The sofa problem

By W. E. Howden*

A program is described which can be used to determine if an irregular two-dimensional object can be moved from one point to another point inside a complicated two-dimensional structure (the two-dimensional sofa problem). The representation of the program in terms of a problem network and an associated search procedure is used to indicate the desirability of a generalised programming schema.

(First received March 1968 and in revised form May 1968)

A problem of both practical and academic interest is to determine if an object A can be moved from one point p_1 to another point p_k inside a complicated structure M. Consider, for instance, the problems of determining whether part of an aircraft can be easily accessed and removed for servicing, or if a complicated structure can be assembled from pre-fabricated parts in a specified order. In some problems there will be no route from p_1 to p_k and in others there will be several routes of which perhaps the shortest must be determined. Some of the difficulties encountered when dealing with complicated shapes can be avoided if the problem is treated numerically rather than analytically, i.e. the boundaries of M and A are described as sequences of points on a grid G (of fineness Δx) rather than as collections of known analytic curves.

Let A be a numeric representation of any two-dimensional object (e.g. a sofa) and M a numeric representation of any two-dimensional structure (e.g. a house). Then if p_1 and p_k are any two points in M, the program described below will determine whether or not A can pass inside M from p_1 to p_k. The program was written in FORTRAN and implemented on the Cambridge University Titan computer.

Sofa program

The input data for the program consists of numeric representations of the boundaries of M and A, and is stored in the two lists L_M and L_A. The coordinates of the boundary of A are relative to a point c_A inside A (the 'centre' of A). If p is some point inside M then say that A is at point p if A is placed so that c_A coincides with p. If A is rotated about c_A through an angle θ, then say that A is rotated through angle θ. Choose an increment of arc $\Delta\theta$ so that if A is rotated through $\Delta\theta$, the maximum displacement of any point in A is less than Δx.

The lists L_M and L_A are in chain-encoded form (Freeman and Garder, 1964). Chain-encoding allows for both economy of storage and ease of manipulation in describing A and its rotations. In this program the type of chain-encoding used requires that each point differ from its predecessor by one unit in exactly one of its coordinates. If (x, y) is a point on the boundary of A or M, then the 'next point' on the boundary is allowed to differ from (x, y) in only one of four ways and only this difference need be stored. For example, suppose the boundary of A, B_A, consisted of the sequence of points (3, 4), (2, 4), (1, 4), (1, 3), (2, 3), (3, 3) then L_A would be of the form (3, 4),1, 1, 2, 3, 3. Only one coordinate, together with a list of numbers representing differences, is needed to represent A and each difference requires only two bits of storage. If B_A is not connected then L_A will consist of several sublists, each of which represents a maximal connected segment of B_A. Let a_i and a_{i+1} be two points on the boundary of A where a_{i+1} is the point 'after' a_i. Suppose that L_A indicates that $(a_{i+1})_x = (a_i)_x + 1$ and $(a_{i+1})_y = (a_i)_y$. The formula becomes only slightly more expensive when rotation through an angle θ is allowed for by using $(a_{i+1})_x = (a_i)_x + \cos\theta$ and $(a_{i+1})_y = (a_i)_y + \sin\theta$. A table in increments of $\Delta\theta$ can be used for storing $\cos\theta$ and $\sin\theta$.

The program is based on certain continuity assumptions about the boundaries of M and A and on the assumption that any route through M for A can be approximated by unit translations of A parallel to the axes, and unit rotations of A about c_A. Suppose it is known that A, when rotated through angle θ, will fit inside M at a point p. Let q be a point in M adjacent to p and suppose that when A is placed at q with rotation θ, the boundaries of M and A do not intersect. Then if Δx was chosen sufficiently small it can be assumed that the 'sofa' represented by A can be moved inside the 'house' represented by M from the point p to the point q (i.e. in moving from p to q the boundary of the 'sofa' represented by A does not 'hop over' or surround a little piece of the 'house' represented by M). Similarly, suppose that the boundaries of M and A do not intersect when A is put at p with rotation $\theta \pm \Delta\theta$. Then if $\Delta\theta$ is sufficiently small it can be assumed that the 'sofa' represented by A will fit inside the 'house' represented by M at p at rotation $\theta \pm \Delta\theta$.

The attempt to discover a route through M is equivalent to finding a path (or shortest path) through a problem network N. In this case N is a network of four-dimensional points such that (x, y, α, β) is a node in N if and only if the boundaries of A and M do not intersect when A is placed at (x, y) in M for each rotation in the interval $[\alpha, \beta]$ but not for the rotations

* *University Mathematical Laboratory, Corn Exchange Street, Cambridge.*

$\alpha - \Delta\theta$ and $\beta + \Delta\theta$. Two nodes $(x_1, y_1, \alpha_1, \beta_1)$ and $(x_2, y_2, \alpha_2, \beta_2)$ in N are connected if (x_1, y_1) and (x_2, y_2) differ by a unit in exactly one of their coordinates and $[\alpha_1, \beta_1]$ intersects $[\alpha_2, \beta_2]$. The possibility that N is not a true network but consists of several disjoint smaller networks is included. The sofa program determines the nodes in the problem network adjacent to a given node and uses the strategy described below for processing this information about N to produce a solution. In this particular program the strategy used was a straightforward exhaustive search procedure. Let n_1 and n_k be the nodes in the problem network which represent the two points p_1 and p_k between which it is desired to move A. Starting from n_1, the nodes in N adjacent to n_1 are determined. Then each of these nodes is considered in turn as a starting point and so on until n_k is reached or there are no more nodes to be considered. Suppose $g(n_i, n_j)$ is a function which measures the 'distance' between any two nodes (n_i, n_j) which are connected (adjacent) in N and that $f(n_j)$ is defined by $f(n_j) = f(n_i) + g(n_i, n_j)$. For each node n_j in N which is being considered, the node n_i before n_j (i.e. the node n_i to which n_j was adjacent) is noted and the value of $f(n_j) = f(n_i) + g(n_i, n_j)$ is determined. If n_k is reached a path through N can be constructed by working backwards from n_k to n_1 since the name of the node 'before' each node has been stored. Suppose n_j is adjacent to a node n_i (i.e. we have got to n_j from n_i) but n_j has been previously discovered (as a node adjacent to some earlier node n_h). If $f(n_i) + g(n_i, n_j) < f(n_h) + g(n_h, n_j)$ then $f(n_j)$ becomes $f(n_i) + g(n_i, n_j)$ instead of its previously assigned value $f(n_h) + g(n_h, n_j)$, the node 'before' n_j becomes n_i instead of n_h, and n_j is considered as a starting point once again. If $f(n_i) + g(n_i, n_j) \geqslant f(n_h) + g(n_h, n_j)$ then n_j is ignored and no action is taken. This technique is similar to the dynamic programming procedure described in Busacher and Saaty (1965). If $g(n_i, n_j) = 1$ for any two nodes n_i and n_j which are adjacent in N then the path with the least number of nodes in it is determined. Other functions g can be used, for example, to help minimise the number of turnings in the first path to be discovered. The 'looking backwards' type of strategy used in this program can be compared to the 'looking ahead' type of strategy of the evaluation function in the Graph Traverser (Doran and Michie, 1966).

If L_M and L_A are used to determine whether or not the boundary of A intersects the boundary of M, every point in L_M will have to be checked against every point in L_A. To avoid this expensive procedure a model of the boundary of M is constructed in the form of a tree of lists T_M in which each list gives all the values of y for a particular x. To determine if the point (x, y) from L_A is on the boundary of M, the list beginning at $T_M(x)$ is examined for the entry y.

Program results

The upper bound l_{max} to the set of all lengths l of rectangles of width w that will pass through a corridor of width W with a right-angled corner in it, is given by $l_{max} = 2.(W.\sqrt{2} - w)$. The program was tested with $w = 2$ and $W = 8$ ($l_{max} \cong 19$). A route was determined for $l = 16$ when $\Delta\theta = \pi/10$. For $l = 16$ an increment of arc of the same order of size as Δx would be $\Delta\theta = \pi/20$. Even with $\Delta\theta$ decreased to $\pi/80$ success could not be achieved for $l = 17$, indicating that accuracy (of approximately three units) is more dependent on Δx, the fineness of G, than on the size of $\Delta\theta$.

The computation time for the simple example described above was approximately 90 seconds. A larger more complicated example, in which a U-shaped object is required to retrace its path several times in order to achieve necessary changes in orientation, required between 5 and 6 minutes computation time. A decrease in Δx to $\Delta x/n$ increases the computation time and storage requirements k to a value less than the possible maximum of $k.n^2$. Let m be the approximate number of points in M. Based on the results of both simple and complicated examples, the amount of working space required by the program is of the order of $9.m/2$ twelve-bit words. The program occupies about 11K of (48-bit word) core store. It should be noted that the running time and storage requirements for this type of sofa program depend directly on the strategy being used.

The program produces diagrams of the 'sofa' and the 'house' on the line-printer, with the solution route marked out in that of the 'house'.

Strategies and heuristic programming

Although the strategy of the sofa program can be described independently, it was not written as a separate piece of program. It is intermingled with the other parts of the program and is not easily altered. Lack of flexibility is a characteristic feature of large heuristic programs and it constitutes one of the major limitations in a heuristic approach to problem solving (Newell and Simon, 1964). Approaches to the problem of inflexibility can be found in the early attempts to formulate problems in terms of a problem network and a separately written general network-processing procedure (e.g. DeFlorio, 1963, and Suurballe, 1962). In a general way this approach is fundamental to the General Problem Solver (Newell and Simon, 1963), as well. Extensive research into the representation of heuristic programs specifically in terms of a problem network and an independent strategy-processor was first described by Doran and Michie (1966).

For those engaged in the construction of particular, large heuristic programs that can be formulated in terms of a problem network, it would be a great advantage if: (i) 'strategy programs' were available which were written in a general format so that they could be applied to contextually differing problems, and (ii) alternative strategies could be applied independently or in combination to the same problem.

Conclusions

The accuracy achieved in the sofa program indicates the feasibility of a numerical approach to the two-dimensional sofa problem. The ideas in the program are readily extendable to three dimensions and a solution to the more general sofa problem in which it is required to determine whether or not a deformable three-dimensional object can be moved between two points inside a three-dimensional structure is being programmed.

Because of its evident formulation in terms of a network search, the sofa problem has led to the idea of programming schema in which different strategy routines can be applied to the same problem or, alternatively, the same strategy routine can be applied to different problems. Such a schema is being used to program the three-dimensional sofa problem and it is hoped that it will provide a technique for minimising storage requirements and computation time. The schema is a method of dividing the different parts of heuristic programs into: (i) overall or global strategies for dealing with problem networks as a whole, (ii) local strategies for dealing with particular nodes in problem networks, (iii) global data programs for providing overall information about a specific problem, and (iv) local data programs for providing particular points of information about a specific problem. Possible increases in computation time may result from the increased cost of communication between the different parts of a program that is split up in this manner. However, since most of the computation time is consumed within single blocks of program (e.g. the routine that checks to see if a given node is in N in the sofa program), the increase in computation time will be small relative to the advantages gained in providing for the variable use of different strategies. The efficiency of a heuristic program rests more in its ability to avoid processing non-relevant information contained in the problem network (i.e. in the efficiency of its strategy) than in the speed with which different parts of the program operate and communicate. The facility for testing and improving the strategy of the sofa program should be an important benefit of the programming schema. 'If you want to choose between selectivity and speed, choose selectivity because it will buy a great deal' (Selfridge, 1965).

Acknowledgements

The author wishes to thank C. A. Lang for his invaluable assistance in the formulation of the sofa program and for helpful discussions about the programming schema. This research was financially supported by the National Research Council of Canada.

References

BUSACHER, R. G., and SAATY, T. (1965). *Finite Graphs and Networks*, New York: McGraw-Hill Book Co.

DEFLORIO, GEORGE (1963). Intelligent Automata and Man-Automaton Combinations: A Critique and Review, *Electrical Engineering*, Vol. 82, p. 200.

DORAN, J. E., and MICHIE, D. (1966). Experiments with the Graph Traverser Program, *Proceedings of the Royal Society*, Vol. 294A, p. 235.

FREEMAN, H., and GARDER, L. (1964). Apictorial Jigsaw Puzzles; The Computer Solution of a Problem in Pattern Recognition, *IEEE Transactions on Electronic Computers*, Vol. EC–13, p. 118.

NEWELL, ALLEN, and SIMON, HERBERT A. (1963). GPS, A Program that Simulates Human Thought, in *Computers and Thought*, New York: McGraw-Hill Book Co.

NEWELL, ALLEN, and SIMON, HERBERT A. (1964). Problem Solving Machines, *International Science and Technology*, Vol. 36, p. 48.

SELFRIDGE, OLIVER (1965). Reasoning in Game Playing by Machines, in *Computer Augmentation of Human Reasoning*, Washington: Spartan Books, Ltd.

SUURBALLE, JOHN W. (1963). Network Algorithms for Combinatorial and Discrete Variable Problems, in *Recent Advances in Mathematical Programming*, New York: McGraw-Hill Book Co.

Appeal from the Book Review Editor

The British Computer Society receives a large number of newly-published books for review and it is becoming difficult to find reviewers for all of them. In the past I have tended to rely on persons I know or who have been recommended to me. I believe there must be many members of the BCS who would like to review some books but are never asked because nobody knows of their willingness to help. Unfortunately, the BCS is unable to pay the reviewers, although they are allowed to keep those books they review.

Would anyone willing to help please write to me, indicating their subject preferences?

Dr. P. A. Samet
BCS Book Review Editor
Computer Centre
University College London
19 Gordon Street
London, WC1