (circle) at the top of the diagram. In this example, the particular subsets into which the full activity space has been divided are represented by the nodes in the lower portion of the diagram. Beneath these nodes are listed the elements contained in each subset. The user can request to have these subsets further divided into additional subsets, and the result displayed in an elaborated diagram. Creating subsets of linked activities can be an important aid to the designer in simplifying and better understanding the structure of the design problems. The preceding example, although small for purposes of illustration, represents a real-world application. In this case, the different subsets became distinct sections of the final building.

(3) The system allows the user to produce a bubble diagram, as characterized earlier, either automatically or in stages. The bubble diagram's two-dimensional representation of the weights between activity spaces is a key aid in conceptualizing possible spatial arrangements. As a visualization tool, it is exceedingly helpful to the designer, and the nonlinear optimization procedure that creates it is an important system component (see fig. 3).

(4) SPDS next provides the user graphic aids to help him produce an initial rough floor plan based on the bubble diagram. At this stage, the user may define adjacency (as opposed to proximity) requirements, and may define circulation ways. A combinatorial search algorithm was designed and programmed to identify ways to rearrange the plan so as to better satisfy the user's input requirements. Parameters may be redefined and partial solutions saved.

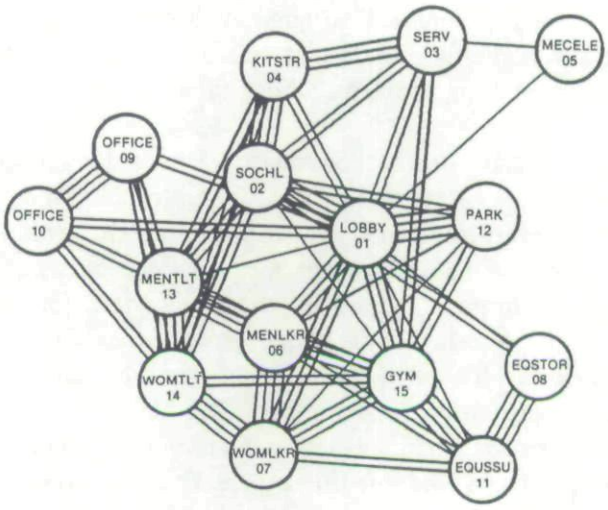

Fig. 3.

The algorithms that have been designed into the system undertake, in each component, to meet the stringent efficiency needs of user interaction. Our overriding guideline has been to permit the architectural designer to freely incorporate his thoughts with the output of the computer. In any of the four stages described above, the user can edit or evaluate the design. Special convenience features allow the user to change his mind regarding initial input at various stages throughout the process.

6. Technical features of the system

We now describe the components of the system from a decision science perspective, briefly characterizing the classes of problems confronted, and the nature of the methods we developed for handling them.

SUBSET CREATION AND EVALUATION ROUTINE

The subset creation and evaluation routine of the SPDS system, as previously indicated, breaks the design problem into linked activity domains, or subsets of highly connected activities. This tool is not only exceedingly useful in its own right, but also facilitates the solution of the bubble problem.

The concept of computerized decomposition and recomposition of elements in an architectural design problem has been addressed by Alexander [1], Davis [3], Milne [9], Owen [11], and Shaviv [12,13]. The full combinatorial complexity of the problem, however, has characteristically been avoided by introducing simplifying assumptions that remove many of its real-world aspects. We undertook to avoid this pitfall by starting from a mathematical programming formulation that captured the problem's key elements explicitly. The result was to characterize the problem as a mixed integer goal programming problem, whose formulation details are provided in the appendix.

We designed a special purpose combinatorial heuristic for the problem that was tested against a number of standard heuristic approaches, and we found that it generated uniformly superior solutions. We also extensively tested the method by manipulating the objective function, determining that the best solutions generated for one objective are not improved upon by those generated for 'neighboring' objectives (a strong measure of reliability). Moreover, the procedure succeeds in solving problems corresponding to MIP's with over 25 000 variables and 50 000 constraints in less than one minute on a V77 minicomputer. The heuristic principles underlying our method are derived from [6] and are elaborated in the appendix.

The solution produced for the subset creation and evaluation problem is translated by our system into a graphic display to aid the designer's understanding and analysis, again in an interactive mode.

THE BUBBLE ROUTINE

The bubble problem requires the determination of a set of two-dimensional coordinates that relate the various activities spatially. The distance between each pair of activities should be inversely proportional to the weight specified between them. The weights will not in general yield exact proportionality on the plane. We seek instead to come as close as possible to satisfying the inverse proportionality condition.

The approach we used is an application of nonlinear programming, utilizing a successive re-evaluation and nonlinear approximation method. A general overview of its operation is as follows. At each stage *ideal distances,* separating each pair of points, are determined from adaptively calculated ranges that currently correspond to the distance options for representing the linkage weights which the user specifies. A successive approximation method then undertakes to minimize deviations of actual distances from given ideals by means of geometric functions that shift the point positions toward those that would more closely correspond to the ideal distances. The ideal distances and the geometric shift functions may be viewed as analogous to 'directional derivatives'. The actual degree of shift may be viewed as a form of 'step size'. The manner of controlling the shift and revising the functions has the same critical importance as in more standard nonlinear programming contexts. In particular, we organized our procedure to carry out these operations by the guidelines found successful in subgradient optimization [7].

To further enhance the value of the solution, 'a psychological component' gives increased emphasis to higher user weights, based on the fact that designers perceive a non-uniform spatial separation to provide the most meaningful visual separation. For example, a user specification of weights such as 1, 2, 3, 4 is translated by the system into a framework where '4' is somewhat more separated from '3' than '3' is from '2' and so forth.

In routine operations, the user has the option of creating a bubble diagram either automatically in one step, or in a more interactive manner. He or she may further elect to use the 'subset creation routine' to subdivide the problem, obtaining a bubble diagram for each subset. Once the diagram appears on the screen, the diagram may be edited: rotating, translating, or mirroring desired sets of activities, obtaining a score for the results as shown in fig. 4.

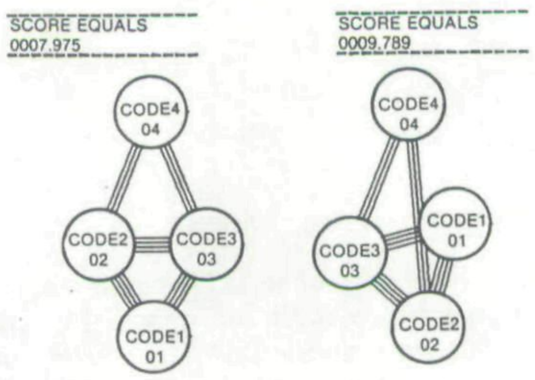

Fig. 4. Obtaining a score.

By invoking the proper system controls, the user may then combine the individual bubble diagrams into one diagram, automatically rotating the several drawings optimally with respect to each other, then automatically adjusting the diagram a little further. At any stage, the user may edit the diagram or obtain a score.

An extra convenience feature, not requiring additional algorithmic components but further aiding the user's analysis, is an option that allows the user to automatically produce what is called a 'space diagram'. The space diagram is similar to the bubble diagram illustrating, in addition, square footages and proportions. The space diagram takes the designer one step closer to the final layout plan, which may then be edited. Figure 5 represents the relationship between three subsets.

While it may be thought that an effort of this magnitude would best be accomplished by a large staff of programmers and analysts, our experience in using only two support personnel in addition to the authors was that the intensity and dedication of a small group enabled goals to be met that quite probably would be unattainable by a larger team, whose members had a more diffuse involvement.

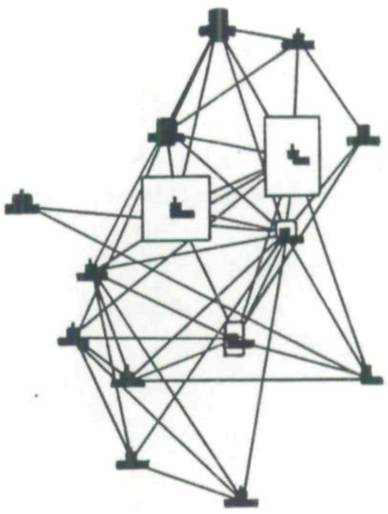

Fig. 5. Space diagram.

A 'BLOCK PLAN IMPROVEMENT' COMPONENT IN SPDS

The *b*lock *p*lan *i*mprovement routine (BPI) constitutes an extension and enhancement of a system proposed by Buffa and Armour [2], called CRAFT. Our BPI represents a significant improvement over CRAFT, however, both in terms of its usefulness to architects and space planners and in terms of its speed. The basic algorithm has been modified not only to improve performance, but to solve a more general architectural problem.

BPI begins with an initial layout plan created interactively by the user – one in which the various rooms or activities have been placed on a 42 by 42 grid cell matrix. BPI then employs a tailored heuristic to improve the layout, seeking a local minimum to a cost function. The algorithm may be characterized as a breadth first, unit depth tree search strategy using an estimate function to evaluate node costs.

The cost function in BPI, as opposed to that in [2], has two components. The first is defined by the proximity weight specified by the user in the Relationship Chart. The second component involves a user-defined adjacency weight. Thus, BPI allows the user to make the distinction between *adjacency* and *proximity*, an important consideration architecturally. In common with many other such techniques (including those of dynamic programming), BPI seeks to make the best exchange which is locally available. In comparison to other methodologies proposed in [2], however, BPI uses both a better cost estimate and a larger range of 3-way moves.

BPI also has an optional activity shape monitor that permits only exchanges that do not drastically alter an activity's shape. In contrast, previous methodologies tend to severely distort shapes. The means for evaluating exchanges is also more comprehensive than in previous approaches. For example, CRAFT estimates the value of an exchange by computing the cost decrease if the centroids of the activities are moved into new positions. BPI estimates the new positions more accurately by considering relative room sizes.

In addition to its increased functionality, BPI is exceedingly efficient. Our tests indicate that it is on the order of 100 times faster than the CRAFT system, yet occupies 5 K fewer words of space in main memory. The efficient memory handling of BPI also gives it the advantage of being able to run on a non-virtual system.

To enhance the value of the BPI routine to the designer, we equipped it to provide a score for part or all of a proposed layout plan. The user has the option of freezing a group of activities in place 'on the plan' before activating the routine. When the user is satisfied with the arrangement, he may want to save it in a library of solutions and make a hard copy of it or plot it, in order to remember it as an alternative. SPDS makes all this possible.

The resulting bubble diagram has been enthusiastically embraced by designers in firms such as Dalton, Dalton, Newport and Marshal Erdwin. As we anticipated, the primary use of the bubble diagram has been to aid in visualizing possible spatial arrangements and in sorting out the various relationships between activities. The graphic diagrams which SPDS produces have been cited as among the most useful components of the system. In addition, the speed of the solution of this complex problem facilitates interactive use, consuming less than one minute on average size problems and under five minutes for the largest problems, on the V77 minicomputer.

7. Conclusions

The SPDS system enhances traditional architectural design methods in a variety of ways. The efficiency of its decision science software when applied to complex problems greatly improves on previously available software. In the case where available software is specialized to solve the same problem, our routine is more efficient by a factor of 100 (and yields higher quality solutions). In cases where the only alternative is general purpose mixed integer and nonlinear programming software, the difference is probably greater. (Attempting to obtain even an LP solution to an MIP with over 25 000 variables and 50 000 constraints would undoubtedly take some hours, in contrast to SPDS's solution time of less than one minute on a minicomputer.)

This efficiency makes a truly interactive system possible and translates into solid practical gains. Based on figures supplied by Westinghouse Furniture Systems, the reduced time required for architects to develop plans is estimated to save between US$ 150 000 and US$ 240 000 per year for a firm with thirty designers.

Yet a more important type of gain is the improvement in design plans. As Marshal Erdwin reports, recognition of this improvement by clients, who are the ultimate beneficiaries, has led to reducing turn around time of client contact to the initiation of a project by a factor of 2.

There is no way to put a dollar figure on the value of improved system design. Yet improved design – particularly in application to medical centers, office buildings and other large complexes – may be SPDS's most useful contribution.

References

[1] C. Alexander, *Notes of the Synthesis of Form* (Harvard University Press, Cambridge, 1964).

[2] E.S. Buffa, G.C. Armour and T.E. Vollman, Allocating facilities with CRAFT, Harvard Business Review 42, 2(1964).

[3] C.F. Davis and M.D. Kennedy, EPS: A computer program for the evaluation of problem structure, in: *Emerging Methods in Environmental Design and Planning,* ed. Moore (MIT Press, 1970).

[4] C.M. Eastman, *Spatial Synthesis in Computer-Aided Building Design* (Wiley, New York, 1975).

[5] R.L. Francis and J.A. White, *Facility Layout and Location: An Analytic Approach* (Prentice – Hall, Englewood Cliffs, NJ, 1974).

[6] F. Glover, Heuristics for integer programming, using surrogate constraints, Decision Sciences 8, 1(1977).

[7] M. Held, P. Wolf and H. Crowder, Validation of subgradient optimization, Mathematical Programming 9(1974).

[8] R. Mather, *Schematic Layout Planning* (CBI Publishing Co., Boston, 1977).

[9] M. Milne, A structure-finding algorith, in: *Emerging Methods in Environmental Design and Planning,* ed. Moore (MIT Press, 1970).

[10] W.J. Mitchell, *Computer-Aided Architectural Design* (Petrocelli Charter, New York, 1977).

[11] C.L. Owen, DCMPOS: An algorithm for the decompostion of nondirected graphs, in: *Emerging Methods in Environmental Design and Planning,* ed Moore, (MIT Press, 1970).

[12] E. Shaviv, R. Hashimshony and A. Wachman, Decomposition of a multi-cell complex a problem in physical design, DMG-DRS 2, 2(1977).

[13] E. Shaviv, R. Hashimshony and A. Wachman, Transforming an adjacency matrix into a planar graph, Technion, Israel Institute of Technology (1979).

Appendix

FORMULATION AND SOLUTION OF THE SUBSET CREATION PROBLEM

Problem formulation

The mathematical formulation of the subset creation problem as a 0-1 mixed integer goal program is as follows.

Define:

N	=	$\{1, \ldots, n\}$, the index set of elements to be allocated to subsets. (Note that n is the maximum possible number of nonempty subsets.)
s_{ij}	=	strength of the relationship between elements $i, j \in N$. (By symmetry we may impose $i < j$.)
L	=	lower bound (goal) on number of elements allowed in any nonempty subset (hence $L \geqslant 1$).
U	=	upper bound (goal) on number of elements allowed in any subset ($L \leqslant U \leqslant n$).
P^+	=	penalty for violating an upper bound goal.
P^-	=	penalty for violating a lower bound goal.

Integer variables:

z_k	=	1 if subset k is created, 0 otherwise; $k \in N$.
x_{ik}	=	1 if element i belongs to subset k, 0 otherwise; $i, k \in N$.

Continuous variables (automatically integer-valued by constraint implications):

y_{ijk}	=	1 if elements i and j belong to subset k, 0 otherwise; $i, j, k \in N$ and $i < j$.
v_k^+	=	number of elements by which set k exceeds its upper bound gaol, $k \in N$.
v_k^-	=	number of elements by which set k falls short of its lower bound goal, $k \in N$.

The problem is then to

$$\text{maximize} \quad \sum_k \sum_{i<j} s_{ij} y_{ijk} - P^+ \sum_k v_k^+ - P^- \sum_k v_k^- \tag{1}$$

$$\text{subject to} \quad \sum_k x_{ik} = 1 \qquad i \in N \tag{2}$$

$$\sum_i x_{ik} \leqslant U z_k + v_k^+ \qquad k \in N \tag{3}$$

$$\sum_i x_{ik} \geqslant L z_k - v_k^- \qquad k \in N \tag{4}$$

$$y_{ijk} \leqslant x_{i}, \ x_{j} \qquad i < j \qquad (5)$$

$$y_{ijk} \geqslant x_{ik} + x_{jk} - 1 \qquad i < j \qquad (6)$$

$$z_{k}, x_{ik} \in \{0. 1\} \qquad i, \ k \in N \qquad (7)$$

$$y_{ijk} \geqslant 0 \qquad i, j, k \in N \qquad (8)$$

$$v_{k}^{+}, \ v_{k}^{-} \geqslant 0 \qquad k \in N. \qquad (9)$$

The objective function (1) maximizes the total strength of the subsets created, reduced by the penalties for failing to satisfy the upper and lower bound goals on the number of elements in each subset. Constraint (2) says that each element must belong to exactly one subset. Constraints (3) and (4) ensure that subset k is empty when $z_k = 0$ and determine the proper v_k^+ and v_k^- values which incur penalties. Constraints (5) and (6) together ensure that y_{ijk} has the value of the product $x_{ik} x_{jk}$, thereby linearizing what would otherwise be a nonlinear problem. These constraints also ensure that y_{ijk} is automatically integer-valued. (Some formulations for linearizing cross products do not.) The remaining constraints express standard bound and integer requirements.

If it is known in advance that a maximum of m subsets will be used, the size of the formulation can be reduced by stipulating that k range from 1 to m. (Note that m is ensured to be less than n if $L \geqslant 2$.) The total number of variables is then $3m + mn + mn(n-1)/2$ and the total number of constraints, disregarding (7)–(9), is $n + mn(n-1)$. In the common size range of $n = 60$, we found that m did not exceed 15 in any solution, including the case where $L = 1$. Thus, the formulation for these problems consists of 27 495 variables and 53 160 constraints.

Heuristic solution method

The method we employed can begin with any initial assignment of the elements to subsets. The indexing of nonempty subsets is maintained to ensure, for some value of m,

$$z_1 = \ldots = z_m = 1 \text{ and } z_k = 0 \text{ for } k > m.$$

The procedure may be viewed as progressively transferring elements among sets, with the features:

(a) transfers that include the empty set are employed;
(b) transfers are managed in a way that avoids being trapped at local optima;
(c) double as well as single element transfers are utilized.

The chief conceptual aspects of the procedure relate to considerations (a) and (b). We slightly simplify the method's description (removing reference to double transfers) to make these aspects more clearly visible.

The evaluation of transfers can be conveniently expressed in the notation previously introduced, where we additionally define

$$V_{ik} = \sum_j s_{ijk} x_{jk} .$$

By convention $s_{ijk} = s_{jik}$, $s_{iik} = 0$, and here we do not restrict i and j so that $i < j$. In words, roughly, V_{ik} is the summed strength of relationships between element i and all elements in set k. Transferring a particular element i out of set k and into a set h corresponds to setting $x_{ik} = 0$ and $x_{ih} = 1$ [continuing to satisfy constraint (2)].

The objective function (1) and the relation $y_{ijk} = x_{ik} x_{jk}$ implies that the *marginal value* of this transfer, neglecting the penalty component, is precisely

$$V_{ih} - V_{ik} .$$

An important aspect of our procedure, embodied in condition (a) mentioned previously, is that we allow the set h to be empty; i.e. specifically, set h can denote the set indexed $m + 1$ (assuming $m < n$). The preceding formula continues to apply. As a consequence, the procedure can both destroy sets and create new ones.

To determine the portion of the total marginal value (objective function change) contributed by P^+ and P^-, let n_k denote the cardinality of set k, i.e.

$$n_k = \sum_i x_{ik} .$$

The marginal penalty that is incurred by setting $x_{ik} = 0$ and $x_{ih} = 1$, given $k \neq h$ and $x_{ik} = 1$ to start, is identified in table A1.

Table A1

Cases	Marginal penalty
$n_k > U$	P^+
$n_k \leqslant L$	$-P^-$
$n_h \geqslant U$	$-P^+$
$n_h < L$	P^-
Exceptions:	
$n_k = 1$	$(L-1)P^-$
$n_h = 0$	$-(L-1)P^-$

The radical discontinuities of the marginal penalties for the cases $n_k = 1$ and $n_h = 0$ are not encountered in the evaluation schemes of other heuristic integer or goal programming procedures we are aware of. (They do not occur, for example, if transfers that start or end with the empty set are not employed.)

These discontinuities motivated another feature of our procedure: to employ criteria for evaluating transfers that differ from those for evaluating solutions. (The latter must strictly abide by the results of table A1.) We found it highly effective for evaluating transfers to smooth the discontinuities at the extremes, replacing the marginal penalties for $n_k = 2$ and 3 by $-(n_k - 1)P^-/3$, and for $n_h = 1$ and 2 by $n_h P^-/3$. In addition, we halved the penalties for $n_k = 1$ and $n_h = 0$.

Combining the preceding marginal values for 'relationship strengths' and for 'penalties' yields the total marginal contribution to the objective function. The logical step, following standard analysis, is to select the transfer that yields the highest evaluation (total marginal contribution), ultimately reaching a point where no improvement is possible – a local optimum. As suggested in condition (b), we avoid being trapped in this situation by introducing a strategy of 'managing' transfers. For this, we adopt the oscillating assignment framework of [6], which persists in making 'best' moves after a local optimum is reached, but relative to a criterion that is modified to prevent cycling. Specifically, we create a tabu list T, whose members are given a temporary 'tabu status' that discourages them from participating in transfers. As soon as an element i is involved in a transfer (corresponding to setting $x_{ik} = 0$ and $x_{ih} = 1$), it is added to the list T, with a tabu penalty. Pragmatically, we found it useful to set this penalty equal to twice the absolute value of the largest evaluation made in the previous twenty transfers. Until twenty transfers, the penalty is simply set large. Thus, when added to T, element i receives the full current tabu penalty. Evaluation of element i is the altered by subtracting this penalty from the usual evaluation measure (since we seek a maximum evaluation). When a transfer is made, the penalty for each element of T is reduced by 1/7 of its original value. After 7 transfers, the penalty for a particular element becomes 0, and that element is removed from T.

Using this approach, the method continues to make transfers when a local optimum is reached, without concern that the best move may be one with a negative evaluation (or may lead to a less desirable solution). Of course, the best move immediately after leaving a local optimum would often be, under normal circumstances, to reverse the move just made. However, the tabu penalty makes this unattractive. In practice, by applying the foregoing rules, a penalized element was generally not selected for transfer until finally removed from T, and was rarely selected before 5 intervening transfers. Thus, for the most part, the penalty operated the same as if preemptively large. However, an element was permitted to 'cheat' on its tabu status, and to be selected earlier, if its evaluation independent of the tabu penalty was strictly better than any evaluation previously received by that element. (As soon as an element is selected, it is freshly added to T, replacing a previous occurrence on the list, if any, by a dummy element.)

The choice of 7 as the decay factor was varied experimentally, and all values from 5 to 9 gave good results. The quality of solutions began to deteriorate outside this range, especially on the lower end. (Cycling behavior, sometimes with periods as great as 20 transfers, were observed when the value was dropped to 4.)

This experimentation underscored the value of managing transfers by the tabu list, although the context of this application differs substantially from that of the first application of related procedures [6]. It may be noted that a standard strategy, which progresses to a local optimum and stops, would correspond closely to a strategy of keeping 0 elements on the tabu list. The correspondence is not perfect because, if stopping is not imposed, improvements over a local optimum may sometimes occur without penalizing immediate reversals of a 'bad' move. (A better local optimum may be close by.) But our solution quality steadily improved as the number of elements of the tabu list progressed through the values 0, 1, 2, 3, 4.

We remark that this implementation differs from the implementation of related ideas in [6], where two tabu lists were used in order to separate the effects of increasing and decreasing variables. However, the tabu list size of 7 proved effective in the earlier application as well. In general, these findings suggest the possible value of the tabu list implementation of oscillating assignment heuristics in other contexts.