The application of interactive computer techniques and graph-theoretic methods to printed wiring board design.

A.K. HOPE

Thesis presented for the Degree of Doctor of Philosophy of the University of Edinburgh in the Faculty of Science, July 1973.
# Table of Contents

1. **Introduction**
   - 1.1 The Interconnection Problem in Electronic Circuits.  
   - 1.2 Printed Wiring Boards.  
   - 1.3 Definition of the Layout Problem.  

2. **Survey of Existing Approaches to the Layout Problem.**
   - 2.1 General Observations.  
   - 2.2 Component Placement Methods.  
     - 2.2.1 Force Placement Methods.  
     - 2.2.2 Sequential Placement Methods.  
     - 2.2.3 Placement Permutation Procedures.  
     - 2.2.4 The Method of Unconnected Sets.  
   - 2.3 Conductor Routing Methods.  
     - 2.3.1 Maze Running and its Derivatives.  
     - 2.3.2 Routing with a "Stepping Aperture".  
   - 2.4 Topological Methods Based on Graph Planarity.  
     - 2.4.1 Graph Planarity in Single-sided Board Layout.  
     - 2.4.2 Graph Planarity in Multi-layer Board Layout.  
   - 2.5 Highly Interactive Layout Procedures.  

3. **Component Placement Through Graph Partitioning.**  
   - 3.1 The Abstract Graph Representing the Circuit.  
   - 3.2 Partitioning the Circuit Graph.  
     - 3.2.1 The Allocation of Clusters to Slots on the Board.  
     - 3.2.2 Specification of the Partitioning Procedure.
3.2.3 Description of the Partitioning Procedure. 28
3.3 The Assignment of Clusters to Specific Slots. 32
3.4 The Assignment of Components to Specific Locations in a Slot. 33
3.5 Refinement of the Component Placement Within a Slot. 35
3.6 Interactive Modification of Component Placement. 37

4 Conductor Routing. 42
4.1 The Subdivision of the Conductor Routing Problem. 43
  4.1.1 The Geometric Definition of Inter-Slot Boundaries. 46
4.2 Outline Description of the Routing Scheme. 49
4.3 Sub-Tree Generation in Route Planning. 49
4.4 Interactive Route Planning for Special Cases. 52
4.5 Conflict Minimisation Using Stepping Aperture. 55
  4.5.1 The Initial Definition and "Stepping" of the Aperture. 59
  4.5.2 The Allocation of Parts of Sub-Trees to Specific Board Sides. 62
  4.5.3 The Minimisation of Conflicts Within a Given Aperture. 66
4.6 Interactive Completion and Modification of the Routing. 67
  4.6.1 The Need for Interaction. 67
  4.6.2 The Facilities Provided for Interaction. 68

5 The Input and Storage of the Circuit Description. 72
5.1 The Master Library. 72
  5.1.1 The Content of a Master Definition. 72
  5.1.2 The Organisation and Operation of the Master Library. 74
5.2 The Input of Circuit Information.

5.3 The Circuit Data Structure.
   5.3.1 Single Level Associations.
   5.3.2 The Data Structure Representation of a Component.
   5.3.3 Hierarchical Associations.
   5.3.4 Building the Circuit Data Structure.

5.4 The Verification of the Input Data.

5.5 The Data Structure Used in Graph Partitioning.

6 The Implementation of the Component Placement Algorithms.
   6.1 k-way Partitioning of the Circuit Graph.
      6.1.1 The Evaluation of k, the Number of Clusters.
      6.1.2 Building the Temporary Data Structure used in Partitioning.
      6.1.3 The Initial Assignment of Components to Clusters.
      6.1.4 The Generation of the Pairwise Refinement Sequence.
      6.1.5 The 2-way Refinement Procedure.
   6.2 The Placement of Clusters in Slots on the Board.
   6.3 The Placement of Components within Slots.
   6.4 The Refinement of the Initial Component Placement.

7 The Implementation of the Conductor Routing Algorithms.
   7.1 The Geometric Data Structure.
      7.1.1 The Overall Organisation of the Data Structures.
      7.1.2 The Storage of Coordinate Data in the Geometric Structure.
      7.1.3 The Geometric Structure Representation of Directed Trees.
7.1.4 The Temporary Data Structure used in Aperture Routing. 124
7.1.5 Building the Initial Geometric Data Structure. 126
7.2 The Assignment of Dummy Pins to Specific Locations. 129
7.3 The Generation of Sub-Tree Route Plans. 131
7.4 The Resolution of Conflicts Between Route Plans. 137
7.4.1 The Generation and Control of the Stepping Aperture. 139
7.4.2 The Initial Allocation of Track Segments to Board Sides. 141
7.4.3 Route Plan Modification Within a Given Aperture. 142

8 The Implementation of the Facilities for Graphical Interaction. 151
8.1 System Software for Graphical Interaction. 152
8.2 Interaction with the Program-Generated Placement. 154
8.2.1 The Facilities Provided for Placement Interaction. 155
8.2.2 The Implementation of Placement Interaction. 156
8.3 The Interactive Insertion of Route Plans. 158
8.3.1 The Approach to Interactive Route Planning. 159
8.3.2 The Facilities Provided for Interactive Route Planning. 160
8.3.3 The Implementation of Interactive Route Planning. 162
8.4 Interaction with the Program-Generated Routing. 167
8.4.1 Graphical Output for the Whole Board. 168
8.4.2 The Facilities Provided for Routing Interaction. 171
8.4.3 The Implementation of Routing Interaction. 174
9 Program Performance.
   9.1 The Selection of Test Circuits.
   9.2 Placement Evaluation in terms of Estimated Wirelength.
   9.4 Interactive Route Planning for the Power and Ground Nets.
   9.5 Overall Evaluation including Routing Interaction.

10 Proposed Extensions Relating to the Layout Program.
   10.1 The Use of Graph Partitioning in Related Tasks.
       10.1.1 The Allocation of Components to Boards.
       10.1.2 The Assignment of Logic Elements to Packages.
       10.1.3 The Placement of Discrete Components.
   10.2 Post-Processing to Obtain Manufacturing Information.

11 Conclusions.
   11.1 Component Placement.
       11.1.1 Deficiencies in the Placement Procedures.
   11.2 Conductor Routing by Program.
       11.2.1 Deficiencies in the Automatic Routing Procedures.
   11.3 Interactive Completion of Conductor Routing.
       11.3.1 Deficiencies in the Routing Interaction Scheme.
   11.4 Originality.

Acknowledgements.

Appendix 1 Data Structure Free-Storage Management.
Appendix 2 The Use of Macro Processing Techniques.
   A2.1 The STAGE2 Macroprocessor
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2.2</td>
<td>Macro Processing as a General Programming Aid.</td>
<td>234</td>
</tr>
<tr>
<td>A2.3</td>
<td>Macro Processing as a Data Structuring Aid.</td>
<td>235</td>
</tr>
<tr>
<td>Appendix 3</td>
<td>Detailed Description of Storage Structure Blocks.</td>
<td>241</td>
</tr>
<tr>
<td>A3.1</td>
<td>The Circuit Structure Blocks.</td>
<td>243</td>
</tr>
<tr>
<td>A3.2</td>
<td>The Geometric Structure Blocks.</td>
<td>246</td>
</tr>
<tr>
<td>Appendix 4</td>
<td>The System Hardware Configuration.</td>
<td>250</td>
</tr>
<tr>
<td>Appendix 5</td>
<td>Layout Program Implementation Details.</td>
<td>251</td>
</tr>
<tr>
<td>Appendix 6</td>
<td>2-way Partitioning.</td>
<td>253</td>
</tr>
<tr>
<td>A6.1</td>
<td>The Procedure due to Kernighan and Lin.</td>
<td>253</td>
</tr>
<tr>
<td>A6.2</td>
<td>The Modified Partitioning Procedure.</td>
<td>255</td>
</tr>
<tr>
<td>Appendix 7</td>
<td>Component Placement Through Graph Partitioning in Computer-Aided Printed Wiring Board Design.</td>
<td>259</td>
</tr>
<tr>
<td>Appendix 8</td>
<td>A Planar Graph Drawing Program.</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td>Glossary of Terms.</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td>Bibliography.</td>
<td>278</td>
</tr>
</tbody>
</table>

Copyright A.K. HOPE
Introduction

Printed wiring boards are used extensively for the purpose of mounting and interconnecting the components making up an electronic circuit. The design of the layout of a printed wiring board is a task generally undertaken by a skilled layout draughtsman, working from the engineer's circuit design. Often it requires a great deal of time. As the complexity of components continues to increase, with increasing use of medium- and large-scale integration, the complexity of the layout design problem will, in some cases, increase, with an increasing probability of errors occurring in the layout process. In some situations, however, the advent of complex new components will actually simplify the board layout problem, as an increasing proportion of the total interconnection problem is absorbed in the design of the packages themselves. The automation of this part of the total design process is thus considered to be highly desirable particularly in view of the potential reduction in design time and the elimination of errors.

A number of attempts have been made to write computer programs to design such layouts, sometimes involving graphical interaction. Though none of those known to the author has succeeded in reducing both layout cost and time while maintaining a standard of layout comparable with that produced by a draughtsman, various programs have been exploited commercially because particular manufacturers value the reduction in design time more highly than the actual increase in the design cost.
This thesis describes a new approach to the computer-aided layout of double sided printed wiring boards, aimed primarily at the very large category of digital circuits currently in use. It is based on graph-theory and uses interactive graphics as a means of making efficient use of the layout designer's talents. The results of the automatic part of the program compare well with manual layouts and the few published results available for other programs in terms of both cost and speed. The interactive parts of the program are at present slow, but it is anticipated that the method could, with suitable development, provide a cost-effective tool in this type of layout.

1.1 The Interconnection Problem in Electronic Circuits.

Any electronic circuit consists of a number of interconnected components. In the case of digital circuits, which make up a large proportion of all present day circuit designs, most of the components are packages - monolithic integrated circuits or hybrid packages, a hybrid consisting of a substrate carrying a number of unencapsulated monolithic chips and discrete components together with their interconnections. Packages of either type may contain a small number of electrically identical elements (e.g. 4 3-input gates) or a collection of basic logic elements interconnected so as to perform some standard function (e.g. B.C.D. to decimal conversion).

Given the task of implementing a circuit design in hardware, the circuit must be broken into sections for allocation to cabinets (assuming the circuit to be large enough to occupy more than one cabinet), and these must in turn be broken down into parts for
allocation to racks and then to individual printed wiring boards. There may also be a problem associated with the allocation of logic elements to packages where a package contains a number of electrically identical elements. Given the solution of these various allocation problems, the remaining problem is the layout of the individual printed wiring boards.

This thesis is concerned with the latter problem - the layout of printed wiring boards - though proposals are made in Chapter 10 for extensions, based on the methods developed for board layout, which would enable the assignment of components or logic elements to boards and the allocation of logic elements to packages to be handled automatically by program.

1.2 Printed Wiring Boards.

A printed wiring board serves the dual function of supporting and interconnecting the set of elements assigned to it. A board may have any number of layers of conductors, but the present description is limited to the most common type of board currently in use - the double sided board, in which a stable, insulating base material (typically fibreglass) supports a conductive copper pattern on each of its two surfaces. Connections between the conductor patterns on either side of such a board are generally made by means of through-plated holes - holes drilled in the board and plated inside first with copper and then with tin-lead or gold.

In terms of the physical support of components, most components are mounted such that their pins or leads are normal to the board
surface and are soldered into an appropriately placed set of through-plated holes. For certain types of package - such as flat-packs - the leads lie parallel to the board surface and are soldered or welded onto appropriate pads in the copper pattern on one surface of the board. This category of component has not been considered in the layout program to be described.

The interconnections between component pins consist of copper tracks on the two board surfaces, interconnected as necessary with through-plated holes. Such tracks will be of a minimum width dictated by their current carrying capacity or, more commonly, by manufacturing limits. Track widths of 0.015" and upwards are currently in common use, though it is feasible to manufacture boards with 0.005" tracks.

The design of a board layout requires the positioning (placement) of the various components on the board, so defining the positions of the through-plated holes for the component pins, and the design of the interconnection pattern of tracks and holes as required. In view of the fact that the tracks have finite resistance, capacitance and inductance, it is desirable that the tracks should be of minimal length and that track separation be large enough to avoid excessive capacitive and inductive coupling between adjacent signal tracks etc. This is particularly important in high frequency circuits, and it should be noted that with digital circuits operating at frequencies up to several hundred megahertz the delay along a few inches of copper track can easily equal the propagation delay of a fast gate, thereby affecting the electrical design considerations.
1.3 Definition of the Layout Problem.

This thesis describes an approach to the computer-aided design of printed wiring board layouts. Ideally a layout program would be sufficiently general in nature to handle any type of circuit, but for present purposes the type dealt with is restricted to that category consisting of packaged components. An explanation for this restriction is to be found in 9.1. In addition to generating a layout for a given circuit, a layout program must be able to cater for special constraints - for example, that a given component should occupy a given board location for purposes of accessibility or heat dissipation, or that a given electrical net should be very short because any delay in it could affect the circuit performance adversely.

Ideally a layout program would solve such constrained layout problems automatically, but in practice it seems doubtful if complex problems of this nature can be solved efficiently using only heuristic algorithms. There is thus a need to introduce graphical interaction at the critical points in the layout procedure such that the skill and pattern recognition ability of the designer can be used effectively to assist the automatic algorithms. This is particularly important where it is necessary to handle constraints of an unusual nature which arise so infrequently as not to merit any attempt at automatic handling.


In Chapter 2 a critical survey is presented of those techniques used by other workers in the field of printed wiring board layout.
Chapters 3 & 4 present an outline description of the approach developed to the layout of double sided boards, while Chapters 5-7 deal with the implementation details of the method. Chapter 8 deals with the interactive facilities, both in terms of general description and implementation detail.

In Chapter 9 the results obtained using this new method are presented, while Chapters 10 and 11 deal with some proposed extensions of the work described and the conclusions drawn.

A Glossary of terms used in the thesis appears after the Appendices.
2 Survey of Existing Approaches to the Layout Problem

2.1 General Observations.

In attempting to program a computer to design layouts for printed wiring boards one is, in essence, attempting to simulate a human capability of a fairly high order, although most draughtsmen appear to be unaware of the "algorithms" which they themselves use in performing such tasks. The essential element is the apparently "natural" human ability to extract relevant information from a mass of detail, mostly superfluous - in short, the human eye-brain pattern recognition ability.

In view of the difficulty of simulation, most of the workers in this field have divided the layout task into two simpler subproblems, namely "component placement" and "conductor routing", and assumed these to be independent. The latter assumption, however, is invalid: the two phases should be inter-related, thus requiring their inclusion in a "feedback loop", involving lookahead to the routing at the time of component placement or modification of the initial placement in the light of the routing. Either process could easily lead to a time-consuming iterative procedure. Thus, in general, the placement is designed to optimise some more or less arbitrary criteria which, on average, are meaningful in terms of easing the problems in the subsequent routing phase. The topological methods described in 2.4 are unique in that they alone make some attempt at providing the required degree of lookahead insofar as board topology is concerned, but unfortunately introduce another artificial division in the process, namely that between the topological and geometric
2.2 Component Placement Methods.

The following consideration of component placement methods includes, for brevity, those procedures which would be more accurately described as "placement refinement" methods, and would obviously be used following an initial placement phase which might be manual, random or by any of the automatic procedures described.

2.2.1 Force Placement Methods.

Fisk et al (17) describe a placement procedure in which the conductors interconnecting components are endowed with "elastic" properties, thus exerting an attractive force between the components at either end of a conductor, proportional to the conductor length. To avoid components "colliding", a repulsive force, inversely proportional to their separation, is added between pairs of component bodies, and likewise between the edges of the board and the component bodies. For a given (perhaps arbitrary) initial placement, the net force acting on each component can be found, and an equilibrium position computed. Each component is allowed to adopt this equilibrium position (or to move some fraction, k, of the way towards it, where 0<k<1), and the process is repeated until movement ceases or falls below some pre-determined threshold value, giving a "minimum energy" solution. If component pins are recognised as geometrically distinct, then the net torque acting on components may be used to determine their optimum orientation.
An alternative, though essentially similar, method is the so-called "centre-of-gravity" method, such as that of Houghton (25). The "centre of gravity" of a node (electrical net) is found, assuming that each component connected to the node contributes unit mass to that node. Each attached component is then moved a fraction $k$ ($0 < k < 1$) of the distance from its present position towards the centre of gravity of the node. This is repeated for all the nodes of the circuit. In order to prevent convergence of the components on the overall centre of gravity of the circuit, they are periodically expanded outwards onto a grid, maintaining their relative positions. The whole process is iterated until the net movement falls below some threshold value.

Both of these methods are restricted because of their dependence on the initial placement, and their lack of any meaningful objective function (meaningful, that is, in terms of providing a good start-point for conductor routing). In favour of these methods is the fact that they can handle any type of component (some methods are biased in favour of handling only packages of similar size), and are reasonably fast in operation.

2.2.2 Sequential Placement Methods.

Dunne (14) describes a method in which components are placed one at a time, the routing from the newly-placed component to the already placed part of the circuit being completed before going on to place and route any further components. Given a set of permissible package locations, and a set of packages to be placed, the next package to be placed is that with most connections to those already placed, and in
the event of ambiguity, that with fewest connections to the unplaced components. The package is placed in that location yielding the shortest wirelength or satisfying some similar criterion.

A similar approach described by Radley (48) chooses the next component to be placed as that with most completed connections (given that it is placed), and places it in that free location nearest to the centre of gravity of the nodes of all of its placed connections (see description of "centre of gravity" of a node in 2.2.1), provided that its connections can be routed in that position.

These methods probably approach more closely than any others a direct simulation of the draughtsman's approach, interleaving the placement and routing phases. Their major disadvantage lies in the fact that no attempt is made to look at the global properties of the layout, since the program deals essentially with one component at a time, whereas the draughtsman would be more likely to consider groups of highly-connected components.

2.2.3 Placement Permutation Procedures.

Working from an initial placement generated using a sequential placement method (see 2.2.2), Garside and Nicholson (18) consider the effect of n-way cyclic interchanges, a given switch being effected if it results in a net improvement in the objective function (which may be total wirelength, or similar), where n is 2 or 3. The procedure is applied iteratively until no 2-way or 3-way interchange results in any improvement in the objective function.
Similar interchange methods have been described by many authors for placement refinement. For \( n > 2 \), the process is very time consuming, but fortunately most improvement can be achieved for \( n = 2 \) (pair-swapping). The method is restricted to dealing with packages of similar size, for obvious reasons.

2.2.4 The Method of Unconnected Sets.

Working from an initial (perhaps arbitrary) placement, Steinberg (53) removes from the board a maximal unconnected set of components - i.e. a set with no mutual interconnections, to which no other component could be added without destroying this property. The placement of any member of such a set is, by definition, independent of the placement of any other member of the same set. Thus one can evaluate (in terms, for example, of total wirelength) all possible placements of each member of the set. Steinberg then uses Munkres' assignment algorithm (42) to solve the resulting linear assignment problem. This is repeated for all unconnected sets until no further improvement in the objective function can be obtained.

Rutman (51) extends this method by allowing interchanges between members of the connected sets as well (c.f. the permutation procedure of 2.2.3), and by using a more realistic objective function (i.e. geometrically more accurate). Rutman also concentrates on minimising the lengths of the longest wires initially, thereafter minimising the total wirelength, thus tending to reduce the deviation in wirelengths and, hopefully, leading to improved "wireability".
These methods have the disadvantage of being restricted to the placement of packages of similar size, but otherwise perform well in minimising the objective function efficiently.

2.3 Conductor Routing Methods.

2.3.1 Maze Running and its Derivatives.

The most frequently quoted routing algorithms are those based on the maze running technique developed by Lee (34), being a cell-marking method based on an exhaustive step-wise search of all possible routes between a given start point and target. The original method did not distinguish between differing routes of similar length, as it accepted the first (shortest) route generated by the algorithm regardless of its complexity, or its effect on subsequent routes. The major disadvantages inherent in the method are the large storage requirement (or the low speed if many cells are packed into one word of storage), the slow speed due to the simultaneous, and largely superfluous, growth of all possible routes in a "wave" moving out from the start cell in all directions, and the total lack of lookahead - i.e. the inability to foresee the effect of an accepted route in terms of congestion at a later stage. Several modifications of this method have been developed which overcome, to a large extent, the first two of these three drawbacks.

Akers (1) has described a cell-marking technique requiring only 2 bits of storage per cell, for a slight increase in the complexity of re-tracing a successful route, and a substantial increase in the time spent in cell-addressing, while Majorani and
Lerda (36) have speeded up the process by constructing lists of marked (admissible) cells, thus avoiding multiple scanning of cells within the "wave".

Perhaps the most significant extension is that due to Mikami and Tabuchi (41), who, in dealing with double-sided boards, use "rays" (horizontal or vertical lines) rather than cells. A "ray" or line descriptor is thus three co-ordinates, regardless of length: \((x_1, y_1, y_2)\) or \((y_1, x_1, x_2)\). This results in a considerable saving in storage space and processing time, and also yields the simplest routes first (i.e. those with fewest bends or through-plated holes), rather than the shortest.

Geyer (19) has further extended this technique to deal with multilayer boards and has reduced the required processing time by restricting the initial search for a path to the rectangle formed with the start point and target as diagonally opposite corners, or a limited extension of this rectangle.

Thus, although the speed and storage problems have been largely overcome, there still remains the major problem: lack of lookahead.

2.3.2 **Routing with a "Stepping Aperture"**

In an attempt to avoid the inherent weakness of the track-by-track maze running approaches, Lass (32) has described a method in which all of the tracks (or parts of tracks) in a given region of a board are routed simultaneously, the region being defined by a hypothetical "aperture" which is "stepped" over the whole board surface.
Each track is given a planned routing prior to the aperture routing phase (this planned route being a near optimal route if conflicts with other tracks etc. are ignored). Thus, while dealing with the detail routing within the aperture, a measure of lookahead is available, allowing some of the global properties of the layout, such as track density, to be considered. The usefulness of this lookahead is limited only by the accuracy of the initial planned routes.

The implementation described by Lass has a number of somewhat arbitrary and unnecessary restrictions, such as the limited form of planned routes, but nonetheless appears to offer distinct speed and storage advantages as compared with maze running, in addition to overcoming, at least partially, the lookahead problem.

2.4 Topological Methods based on Graph Planarity.

There are a number of ways in which an electronic circuit can be represented by an abstract graph, the two basic representations equating the nodes of the graph with electrical nodes (nets; sets of points at a common potential) or with components. In the former case the branches of the abstract graph will correspond to 2-terminal components, and in the latter to conductors. For present purposes we shall consider the former representation, and designate it the "circuit graph".

2.4.1 Graph Planarity in Single-sided Board Layout.

In order to design a single-sided layout for a given circuit, one requires that the circuit graph be planar (or have a limited
number of "permissible non-planarities", as described below), thus guaranteeing a topologically feasible layout. Bader (2) and Fisher and Wing (16) have described procedures for generating planar representations of maximal planar subgraphs of such graphs. Rose (49), by extending the circuit graph to include subgraph representations of multi-terminal components, has generalised this, and has developed a heuristic technique for finding a planar subgraph of the circuit graph. In general, such a circuit graph will not be planar, and Rose describes a technique for "planarising" such a graph by permitting a certain class of non-planarity, corresponding to the bridging of a conductor by a component. The resulting graph is described as "pseudo-planar". The topology of this pseudo-planar circuit graph is then used as the basis for the construction of a layout.

The major disadvantage in these methods lies in the fact that a topologically sensible solution (e.g. minimum number of permissible non-planarities in the insertion of a non-planar branch in the planarising process) may be geometrically inferior to one which is, topologically, less simple. Also, though a solution is guaranteed topologically if a pseudo-planar graph can be generated which includes all of the circuit graph, a real solution (i.e. a physical layout with a defined geometry as well as topology) can only be guaranteed in the case of unlimited board area being available. There is one other inherent drawback, in that it is generally assumed desirable to extract a maximal planar subgraph of the circuit graph initially, thereby minimising the amount of "bridging" of conductors by components, and so to some degree the
total board area occupied is maximised (since, to a first approximation, the conductor area will be constant, and the above procedure tends to avoid superimposing the conductor and component areas, thus increasing the total area requirement).

Thus where non-topological methods fail by separating the task into independent placement and routing phases, the above method fails by separating the task into topological and geometric (layout) phases, again independent. Again the solution requires lookahead from the topological to the geometric phase, or feedback in the reverse direction, to avoid this shortcoming. This could, for example, be achieved by laying out the planar portion of the circuit prior to the insertion of the non-planar branches, thus providing some ability to include geometric information during the planarisation phase. Such a process could be enhanced if it were made iterative, updating the layout with each modification (addition) to the topology, although this could become very time-consuming.

2.4.2 Graph Planarity in Multi-layer Board Layout.

Kodres (29) has developed a method for the layout of double-sided boards, based on the factorisation of the circuit graph into two planar subgraphs whose union is the circuit graph. The word "union" is used here in a specialised sense, as it includes superpositioning, since component pins occupy geometrically identical positions in each layer. Factorisation is achieved through the use of graph colouring theorems, as for example in Ore (45), and theorems relating to the "thickness" of a graph, as in Hobbs (22), and certain operations on the graph, such as those corresponding to
the bridging of conductors by component bodies, the end result being a biplanar factorisation of the graph. For boards with more than two layers, efficient graph colouring techniques are not available, though non-optimal solutions could be obtained. Having factorised the graph, Koclres then uses a piecewise-linear transformation to map the graph into the rectilinear cartesian co-ordinate system of the real board. This transformation is such as to limit the usefulness of this approach to circuits containing only packages of similar size, though the user could assign discrete components to groups of such a size, in the input stages.

This procedure, like those for dealing with single-sided boards, suffers from having no feedback from the layout phase to the factorisation phase, and since it deals with double-sided boards, this is a potentially greater source of inefficiency in achieving a high packing density.

Wiendling and Golomb (56), in considering the layout of hybrid circuits (using unencapsulated i.c. chips and deposited rather than etched conductors, but with essentially similar design requirements to those for printed wiring boards), propose a method, based on graph colouring theorems, for minimising the number of layers required to implement a given circuit, and for establishing the planar topology for each layer. This is done by determining all of the maximal planar subgraphs of the circuit graph (not necessarily disjoint subgraphs), and finding sets of these subgraphs whose union is the complete graph. That set with fewest subsets (i.e. fewest maximal planar subgraphs, corresponding to fewest colours in graph colouring
algorithms) corresponds to the optimum solution. No technique is
described for transforming the resulting set of planar graphs (one
per layer of the board) into a suitable coordinate system for
physical implementation.

2.5 **Highly Interactive Layout Procedures.**

A number of programs have been developed which can best be
described as "automated drawing-board" facilities, in which the
computer merely acts as a storage device, sometimes applying
dimensional checking and checking for consistency with the stored
circuit, but in which the user designs manually with the aid of a
graphical output device (c.r.t.) and light-pen, or some similar
combination.

Programs such as that of Koford, Strickland et al (30), and
Cullyer, Tubbs and Stockton (13) for the layout of hybrid thin-film
circuits fall into this category.
Component Placement through Graph Partitioning.

The major shortcoming in the various approaches to the automatic layout of printed wiring boards outlined in Chapter 2 lies in the artificial division of the problem into two or more independent phases, while an optimal solution requires that these phases be highly interdependent. This is true whether the division is into component placement and conductor routing phases, or into topological and geometric (placement/routing) phases.

Since the processing cost incurred by extending existing methods to include lookahead or feedback (conceptually identical mechanisms) between these phases would appear in general to be prohibitive in terms of providing a cost-effective design aid, the only remaining course is to establish more meaningful design aims for the first phase.

The approach to be described retains the division into component placement and conductor routing phases, while attempting to provide a greater degree of lookahead at component placement time through consideration of the most significant global property of the circuit, namely connectivity. A brief description of this placement procedure has already been published by the author (see Appendix 7). Intuition tells us that one of the properties of a "good" placement is that highly connected components will be placed close to each other. This leads us to a hierarchical approach: the component placement is defined globally by consideration of the circuit connectivity, the more common "minimal total wirelength" criterion being used only at the local level. Component placement is thus divided into
three distinct phases, namely the partitioning of the circuit into maximally internally connected (and therefore minimally interconnected) clusters of components, the assignment of each cluster to a geometrically defined region of the board, and the detailed placement of the components of a cluster within the board region allotted to that cluster.

This procedure would appear, at least superficially, to correspond to that used by a draughtsman more closely than any of the methods described in Chapter 2, in that the draughtsman tends to isolate natural "clusters" of components, placing each cluster according to its overall connectivity with the remainder of the circuit, and dealing with the detailed placement and routing of each cluster with reference only to local constraints.

LoDato (38) has shown, by inference from the statistical properties of graphs, that while "minimum total wirelength" is a good local criterion in a placement algorithm, it is poor as a global criterion, and further, that coupling a partitioning algorithm (to provide global control) with a minimal length algorithm (for local control) gives a significant increase in wireability over algorithms based on minimal length alone.

3.1 The Abstract Graph Representing the Circuit.

Since, for the purpose of component placement, we are concerned with the interconnections between components, we shall use the second of the two graphical representations of a circuit outlined in 2.4, in which the nodes of the abstract graph correspond to the components of the circuit. An electrical net (conductor) joining only two
component pins is represented by a branch of the abstract graph incident with the nodes corresponding to the appropriate two components - these are essentially distinct nodes as self-connections are of no relevance in placement. For a net joining n pins where \( n \geq 3 \), however there are \( n^{n-2} \) distinct trees, each with \((n-1)\) branches, any one of which would serve to interconnect the n pins, assuming the simple case in which only direct pin-to-pin connections are allowed. Which of these trees would ultimately be used in routing is indeterminate prior to the completion of the placement phase. Thus, in order to avoid making unnecessary arbitrary decisions, the abstract graph includes for any such n-terminal net the \( n(n-1)/2 \) branches forming the complete graph on these n terminals. As we are interested in the connectivity of components, and not in circuit details, there is no reason to distinguish between different electrical nets connecting any given pair of components. The number of nets common to any two components can be indicated by the simple expedient of weighting the appropriate branch. Thus any two nodes whose corresponding components share n common electrical nets would be joined by a single branch of weight n in the abstract graph. It will be readily seen that the abstract graph so defined can have no loops and no parallel branches, and is thus simple (in the graph-theoretic sense).

A trivial example of a circuit and the corresponding circuit graph (the abstract graph referred to above), showing the appropriate weights on its branches, is shown in Figure 3.1 (unity-weight branches are unmarked).
3.2 Partitioning the Circuit Graph.

To avoid ambiguity, the term "partition" will be used in relation to the circuit graph, while the term "cluster" will denote the set of components corresponding to a partition of the graph (i.e. a subset of the set of nodes of the graph).

Having established the desirability of partitioning the circuit graph, thereby generating a set of minimally interconnected clusters of components, we must now consider the specific objectives of such a procedure in terms of the number and size of the clusters into
which a given circuit should be broken. In order to do this we must also consider the "mapping" to be used in allocating clusters to specific (i.e. geometrically defined) regions of the board.

The obvious, and perhaps optimal, approach would break the circuit into an arbitrary number (with a specified minimum value) of clusters of arbitrary size with the sole objective of creating a minimal-cost partitioning. These clusters would in turn be assigned to specific board regions of arbitrary shape in such a manner as to minimise the total wirelength of the cluster interconnections. This, however, was considered to be unnecessarily complex in view of the excessive number of permitted degrees of freedom. Consideration of the general features of double-sided board technology led to the simple and easily defined scheme outlined below.

3.2.1 The Allocation of Clusters to Slots on the Board.

It has been found from experience (see, for example, Scarlett (52), p. 109) that for double-sided boards a conductor pattern favouring orthogonal routes on the two board surfaces, with the insertion of through-plated holes as necessary, will allow a higher packing density (no. of packages of given size per unit board area) than will the allocation of complete electrical nets, or even single pin-to-pin connections, to individual surfaces of the board. It should, however, be noted that owing to the large "area requirement" of through-plated holes, this bias swings gradually to the other extreme as the number of board layers increases, the break-even point being some function of circuit complexity and geometric detail (such as the ratio of through-hole land diameter to track width).
restriction of routes to the use of mainly one axis per board surface is also desirable from the manufacturing viewpoint as it aids the wave-soldering process commonly used. This it does by avoiding the formation of solder "bridges" between tracks lying at right angles to the direction of movement of the board over the solder wave, since such tracks can be largely avoided by orientating the board such that most tracks on the "solder-side" lie along the direction of travel.

Thus it would appear reasonable to associate a cluster of components with a narrow strip of the board, which we shall term a "slot", such that the internal connections of the cluster lie along the major axis of the slot and so are largely restricted to one side of the board, leaving the other side relatively free for the small number (by definition) of potentially more complex cluster interconnections. Specifically, we shall map a cluster onto the board as a single column of packages, perpendicular to the edge connector, as outlined in Fig. 3.2, in which the inter-slot boundaries are shown as dashed lines. For simplicity we shall restrict our description to the case of packages of uniform size, but it will be shown subsequently that the methods employed can be extended to handle non-uniform package size and discrete components.
In order to aid the accommodation of the high connection density within slots, components will be oriented such that they are most "transparent" to conductors along the major axis of the slot, unless the user specifically directs otherwise. The transparency of a component in a given axis is the number of tracks lying in that axis which can pass, unhindered by the component pins, within the limits of the component body in the other axis, expressed fractionally - i.e. as the number of tracks per unit length of the component body.

Given such a mapping of clusters into slots, the determination of the number of clusters and the maximum size of cluster for a given circuit is reduced to a simple geometric exercise involving knowledge of the board and component dimensions and the orientations of components in slots, with the objective of achieving a uniform packing density in each axis of the board.
READ IN CIRCUIT DATA, CHECK FOR CONSISTENCY, AND BUILD DATA STRUCTURE.

ESTABLISH THE NUMBER, K, OF CLUSTERS (PARTITIONS) INTO WHICH THE CIRCUIT SHOULD BE BROKEN.

BREAK THE CIRCUIT INTO K MINIMALLY INTERCONNECTED, APPROXIMATELY EQUAL-SIZED, CLUSTERS (OF COMPONENTS).

DEFINE THE SLOT BOUNDARIES GEOMETRICALLY, AND ASSIGN SPECIFIC CLUSTERS TO SPECIFIC SLOTS, SO AS TO MINIMISE THE TOTAL SLOT INTERCONNECTION COST.

FOR EACH SLOT, ASSIGN COMPONENTS TO SPECIFIC LOCATIONS, SO AS TO MINIMISE THE TOTAL INTERNAL CONNECTION COST (i.e. INTERNAL TO THE SLOT).

ACCOUNTING FOR INTER-SLOT, AS WELL AS INTERNAL CONNECTION COSTS, REFINE THIS PLACEMENT UNTIL IT IS PAIRWISE OPTIMAL (WITHIN INDIVIDUAL SLOTS ONLY).

ALLOW THE USER TO MODIFY THIS PLACEMENT INTERACTIVELY, IF SO DESIRED.

Fig. 3.3 Outline Flowchart for Placement Scheme.
A flowchart outlining the major steps involved in the placement scheme is shown in Fig. 3.3.

3.2.2 Specification of the Partitioning Procedure.

Having established, in outline, the mapping used in relating clusters to geometric regions of the board, and the broad objectives of partitioning, we may now set out in some detail the requirements which must be satisfied by a suitable partitioning procedure.

It must operate on the circuit graph described in 3.1, and be capable of partitioning the graph in such a manner as to yield a specified number, \( n \), of minimally interconnected clusters of components, each cluster being limited in size such that the "length of slot" required does not exceed the appropriate board dimension (see Fig. 3.2).

Where the packing density on a board is low, and the user specifies an unnecessarily large number of slots (in order, for example, to achieve a layout with packages lying on a standard grid), a good partitioning algorithm would tend to utilise a near-minimal number of slots (thereby minimising the total number of inter-cluster connections), so yielding a potentially poor solution in terms of total wirelength because of the inequality in packing density in the two axes. The extreme case would be that in which a small circuit was assigned to a single slot, resulting in a placement comprising a single column of components running the full length of the board, and with a decidedly non-minimal total wirelength. To overcome this hazard, we must stipulate that clusters should be of similar size, where by "size" we refer to the "length of slot" occupied by the components of the cluster.
We further require that the process be efficient in terms of computer storage requirements and speed of execution.

3.2.3 Description of the Partitioning Procedure.

Kernighan and Lin (27) have described a heuristic procedure for partitioning graphs. Their procedure solves the combinatorial problem of dividing the nodes of a graph, with costs (weights) on its branches, into subsets no larger than a given maximum size so as to minimise the sum of the costs of all branches cut.

The basic "unit" of their procedure is the refinement of a 2-way starting partition of a set of 2n nodes, resulting in two minimally interconnected subsets of size n. This is extended to the partitioning of kn nodes into k sets of size n by starting with some (perhaps arbitrary) k-way partition and refining this by repeated application of the 2-way refinement procedure to pairs of subsets until the partition is pairwise-optimal.

The starting partition could be generated by a modified version of the basic procedure, either by repeated 2-way partitioning, first of the whole set of nodes, and then successively on the resulting subsets, where k is a power of 2, or else by sequential breakoff in which the set of kn nodes is partitioned into sets of n and (k-1)n, the set of (k-1)n then being partitioned into n and (k-2)n, and so on for all k sets. These starting partitions would then be pairwise-optimised as above. The former method, however, apart from the difficulty of catering for \( k \neq 2^i \), i an integer, has the disadvantage that each step, by maximising the internal connection cost for each subset, makes the next step - partitioning these subsets - more difficult, except in
the special case where the kn nodes happen to fall into k natural clusters of equal size. The latter method has the disadvantage that a bad choice made early in the sequence can adversely affect all subsequent partitions, this being most dangerous where k is large.

In view of the above disadvantages and the fact that Kernighan and Lin claim only a slight average improvement through using the above starting partition mechanisms as against arbitrary starts, together with the cost of implementation, the version of the partitioning procedure implemented employs an arbitrary k-way starting partition.

To satisfy the requirements of 3.2.2 in order that we can handle the "real" problem, as opposed to the abstract combinatorial problem described by Kernighan and Lin, a few changes and additions to their procedure are required, as outlined below.

In their procedure the "size" of a cluster (subset of the set of nodes) refers simply to the cardinality of the subset (i.e. the number of nodes in the subset), while we require a more literal interpretation of the "size" of a cluster as the "length of slot" occupied by the components of that cluster with the components oriented as described in 3.2.1. Thus we shall attribute a property called "size" (as above) to each node (component), and define the size of a cluster to be the sum of the sizes of its nodes.

Kernighan and Lin suggest that a node of size k should be represented by k nodes of size 1 bound together by high-cost branches, but this could be very expensive as the computer storage requirement is proportional to \( n^2 \) and the execution time is proportional to \( n^{2.4} \) approximately, for this procedure, where \( n \) is the number of nodes (see Kernighan and Lin (27), p. 301, for justification of these figures).
To avoid this expense, and retain the ability to handle variable-size nodes, the implemented procedure includes, in the evaluation of the cost of a particular partition, a factor which is a function of cluster size, where the size of a cluster is the sum of the sizes of its associated nodes (as in the foregoing paragraph). This factor is signed to favour exchanges in the refinement procedure resulting in cluster sizes nearer to equality, and vice-versa. A cubic polynomial is used to generate this factor, the coefficients being chosen such that the "plateau" near the origin, as depicted in Fig. 3.14, permits a reasonable variation in cluster size without significantly affecting the primary objective of minimising the cost of branches cut, while providing increasingly strict control, regardless of the "goodness" of the resulting partitioning, outside of that range.

The function used to determine this factor, termed "COST", for a given exchange, is:

\[ \text{COST} = a \times \text{RATIO} + b \times \text{RATIO}^3, \]

where \( \text{RATIO} = (\text{RATIO}_B - \text{RATIO}_A)/\text{RATIO}_A, \)

and where \( \text{RATIO}_B \) and \( \text{RATIO}_A \) are the ratios of the cluster sizes before and after the exchange being considered, expressed as a fraction, \( 0 < \text{RATIO}_A / \text{B} \leq 1.0. \)

The coefficients "a" and "b" are functions of the initial cluster interconnection cost.
In this quadrant COST is negative, encouraging any exchanges.

In this quadrant COST is positive, discouraging any exchanges.

Fig. 3.4 The "COST" of an Exchange as a Function of Cluster Size Ratio

As we are dealing with nodes of variable size and clusters of similar, but not necessarily equal, size, there is a need to introduce "slack" into the problem in the form of a small number (2 as implemented) of unconnected (degree zero) nodes, of zero size, added to each cluster, thereby permitting exchanges involving unequal numbers and/or sizes of components in the cluster refinement procedure (since these "dummy" nodes are not associated with "real" components). It should be noted that this use of "slack" does not correspond to that proposed by Kernighan and Lin.
In this section, the 2-way refinement procedure, the kernel of the method due to Kernighan and Lin, has been considered as a "black-box". A detailed description of the basic algorithm is to be found in Appendix 6, and the implementation details, including the foregoing extensions, are to be found in Chapter 6.

3.3 The Assignment of Clusters to Specific Slots.

Given a particular k-way partition of a circuit graph, defining the subsets of the set of components which have to be assigned to individual slots, and a set of k slots on the board, as depicted in Fig. 3.2, we have to solve the problem of assigning specific clusters to specific slots in order to achieve minimal total wire-length. For this purpose no generality is lost by considering each cluster as a single element.

Thus the problem can be re-stated as that of finding a linear ordering for a set of k interconnected elements, with cost matrix \( C_{k,k} \) corresponding to a connection cost \( c_{i,j} \) between the \( i \)'th and \( j \)'th elements, such that the total interconnection cost is minimised. Assuming unity spacing between adjacent elements, the total cost is given by:

\[
\text{COST} = \sum_{k=1}^{k-1} \sum_{j=1}^{k-j} c_{i,i+j}
\]

In view of the low interconnection costs involved (resulting from the partitioning procedure) it was considered unnecessary to guarantee an optimal solution for this assignment problem, and the simple heuristic scheme outlined below has been implemented.
Operating on the cost matrix, \( C(k,k) \), that element \( i \) with the two associated costs \( c_{i,n} \), \( c_{i,m} \), whose sum is highest for all \( i \neq k, \: n \neq k, \: m \neq k \), \( i,n,m \) distinct, is selected, and the ordering \( n,i,m \) established for the three appropriate elements. Given the ordering \( x,n,i,m,y \), contenders for either the \( x \) or \( y \) location are considered from among the unordered elements, that element being selected which is most highly connected to those already placed and that location \((x \text{ or } y)\) being selected to minimise the total cost for all of the ordered elements. This process is repeated, adding one element at a time to either end of the "chain" of ordered (with respect to each other, that is) elements, until all of the elements have been ordered. The elements, thus ordered, are assigned to specific slots on the board.

3.4 The Assignment of Components to Specific Locations in a Slot.

For each cluster, allocated to a slot as in 3.3, we must establish an optimal placement of the components within the slot. As indicated in 3.2.1, this placement will simply be a linear ordering of the components along the major axis of the slot such that the total wirelength is minimised. Thus, as in 3.3, the problem is that of establishing such a linear ordering for the set of \( n \) interconnected elements (this time, components), with cost matrix \( C(n,n) \) corresponding to a connection cost \( c_{i,j} \) between the \( i \)'th and \( j \)'th elements, such that the total interconnection cost is minimised. In this case, however, there are two further constraints, in that there may be "fixed" components at either or both ends of the slot, corresponding to the edge connector(s), and further, that some, if not all, of the elements will have connections external to
the cluster which, although minimised in number by the partitioning procedure, must be considered as regards their effect on total interconnection cost in the placement procedure.

In this case the elements to be placed are, at least to some extent, maximally interconnected, this being the primary objective of the partitioning procedure and so the "naive" algorithm used to solve the corresponding cluster placement problem is unlikely to yield satisfactory results. For this reason an initial placement algorithm of the "sequential" type (see 2.2.2) has been implemented, in which the next component to be placed is that which is most highly connected to those already placed, and the location selected for it is that free location for which the total internal (to the cluster) interconnection cost is minimised. The start point for the procedure is the fixed edge connector, which is generally highly connected to the components of the slot. Connections external to the cluster are entirely ignored for the purposes of this initial placement. The resulting placement is subsequently refined by a 2-way interchange procedure, as described in 3.5, which takes inter-cluster connection costs into account.

It should be noted that this placement procedure arranges the packages centrally in the slots in the x-axis (parallel to the slot minor axis), and on a basic 0.550" grid in the y-axis (parallel to the slot major axis), in both cases rounded such that the component pins all lie on a 0.100" grid relative to the board datum. The choice of the 0.550" grid in y is based on its being the "highest common factor" accommodating, at a reasonable packing density, all commonly used package sizes. The program cannot currently handle
discrete components, but when this facility is added the above "coarse grid" will still serve, as discrete components will be clustered into groups, those groups being handled like packages at this stage. A more detailed discussion of this proposed extension is to be found in 10.1.3.

3.5 Refinement of the Component Placement Within a Slot.

Given the initial placement of the components of a cluster within the appropriate slot, we require to optimise that placement so as to minimise the total interconnection cost of all connections to all elements (components) of that cluster. The procedure adopted is a combination of the 2-way interchange and "centre-of-gravity" methods (see 2.2.3 and 2.2.1 respectively).

Given two components, A and B, adjacent in a slot, as depicted in Fig. 3.5, where \( y_{pn} \) is the y-coordinate of pin \( n \) and \( y_{cn} \) is the y-coordinate of the centre of gravity of the electrical net of pin \( n \), and given that forces are proportional to distance, then the net upward force on A is given by:

\[
\text{UPWARD FORCE} = k \sum_{n=1}^{k} (y_{cn} - y_{pn}), \quad k \text{ being the number of pins of A.}
\]

If A is allowed to move freely then it will reach equilibrium after a move \( \delta y_A \) upwards, where \( \delta y_A \) satisfies:

\[
\sum_{n=1}^{k} (y_{cn} - y_{pn} + \delta y_A) = 0
\]

\[
\therefore \quad \delta y_A = \frac{\sum (y_{cn} - y_{pn})}{\sum \text{(connections)}}, \quad \text{given that the } y_{cn} \text{ values remain unchanged by the move, which is true to a first approximation.}
Fig. 3.5 Forces acting on a Component in Placement Refinement

The components A and B will be swapped if and only if (iff) their equilibrium positions, under the action of the forces described above, are reversed in the slot ordering as compared with their initial positions, i.e.

\[ \text{iff } \delta_{y_A} - \delta_{y_B} > \delta_{AB}. \]

Thus for a given slot, all adjacent pairs of components are considered as candidates for swapping, the highest mutual cost pairs being checked first (as any swapping of such pairs is likely to have the most far-reaching consequences), and the refinement progresses pairwise until a complete pass has been made without any swaps being made, thus guaranteeing that slot to be pairwise optimal. This process is repeated for all slots, but it should be noted that since inter-slot connections are taken into consideration, a swap in any
slot may require further refinement of any or all of the slots which share electrical nets with the swapped component pair, and thus to guarantee overall pairwise optimality, a complete pass over all of the slots must be made in which no swaps take place.

There is one further facility in the implemented procedure which allows maximal freedom in the utilisation of the available board area and assists in reducing that component of the total interconnection cost which is due to the y-component of the inter-slot connections. Briefly, it consists in the addition of sufficient "dummy" components, with no interconnections and occupying 1 grid unit (0.550") in the y-axis, to fill each slot, thus permitting use of the initially "empty" parts of the slot, if any exist.

3.6 Interactive Modification of Component Placement.

As has already been stated in 2.1, the human designer has a most impressive ability in the field of pattern recognition, and in particular is capable of extracting the essential information for a given task from a pictorially presented "overview" of a situation. Although the placement program has been designed with the handling of global properties in mind, it is nonetheless essential to provide the user with a pictorial representation of the program-generated placement, and to provide him with appropriate "tools" in order that he can modify this placement if he feels it desirable to do so.

Such a scheme also provides an interim solution to the problem of the insertion of specific constraints, such as the requirement that a given component should occupy a given board location (for example, a potentiometer which must be accessible for adjustment), which
currently the program cannot handle, although it is proposed that the program be extended in order that such constraints would be handled automatically (see 11.1.1).

As implemented, graphical interaction using the 340 display and a light-pen is utilised, although "off-line" interaction using a hard-copy output device (plotter) and keyboard input would be feasible for this process, as one would expect all required changes to be correctly anticipated in a single output/input cycle.

The program displays the automatic placement on the 340 display as a set of appropriately scaled and labelled package outlines, with the option of displaying the basic coarse grid (slot width in x, 0.550" in y), and a similar option on the display of the straight-line pin-to-pin connections. For this purpose it is assumed that all of the pins of a given net lying within a given slot will be joined in order of ascending y-coordinate (i.e. in order along the major slot axis), while inter-slot connections will be as nearly horizontal as possible (i.e. minimal length), given that they must go direct from pin-to-pin. The justification for the selection of this particular form of spanning tree appears in 4.1.1. Also as an aid in assessing the merit (or otherwise) of any changes made, the estimated manhattan wirelengths (i.e. \( \Sigma (\delta x+\delta y) \)) for all pin-to-pin connections are displayed for the current placement and for the initial (i.e. prior to any interactive modification) placement.

In the display of pin-to-pin connections and also in the estimation of total wirelength, all power and ground connections are omitted, as it is assumed that these will connect to all packages, and so any changes made to the placement will result in a very minor net effect in such connections.
The user, with the aid of the light-pen, can select specific modes of operation (specifically, the user manipulates the program "transfer of control" in such a manner as to select blocks of user code which execute the required functions, possibly with the aid of further interaction). The facilities currently implemented allow the user to move a selected component to a new position by "dragging" it with the light pen to that position, or to swap two selected components, the program undertaking the necessary rounding onto the grid, thus only requiring an approximate indication of the new position from the user. Rotation of components (through \( n \times 90^\circ \), \( n \) an integer) is planned, but not as yet implemented. The user can also initiate "dumping" of the current display file onto a disc file which can subsequently be processed to generate plotter commands for driving the Calcomp drum plotter, thus providing a "hard-copy" of the dumped picture.

Figs. 3.6 and 3.7 show the placement of one of the circuits used in testing the program, prior to and after interaction, respectively.
The placement of Component after Placement Interaction.
Conductor Routing.

As indicated in 2.3.1, maze running and its various derivatives have one major, and unfortunately inherent, deficiency - tracks are processed sequentially without any form of lookahead. This inevitably results in an ever increasing probability of failure to complete a given route as the number of completed routes on a given layout increases. Furthermore, those routes which are completed late in the routing sequence are likely to be unnecessarily long and/or complex, which can be undesirable in terms of electrical function even if a complete layout is achieved.

In order to avoid this problem it is necessary to adopt a scheme permitting lookahead, such as the "stepping aperture" approach due to Lass (32), outlined in 2.3.2. In this approach a planned route is generated for each electrical net, regardless of any conflicts arising between such routes (e.g. where 2 or more tracks contend for the use of one part of the board surface). An imaginary "aperture" is then stepped over the board, conflicts within the aperture being resolved at each step by perturbation of the planned routes. An attempt is also made to minimise future conflicts, as predicted from the route plans, by re-routing within the aperture where this is feasible. This method thus avoids the complete lack of lookahead of the maze running methods, although the usefulness of the lookahead is limited in proportion to the perturbation of the route plans required in conflict solving.

The scheme implemented is, in essence, similar to that described by Lass, though entirely different in detail. Before considering it
in greater detail, however, it is necessary to examine one particular aspect of the routing problem, and its consequences - namely, the scale of the problem.

4.1 The Subdivision of the Conductor Routing Problem.

One of the problems met in the implementation of any conductor routing program is that of the storage of the necessarily large volume of geometric data in an associative data structure in which sufficient relationships between datums (data items) are expressed explicitly to allow of its being reasonably fast in operation - the usual speed/storage trade-off. Given a suitable data structure, such as that used in this program implementation (described in 7.1.3 and Appendix 3), the storage requirement for a board of 50 i.c's, interconnected by some 200 nets, and using tracks on a 0.050" grid, might typically be 100K bytes (1 byte = 8 bits) for the geometric information alone. As it is necessary to solve realistic problems on realistic machines - i.e. those commonly available within industry - it is unreasonable to contemplate an in-core data structure of this size in addition to the user program and other user data, such as the circuit data. To reduce the in-core storage requirement it is necessary to segment the geometric data structure, preferably without incurring the time overheads of a virtual memory system or of an arbitrary paging of the structure (see Van Dam and Tompa (55) for a discussion of segmentation and paging schemes).

Fortunately such a division already exists, in essence, insofar as the circuit has been broken into minimally interconnected clusters of components, each assigned to a specific slot on the board. Thus
we might consider the routing of each slot as a subproblem of the whole, with a corresponding division of the geometric data structure into segments, each associated with a particular slot. If these subproblems were independent then only one data structure segment would be required in-core for the solution of any given subproblem (the other segments being held in backing-store), so avoiding the potentially large amount of segment swapping in the handling of all of the inter-slot connections. To achieve this independence, the detailed geometry of each of the inter-slot boundaries must be established.

For the sake of brevity, a few terms will be defined before going on to examine the problem of inter-slot boundary geometry. In the placement phase, the nodes of the circuit graph were equated with components. As routing is concerned with individual nets, it is now necessary to re-define a **NODE** as a component pin. A **SPANNING TREE** is a set of branches joining each of a given set of nodes, with no loops and no parallel branches. A **STEINER TREE** is a special case of a spanning tree in which extra nodes, referred to as **STEINER POINTS**, may be added, in order to minimise the total branch length. A **STEINER-MINIMAL TREE** is a Steiner tree of minimal total branch length, spanning a given set of nodes. It is assumed here that a Cartesian coordinate system is used, and that, as in Hanan's formulation of Steiner's problem (21), rectilinear distance is defined with respect to this coordinate system. Fig. 4.1 (a) depicts such a Steiner-minimal tree with orthogonal branches (see Chang (9) and Hanan (21)). In the case of a net spread over a number of slots of the board, as in Fig. 4.1 (b) the intersections of the inter-slot boundaries and the
Steiner tree (which may or may not be minimal) will be termed **DUMMY PINS**. Thus, for the planning of routes within a slot, dummy pins, after assignment to specific coordinates, will be considered as nodes, along with component pins and Steiner points. That part of a Steiner tree lying within a slot, and bounded at the slot edges by dummy pins (if any), will be referred to as a **SUB-TREE** of the Steiner tree.

**Fig. 4.1 Steiner Trees.**
4.1.1 The Geometric Definition of Inter-Slot Boundaries.

The geometric definition of an inter-slot boundary is simply the assignment of all dummy pins, one for each connection crossing the boundary, to specific board coordinates. Before this can be done, the topology of each spanning tree crossing that boundary must be established in order to determine, for each net, the number of dummy pins lying on that boundary.

Fig. 4.2 (a) depicts the complete graph on 4 nodes (component pins), with two nodes in each of two adjacent slots, as was used in the placement procedure (see 3.1), while Figs. 4.2 (b) - (e) show four of the 16 spanning trees ($16 = n^{(n-2)}; n = 4$). The inter-slot boundary is shown in each case as a dashed line separating the two pairs of nodes.

Fig. 4.2 The Complete Graph and 4 Spanning Trees on 4 Nodes.
It might be considered that the ideal solution to the problem of inter-slot boundary definition - both topological and geometric - would be the generation of a Steiner tree for each net, regardless of how many slots it spread over, thus defining the coordinates of the dummy pins in terms of the intersections of the Steiner tree branches with any inter-slot boundary. This would also furnish the route plans required for aperture routing. Where the number of nodes, \( n \), is greater than 5, however, the generation of Steiner trees is non-trivial (see, for example, Yang and Wing (58)), while for small \( n \) the partitioning procedure results in a high probability of all \( n \) nodes being in one slot. It will also be shown in 4.3 that Steiner sub-trees are not necessarily useful route plans owing to technological considerations. Thus, in view of the high cost of generating Steiner trees, and the dubious advantages offered, the above approach was replaced by the simpler heuristic scheme outlined below.

For a given net, with nodes in more than one slot, as for example in Fig. 4.2, only one inter-slot connection is permitted between each of the adjacent pairs of slots, thus requiring a spanning tree of the general form of Fig. 4.2 (b) or (c), neither (d) nor (e) being permitted. This, for a given partitioning of the circuit, minimises the total number of inter-slot connections, so reducing the possibility of conflicts for specific dummy pin locations. Thus, in fixing the coordinates of a given dummy pin, one is dealing with one of the four cases depicted in Fig. 4.3, in which varying degrees of overlap in the \( y \)-axis exist between the sub-trees in the two slots. The \( x \)-coordinates of the nodes (component pins) are irrelevant to this discussion, and so the simple case is considered in which each pin lies centrally in \( x \) in the
appropriate slot. Fig. 4.3 indicates $y_{\text{max}}$ and $y_{\text{min}}$, the upper and lower bounds of the preferred y-coordinate for the dummy pin, for each case.

Where, as in Figs. 4.3 (a), (c) and (d), $y_{\text{max}} \neq y_{\text{min}}$, the choice of the dummy pin y-coordinate is made so as to use the least densely packed part of the permitted range, $y_{\text{min}} < y_{\text{dummy}} < y_{\text{max}}$ (least dense, that is, in terms of other dummy pins). Where no location is available within the preferred range, that free location nearest to it is used.

Thus, by the division of each Steiner tree into an appropriate set of sub-trees, joined at slot boundaries by dummy pins, and the assignment of specific coordinates to these dummy pins, a subdivision of the whole routing problem into a set of independent subproblems, one for each slot, is achieved. A segment of the geometric data structure is associated with each slot, and so with each subproblem.
4.2. Outline Description of the Routing Scheme.

The desirability and feasibility of problem subdivision having been established in 4.1, the overall approach to the routing problem assumes the form outlined in Fig. 4.4.

4.3 Sub-Tree Generation in Route Planning.

Given the problem subdivision outlined in 4.1 and the geometric "fixing" of dummy pins as described in 4.1.1, the route planning problem is reduced to that of finding a suitable Steiner tree spanning
CREATE DATA STRUCTURE "SEGMENT" FOR EACH SLOT, WITH DESCRIPTION OF GEOMETRICALLY "FIXED" ITEMS (COMPONENT PINS)

BREAK DOWN SPANNING TREES INTO SUB-TREES, ONE PER SLOT, AND ASSIGN COORDINATES TO THE RESULTING DUMMY PINS.

INSERT, INTERACTIVELY, ROUTE PLANS FOR POWER/GROUND NETS.

FOR EACH SLOT DO:

CREATE ROUTE PLANS FOR ALL NETS, EXCEPT POWER/GROUND, IGNORING CONFLICTS

MINIMISE CONFLICTS BETWEEN ROUTES USING THE STEPPING APERTURE APPROACH

INTERACTIVELY COMPLETE AND/OR MODIFY THE AUTOMATICALLY GENERATED ROUTING

EXIT

Fig. 4.4 Outline Flowchart of Routing Scheme.
the set of nodes (component pins and dummy pins) of a given net in a given slot. Such a tree will be referred to simply as a sub-tree. A "suitable" sub-tree is one which is near-minimal length, and which has certain technology-dependent characteristics, as outlined below.

Key:  

- **O** Component/Dummy pins
- **X** Steiner points

**nodes of sub-tree**

*Fig. 4.5 Some Steiner Trees for 3 Sets of Nodes.*
In Fig. 4.5 two alternative Steiner trees are shown for each of three sets of nodes: (a) and (b), (c) and (d), and (e) and (f). Figs. 4.5 (b), (d) and (f) – the lower three – are Steiner-minimal trees, while (a) and (e) are non-minimal and (c), though minimal, is unnecessarily complex. Despite this, (a), (c) and (e) are preferred as route plans for the following reasons. The program design, so far, tends to yield a high interconnection density in the y-axis and so, to avoid congestion, it would seem desirable to simplify, as far as possible, the y-component of sub-trees. Thus, for example, (a) is preferable to (b). Also, the program will frequently have to deal with slots containing a column of similar packages, in which case there would be a high probability of conflict between a route travelling directly in the y-axis, from a pin, with other pins at the same x-coordinate. Thus in all three of the illustrated cases the use of an intermediate x-coordinate for the y-component of the sub-tree is preferred. The scheme adopted for the generation of sub-tree route plans is outlined in the flowchart in Fig. 4.6.

4.4 Interactive Route Planning for Special Cases.

The topological restriction, permitting only one branch of a given net to cross a given inter-slot boundary, imposed by the "boundary definition" algorithms outlined in 4.1.1, can in certain cases result in a poor overall solution. Such a situation, for example, arises frequently in the case of power and ground nets for a regular matrix of i.c's. The inter-slot boundary definition and route planning algorithms, as outlined in 4.1 and 4.3 respectively,
CREATE List of nodes (component and dummy pins) of sub-tree, in order of ascending y-coordinate. Store data for each node in array IWORK.

Find mean X, the mean of the x-coordinates of all of the sub-tree nodes. Modify mean X if it is coincident with the x-coordinate of a comp't pin.

Any nodes left in list?

Get next node from list in IWORK which has still to be added to the sub-tree, and find the nearest feature to it which is already in the sub-tree. Designate these I and J respectively.

If node I is superimposed on a sub-tree branch, break the branch and add I, otherwise join I to J (J is already a part of the sub-tree) by a simple route, preferring the use of \( x = \text{mean} X \) for any y-component of the route. Where a steiner point (bend, tie) is added to the tree, an appropriate entry is appended to the list in IWORK.

Fig. 4.6 Generating a Sub-Tree Route Plan for a Given Set of Nodes.
generate a route plan, for a simple case of this type, as illustrated in Fig. 4.7 (a). Fig. 4.7 (b) shows an alternative route plan for this net which is superior in that it avoids the heavy use of tracks in the y-axis, where congestion is most likely, this being particularly important where double-width tracks are used for power and ground, as is common practice. Also, the latter solution is electrically superior in that the distance from the edge connector to the most remote part of the net is shorter than that in Fig. 4.7 (a), although this is not necessarily always the case.

![Power/Ground Route Plans for Regular Matrix of i.c's.](image)

Fig. 4.7 Power/Ground Route Plans for Regular Matrix of i.c's.
It was thus considered essential to provide a "tool" permitting the user to insert route plans interactively, allowing control of both topology and geometry. In the implemented version the application of this facility is restricted to the power and ground nets, all others being handled automatically as described in 4.1 and 4.3.

The program, for a given power or ground net, displays the board outline, the slot boundaries and the pins of all components, and adds a marker to each node (pin) of the net to be planned. Fig. 4.8 shows a typical example of the picture displayed prior to any interaction. The user, with the aid of the light-pen, then outlines sub-trees interactively by joining the set of nodes for a given sub-tree, adding dummy pins as necessary, and geometrically fixing such dummy pins at the desired y-coordinate on the appropriate slot boundary. By way of assisting the user, the program adds straight-line node-to-node connections to the picture as nodes are added into the sub-tree. When the user has defined a sub-tree (i.e. listed its nodes - component and dummy pins, and geometrically fixed any dummy pins), the program stores the relevant information. On completion of the interactive definition of all of the sub-trees, the program takes each in turn and generates the appropriate sub-tree route plan, as described in 4.3. Figs. 4.9 and 4.10 show the displayed picture (for the same board as in Fig. 4.8) after the manual entry of one sub-tree, and after the entry of all sub-trees, respectively, for one net.

4.5 Conflict Minimisation Using Stepping Aperture.

So far the division of the problem into a number of independent subproblems, one for each slot on the board, and the planning of
Fig 4.8 The Markers on the Nodes of the Ground Net of CIR16 Prior to Interactive Route Planning.
Fig 4.9 The Ground Net of CIR16 after the Interactive Insertion of one Sub-Tree.
Fig 4.10  The Ground Net of CIR16 after the Interactive Insertion (Planning) of the Entire Net.
routes within each slot, have been considered. It is now necessary to consider the removal of conflicts between tracks, which so far have been largely ignored, for each slot in turn.

4.5.1 The Initial Definition and "Stepping" of the Aperture.

As outlined in 4.3, the route plans are Steiner trees, with branches directed away from the roots. The "root" of a sub-tree in a given slot is arbitrarily defined as that node of the sub-tree occupying the highest y-coordinate of all of the nodes of that sub-tree in that slot. The change at this stage from undirected to directed trees is due entirely to the resulting simplicity in the geometric data structure, and in no way affects the validity of the graph-theoretic model (see Busacker and Saaty (7), pp. 29 ff.). Thus, in following the branches of a tree away from its root, the corresponding geometric movement will be horizontal (left or right) or vertically down within the slot. Since writing this part of the program it has been realised that this precludes certain geometric track configurations, such as a U-shape, but the extra code required to handle such "special-cases" is probably small in relation to the saving through the use of directed trees, though the model is no longer as elegant as it originally appeared to be.

Following this top-to-bottom convention, the aperture, of width equal to that of the slot, will be stepped from top to bottom in a given slot. The top of the initial aperture will be the highest y-coordinate occupied by any root in the slot (the root being the highest part of any given tree). As we are interested in the "features" of the sub-trees, where by "feature" is meant node, bend
or through-plated hole, etc., it would seem desirable to use a "variable-height" aperture, such that the bottom of the initial aperture corresponds to the next-lower y-coordinate occupied by any feature of any sub-tree in the slot (next-lower than the top of the aperture, that is). This avoids unnecessary wastage of time in stepping a fixed-height aperture over featureless regions of the slot, where each successive part of the slot, "viewed" through the aperture, is identical to the preceding part.

On stepping the aperture down, the top of the new aperture is made coincident with the bottom of the old, and the bottom of the new aperture again corresponds to the next-lower feature of any sub-tree in the slot. Fig. 4.11 shows, for part of a slot, the successive aperture positions, given the sub-trees as drawn.
If, in the process of conflict minimisation within a given aperture, a new y-coordinate, intermediate between the top and the bottom of the aperture, is used, then the original aperture, as for example in Fig. 4.12 (a), will be updated such that its bottom corresponds to the y-coordinate of the newly added feature, as in Fig. 4.12 (b).
4.5.2 The Allocation of Parts of Sub-Trees to Specific Board Sides.

So far we have considered sub-trees as two-dimensional: this however is a simplification as two board surfaces are available for routing, and the situation may best be described as "2½-D". The first step, then, in conflict minimisation, is the allocation of parts of each sub-tree to appropriate sides of the board, with the addition of through-plated holes as necessary.

As indicated in 3.2.1, it is desirable to allocate "track segments" - straight-line parts of a track, each bounded by two geometric features - such that orthogonal segments occupy opposite sides of the board. There is, however, a limit to the desirability of such a solution, in that through-plated holes use considerable space on both board surfaces, so that for short track segments the use of the "wrong" board surface is generally desirable. (The solder side is
used mainly for vertical tracks: any horizontal track on the solder side is said to be on the "wrong" side, and vice-versa). Thus, for example, the use of the wrong board side for the crank, as shown in Fig. 4.13 (b), is preferable to the addition of two through-plated holes with the crank on the "correct" side as shown in Fig. 4.13 (a). In the latter case no benefit at all is derived from the addition of the two through-holes: the number of tracks blocked in both axes is one in each case.

The object, then, of the side-allocation algorithm, is to achieve such a degree of orthogonality of the routes on the two surfaces as to satisfy soldering requirements (see 3.2.1) and simplify the routing problem, while avoiding situations such as that depicted in Fig. 4.13 (a).

**Fig. 4.13** Alternative Board Sides for Simple Crank in Track.
To cater for the above, a "permitted span" in the wrong axis is defined to be the distance a track segment may travel on the wrong side of the board before being forced onto the other side, with the consequent addition of through-holes. For a part of a sub-tree this definition is extended such that no part of a sub-tree, bounded by dummy or component pins, and through-holes, may spread by more than the "permitted span" on the wrong side of the board. For example, if in Fig. 4.14, $\delta x_1 + \delta x_2$ is greater than the "permitted span", then one (at least) of the horizontal segments would be transferred to the opposite side of the board. (Since $\delta y > \delta x$, this tree is considered for allocation to the solder side, horizontal track being, therefore, on the "wrong" side).

![Diagram of permitted span on wrong side of board](image)

**Fig. 4.14** The "Permitted Span" on the Wrong Side of the Board.

Side allocation, then, is achieved as follows. Starting from the root of the sub-tree, extract as many branches of the sub-tree as possible without "crossing" any pin or through-hole, and stopping just before the "permitted span" on the wrong side of the board is exceeded...
in both axes. If the process is stopped because the extracted branches are entirely bounded by pins or through-holes, then the set of extracted branches is simply allocated to the appropriate side of the board. If, however, it is stopped because the "next" branch to be extracted would cause the permitted span to be exceeded in both axes, then a through-hole is inserted at the junction of that branch with the extracted part of the tree, and the extracted part allocated appropriately. Further groups can then be extracted with each of the boundary pins/through-holes of the initial group replacing, in turn, the sub-tree root in the above, and so on until the entire sub-tree has been allocated.

In order to take maximum advantage of the lookahead available, it is desirable to defer the side allocation of any given part of a tree for as long as possible - i.e. until the aperture is as near to it as possible, or is actually enclosing it, such that the decisions made in side allocation are based on the most up-to-date lookahead information available. Thus side allocation is undertaken as the aperture progresses down the slot, only allocating as far below the current aperture position as is necessary, rather than as a separate phase prior to aperture routing.

In the implemented program the "permitted span" on the wrong side of the board is a fraction of the slot width (this axis being the more constrictive), thus permitting greater freedom in the use of the "wrong" side where slots are wide. This appears to operate satisfactorily, though it may be argued that the permitted span on the wrong side should be a fixed multiple of the standard track-to-track spacing.
### 4.5.3 The Minimisation of Conflicts Within a Given Aperture.

After the assignment of any unallocated track segments to appropriate sides of the board, an attempt is made to remove any conflicts existing between the track segments, including component pin and through-hole lands, on each side of the board, by modification of the planned routes.

Lass (32) uses a secondary aperture, stepping across the width of the primary aperture, and of width equal to the height of the primary aperture, in the conflict-solving process, thus only looking at a very small portion of the board at any instant. To avoid this, the approach adopted takes complete track segments (or the parts of them bounded by the aperture limits) in each of a succession of categories, and checks for conflicts, "fixing" a segment if it does not conflict with any previously-fixed segment. The first category, for example, to be checked, is that in which a track joins two "features" sharing the same x-coordinate and board side, one at the top and the other at the bottom of the aperture. The ordering of such categories in terms of a "priority rating" is based on an estimate of the cost of not using the simplest available route, and hence the general order from simply defined to complex situations.

By way of lookahead beyond the current aperture (i.e. to lower y), a "profile" of the slot is maintained. This profile, for a given x-coordinate, has magnitude equal to the y-coordinate of the next fixed obstacle (e.g. component pin) below the current aperture, at that x-coordinate. Thus, for example, in a track planned to run vertically down from the aperture to some target, the profile is checked for any
fixed obstacle at the appropriate x-coordinate, and if any exists, a crank is inserted in the route plan to avoid the obstacle. Such checking and modification takes place after conflict minimisation within the aperture as the latter is, at any given time, more urgent.

4.6 Interactive Modification and Completion of the Routing

4.6.1 The Need for Interaction.

So far, a set of procedures based on heuristics has been outlined, whose function is the generation of route plans and the subsequent removal of conflicts between tracks. These procedures, unlike the "topologically" based methods outlined in 2.4, cannot guarantee a 100% conflict-free routing for any given circuit. The "100% guarantee" in the case of the topological methods is dependent, of course, on the successful planarisation of the circuit graph(s), and on there being no geometric restrictions (i.e. unlimited board area available). There are a number of potential causes of failure, prominent among them being the difficulty of defining an heuristic scheme capable of handling every possible topological/geometric configuration. This is also, in part, a matter of economics: the cost of implementing an algorithm to deal with a very rarely-occurring situation may not be justified where an alternative method of handling such situations, as they arise, is available, as with the interaction to be described. The other important cause of failure lies in the deficiencies in the approach adopted, which is necessarily designed to cope with the "average" board and may consequently perform badly in dealing with certain circuits, or alternatively when dealing with certain situations in
any layout.

There is a particular use of interaction, related to the above, in the context of its being a "program design-aid", allowing the evaluation of the automatic algorithms and suggesting the most fruitful areas for further automatic procedures. The latter would correspond to the provision of procedures to deal with those situations most frequently solved interactively and which are amenable to automatic solution.

The major use of interaction, however, is in the provision of "tools" which allow the user to modify (and so complete) the automatically generated layout. It is assumed, at this stage, that component placement is "frozen", as are the coordinates of the dummy pins on the inter-slot boundaries, so that a slot-by-slot approach is adopted.

4.6.2 The Facilities Provided for Interaction.

The graphical and interactive facilities available are divided into two distinct sections, the first of which is the "control" phase. In this, the user is permitted to display, for the entire board, all of the conductors on either or both sides, conflict-free or unrouted, as selected. This provides an overview of the whole board, the interactive ability in this phase being restricted to the selection of the view to be displayed, as above, and to the selection of a specific slot for interaction, and the initiation of interaction for that slot. On termination of interaction with a particular slot, the user can "escape" to the control phase and go on to select further slots for interaction.
Alternatively, on completion of all of the slots, the dumping of the necessary picture descriptions to disc files can be initiated, these files subsequently being processed for plotting.

The second and most important section is that permitting modification of the routing of a particular slot. The first requirement for such interaction is the provision of graphical output such that the user can easily determine the current state of the detailed slot geometry. The program provides the following, which are considered the "minimum essentials":

(i) a picture of the "solder-side" of the slot, showing component pins, through-holes, and all conflict-free track on that side,

(ii) as for (i), but for the "component-side" of the board,

(iii) an "overview", showing (i) and (ii) superimposed, and

(iv) a picture showing only pins and conflicting tracks.

In the implemented program, (i) and (ii) are not drawn for the whole slot, but are drawn at double-scale for a "window" on the slot, which the user may interactively move up and down the slot, thus affording better resolution. For (iii), being an overview, the whole slot is drawn, with some detail omitted (e.g. through-holes), and likewise for (iv). Fig. 4.15 shows a set of pictures, being, from left to right, (iv) (iii), (i) and (ii) as above.
Fig 4.15  The Display of Slot No. 2 of CIRO2, Showing the four views, Prior to Routing Interaction.
The interactive facilities allow the user to identify and move or delete any geometric feature, to insert a new feature of any type on a specified track segment (e.g. adding a bend to a track), and to change the type of any feature (e.g. changing a bend to a through-hole). Also, the user may switch a track segment from one side of the board to the other, and, after conflict removal, cause previously conflicting track segments to be "fixed". Thus, assuming that 100% routing is attainable for the given slot, the user is able to complete any incomplete tracks and modify, if so desired, any of the automatically generated routing.

As implemented, the program gives the user absolute freedom in this phase, including the freedom to join electrically separate points "accidentally", which is obviously undesirable. A necessary addition, then, is a checking process which rejects any user action which would result in a conflict with the stored circuit description.
The description of any circuit, for layout purposes, can be divided into two parts - the components list, and the interconnection data. The former must contain, explicitly or implicitly, geometric descriptions of the listed components. To avoid the need to provide these somewhat bulky descriptions explicitly, a "library" containing the necessary description of each component is held permanently in backing-store (disc), the components list simply relating specific "instances" of a given type with the appropriate entry in the library. This library is designated the "master library", any entry in it being termed a "master definition", or simply "master".

Before going on to consider circuit input, in 5.2, the content and manipulation of the master library must first be considered.

5.1 The Master Library.

5.1.1 The Content of a Master Definition.

The geometric information required for each component, for layout purposes, is listed in Table 5.1, the abbreviated descriptor for each item corresponding to that used in the example shown in Fig. 5.1 (a 14 pin dual-in-line package), and subsequently in this section. A number of potentially useful datums have been omitted, such as the "height" of the component, which would be required in checking clearance between boards in a rack for example, because the program as implemented does not do any such checking, and the subsequent addition of such items is a trivial matter (see Appendix 2).
Referring to Table 5.1 and Fig. 5.1, it will be noted that a specific orientation has been assumed (in defining "x" and "y"): the orientation chosen for the master definition is a matter of convenience (or of convention), as the placement program will rotate an "instance" of a master relative to the orientation in which it was defined, if necessary. The user is only required to be consistent.

It will also be noted that a datum has been assumed, relative to which the pin coordinates are defined. In creating a new master definition, the user supplies the x- and y-outline dimensions along with the x- and y-pin coordinates referred to the datum (x-outline=0, y-outline=0). The program which generates the master library computes the x- and y-pitch from this information, together with the FINSIZE, and adjusts the x- and y-pin coordinates such that they refer to the new datum, (x-pitch=0, y-pitch=0), which is subsequently used throughout the layout programs. The x- and y-pitch may best be thought of as defining the size of a rectangular region of the board which "contains" the component, including the lands (pads) associated with the through-holes for the component pins (leads). The distinction between "pitch" and component body "outline" is particularly significant in the case of wire-ended devices, such as resistors, in which the "pitch" extends considerably beyond the "outline" in the major axis (assuming, that is, that the component has axial leads along its major axis).

At the time of writing this part of the program, the fact that the "outline" dimensions of components are frequently very loosely tolerated was overlooked. The use of the datum (x-outline=0,
y-outline=0) is thus somewhat dubious. It would appear sensible to switch the datum used for the input of master data either to one of the component pins, or to the geometric centre of the component body (defined relative to the pins, not the body outline), the latter corresponding to the convention adopted by a number of major component manufacturers.

In terms of the storage of a master definition in a data structure, a master will occupy a block (a number of contiguous words) consisting of a fixed-length "head", containing items 1 to 5 of Table 5.1 together with a pointer used in accessing library entries, and a variable-length "tail" containing the coordinates of all of the pins (length NOPINS). Such a block is referred to as a "master block". The detail description of a master block, for the implemented version of the program, is to be found in Appendix 3.

5.1.2 The Organisation and Operation of the Master Library.

The master library is a file, which may be thought of as an array, normally stored on disc, but brought into core as required, either for updating or access. Storage space in the array is manipulated by the free-storage scheme outlined in Appendix 1, thereby facilitating the creation and deletion of variable-length master blocks, as required.

Master blocks, as described in 5.1.1, are linked by pointers into a ring, the head of which resides in the 1st word of the free-store array (which, it will be noted from Appendix 1, is a reserved word), as indicated in Fig. 5.2. The master blocks shown in Fig. 5.2 are simplified for clarity, the actual organisation being described in detail in Appendix 3.
<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>ABBREVIATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The unique descriptor through which the library entry is referenced (5 ASCII characters).</td>
<td>NAME</td>
</tr>
<tr>
<td>2. The number of pins (leads) of the component</td>
<td>NOPINS</td>
</tr>
<tr>
<td>3. The pin (lead) diameter.</td>
<td>PIN-SIZE</td>
</tr>
<tr>
<td>4. The x- and y-dimensions of the component body.</td>
<td>X-outline, y-outline.</td>
</tr>
<tr>
<td>5. The x- and y-dimensions of the minimum board area required for the component, including the lands (pads) associated with its pins (leads).</td>
<td>X-pitch, y-pitch.</td>
</tr>
<tr>
<td>6. The x- and y-coordinates of each pin, referred to a suitable datum</td>
<td>$x$-pin&lt;sub&gt;n&lt;/sub&gt;, $y$-pin&lt;sub&gt;n&lt;/sub&gt;, $n = 1$, NOPINS.</td>
</tr>
</tbody>
</table>

Table 5.1  The Geometric Data Associated with a Master Definition.

![Diagram](image.png)

**Fig. 5.1** Component Dimensions and Corresponding Land Pattern.

THE PIN NUMBERS ARE AS INDICATED WHEN VIEWED FROM ABOVE (component side).
Fig. 5.2 Ring of Master Blocks in Master Library.

The program which adds to, or modifies, the master library is best described by the flowchart shown in Fig. 5.3, given the following conventions relating to the data supplied to this program. Where \( \text{NOPINS} = 0 \), the associated master is deleted from the library, and where \( \text{NOPINS} < 0 \), \( |\text{NOPINS}| \) is used as the number of pins, while a flag is set indicating that this master definition describes a board (including the edge-connector(s)). In the event of a master being defined which already exists in the library, the existing one is deleted and replaced by the new one (on the assumption that this was an update, and not a user error).
ENTRY 1: To clear the library and rebuild it.
ENTRY 2: To add to, or modify, the library according to the input data supplied.

Fig. 5.3 Master Library Manipulation.

5.2 The Input of Circuit Information.

The information required for a given circuit, for layout purposes, consists primarily of the components list and the data describing the
interconnections of these components. The geometric descriptions of the components (including the board itself) are implicit in the master name associated with each component (see 5.1). In addition to this basic information, there may be specific layout constraints to be taken into consideration by the program, relating either to the components list or to the interconnection data. An example of the former type of constraint would be the requirement that a specific component occupy a specified board location, and of the latter, that a particular electrical net be kept as short as possible.

The components list is simply a list of the components used in the circuit, including the board and edge-connector(s), each being referred to by a unique user name or label, and each being associated with the name of the master definition describing its geometry. Where a constraint applies to a particular component a suitably coded description of the constraint type, together with any necessary parameters, will follow immediately after the appropriate entry in the components list (see Tables 5.2 and 5.3).

The interconnections of the components of a circuit can be described systematically in two ways, each requiring that each electrical net be labelled with a unique identifier - a positive integer in the implemented program. The first form of description deals with each component in turn, listing (in order) the net identifiers associated with each pin, using a special identifier for unconnected pins. The second form deals with each net in turn, listing the component name and pin number for each pin of a given net. The latter description, being the one adopted, has certain advantages, in
that the extraction of data in this form from a circuit diagram is simpler than in the first form, and also in that the insertion of constraint information relating to specific nets is facilitated. Thus, where a constraint applies to a specific net, a coded description of the constraint type, together with any associated parameters, is inserted in the interconnection list at any point within the range of the appropriate net (see Tables 5.2 and 5.3).

A fragment of the input data of a circuit is shown in Table 5.2 and a list of constraint "codes", including special descriptors and other "reserved" names is shown in Table 5.3.

It should be noted that a given net may be defined in the input data in a number of disjoint parts, thus facilitating data preparation. The program, however, prints a warning message in these cases, as such a situation could be arrived at accidentally, therefore meriting an additional user check.
<table>
<thead>
<tr>
<th>INPUT DATA</th>
<th>COMMENT/DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>EJ01 EDGE</td>
<td>MASTER NAME, COMPONENT NAME (&quot;INSTANCE&quot; of MASTER).</td>
</tr>
<tr>
<td>ICO1 A1</td>
<td>Where the master name is omitted, the last-named master is assumed (ICO1 in this case).</td>
</tr>
<tr>
<td>ICO2 B1</td>
<td>CONSTRAINT referred to component Bl - see Table 5.3</td>
</tr>
<tr>
<td>FIXY 200,400</td>
<td>&quot;STOP&quot; FLAGS END OF COMPONENTS LIST - see Table 5.3</td>
</tr>
<tr>
<td>COOL C11</td>
<td>&quot;NET&quot;, UNIQUE NET IDENTIFIER (+ve integer)</td>
</tr>
<tr>
<td>STOP</td>
<td>COMPONENT NAME, PIN NUMBER</td>
</tr>
<tr>
<td>NET 1</td>
<td>CONSTRANTR referring to NET 90 - see Table 5.3</td>
</tr>
<tr>
<td>EDGE 27</td>
<td>&quot;STOP&quot; FLAGS END OF INTERCONNECTION LIST - see Table 5.3</td>
</tr>
<tr>
<td>NET 2</td>
<td>&quot;NET&quot;, UNIQUE NET IDENTIFIER (+ve integer)</td>
</tr>
<tr>
<td>EDGE 60</td>
<td>COMPONENT NAME, PIN NUMBER</td>
</tr>
<tr>
<td>A1 2</td>
<td>CONSTRAINT referring to NET 90 - see Table 5.3</td>
</tr>
<tr>
<td>A1 13</td>
<td>CONSTRAINT referring to NET 90 - see Table 5.3</td>
</tr>
<tr>
<td>NET 3</td>
<td>COMPONENT NAME, PIN NUMBER</td>
</tr>
<tr>
<td>NET 90</td>
<td>CONSTRAINT referring to NET 90 - see Table 5.3</td>
</tr>
<tr>
<td>POWR 5</td>
<td>CONSTRAINT referring to NET 90 - see Table 5.3</td>
</tr>
<tr>
<td>A1 14</td>
<td>CONSTRAINT referring to NET 90 - see Table 5.3</td>
</tr>
<tr>
<td>A2 14</td>
<td>CONSTRAINT referring to NET 90 - see Table 5.3</td>
</tr>
<tr>
<td>EDGE 10</td>
<td>CONSTRAINT referring to NET 90 - see Table 5.3</td>
</tr>
<tr>
<td>TRAK 50</td>
<td>CONSTRAINT referring to NET 90 - see Table 5.3</td>
</tr>
<tr>
<td>STOP</td>
<td>CONSTRAINT referring to NET 90 - see Table 5.3</td>
</tr>
</tbody>
</table>

Table 5.2 Fragment of Typical Circuit Input Data.
<table>
<thead>
<tr>
<th>INPUT DATA FORMAT</th>
<th>DESCRIPTION/FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESERVED NAME ACTING AS &quot;FLAG&quot; FOR PROGRAM</td>
<td></td>
</tr>
<tr>
<td>* STOP</td>
<td>Marks end of section of input data.</td>
</tr>
<tr>
<td>* NET N</td>
<td>Marks beginning or continuation of interconnection list of net with identifier N.</td>
</tr>
<tr>
<td>CONSTRAINT REFERRING TO SPECIFIC COMPONENT</td>
<td></td>
</tr>
<tr>
<td>FIXY X,Y</td>
<td>Assign component to board coordinates (X,Y).</td>
</tr>
<tr>
<td>* FIXO θ</td>
<td>Place component with orientation =θ (θ = 0, 1, 2 or 3).</td>
</tr>
<tr>
<td>DEFAULT ASSUMPTIONS REFERRING TO ALL NETS</td>
<td></td>
</tr>
<tr>
<td>* DTRK W</td>
<td>Where not specified otherwise, track width is W for all nets.</td>
</tr>
<tr>
<td>* DSEP S</td>
<td>Where not specified otherwise, track separation is S for all nets.</td>
</tr>
<tr>
<td>DESCRIPTOR REFERRING TO SPECIFIC NET.</td>
<td></td>
</tr>
<tr>
<td>* POWR V</td>
<td>Indicates that the net is a power net, at voltage V.</td>
</tr>
<tr>
<td>* GRND V</td>
<td>Indicates that the net is a ground net, at voltage V.</td>
</tr>
<tr>
<td>CONSTRAINT REFERRING TO SPECIFIC NET</td>
<td></td>
</tr>
<tr>
<td>* CRIT W</td>
<td>Attach weight W to the importance of the net being short, (1 ≤ W ≤ 15). Default value is W = 1.</td>
</tr>
<tr>
<td>TRAK W</td>
<td>Route this net with track of width W. The default value is given by DTRK above.</td>
</tr>
<tr>
<td>SPAC S</td>
<td>Route this net with track separation S from all adjacent nets and obstacles. The default value is given by DSEP above.</td>
</tr>
<tr>
<td>SIDE S</td>
<td>Route this net entirely on side S of the board, S = 0 or 1.</td>
</tr>
</tbody>
</table>

Table 5.3  Reserved Names for Input Data Constraints, etc.

Note: Those items marked * are currently implemented. The program reads and stores the relevant information for the remainder, but does not act upon it.
5.3 The Circuit Data Structure.

An associative data structure is required for the storage of information relating to the elements - components and nets - of a given circuit, and of the associations between these elements. Such associations link elements into sets with common properties, such as the set of all components, and may be hierarchical in nature, as in the case of an electrical net, where each net is linked with a set of associated component pins at a lower level in the structure.

A ring-based data structure is used in which a given element of a circuit is represented by a block (a set of contiguous storage locations), in which the attributes of the element are stored. The associations between elements are represented by pointers which link the associated blocks into rings. The structure is built within a 1-dimensional FORTRAN array, the free-storage scheme described in Appendix 1 being used to allocate and return storage as required.

5.3.1 Single Level Associations.

Four basic types of block are required for the storage of circuit and placement information. These are component, master, net and cluster blocks. The blocks of each type are tied into a ring, the first pointer of which resides in a reserved location in the data structure array, known as the ringhead. As each block contains a marker indicating its type (see Appendix 3), a ringhead is distinguishable from the elements of its ring. The indices (addresses) of the ringheads are held in a FORTRAN COMMON block, and so are accessible throughout the program. In searching the data structure
for a given element, the structure would always be entered through one of these ringheads. Fig. 5.4 shows, in outline, the organisation of the ring of masters, the other rings in this category following a similar pattern, based on the appropriate ringheads. Only that detail essential to the point being made is shown in Fig. 5.4, and in all subsequent data structure diagrams in the main text of the thesis. A complete description of the various data structure blocks, for the current program implementation, is to be found in Appendix 3.

Fig. 5.4  Outline Data Structure: the Master Ring.

5.3.2  The Data Structure Representation of a Component.

In order that the interconnections between component pins can be represented, it is necessary that a block must exist in the data structure for each connected pin of each component. Such "pin blocks"
would be associated with the appropriate component block. Rings, however, are costly in terms of both storage and access speed, and so it is desirable to avoid their use unnecessarily. The primary justification for the use of a ring data structure lies in its flexibility - relationships can be added, deleted, or amended, as and when desired, within the limits of the structure design. The relationship between a component and its pins, however, is static and pre-determined. Consequently a composite structure is adopted, using a single block to represent a component, including its pins. That part of the composite block, referred to as the "component block", apart from the pin blocks, is referred to as the "component head" block. For consistency, the pins remain as if they were independent blocks as viewed from "outside" the component block, but can be accessed from the component head in a single array indexing operation (pin index = head index + length of head block + length of pin block \times (pin number -1)).

To access the component head from any of the component pins, each pin must contain either an explicit pointer to the head block, or the appropriate pin number, such that the head can be found from the inverse of the above equation (i.e. head index = pin index - length of head block - length of pin block \times (pin number -1)). The former has been chosen in view of the frequency of such accesses, trading-off a small amount of storage for an increase in speed (as the field required to hold a pointer is greater than that required to hold a pin number).

Fig. 5.5 shows, in outline, the arrangement of a component block. It will be noted from Fig. 5.5 that each component head contains a pointer to the appropriate master block, containing the detailed geometric description of the component.
5.3.3 Hierarchical Associations.

Given the block types listed above - component heads and pins, masters, nets, and clusters - the remaining two associations required are easily expressed. The first is the linking of the pin blocks associated with a given net into a ring, along with the net block, as depicted in Fig. 5.6. The second is the linking of the head blocks of those components allocated to a particular cluster, with the appropriate cluster block, as outlined in Fig. 5.7. The latter associations obviously do not exist initially: they are added when components are assigned to specific clusters, on completion of the partitioning of the circuit graph (see 3.2).
Fig. 5.6  Data Structure: Net - Component Pin Associations.

Fig. 5.7  Data Structure: Cluster - Component Associations.
5.3.4 Building the Circuit Data Structure.

Given the structure outlined in 5.3.1 - 5.3.4, the building of such a structure from the input data described in 5.2 is a trivial matter.

On reading the components list, a "skeleton" component block is generated for each component, with the appropriate number of pin blocks, and tied into the ring of components on the component ringhead. Master definition blocks are copied from the master library as required, and similarly tied into the ring of masters on the master ringhead.

On reading the interconnection data a new net block is created for each net appearing for the first time, the unique net identifier is added to the net block, and the block is tied into the ring of nets on the net ringhead. On reading the component pins of a given net, the appropriate pin block is found from the component head and pin number (see 5.3.2), and is tied into the ring of pins of that net (see Fig. 5.6).

When a constraint is met in the input data, the constraint parameters are inserted in the appropriate blocks (see Table 5.3), or into COMMON in the case of the default values for track width and separation (DTRK and DSEP in Table 5.3).

5.4 The Verification of the Input Data.

The verification of the input data falls broadly into two categories - checking for consistency, which can be done by program, and checking for accuracy in the transcription from circuit diagram to program input data. The latter cannot easily be automated, but
as a faulty layout arising from erroneous data can be very expensive, the independent compilation of input data by two individuals, followed by an automated comparison at the time of building the circuit data structure, would appear to be justified.

Returning to the former category - checking for consistency - the program performs a number of trivial checks, aimed at establishing the existence of errors as much as at their location, which is frequently indeterminate. For example, the fact that, on completion of the building of the circuit structure, a two-pin device has only one pin connected, suggests the existence of an error. Whether the error lies in the complete omission of the other connection from the input data, or in its accidental assignment to some other component is indeterminate without reference to the circuit diagram.

Apart from the obvious checks which must be made, such as ensuring that a stated pin number for a given component does in fact exist, the most significant additional checks are as outlined in Table 5.4.
CHECKS MADE THROUGH-OUT DATA INPUT.

1. Component multiply declared in components list.
2. Master with given name not present in master library.
3. Component in interconnection data not declared in components list.
4. Component pin declared as connected to more than one net.
5. Component pin multiply declared as connected to a given net - not of itself an error, but may be associated with one.
6. Net with given identifier appearing more than once in the interconnection data - not of itself an error, but may be associated with one.

CHECKS MADE ON COMPLETION OF DATA INPUT.

7. Net has fewer than two associated component pins.
8. Component has fewer than two connected pins.
9. A two-pin component has both pins connected to the same net.

Table 5.1 Check for Input Data Consistency.

5.5 The Data Structure Used in Graph Partitioning.

The partitioning algorithm as described by Kernighan and Lin (27), and upon which the adopted procedure is based (see 3.2.3 and Appendix 6), originally operated on the connectivity matrix (weighted adjacency matrix) of the circuit graph (in which components are represented by nodes, and electrical connections by branches). For large circuits this can be very costly in terms of the storage requirement, particularly as the required information regarding connectivity already exists in the circuit data structure (see 5.3), albeit in a somewhat scattered form. In order to avoid the storage penalty involved in using the connectivity matrix, and the speed penalty involved in accessing the
existing structure repeatedly to obtain the connectivity information, some extensions are made to the circuit data structure, in the form of "component sub-blocks", to hold the required information in an accessible manner. These extensions are temporary, lasting only for the duration of the partitioning process.

A sub-block is associated with each component. Each sub-block consists of a fixed length "head" containing a pointer to the associated component, a pointer in the ring of cluster/sub-block pointers, and some "work-space" used by the partitioning algorithm, and a variable length "tail". The tail of a sub-block contains a set of pointers, one to the sub-block of each component connected to that associated with the given sub-block, a weight being associated with each pointer to indicate the total number of such shared connections. Fig. 5.8 shows a fragment of such a sub-block structure.

It will be noted from Fig. 5.8 that the structure, like the connectivity matrix, is symmetric: for any two connected components, the sub-block of each contains a pointer to that of the other, with identical weights. While doubling the storage cost, this decreases the computation time by more than half as compared with asymmetric storage.

It should also be noted that the time required for the building and dismantling of such a temporary structure is low as compared with the time spent by the partitioning algorithms in accessing it, and so the inefficiency of the basic circuit structure as regards the retrieval of connectivity information in this form is of negligible consequence.
Fig. 5.8  Data Structure: Temporary Structure Used in Partitioning.
The Implementation of the Component Placement Algorithms.

The input of circuit data, and the data structure used for its storage, have been described in 5.2 and 5.3 respectively. On completion of data input and checking, the circuit data structure contains a complete description of a given layout problem. The implementation of the placement algorithms, outlined in Chapter 3, may now be considered in some detail. These algorithms operate on the circuit structure, adding parts of the "solution" to it as they are generated, such as the datum coordinates of components, after placement.

6.1 k-way Partitioning of the Circuit Graph.

6.1.1 The Evaluation of $k$, the Number of Clusters.

As indicated in 3.2.1, the choice of the number of clusters (of components) into which a given circuit should be broken is based largely on geometric considerations, unless the program user wishes to override the program choice in order to guarantee the use of a specific grid spacing in $x$ for package placement.

The first step is to establish the orientation for each component. Where this is not stated explicitly by way of an input constraint (see Table 5.3), and is not "forced", through the $x$- or $y$-pitch of the component being greater than the slot width, it is derived from consideration of the "transparency" of the component (see 3.2.1). Any given component is oriented such that its more "transparent" axis lies parallel to the major slot axis (the $y$-axis). The term "$x$- or $y$-transparency" denotes a measure of the number of conductor paths...
which can pass under the body of a component in x or y, respectively, unobstructed by the lands associated with the component pins, and normalised so as to yield transparency values in units of tracks/inch. The default values of track width and separation are assumed (see Table 5.3). For example, in Fig. 6.1, 8 conductors can pass under the component body, parallel to the x-axis, while the component spans 0.800" normal to the x-axis (i.e. y-pitch = 0.800"), yielding a normalised x-transparency of $8 \times \frac{1,000}{0.800} = 10$ tracks/inch.

It should be noted that, with the above definition of transparency, orienting components for maximal transparency along the major slot axis does not necessarily lead to a maximal number of unobstructed y-axis paths being available in a given slot, but trades-off packing density in the slot against this number of unobstructed paths. Fig. 6.2 shows one example of this, where 1 potential y-axis route is sacrificed for an increase in packing density by a factor of 5 (in idealised circumstances!).

As most circuits will contain a number of components with a common master (see 5.1), or a number of such groups, and transparency is a function of component geometry, the x- and y-transparency is computed for each master, the two values being stored in the master block (see Appendix 3), thereby minimising the computation involved in establishing the orientation for each component. In assessing the preferred orientation for a given component, the x- and y-transparency values are found from the appropriate master block, and hence the orientation.

Given then, the preferred orientation for each component (mandatory only where forced by input constraint), the "total slot length" required to accommodate all of the components (except, of
Fig. 6.1 The X- and Y-transparency of a 14 pin dual-in-line Package.

Fig. 6.2 The X- and Y-transparency of a Two-pin Component.
course, the board itself!) is found, this being the sum of the lengths of each component, measured along the y-axis with the component in its preferred orientation, and rounded up to the next higher grid unit before summation (in y, grid size = 0.550" : see 3.4). Similarly, the "mean width" of the hypothetical column of components of length = "total slot length", is found, being the total component area divided by the total slot length. From the "total slot length", L, the "mean slot width", W, and the board size, (X,Y), the actual slot width can be found, such that the mean packing density over the board is approximately equal in x and y, from:

Maximum slot length, M, of width W, on X × Y board is

\[ M = Y \times \left( \frac{X}{W} \right) \]

Hence the fractional utilisation of the board, U, (area-wise) is

\[ U = \frac{L}{M} \]

For equal x and y utilisation (packing density), the slot width, S, will be

\[ S = \frac{W}{\sqrt{U}} \]

and hence the number of slots, k, will be

\[ k = \frac{X}{S}. \]

Having computed the actual slot width, S, each component must be checked to ensure that, in its current orientation, it does not exceed the slot width in x, and re-oriented if necessary.
6.1.2 Building the Temporary Data Structure used in Partitioning.

The data structure used to store the connectivity information required by the partitioning procedure is described in detail in 5.5. The building of that structure, from information contained in the basic circuit structure, is best described by the flowchart in Fig. 6.4. It will be noted from Fig. 6.4 that all of the interconnections of any given component are searched twice. In the first
Foreach net, do:

- Set TAG=1 in net block if net is power or ground net, or if it connects the pins of only one component, else set TAG=0.

For I=1, 2

- Foreach component do:
  - N=0
  - Foreach component pin, do:
    - Find mutual pointers in 2 component sub-blocks, creating them if not already there. Increment associated weight in sub-block by weight of net.

- Add sub-block to component

Find net to which pin belongs

Get weight of net block

Tag(NET)=1?

- Foreach comp't pin of net do:
  - Yes
    - I=1?
      - Yes
        - N=N+1
      - No
        - Yes
          - I=1?
            - No
              - No
              - Yes
                - Yes

Fig. 6.4 Creating the Temporary Sub-Block Structure.
search the number of components connected by any net(s) to a given component is established, hence putting an upper bound on the length of the sub-block required for that component. It is only an upper bound because, in this search, multiply-connected components are not detected. In the second search the appropriate pointers between the sub-blocks of connected components, and the weights associated with those pointers, are inserted in the sub-blocks.

6.1.3 The Initial Assignment of Components to Clusters.

As indicated in 3.2.3, the k-way starting partition is arbitrarily selected. The selection, in fact, is not entirely arbitrary, as the starting partitions are made to be of similar size (the size of a cluster, as defined in 3.2.3, being the "length of slot" occupied by the components of that cluster), though no attention at all is paid to connectivity. The procedure adopted is as follows.

The k cluster blocks (as described in 5.3.3) are created, and tied into a ring on the cluster ringhead. Each block has a location in which is stored the cluster size, this being set to zero initially (see Appendix 3 for details of cluster block). Components are then taken, one at a time, in descending order of size, and added to that cluster having the smallest size by tying the component sub-block into the ring of sub-blocks of that cluster, and incrementing the cluster size by the size of the component.

After the allocation of all of the components to clusters, two "dummy" component sub-blocks are created and added to each cluster, representing the dummy nodes described in 3.2.3. Being of zero size, these do not affect the cluster sizes.
6.1.4 The Generation of the Pairwise Refinement Sequence.

As indicated in 3.2.3, the 2-way refinement procedure, outlined in Appendix 6, is applied to pairs of subsets (of the circuit graph), until the k-way partition is pairwise optimal. To achieve pairwise optimality in a near-minimal time requires a sensible selection of the order in which pairs of subsets must be refined. It should be noted that the costs referred to hereafter are the numbers of connections between components or clusters, as appropriate. As indicated in 3.1, a net connecting n component pins is represented by the complete graph on those n pins.

To make this selection, the mutual costs of all k subsets are required, together with a means of marking those pairs which have reached minimal cost. A $k \times k$ 2-dimensional FORTRAN array, "IRAY", is used to store the symmetric cost matrix for the k subsets, where IRAY(i,j) is the total interconnection cost between the i'th and j'th subsets, and where IRAY(i,i) = 0 for all i, and IRAY(i,j) = IRAY(j,i) for all $i,j = 1 .... k$. The mutual cost for any given pair of subsets is found by summing the weights associated with the sub-block pointers (see 6.1.2) which point from the sub-blocks of the components (nodes) of one subset to those of the other. The means used to indicate those pairs of subsets having reached minimal cost is a single marker bit set in the appropriate location of IRAY, corresponding to an impossibly high interconnection cost (so that the marker bit could never be "accidentally" set). It should be noted that if a change is made to subset n in the process of minimising the cost in IRAY(n,m), then the minimal cost property which n may have had with respect to any other subset p, $p \neq m$, may be destroyed, and likewise for m.
Thus, on altering any subset $n$, any markers involving $n$ must be cleared. Thus requires simply the clearing of any markers in row and column $n$ of $\text{IRAY}$ which may have been set.

The next pair of subsets to be refined at any given time is simply that pair with the highest mutual cost (in $\text{IRAY}$), which are not "tagged" (as above). This choice is based on the assumption that, in general, more changes will be made (swaps) in the refinement of a high-cost than of a low-cost pair, thus involving a greater number of changes in the mutual costs of the given pair with the remaining subsets. The aim, then, is to refine those pairs first with the most far-reaching consequences, in the hope that the process will converge rapidly. Experience with the partitioning procedure bears this out, pairwise optimality typically being achieved in 3 passes (where by a pass is meant the refinement sequence between two successive refinements of a given pair of subsets).

6.1.5 The 2-way Refinement Procedure.

The 2-way partitioning procedure is described in some detail in Appendix 6. Only a few relatively minor implementation details need be dealt with here, mainly concerning the storage of "working data", but for ease of reading these are embedded in a very brief description of the procedure.

In refining the partition of a given pair of subsets (of the set of nodes), the "D values" (the differences between external and internal connection cost) are first computed for each node of each subset. The connectivity information required for the establishment of the D values is contained in the sub-blocks of the components corresponding
to the nodes of the two subsets (see 6.1.2). On computing the D values, they are stored in the appropriate sub-block heads (see Appendix 3 for details). In Appendix 6 it is indicated that the best nodes to be swapped at any given time will generally correspond to one of the three highest D values for each of the two subsets. To assist the search for the best pair of nodes to swap, the 3 highest D values for each of the two subsets are saved in a $2 \times 3$ FORTRAN ARRAY, $D(i,j)$, and the addresses (pointers) of the corresponding sub-blocks in a similar array, $ADD(i,j)$. In $D(1,j)$ is stored the $j$'th highest D value for the 1st subset, $ADD(1,j)$ containing a pointer to the corresponding sub-block, and similarly $D(2,j)$ and $ADD(2,j)$ for the 2nd subset.

The "best" (highest gain) pair of nodes to be considered for swapping is found from an exhaustive check on each pair of sub-blocks with addresses in $ADD(1,i)$ and $ADD(2,j)$, $i,j = 1,2,3$, as described in A6.2. The "best" sub-blocks are temporarily removed from the cluster rings to which they were tied, and are re-tied into two "swap rings", starting on "swap blocks" similar to the cluster blocks described in 5.5, one for the sub-blocks of each subset. On transferring a sub-block to one of the swap rings, its D value is no longer required, and so is replaced in the sub-block by the value of the net gain, $g_i$, resulting from that swap, which is subsequently required. Also on removal of a node (sub-block) from a subset the cluster size, held in the cluster block, is updated appropriately. Having removed the "best" node from each subset, the D values for the remaining nodes of the subsets are updated, the largest three for each subset again being saved in $D(i,j)$, and so on. The extraction of nodes continues until either of the two subsets is empty.
The number of sub-blocks, $k$, the swapping of which results in \( k \) maximal total gain, \( G = \sum_{i=1}^{k} g_i \), is found by scanning the gain values \( g_i \), in the sub-blocks on one of the swap rings. If the maximum gain is less than or equal to zero, no swaps have to be made, and so all of the sub-blocks are untied from the swap rings and re-tied to their original cluster rings, re-setting the cluster sizes held in the cluster blocks. The partition refinement being complete for these subsets, control is returned to the routine which, after setting the appropriate tags in IRAY, selects the next pair of subsets for refinement. If, however, the maximum gain is positive then the first \( k \) sub-blocks from each swap ring are untied and re-tied to the "other" cluster rings, any remaining sub-blocks on the swap rings being untied and re-tied to their original cluster rings, the cluster sizes being modified appropriately. The D values are then re-computed for each sub-block of each subset, and a further iteration of the 2-way refinement procedure is initiated.

6.2 The Placement of Clusters in Slots on the Board.

As outlined in 3.2.1 and 3.3, each cluster of components, as generated by the partitioning procedure, is allocated to a specific slot on the board, the objective of the allocation being the minimisation of the total cluster interconnection cost, taking into account only the x-component of the cost, y-coordinates at this stage still being undefined. The cost of any particular linear ordering of clusters (i.e. a particular mapping of clusters into slots) is given by:

\[
\text{COST} = \sum_{j=1}^{k-1} \sum_{i=1}^{k-j} c_{i,j}, \text{ where } c_{i,j} \text{ is the}
\]

\[
\sum_{i=1}^{k} g_i, \text{ in the sub-blocks on one of the swap rings. If the maximum gain is less than or equal to zero, no swaps have to be made, and so all of the sub-blocks are untied from the swap rings and re-tied to their original cluster rings, re-setting the cluster sizes held in the cluster blocks. The partition refinement being complete for these subsets, control is returned to the routine which, after setting the appropriate tags in IRAY, selects the next pair of subsets for refinement. If, however, the maximum gain is positive then the first \( k \) sub-blocks from each swap ring are untied and re-tied to the "other" cluster rings, any remaining sub-blocks on the swap rings being untied and re-tied to their original cluster rings, the cluster sizes being modified appropriately. The D values are then re-computed for each sub-block of each subset, and a further iteration of the 2-way refinement procedure is initiated.

6.2 The Placement of Clusters in Slots on the Board.

As outlined in 3.2.1 and 3.3, each cluster of components, as generated by the partitioning procedure, is allocated to a specific slot on the board, the objective of the allocation being the minimisation of the total cluster interconnection cost, taking into account only the x-component of the cost, y-coordinates at this stage still being undefined. The cost of any particular linear ordering of clusters (i.e. a particular mapping of clusters into slots) is given by:

\[
\text{COST} = \sum_{j=1}^{k-1} \sum_{i=1}^{k-j} c_{i,j}, \text{ where } c_{i,j} \text{ is the}
\]

\[
\sum_{i=1}^{k} g_i, \text{ in the sub-blocks on one of the swap rings. If the maximum gain is less than or equal to zero, no swaps have to be made, and so all of the sub-blocks are untied from the swap rings and re-tied to their original cluster rings, re-setting the cluster sizes held in the cluster blocks. The partition refinement being complete for these subsets, control is returned to the routine which, after setting the appropriate tags in IRAY, selects the next pair of subsets for refinement. If, however, the maximum gain is positive then the first \( k \) sub-blocks from each swap ring are untied and re-tied to the "other" cluster rings, any remaining sub-blocks on the swap rings being untied and re-tied to their original cluster rings, the cluster sizes being modified appropriately. The D values are then re-computed for each sub-block of each subset, and a further iteration of the 2-way refinement procedure is initiated.

6.2 The Placement of Clusters in Slots on the Board.

As outlined in 3.2.1 and 3.3, each cluster of components, as generated by the partitioning procedure, is allocated to a specific slot on the board, the objective of the allocation being the minimisation of the total cluster interconnection cost, taking into account only the x-component of the cost, y-coordinates at this stage still being undefined. The cost of any particular linear ordering of clusters (i.e. a particular mapping of clusters into slots) is given by:

\[
\text{COST} = \sum_{j=1}^{k-1} \sum_{i=1}^{k-j} c_{i,j}, \text{ where } c_{i,j} \text{ is the}
\]
total interconnection cost, unweighted for distance, between the clusters occupying the i'th and j'th slots.

The cost matrix, IRAY(k,k), containing the mutual interconnection costs for all of the clusters, is available from the partitioning procedure, which made use of it in controlling the application of the 2-way refinement procedure (see 6.1.4). The procedure used for cluster placement has already been described in 3.3, the implementation utilising a scheme for "marking" elements of the cost matrix, IRAY, as the rows/columns (clusters) corresponding to those elements are placed relative to each other, similar to the marking scheme used in generating the cluster refinement sequence (see 6.1.4).

On establishing the ordering of the first three clusters, n,i,m, the elements of row and column i of IRAY are each assigned a tag of value N (N = 5000), and the four elements IRAY(i,n), IRAY(n,i), IRAY(i,m) and IRAY(m,i) are each assigned a tag of value 2N (= 100000). Any row or column of IRAY containing two elements with tags of value 2N corresponds to a cluster whose linear ordering is fixed, and so is not considered further in the ordering process. The ends of the currently ordered "chain" of clusters correspond to those two rows/columns of IRAY containing only one element with a tag of value 2N. The next cluster to be added to one end of the chain is found by a simple search of the elements of the rows/columns of IRAY corresponding to the ends of the chain, the joining of the two ends of the chain to form a loop not being permitted. On finding the next cluster to be placed, the tag of the appropriate element is set to 2N, and those of the remainder of the appropriate row and column are set to N.
The process is repeated until \( k-2 \) of the rows/columns of \( \text{IRAY} \) have 2 tags of value \( 2N \), the remaining two having only one tag of value \( 2N \), corresponding to the ends of the now ordered chain of clusters. This process is illustrated for a typical \( 7 \times 7 \) cost matrix in Table 6.1, showing the initial cost matrix and each stage in the ordering sequence, together with the corresponding sequence (ordering) at each stage. On termination of the ordering process, all tags of value \( N \) are cleared, and the ordering is extracted from the matrix by following from one end of the chain through alternate rows and columns of the matrix, changing from row to column, or vice versa, on "hitting" a tagged element, as illustrated diagrammatically in Fig. 6.5.

It should be noted that this scheme does not guarantee a minimal cost solution, but as inter-cluster costs are low the probable improvement in a globally optimal solution was considered to be too low to justify the cost of writing code to achieve such optimality.

On completion of the allocation of clusters to slots, the component sub-block structure used in partitioning is deleted, the component head-blocks being tied into the cluster-rings in place of the sub-blocks (see Fig. 5.7). The cluster blocks are re-ordered in the ring of clusters such that their ordering in the ring corresponds to the newly established allocation of clusters to slots, and the cost matrix, \( \text{IRAY}(k,k) \), is cleared.
<table>
<thead>
<tr>
<th>THE SYMMETRIC COST MATRIX, IRAY(7,7)</th>
<th>COMMENT/ORDERING</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 19 4 27 17 60 8</td>
<td>INITIAL</td>
</tr>
<tr>
<td>19 0 0 11 12 12 14</td>
<td>(UNTAGGED)</td>
</tr>
<tr>
<td>4 0 0 11 8 0 2</td>
<td>MATRIX</td>
</tr>
<tr>
<td>27 11 12 0 9 55 3</td>
<td>ORDERING</td>
</tr>
<tr>
<td>17 1 8 9 0 10 3</td>
<td>4-6-1</td>
</tr>
<tr>
<td>60 12 0 55 10 0 3</td>
<td>OF FIRST</td>
</tr>
<tr>
<td>8 14 2 3 3 3 0</td>
<td>THREE CLUSTERS</td>
</tr>
<tr>
<td>0 19 4 27 17 10060 8</td>
<td>ORDERING</td>
</tr>
<tr>
<td>19 0 0 11 12 12 14</td>
<td>4-6-1-1</td>
</tr>
<tr>
<td>4 0 0 11 8 0 2</td>
<td>OF FIRST</td>
</tr>
<tr>
<td>27 11 12 0 9 10055 3</td>
<td>THREE CLUSTERS</td>
</tr>
<tr>
<td>17 1 8 9 0 5010 3</td>
<td>ORDERING</td>
</tr>
<tr>
<td>10060 5012 5000 10055 5010 5000 5003</td>
<td>4-6-1-2</td>
</tr>
<tr>
<td>8 14 2 3 3 5003 0</td>
<td>ORDERING</td>
</tr>
<tr>
<td>5000 10019 5004 5027 5017 10060 5008</td>
<td>4-6-1-2</td>
</tr>
<tr>
<td>10019 0 0 11 1 5012 14</td>
<td>ORDERING</td>
</tr>
<tr>
<td>5004 0 0 12 8 5000 2</td>
<td>4-6-1-2</td>
</tr>
<tr>
<td>5027 11 12 0 9 10055 3</td>
<td>ORDERING</td>
</tr>
<tr>
<td>5017 1 8 9 0 5010 3</td>
<td>4-6-1-2</td>
</tr>
<tr>
<td>10060 5012 5000 10055 5010 5000 5003</td>
<td>ORDERING</td>
</tr>
<tr>
<td>5008 14 2 3 3 5003 0</td>
<td>4-6-1-2-7</td>
</tr>
<tr>
<td>5000 10019 5004 5017 10060 5008 5008</td>
<td>ORDERING</td>
</tr>
<tr>
<td>10019 5000 5000 5011 5001 5012 10014</td>
<td>4-6-1-2-7</td>
</tr>
<tr>
<td>5004 5000 0 12 8 5000 2</td>
<td>ORDERING</td>
</tr>
<tr>
<td>5027 5011 12 0 9 10055 3</td>
<td>4-6-1-2-7</td>
</tr>
<tr>
<td>5017 5001 8 9 0 5010 3</td>
<td>ORDERING</td>
</tr>
<tr>
<td>10060 5012 5000 10055 5010 5000 5003</td>
<td>4-6-1-2-7</td>
</tr>
<tr>
<td>5008 10014 2 3 3 5003 0</td>
<td>ORDERING</td>
</tr>
</tbody>
</table>
Table 6.1 Establishing the Linear Ordering of 7 Clusters.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10019</td>
<td>4</td>
<td>27</td>
<td>17</td>
<td>10060</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>10019</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>12</td>
<td>10014</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>10012</td>
<td>10008</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>27</td>
<td>11</td>
<td>10012</td>
<td>0</td>
<td>9</td>
<td>10055</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>1</td>
<td>10008</td>
<td>9</td>
<td>0</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>10060</td>
<td>12</td>
<td>0</td>
<td>10055</td>
<td>10</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>10014</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Linear Ordering is 5-3-4-6-1-2-7

Fig. 6.5 Extracting the Linear Ordering from the Tagged Cost Matrix.
6.3 The Placement of Components within Slots.

As outlined in 3.4, the component placement problem is complicated by the (possible) existence of fixed components at either end of the slot, corresponding to the edge connector(s), and by the high interconnection density within the slot resulting from the partitioning procedure. The method used to generate the initial placement uses the symmetric cost matrix \( C(n,n) \), where \( c_{i,j} \) is the interconnection cost between the \( i \)'th and \( j \)'th components in the slot. For a given slot, the cost matrix \( C \) is established in the 2-dimensional FORTRAN ARRAY, \( IRAY(n,n) \), in a similar manner to the sub-block construction prior to partitioning. The sub-block structure is not retained and used as the source for this information as it does not include the edge connector(s) among its components. The initial component ordering is taken to be the order in which the component head-blocks happen to be tied into the cluster ring, and so is quite arbitrary.

Operating on the cost matrix, \( IRAY \), contenders for the "next" free location in the slot are evaluated, that one being accepted for placement which has the highest weighted (in terms of conductor length) connection cost to the placed components. A 1-dimensional FORTRAN ARRAY, \( ICOST(n) \), is used to store these costs temporarily for each unplaced component, in order to assist in the selection of the next component to be placed. The "next" free location is that one nearest to the edge connector, as the edge connector is likely to be connected to some of the components of the slot, thereby providing a start point for the placement. Only the \( y \)-component of the connection cost (length) is considered, and that is approximated by the component centre-to-centre distance. The initial placement scheme is outlined by the flowchart in Fig. 6.6.
On completing the initial placement for a given slot/cluster, the component head blocks are re-ordered in the ring of components of that cluster, such that their order in the ring corresponds to their newly-defined ordering in the slot.
6.4 The Refinement of the Initial Component Placement.

The initial placement of components in slots, described in 6.3, does not take inter-slot connections into account, and is not necessarily optimal even with the respect to the internal connections in a given slot. Consequently, the placement of the components within a given slot is refined, as outlined in 3.5, by a 2-way interchange process based on the centre of gravity method, in which elastic forces are considered to act on component pins, parallel to the y-axis, and proportional in magnitude to the y-separation of any given pin and the centre of gravity of its associated net.

Two steps are required prior to the placement refinement stage. The first is the assignment of exact datum coordinates to each component - so far components have only been assigned an "ordering" in a specific slot. In x, components are placed centrally in their respective slots, rounding down the datum x-coordinate where necessary to ensure that the component pins occupy a 0.100" grid relative to the board datum. In y, it will be noted from 3.4, components occupy an integral number of grid units of 0.550" each, and so each component is centred in y within its allotted number of grid units, the datum y-coordinate, as with x, being rounded where necessary to ensure that the component pins occupy a 0.100" grid relative to the board datum. The second pre-requisite is the addition of a number of dummy components, each occupying 0.550" in y and having no pins, and hence no interconnections, as outlined in 3.5. Sufficient dummy components are added to each slot to fill the slot entirely in y, thus permitting full use to be made of any required part of the slot by swapping dummy components with real ones and so creating gaps between the real components, corresponding to the dummy component positions.
Two basic routines are used by the refinement procedure, and are best described at this stage to simplify the subsequent description. The first is a routine which finds the y-coordinate of the "centre of gravity" of a net \( y_{cn} \). The routine simply sums the y-coordinates of all of the pins of the net \( y_{pn} \), finds the degree (number of pins) of the net, and stores the centre of gravity y-coordinate \( y_{cn} \), being \( Ey_{pn} / \text{degree} \) of the net in the net block (see Appendix 3). When a component is moved, and hence one or more pins of a net, the centre of gravity of that net is re-computed - it could be suitably adjusted, but using integer arithmetic, this leads to cumulative errors (due to rounding onto 0.1" grid on which component pins lie). The second basic routine is that which computes the net upward and downward components of the "force" acting on a component. The net upward force is simply the sum \( \Sigma (y_{cn} - y_{pn}) \), for each pin of the component for which \( y_{pn} \) lies below the corresponding \( y_{cn} \), and vice-versa for the net downward force. The terminology used here is identical to that already defined in 3.5. These two forces, together with the number of connections, are stored for each component in an array as outlined below. Where a net has non-unity weight (i.e. has been declared of CRITical length by the user, on input - see Table 5.3), the force on each pin of that net is scaled by the appropriate weight, and such a connection is considered to count as a number of connections, equal to the weight, in the connection count for each component affected.

The working storage, used to hold the above forces and connection counts for each component, utilises a part of the data structure array, IARR. For convenience of description this working space will subsequently be referred to as if it were a 1-dimensional FORTRAN ARRAY, IFORCE(n), where \( n = 4 \times (\text{no. of slots}) \times (\text{max. no of components}) \).
per slot). The format of IFORCE is best described by the diagram in Fig. 6.7. To make IFORCE easily addressable, each slot must be assumed to have an equal number of components, this number being the maximum number which a slot could contain - i.e. the number of 0.550" grid units in a slot.

Given these basic routines and the above working data storage scheme, the refinement procedure may be described in some detail. It has already been observed in 3.5 that, as inter-slot connections are taken into consideration, a swap in any given slot may affect the pairwise optimality of the placement in any other slot containing components connected to the swapped pair. Consequently a control routine is required, similar to that used to control the application of 2-way partition refinement to k-way partitioning, requiring a complete pass over all slots, with no interchanges resulting, before exiting.

A k-word FORTRAN ARRAY, ICOST(k), is used to store a "cost" for each slot, the cost being the arithmetic sum (not vector) of the upward and downward forces on each component in a given slot. When a slot reaches pairwise optimality, a TAG, of value $2^{34}$, is added into the appropriate word of ICOST. On making any alteration to the placement in any slot, all of the tags in ICOST are cleared, the extra computation involved being small in relation to the cost of providing code to check which slots share nets with the affected components. As in partitioning, the slot with the highest cost, being the most likely candidate for modification, is refined first, in an attempt to refine those slots with the most far-reaching consequences early in the sequence, so favouring rapid convergence on overall pairwise optimality.
Array "IFORCE"

| UPWARD FORCE | COMPONENT No. 1 |
| DOWNWARD FORCE | COMPONENT No. 1 |
| UPWARD FORCE | COMPONENT No. 2 |
| DOWNWARD FORCE | ETC. |

| No. of CONNECTIONS |
| No. of CONNECTIONS |

| UPWARD AND DOWNWARD FORCES FOR ALL COMPONENTS OF ALL SLOTS. |
| SLOT No. 2 |
| SLOT No. 2 |
| ETC. |

| NUMBER OF CONNECTIONS FOR ALL COMPONENTS OF ALL SLOTS. |
| SLOT No. 2 |
| ETC. |

Fig. 6.7 The Organisation of Working Storage for Placement Refinement.
The control routine and the refinement routine are best described by the flowcharts in Figs. 6.8 and 6.9 respectively.

On completion of the refinement, the dummy components are deleted and the components are re-ordered in the cluster rings such that their order corresponds to their (new) ordering in the slots.

The interactive modification of component placement is described in detail in Chapter 8.

Fig. 6.8 Generation of Placement Refinement Sequence.
**ENTER**

**IBEST=0**

**FOREACH COMPONENT PAIR, DO:**

- **FIND COMP'T DATUM Y-COORDS.**
- **FIND DATUM Y-COORDINATES WITH ELASTIC FORCES ON PINS AT EQUILIBRIUM (FROM IFORCE).**
- **SAVE COMP'T POINTERS. IBEST=GAIN.**

**IBEST=0 ?**

- **yes**
  - **SWITCH THE SAVED COMP'TS IN THE SLOT**
  - **UPDATE THEIR DATUM COORDS.**
  - **RE-COMPUTE \( y_{cn} \) VALUES & FORCES**

- **no**
  - **ANY SWAPS IN THIS SLOT ?**
    - **yes**
      - **CLEAR TAGS IN ICOST FOR ALL SLOTS.**
    - **no**
      - **SET TAG IN ICOST FOR CURRENT SLOT.**

**EXIT**

*The nomenclature here corresponds to that used in 3.5*

**Fig. 6.9 The Refinement of Component Placement in a Given Slot.**
The Implementation of the Conductor Routing Algorithms.

On termination of component placement the circuit data structure contains a complete description of the layout problem, together with those parts of the solution which have been defined - namely the datum coordinates of the components relative to the board datum, and the related information associating clusters of components with the hypothetical slots on the board. The conductor routing algorithms, about to be considered in detail, must operate on this structure, extending it to include the detailed geometric information describing the tracks as they are defined.

7.1 The Geometric Data Structure.

The conductor routing problem is divided into a set of subproblems, one associated with each slot on a board, there being a corresponding division of the geometric data structure into segments, as outlined in 4.1. The design of the geometric data structure for such segments must be considered in the light of the requirements imposed by the routing scheme outlined in 4.2 - 4.6.

7.1.1 The Overall Organisation of the Data Structures.

Each segment of the geometric structure is a self-contained, ring-based data structure, occupying a 1-dimensional FORTRAN ARRAY, and using the free-storage scheme described in Appendix 1. Two segments of the entire structure (one segment per slot) can be resident in core at a given time, the remainder residing on disc, or similar fast backing store.
The circuit structure, described in Chapter 5, occupies an array, IARR. It should be noted from Appendix 1 that the circuit structure will be built from the "bottom" (high index) end of IARR, because of the method of free store allocation. The two arrays, designated PAGEA and PAGEB, used to hold the two in-core segments of the geometric structure, are taken from the "top" (low index) end of IARR by the special block allocation routine described in Appendix 1, which permits a block to be taken from free-store at a given index, if available. This permits the use of FORTRAN EQUIVALENCE statements to relate PAGEA and PAGEB to IARR, such that PAGEA and PAGEB can be addressed directly, or by way of IARR with a suitable offset. The overall arrangement is shown schematically in Fig. 7.1. The array dimensions shown in Fig. 7.1 refer to the PDP-10 implementation of the program.
The main advantage in the above scheme is that all of IARR is available during component placement, so accommodating the temporary sub-block structure used in partitioning, and thus minimising the total storage space requirement. Also, after establishing the inter-slot boundary geometry (see 4.1), only one segment is required in-core at any time, and so the array, PAGEB, used to hold the 2nd in-core segment, is freed for use as working space.
The design of the geometric data structure must take into account the requirements of the routing scheme outlined in 4.2 - 4.6, such that frequently required information is easily accessible, both in terms of speed and simplicity of the necessary coding. In addition, the structure should be compact and easily built and modified, the latter being particularly important in aperture routing and interaction, where the initial routing is modified in the process of conflict removal (see 4.5 and 4.6). The most important requirements in terms of accessing the structure are listed in Table 7.1.

<table>
<thead>
<tr>
<th>REQUIRED CAPABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Find the x- and/or y-coordinates of a feature.</td>
</tr>
<tr>
<td>2. Find what, if anything, occupies a given location.</td>
</tr>
<tr>
<td>3. Permit multiple occupation of a given location, i.e. allow features to be superimposed.</td>
</tr>
<tr>
<td>4. Follow backwards or forwards through the features of a directed tree (sub-tree).</td>
</tr>
<tr>
<td>5. Find specific attributes of a given feature, or of the track following or preceding that feature, in a directed tree.</td>
</tr>
</tbody>
</table>

Table 7.1 The Basic Requirements of the Geometric Structure.

Given that there is a conflict between achieving maximal speed and minimal storage which results in a compromise situation, the structure adopted uses a block to represent each of the geometric features of a sub-tree (see 4.5). Each feature block carries two sets of pointers (associations), one set concerned with coordinate information, and the other with the structure of the directed tree (sub-tree) of which the feature is a part.
7.1.2 The Storage of Coordinate Data in the Geometric Structure.

As it is necessary to access blocks occupying given coordinates, as well as find the coordinates of a given block, each block is tied into two rings. One of these rings links all of the blocks at a given x-coordinate in a slot, and the other links all of those at a given y-coordinate. Each of these rings has a ringhead - the fixed-x and fixed-y ringheads, respectively - containing the value of the appropriate coordinate, and these ringheads are in turn linked into one of two rings, being the ring of fixed-x ringheads or the ring of fixed-y ringheads as appropriate. The ringheads on these rings are arranged in ascending order of coordinate value, and similarly the blocks on a fixed-x ringhead are ordered in ascending order of y-coordinate, and vice-versa. As with the circuit structure, the pointers to the rings of fixed-x and fixed-y ringheads reside in reserved locations in the data structure array, whose addresses (indices) are held in a FORTRAN COMMON block, so being accessible from any part of the program. These addresses are identical, of necessity, for each segment of the geometric structure, thus enabling the structure to be entered in identical fashion for each segment.

This structure is best described by the simplified sketch shown in Fig. 7.2. Each fixed-x or y ringhead contains a pointer to the previous, as well as to the next, ringhead, but these have been omitted from Fig. 7.2 for clarity. These backward pointers facilitate the searching of a region of a slot adjacent to a given location, for example.
A detailed description of all of the above block types is to be found in Appendix 3.

Fig. 7.2  Coordinate Storage in the Geometric Structure.
7.1.3 The Geometric Structure Representation of Directed Trees.

Each sub-tree in a given slot (and hence in a given segment of the data structure) has a feature block designated as its root (see 4.5.1). In order to access the geometric structure by way of the sub-tree roots, rather than by coordinates, a ring of root-pointer blocks based on one of the reserved pointer words is used, each root-pointer block containing a pointer to a sub-tree root. This is illustrated in outline in Fig. 7.3. It will be noted from Fig. 7.3 that each root block (a feature block) contains a pointer to its root-pointer block, which in turn contains a pointer to the net block in the circuit structure with which the sub-tree on the root is associated. Thus each tree is linked indirectly, via its root, with its associated net.

The directed tree structure representing sub-trees is, in essence, straightforward, a pointer being used to point from each feature block to the following one in a tree. Reverse pointers are also held, permitting movement towards, as well as away from, the root of a tree. For the root block of a tree it is this pointer which points to the root-pointer block (see Fig. 7.3). Each feature block also contains a pointer to the root of its tree, hence permitting easy access from any feature block to its associated net block in the circuit structure.

Such a structure would be adequate if the trees were no more than chains of branches, but this is not the case and so TIE blocks are added to the list of feature types. The junction of n track segments in a tree is represented structurally by a ring of (n-1) tie blocks (a TIERING), one for each of the track segments directed away from the
root of the tree. The end of each branch points back to the root of the tree. Fig. 7.4 depicts a simple tree consisting of two pins and a dummy pin with the track forming a T-junction. The coordinate rings and root and backward tree pointers have been omitted from Fig. 7.4 for clarity.

Fig. 7.3 The Arrangement of Root-Pointer Blocks.
Fig. 7.4 Sub-Tree Representation in the Geometric Data Structure.
Where a T-junction, or similar arrangement, involving the meeting of more than two track segments of a sub-tree (whether on the same side of the board or not) is coincident with a distinct feature block - PIN or THROUGH-HOLE - the ring of \((n-1)\) ties is built following the pin or hole in the tree structure. The pin or hole and each of the ties are inter-dependent, occupying the same coordinate rings. A marker is set in each block of such an inter-dependent set, because it is necessary, for example when moving a pin or hole, to consider the "following" structure. An independent pin has only one "following" track segment, while a dependent pin has \((n-1)\) "following" track segments where there are \((n-1)\) ties in the dependent tiering. The marker bit is thus useful in "flagging" such situations.

A detailed description of all of the feature block types is to be found in Appendix 3.

7.1.4 The Temporary Data Structure used in Aperture Routing

It will be remembered from 4.5 that the "aperture" used in conflict minimisation is the width of a slot, its height being defined, at any instant, by the separation of two adjacent features in the slot in the \(y\)-axis. The top and bottom of the aperture will, therefore, correspond to two adjacent fixed-\(y\) rings of feature blocks. It is desirable, in order to simplify coding and reduce processing time, to have more information directly available for aperture routing than is stored in the feature blocks. For example, the storage of coordinate values, rather than pointers which access those values by following a ring to a coordinate ringhead, is to be preferred in terms of speed.
and simplicity of coding. Also, extra information, such as the coordinates of the next block in the tree, is stored.

Consequently a subsidiary structure is built, consisting of two rings of blocks, one ring associated with the fixed-y ring at the top of the aperture, designated the TEMPORARY ring (abbreviated to TMP), and the other associated with the fixed-y ring at the bottom of the aperture, designated the IMPEDIMENT ring (because many of the features at the bottom of the aperture impede the downward passage of tracks - abbreviated to IMP). The ringheads of these rings reside in the reserved pointer words of a given segment, along with the root-pointer ringhead, etc. Each of these rings contains a block associated with each of the feature blocks in the corresponding fixed-y ring, and, in addition, a block corresponding to each track segment cutting across the appropriate y-coordinate (i.e. those track segments, parallel to the y-axis, with one feature block above, and one below, the appropriate y-coordinate). This is the only instance in the data structure of a point on a track segment, other than one of its ends (features), having an explicit structural representation. These two categories of TMP and IMP blocks will be designated "feature" and "conductor" TMP and IMP blocks, respectively. Each block, TMP or IMP, contains a pointer to its associated feature block or, where it represents a point on a track segment, to the "first" block of that segment (i.e. the one nearer to the root of the tree).

A detailed description of these block types is to be found in Appendix 3.
Fig. 7.5 (b) shows a simplified example of a fixed-y ring and the corresponding temporary ring for the trivial situation depicted in 7.5 (a), in which only a pin and a vertical track segment exist at the given y.

7.1.5 Building the Initial Geometric Data Structure.

For each slot on a given board it is necessary to generate a segment containing the geometric structure relevant to that slot. These segments are of identical, pre-determined, length (see 7.1.1), and are stored in a FORTRAN RANDOM ACCESS FILE held on disc, the n'th record of the file containing the segment describing the n'th slot on the board. The first step, then, in creating the geometric structure, is the creation of a random access file on disc, with k records of length as above, where k is the number of slots on the board. As each segment (record) of the structure has its own free store management scheme, the reserved words used for this (see Appendix 1) are set up in PAGEA, after allocating PAGEA and PAGEB from the free store of IARR as described in 7.1.1. Also, in order that the ringheads (by way of which a segment is accessed) for each segment are addressable in identical manner, the block of reserved pointer words is taken from PAGEA free store, the appropriate pointers (indices) to these words being set into the COMMON locations used for accessing these ringheads. PAGEA is then copied into each of the k records of the random access file.
Fig. 7.5  The Temporary Ring Structure used in Aperture Routing.
ENTER

TAKE PAGEA AND PAGEB FROM FREE STORE OF IARR.
INITIALISE FREE STORE SCHEME IN PAGEA.
TAKE BLOCK OF RESERVED POINTER WORDS FROM PAGEA.

CREATE RANDOM ACCESS DISC FILE
WITH k RECORDS AND COPY PAGEA
INTO EACH OF THE k RECORDS.

FOREACH SLOT, I = 1,k:

READ I'th RECORD FROM DISC TO PAGEA

FOREACH COMPONENT IN SLOT I, DO:

CREATE PIN COMPONENT IN BLOCK IN PAGEA.

TIE PIN BLOCK TO X & Y RINGS.

WRITE PAGEA TO I'th DISC RECORD.

EXIT

Fig. 7.6 Initialising the Geometric Structure.
With the random access file thus initialised, those parts of the geometric structure which are fixed, namely the component pins, can be added to the appropriate segments. For the n'th slot this involves reading the n'th record from the disc file into PAGEA, adding the appropriate pin blocks, together with the necessary x and y ringhead blocks, and copying PAGEA back into the n'th record of the disc file.

This procedure is outlined in the flowchart in Fig. 7.6.

7.2 The Assignment of Dummy Pins to Specific Locations.

As outlined in 4.1, the subdivision of the routing problem into a set of independent subproblems, one for each slot of the board, requires that the geometry of the inter-slot boundaries be defined.

Given that for any net only one connection will exist between any two adjacent slots containing sub-trees of that net, the problem reduces to the allocation of dummy pins to specific y-coordinates on the inter-slot boundaries, the x-coordinates being implicit in the boundaries, as determined by the placement routines.

The basic strategy used in assigning dummy pins to specific y-coordinates is outlined in 4.1.1, the main objective being the avoidance of unnecessary track segments parallel to the y-axis. This is achieved by favouring the use of the "overlap" regions, shown in Fig. 4.3, for the horizontal inter-slot connections, and hence for the dummy pins.

In addition, however, it is desirable that the inter-slot connections should avoid the use of those y-coordinates occupied by
component pins, as these pins, in many cases, would act as obstacles to the horizontal inter-slot routes. The avoidance of these coordinates results in a slightly less complex routing with a resultant increase in wireability. Further, it is desirable that dummy pins should be spread uniformly along a given boundary, within the limits of the basic strategy, in order to avoid congestion in any particular region of a slot. Both of these secondary objectives are achieved, as best they can be, through the use of an "obstacle profile" along the y-axis of the board. This is held in a 1-dimensional FORTRAN ARRAY, "IXOBS" (actually taken from the free store of IARR), which has one word corresponding to each potential dummy pin location along an inter-slot boundary, given that such locations are separated (in y) by the default value of track width and separation. Each word of IXOBS contains a specific value indicating the status of the corresponding y-coordinate for the boundary under consideration. Where a component pin exists at a given y-coordinate (and at any x on the board), the corresponding word in IXOBS is set to -1, indicating that the use of that y for a dummy pin is undesirable, though not forbidden. Where a dummy pin is allocated to a particular y-coordinate, a pointer to the dummy pin block is placed in the corresponding word of IXOBS. For the remaining words in IXOBS, corresponding to the remaining usable dummy pin locations, a value of -10n is set into each word, where n is the number of adjacent free locations in a given group. The highest value of n, therefore, corresponds to the largest "gap" in the y-axis (in terms of pins and dummy pins, that is). Where a choice is available in siting a dummy pin (the "overlap region" referred to above), that location with the highest negative value in IXOBS is to be preferred, -1 being "undesirable", as indicated above.
In defining dummy pin coordinates, the coordinates of each (component) pin on the board are available from the circuit structure. Slots are considered from left to right across the board, the "current" slot being held in PAGEA and the "next" slot in PAGEB, while defining the dummy pin coordinates at the right hand boundary of the current slot. Fig. 7.7 shows a part of two adjacent slots and the corresponding entries in IXOBS for the inter-slot boundary. For simplicity, it has been assumed in Fig. 7.7 that no pins exist at other y-coordinates in other slots, and so the -1 values in IXOBS are as indicated.

The procedure which allocates the dummy pins to specific y-coordinates is outlined in the flowchart in Fig. 7.8. It should be noted from this that, where a net occupies slots n and m, m > n + 1, but not the intermediate slot(s), the program endeavours to route horizontally between slots n and m, such routes being given priority in the allocation of dummy coordinates. This makes for simplicity in the routing, and consequently for high wireability. Dummy pins are not handled by this procedure for the power and ground nets, these being dealt with interactively as described in 8.3.

7.3 The Generation of Sub-Tree Route Plans.

After the allocation of dummy pins to specific y-coordinates, as described in 7.2 and 8.3, the routing of each slot may be considered as an independent subproblem. The first step in the routing of a given slot, as indicated in 4.2, is the generation of near-minimal Steiner trees, one for each net or part of a net in the slot, disregarding conflicts between such trees, these being the route plans. Given that such trees will be modified, in general, in the
subsequent conflict removal stages, there is no advantage to be gained from generating Steiner-minimal trees (at a high cost in terms of execution time, program code size, and coding effort). Minimal length is thus a secondary objective in the tree generation process, the primary objective being the attainment of high wireability through the use of "simple" routes parallel to the y-axis where the interconnection density is likely to be high. The scheme adopted is outlined in 4.3, and only a few minor implementation details need be dealt with more fully.

Fig. 7.7  The Obstacle Profile (in IXOBS) at a Slot Boundary.
ENTER

FOREACH SLOT PAIR, I, I+1, DO:

GET RECORDS I AND I+1 (FOR SLOTS I AND I+1) INTO PAGEA AND PAGEB.
SET OBSTACLE PROFILE IN IXOBS.

SET L = C = N = 0.
(L=LAST, C=CURRENT, N=NEXT).
IF SLOT K HOLDS FEATURE OF NET, SET:
C=I IF K=I
L=I IF K<I
N=I IF K>I
FOR THAT K NEAREST TO I.

FOREACH NET OF CIRCUIT, DO:

EXIT

ADD DUMMY TO R.H.S. OF SLOT I+1

ADD DUMMY TO L.H.S. OF SLOT I+1

ADD DUMMY AT R.H.S. OF SLOT I.

SEARCH OUT FROM OVERLAP FOR BEST Y

FIND BEST Y IN OVERLAP FROM IXOBS.

BEST Y FOUND?

FIND MIN AND MAX Y IN SLOTS C & N.

HENCE FIND OVERLAP(Y) RANGE.

ALL OF NET LEFT/RIGHT OF SLOT I?
yes
no

ANY OF NET TO RIGHT OF SLOT I?
yes
no


Fig. 7.8 Dummy Pin y-coordinate Allocation.
The first step in designing the route plan for a given net in a
given slot is the creation of a list of the nodes (component and dummy
pins) of the tree, in order of ascending y-coordinate. An array,
IWORK (actually taken temporarily from the free storage of IARR) is
used to hold this list, each member of the list occupying a 3-word
block in IWORK, the first of these words holding a pointer to the node
(pin), and the remaining two holding the x and y coordinates of the
node. A variable, NOPINS, holds the number of such blocks (nodes)
in IWORK for a given tree. As the tree is built, with the addition
of bends and ties as necessary, further entries are made in IWORK
for each such block, and another variable, NEWNO, holds the total
number of entries at any time (NEWNO = NOPINS + no. of bends and ties).
Having ordered the initial set of nodes thus (the component and dummy
pins, that is), the last in the list (block no. NOPINS) is taken as
the root of the tree (see 4.5.1) and a root-pointer block is created
pointing to it, as described in 7.1.3. Having selected the root, a
pointer to it is added into each of the node blocks (pins and dummies),
as described in 7.1.3. The arrangement of blocks in IWORK is
illustrated in Fig. 7.9 for a tree containing a number of bends and/
or ties.

Before the planning of the sub-tree can be undertaken it is
necessary to find the mean x-coordinate of the list of nodes in IWORK
(at this stage IWORK contains only the pin blocks, no bends or ties
having been added). The use of this x-coordinate is preferred for
those parts of the tree running parallel to the y-axis, as indicated
in 4.3. It may be observed, however, that many nets in a slot will
have a mean x which is near to the centre of the slot. While not
attempting to avoid specific conflicts between trees at this stage, it is nonetheless desirable to avoid global problems which are predictable, such as the overcrowding of vertical routes near to the centre of the slot through using the scheme as stated above. To relieve this predicted congestion, the mean x coordinates are modified in such a manner as to spread them out from the centre of the slot while maintaining their positions relative to each other. This is achieved through the use of a simple sine function, as follows.

Fig. 7.9 List of Nodes of Sub-Tree in "IWORK".
Given MEANx, the actual mean of the x-coordinates of the nodes, expressed relative to the board datum, IXCEN, the x-coordinate of the centre of the slot, and ISTRIP, the width of the slot, the mean x, MX, expressed relative to the centre of the slot is given by:

\[ MX = MEANx - IXCEN \quad \text{where} \quad -ISTRIP/2 < MX < ISTRIP/2 \]

This is then transformed using the sine function (argument in degrees), as follows:

\[ MX' = \sin((MX \times 180)/ISTRIP) \times ISTRIP/2. \]

After transformation the mean x coordinates are referred back to the board datum and rounded onto a grid of size equal to the sum of the default values of track width and separation (i.e. the centre-to-centre track spacing). This transformation is depicted in Fig. 7.10 which shows, in schematic form, its effect on conductors passing down the slot "through" the transform.

Having dealt with the preliminaries, the generation of the sub-tree spanning the "NOPINS" nodes in IWORK may now be considered. The tree initially consists of the root block alone. Blocks are then added into the tree, one at a time, in descending order of y-coordinate, as outlined in the flowchart in Fig. 7.11. As bends and ties are created, an appropriate 3-word entry is added for each to the end of the list in IWORK. Searching the blocks of the tree is then a simple list search rather than the more complex tree search otherwise required.

The data structure generated in tree building is described in 7.1.3, the various block types being described in detail in Appendix 3.
7.4 The Resolution of Conflicts Between Route Plans.

Having described the generation of sub-trees (route plans) for each slot, it is now necessary to consider the removal of conflicts between these trees. This is achieved in part automatically, through the use of a stepping aperture, as outlined in 4.5, and in part interactively, as outlined in 4.6. The implementation of the latter is described in detail in 8.4, along with the description of the other interactive parts of the program.

Fig. 7.10 Spreading the Mean x-coordinates out from the Slot Centre.
CREATE ROOT-POINTER BLOCK.

FOREACH NODE, IBLOC, IN IWORK:

FIND JBLOC IN TREE, NEAREST TO IBLOC.

BREAK BRANCH, INSERT IBLOC INTO BRANCH.

DELETE JBLOC REPLACE IN TREE BY IBLOC.

INSERT IBLOC BEFORE JBLOC IN TREE.

WHERE JBLOC BELONGS TO A HORIZONTAL TRACK SEGMENT SPANNING IBLOC IN X, BREAK THE SEGMENT AND INSERT A TIE, CALLING IT JBLOC.

ADD IBLOC TO TREE AFTER JBLOC.

JOIN IBLOC TO JBLOC, WITH 1 or 2 BENDS, KEEPING VERTICAL PART OF TRACK AS NEAR TO MX AS POSSIBLE.

Fig. 7.11 Building the Sub-Tree for a (part of a) Net in a Slot.
7.4.1 The Generation and Control of the Stepping Aperture

An aperture of variable height is stepped from top to bottom of a given slot, as outlined in 4.5.1, conflicts being resolved in that part of the slot "covered" by the aperture at any given time. In addition, lookahead is used to predict future (lower in the slot) conflicts, an attempt being made to avoid these, where possible, by modification of the routing within the "current" aperture. By "current" aperture is meant the aperture in its current position - though only one aperture exists, covering different regions of the slot at different times, it is convenient for descriptive purposes to refer to a plurality of apertures, one for each position which the actual aperture occupies. The aperture is bounded in x by the slot boundaries, and in y by successive pairs of adjacent fixed-y rings (see 7.1.2), so that the aperture height corresponds to the y-separation of features in the slot which occupy adjacent (but not equal) y-coordinates. The top and the bottom of the aperture are represented by two rings of blocks in the data structure, the temporary and impediment rings respectively (abbreviated TMP and IMP), as described in 7.1.4. The "generation" or "building" of an aperture refers to the creation of these two rings.

The flowchart in Fig. 7.12 describes the control of the stepping aperture for a given slot. It should be noted that while the aperture may be modified in the process of conflict removal, as indicated in 4.5.1, this does not affect the procedure in Fig. 7.12 as only the bottom (impediment ring) of the aperture is altered in such instances.
While the initial allocation of each track segment to a specific side of the board (solder or component side) might properly be considered a part of route planning, it is included prior to the minimisation of conflicts within each aperture (see Fig. 7.12). This, as indicated in 4.5.2, makes use of the most accurate geometric
information available (without resort to backtracking), and so results
in the best initial allocation. As with the x, y coordinates of the
route plans, the initial side allocation may be amended by the conflict
minimisation routines described in 7.4.3.

7.4.2 The Initial Allocation of Track Segments to Board Sides.

From Fig. 7.12 it will be noted that, for a given slot, side
allocation takes place for unallocated track segments with a feature
(the start of the segment) on the TMP ring for the first (top) aperture
in the slot, and for those with a feature on the IMF ring for each
aperture in the slot. The side allocation procedure, as outlined in
4.5.2, does not necessarily allocate track segments one at a time, but
allocates connected parts of a sub-tree, bounded by through-holes or
component or dummy pins, or else limited through the permitted span on
the wrong side of the board being exceeded in both axes, the latter
forcing the insertion of a through-hole, either replacing a bend or
associated with a tiering (and therefore inter-dependent with that
tiering - see 7.1.3).

As track segments due for allocation to a particular side of the
board are extracted from a sub-tree by the allocation routines, some
information regarding each feature block of the sub-tree associated
with such segments is stored in temporary workspace for use by the
allocation and associated tree-searching routines. The workspace is
an array, KWORK (taken temporarily from the free store of PAGEA -
see Fig. 7.12), which holds a list of entries, one for each extracted
feature block. Each entry in KWORK occupies 4 words, containing a
pointer to a feature block, together with its x and y coordinates and
its type (see Appendix 3).

Fig. 7.13 outlines the side allocation procedure for a given TMP or IMP ring. For clarity, certain complications have been omitted from this flowchart. For example, where a through hole is added to a sub-tree at the same $y$-coordinate as that associated with the TMP or IMP ring being processed, it is necessary to restart the procedure at the beginning of that ring to ensure that the track segment(s) starting on the newly added hole are allocated by the procedure.

7.4.3 Route Plan Modification Within a Given Aperture.

As indicated in 4.5.3, modification of the $x$, $y$ coordinates and side allocation of the route plans within a given aperture of a given slot is used as a means of minimising conflicts, both within that aperture and, with the aid of lookahead, in subsequent apertures (at lower $y$ in the slot).

The process used is not a true conflict "minimisation" process (which would require an iterative procedure, or equivalent), but does attempt, in a simple-minded fashion, to minimise the number of conflicts by "fixing" track segments within the aperture in a specific order, according to the category into which each segment falls, dealing in general with the simplest cases first. The start-point for the minimisation of conflicts in a given aperture is the set of route plans (tree structures with coordinates and board side allocated for each feature and track segment) which lie partially or wholly within a given aperture. The component and dummy pins of such route plans are "fixed" - i.e. their coordinates may not be modified in the conflict minimisation process. The conflict minimisation routines deal with the track segments
ENTER

FOREACH TMP/IMP BLOCK, DO:

CONDUCTOR BLOCK? yes no

GET MBLOC, THE ASSOCIATED FEATURE BLOCK.

MBLOC IS TIE, BEND OR BRANCH END? yes no

ASSIGN PART OF SUB-TREE TO SIDE

IF SIDE = 2, SET SIDE=0/1 FOR $\delta x/\delta y$.

GET NEXT TRACK SEGMENT FROM SUB-TREE. ADD DATA INTO KWORK. IF FIXED SIDE, SET SIDE = 0/1.

DELETE BLOCK(S) FROM KWORK FOR END OF SEGMENT CAUSING CONFLICT. FIND IBLOC AT START OF CONFLICTING SEGMENT.

IBLOC=BEND? yes no

CONVERT BEND INTO THROUGH-HOLE

INSERT HOLE BEFORE IBLOC (TIE).

END TREE SEARCH ON THIS BRANCH

SET DEPENDENT MARKERS IN HOLE & TIES.

CONFLICT IN SIDE ALLOCATION? yes no

MORE TRACK SEGMENTS TO ADD?

ADD MBLOC DATA INTO KWORK AND SET SIDE=2. IF MBLOC FIXED SIDE SET SIDE=0/1 (INPUT CONSTRAINT)

Fig. 7.13 The Initial Side Allocation Procedure.
occupying the aperture, ordered by categories as indicated above, fixing (after modification where necessary) the coordinates and side allocation of each in turn. Where necessary the tree structure itself may be altered in this process. When the features at each end of a track segment are fixed, and no conflicts exist along the length of the track, then that track segment is said to be fixed.

It will be noted from 7.4.1 that no feature may exist between the top and bottom of an aperture, and so the information contained in the TMP and IMP rings is a complete description of the aperture as regards tracks parallel to the y-axis and features at the aperture boundaries. For tracks on the TMP or IMP rings, parallel to the x-axis, however, the description is inadequate as only the ends (feature blocks) of such track segments have an explicit representation in the data structure. To avoid the excessive program code and processor time required to deal with this implicit representation of such track segments, an array, NEXUS (x-usage; taken from the free store of PAGEA - see Fig. 7.12), is used to hold a detailed "map" of each side of the board for a given aperture. The aperture is divided in x into units equal to the track-to-track spacing in use, each such increment being termed a "via". Each location of NEXUS corresponds to a specific x,y coordinate (via, TMP/IMP) and a specific board side, as can be seen from Fig. 7.14. Where a sub-tree block is fixed, the corresponding NEXUS location contains a pointer to the associated TMP/IMP block, and where a horizontal track segment is fixed, each via intermediate between its ends (fixed feature blocks) contains a pointer to the TMP/IMP block associated with the start of the track segment. To determine whether a given location in the
current aperture is occupied by a fixed block or a part of a fixed track segment, it is only necessary to examine the appropriate word in NEXUS, the offset being given by:

$$\text{OFFSET} = \text{VIA No.} \times 4 - \text{SIDE} \times 2 - \begin{cases} 1 & \text{for TMP} \\ 0 & \text{for IMP} \end{cases}$$

If $\text{PAGEA}(\text{NEXUS} + \text{OFFSET})$ is zero, then the location is free. Where a TMP or IMP block is fixed (feature or conductor), it contains the offset in NEXUS corresponding to its location in the aperture (see ALLOX in Table A3.10).

![Diagram](image)

**Fig. 7.14** Arrangement of Aperture "Map" in NEXUS.
As outlined in 4.5.3, a profile of the slot (in x) is maintained as an aid to the use of lookahead in the avoidance of future (lower y) conflicts. The profile is held in an array IYOBS (y-coordinates of obstacles; taken from the free store of PAGEA – see Fig. 7.12), which has one word for each via position across the slot (including the slot boundaries). For a given aperture, IYOBS contains, for each via, the y-coordinate of the highest fixed obstacle in that via below the bottom of the aperture (or zero if none exists).

Having outlined the approach to conflict minimisation in 4.5.3, and having described the necessary data storage mechanisms in 7.1.4 and above, the scheme may now be considered in some detail. The complete process involves the application of a set of routines to a given aperture, each routine dealing with a specific arrangement of track segments and/or feature blocks in a given sub-tree. Each routine must, therefore, recognise the arrangements with which it must deal, and endeavour to fix as many as possible of those existing in a given aperture, modifying coordinates, side allocation and the tree structure itself, where necessary. Table 7.2 lists these routines in the order in which they are applied to a given aperture. For each routine an example is shown of the type of situation dealt with, the left hand diagram corresponding to the situation before the application of the routine, and the right-hand one to that after its application. Planned (unfixed) tracks are shown dashed, while fixed track segments are shown as continuous lines.
1. Where a track segment starts on a fixed TMP block and ends on a fixed IMP block, both at the same x, the track segment is fixed. Fixed segments are shown solid, unfixed ones dashed.

e.g.

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FIXED FEATURES</td>
<td>SLOT BOUNDARIES</td>
</tr>
</tbody>
</table>
-----------------------------------------------
```

2. Where a track segment starts on a fixed TMP block and ends on an unfixed IMP block at the same x, and no conflict exists, the IMP block and the track segment are fixed.

e.g.

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FIXED FEATURES</td>
<td>UNFIXED FEATURES</td>
</tr>
</tbody>
</table>
-----------------------------------------------
```

3. Where a track segment starts on an unfixed TMP block and ends on a fixed IMP block at the same x, and no conflict exists, the TMP block and the track segment are fixed.

e.g.

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TMP</td>
<td>IMP</td>
</tr>
</tbody>
</table>
```

Table 7.2 Continued Overleaf.
4. Where a track segment starts on a fixed TMP block and ends on a fixed TMP block, and no conflict exists along the length of the track, the track segment is fixed. If a conflict exists, an attempt is made to switch the track to the other side of the board with the addition of any necessary through-holes.

   e.g.

   ![Diagram](image)

5. Where a track is planned to go vertically down from a fixed TMP block to a block at the same x below the IMP ring, and is obstructed in the IMP ring or at some lower point defined in the obstacle profile IYOBS, the track is cranked out to avoid the obstacle.

   e.g. (1)

   ![Diagram](image)

   e.g. (2)

   ![Diagram](image)

Table 7.2 Continued Overleaf.
6. Where a TMP block, other than a conductor block, is not fixed, endeavour to find a suitable x at which it may be fixed, causing no conflicts in any of its associated track segments. If successfully fixed, any associated track segments in the TMP ring are fixed.

-e.g. (1)

![Diagram](image1)

-e.g. (2)

![Diagram](image2)

7. Where a track segment goes vertically down from a fixed TMP block (which may be a conductor block), to an unfixed IMP block (which again may be a conductor block) at the same x, and no conflicts exist in the IMP ring, then the IMP block is fixed. If the IMP block is a feature block, the track segment is also fixed.

-e.g.

![Diagram](image3)

Table 7.2 Continued Overleaf.
8. Where a track segment starts on a fixed TMP block and goes vertically down below the IMP ring, and the TMP block x-coordinate has been altered by 5. or 6. above such that the new x no longer corresponds to the target x below IMP, a bend is inserted at (new x, target y), unless the target is a bend or tie, in which case its x-coordinate is modified.

   e.g.

   ![Diagram](image1)

9. Where the next fixed-y ring lower than the IMP ring is at least 2 via widths below the IMP ring, and where a route is planned to go vertically down from an IMP block and then horizontally at this next lower y, and is blocked by a fixed obstacle, a bend is inserted such that the horizontal part of the route is in the next lower via than IMP, but above the obstacle.

   e.g.

   ![Diagram](image2)

Table 7.2 The Steps in Conflict Minimisation.
The Implementation of the Facilities for Graphical Interaction.

Graphical interaction is used in the layout program to:

(i) modify the program generated placement, before routing (see 3.6.),
(ii) insert the route plans for the power and ground nets (see 4.4),
and (iii) complete and/or modify the program generated routing (see 4.6).

Interaction is seen as fulfilling a number of roles. The first is that of using the designer's ability to locate and remove the source of any problems (local or global), thereby supplementing the program algorithms which have a very limited capability in this respect. The second is that of supplementing the program algorithms where they are simply not capable of handling (i.e. not designed to handle) specific situations. This arises, for example, in the routing process where the heuristic scheme described in Chapter 7 does not cater for all of the situations that can arise. There is, inevitably, a trade-off between the cost of writing program code to deal with infrequently occurring situations and the cost of handling such situations interactively as they arise. This leads to the third use of interaction, namely, as a program design aid, where the user, with experience, will be able to define those situations which the program is most frequently unable to handle, so determining where any effort in extending the automatic algorithms would be best spent. It may be thought that interaction could be used to "design itself out" of the program in the above way, but it seems highly improbable that, for complex and unique layout design problems, one could ever hope to dispense with interactive aid.
Before considering the details of the interactive facilities in 8.2 - 8.4, it is necessary to describe briefly the system software available for graphical output and interaction.

8.1 System Software for Graphical Interaction.

The hardware configuration of the system on which the layout program has been developed is described in Appendix 4. It is sufficient to note here that the host machine (PDP-10), in which the layout program runs, is connected by a high-speed 2-way link to the graphics satellite (a PDP-7 driving a 340 refresh display, with a light pen). A set of FORTRAN-callable subroutines is available for the generation and manipulation of 2-dimensional pictures on the 340 display (the SPINDLE package (28)). The user program in the host machine, through appropriate use of the SPINDLE subroutines, generates a "display file" (display code for a given picture) which is sent via the link to the satellite machine. In the satellite an executive program interprets the display file continuously, driving the refresh display.

The graphical output routines include point setting, absolute and incremental line drawing (scale and intensity being set by a parameter setting routine), and character drawing, among others. A subpicture facility is available, similar to the FORTRAN subroutine facility, which simplifies the generation, and reduces the length, of a display file, where multiple copies of a given picture part are required at different points in a picture. This is used, for example, in the drawing of components, where a subpicture is generated for each master (see 5.1), each instance (component) of that master being drawn using
The layout program makes use of the light pen as a means of man to machine communication. When a light pen hit on a given picture part is recorded, an "attention block" is generated in the satellite, containing details of the hit - the display x and y coordinates of the hit and the index in the display file of the start of the display segment containing the picture part which was hit (the "system name" of the segment). Routines are available which enable the user program to "read" such attention blocks from the satellite. In order that light pen hits may be directly related to specific entities in the user data structure, it is desirable that the system name of each segment be stored in the appropriate parts of the layout data structure. Thus, for example, each component block contains the system name of the appropriate display segment in the placement interaction program (see COMPIC in Table A3.2). Similarly, as each display segment has a "user name", supplied at the time of
creating the segment, the user name of a "component" display segment is a pointer to the component block in the circuit data structure. In this way the display segment can be accessed directly from the component block, or vice-versa.

In order that the light pen can be used to indicate display screen positions when nothing is being displayed, a tracking cross facility is provided. The tracking cross is initialised by a routine called from the user program and can be dragged over the display screen by the light pen. A display segment may, under user program control, be dragged with the tracking cross. The loss of tracking (by closing the shutter on the light pen) causes the creation of an attention block containing the coordinates of the tracking cross at the time at which tracking was lost. In component placement interaction, for example, when moving a component, the tracking cross is set on the component datum and dragged by the light pen, along with the "component" display segment, to the desired location. Tracking is then intentionally lost, returning the new display datum coordinates of the component to the user program for appropriate action.

The above description is intended only to provide an outline of the facilities used, and the reader is referred to the SPINDLE Users Manual (28) for a comprehensive description of the available graphics facilities.

8.2 Interaction with the Program-Generated Placement.

The primary function of placement interaction is that of making
use of the user's ability to recognise problem areas (e.g. regions of high interconnection density) and to see good ways of removing such problems by suitable modification of the placement. There is a secondary function in that placement interaction permits the user to modify the placement to take account of specific constraints (for example, that a given component should occupy a specific board location, or that a given region of the board should be unoccupied). Such constraints are not currently handled by the automatic program, though it is proposed that the program be extended to cater for them (see 11.1.1).

8.2.1 The Facilities Provided for Placement Interaction.

The graphical output generated by the program consists of the following:

(i) The outline of the board, scaled so as to maximise the use of the display screen.

(ii) The component outlines in their current positions, drawn to the same scale as (i) above, and labelled with the user-provided name.

(iii) The "coarse grid" on which components are placed (the slot width in x, and 0.550" in y). The display of this grid is optional (see "GRID" in Table 8.1).

(iv) The straight-line pin to pin connections for each net except the power and ground nets. The particular spanning tree drawn for a given net is as outlined in 4.1.1. The display of the pin to pin connections is optional (see "PIN-PIN" in Table 8.1).
(v) The total Manhattan wirelength of the spanning trees as in (iv) above for the current placement and for the initial placement (prior to any interaction). This is provided to assist the user in assessing the merit of any changes made.

(vi) A set of "light buttons", as listed in Table 8.1.

Fig. 3.6 shows the picture generated for placement interaction for one of the circuits used to test the program.

The light buttons mentioned in (vi) above consist simply of display segments, each containing the display code for a character string such as "MOVE", etc. Light buttons are used primarily to allow the user to control the program interactively (within the limits of the program design, of course), a light pen hit on a given light button causing transfer of control to a specific part of the user program, and so initiating a particular sequence of events, as described in 8.2.2. The light buttons and their functions are as listed in Table 8.1.

8.2.2 The Implementation of Placement Interaction.

The interactive procedures are probably best described by the flowchart/state diagram of Fig. 8.1 in which circles are used to represent the state of the program while waiting for a response from the user. An invalid user response results in no net action, this being indicated by the small loop leaving and re-entering each waiting state in Fig. 8.1. The trivial actions (DUMP, PIN-PIN, and GRID) have been omitted for clarity.
Fig. 8.1 State/Flow Diagram for Placement Interaction.
<table>
<thead>
<tr>
<th>LIGHT BUTTON</th>
<th>ACTION INITIATED BY LIGHT PEN HIT ON BUTTON</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT</td>
<td>Remove program control from the user and exit from the placement interaction routines (to the main program).</td>
</tr>
<tr>
<td>DUMP</td>
<td>Save a copy of the current display file on disc for subsequent processing and plotting.</td>
</tr>
<tr>
<td>PIN-PIN</td>
<td>Switch on or off the display of the pin to pin connections, if it was off or on, respectively.</td>
</tr>
<tr>
<td>GRID</td>
<td>As for PIN-PIN above, but for coarse grid.</td>
</tr>
<tr>
<td>SWAP</td>
<td>Swap the two components defined by the next two light pen hits.</td>
</tr>
<tr>
<td>MOVE</td>
<td>Move the component defined by the next light pen hit. When hit, the tracking cross is set at its datum and the user can drag it to the desired position and lose tracking. The program then rounds the coordinates to fit the coarse grid.</td>
</tr>
</tbody>
</table>

Table 8.1  The Light Buttons used in Placement Interaction.

The program does not check for illegal user actions, such as moving a component so that it overlaps another one, as this allows faster interaction in certain cases, and it is assumed that the user will correct such conflicts.

Figs. 3.6 and 3.7 show the displayed picture before and after placement interaction, respectively, for one of the circuits used to test the program.

8.3  The Interactive insertion of Route Plans.

As indicated in 4.4, the program-generated route plans, permitting only one branch of a given net to cross a given inter-slot boundary, do not provide a "good" solution to the routing problem in
certain cases. "Good" in this context is taken to mean of near minimal wirelength and such as to favour high overall wireability of a given board. Specifically, the automatically generated solution for power and ground nets would, in general, lead to an electrically poor solution for such nets and low overall wireability due to congestion in the y-axis of each slot (see Fig. 4.7). The route plans for such nets, therefore, are not generated by the automatic program, but by the user, by means of the interactive routines described below.

8.3.1 The Approach to Interactive Route Planning.

Given the component pins of a net, each occupying a specific location on the board, and the inter-slot boundaries, a graphical input scheme is required which permits the user to "define" the sub-trees which connect the pins of the net. As automatic routines exist for generating a sub-tree on a given set of geometrically fixed nodes (component and dummy pins) within a slot, as described in 4.3 and 7.3, it is both unnecessary and wasteful to perform this part of the task manually. Given, then, that the detailed generation of sub-trees should be handled by the automatic procedures available, the only information required for the definition of a given sub-tree is a list of the nodes of that sub-tree, each geometrically fixed. In terms of interactive input this requires that the user create such dummy pins as are necessary for a given sub-tree, define their coordinates, and list all of the component and dummy pins of that sub-tree.
As implemented, the program stores the interactively generated information for all of the sub-trees of all interactively planned nets in temporary workspace (see 8.3.3). On completion of interactive route planning, this data, suitably processed, is passed to the automatic sub-tree generation routines, as described in 8.3.3. As the detailed route plans are not generated at the time of interactive input, and so cannot be displayed, a simplified picture is generated consisting of straight line connections joining the nodes of a given sub-tree as they are listed by the user. This enables the user to see readily which pins of a given net have been connected.

8.3.2 The Facilities Provided for Interactive Route Planning.

The graphical output required by the user in route planning consists of the following:

(i) The inter-slot boundaries, defining the limits of sub-trees in the x-axis.
(ii) The pins of all components.
(iii) Easily recognisable markers on each component pin of the net for which the user is currently defining route plans (sub-trees).
(iv) For a given net, straight line connections depicting those sub-trees already defined by the user at any given time (see 8.3.1).
(v) A set of light-buttons, enabling the user to control the program, as listed in Tables 8.2 and 8.3.

Examples of the above graphical output are to be seen in Figs.
4.8–4.10, showing different stages in the planning of a net.

The various program actions which the user may initiate are listed in Table 8.2, one for each of the light buttons mentioned in (v) above.

<table>
<thead>
<tr>
<th>LIGHT BUTTON</th>
<th>ACTION INITIATED BY LIGHT PEN HIT ON BUTTON</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT</td>
<td>Remove program control from the user and exit from the interactive routines (to the main program).</td>
</tr>
<tr>
<td>DUMP</td>
<td>Save a copy of the current display file on disc for subsequent processing and plotting.</td>
</tr>
<tr>
<td>SAVE</td>
<td>Having completed the planning of routes for a given net, save the necessary data and clear the pin markers and connections from the display.</td>
</tr>
<tr>
<td>KILL</td>
<td>Delete the data for, and the display representation of, the most recently defined sub-tree of the current net.</td>
</tr>
<tr>
<td>JOIN</td>
<td>Indicates to the program the start or end of a sub-tree definition, as appropriate.</td>
</tr>
<tr>
<td>NEXT</td>
<td>Find the next power or ground net to be manually defined if any exists, and set display markers on the appropriate component pins.</td>
</tr>
</tbody>
</table>

Table 8.2 The Light Buttons used in Interactive Route Planning.

One of the necessary facilities is not included in Table 8.2 - the insertion and geometric fixing of dummy pins. Where a dummy pin is required, the user points the light pen at the appropriate inter-slot boundary, as near to the desired y-coordinate as possible. The program then displays a marker on that boundary, the y-coordinate of which is rounded to the same grid as that occupied by the component pins, accompanied by three supplementary light buttons. The
x-coordinate of the dummy pin is that of the slot boundary, and so is defined, while the user can modify and fix the y-coordinate through appropriate use of the supplementary light buttons, as indicated in Table 8.3.

<table>
<thead>
<tr>
<th>LIGHT BUTTON</th>
<th>ACTION INITIATED BY LIGHT PEN HIT ON BUTTON</th>
</tr>
</thead>
<tbody>
<tr>
<td>UP</td>
<td>Move the dummy pin up the slot boundary by the default track centre-to-centre spacing (Via size).</td>
</tr>
<tr>
<td>DOWN</td>
<td>Reverse of UP.</td>
</tr>
<tr>
<td>FIX</td>
<td>Fix the dummy pin in its present position, add it to the list of nodes of the current sub-tree, and remove the supplementary light buttons.</td>
</tr>
</tbody>
</table>

Table 8.3  The Light Buttons used in Fixing Dummy Pin y-coordinates.

8.3.3  The Implementation of Interactive Route Planning.

The data pertaining to the interactively defined route plans is held in three 1-dimensional FORTRAN arrays, PLIST, NLIST, and PAGEA. PLIST and NLIST are equivalenced to parts of PAGEB, neither PAGEA nor PAGEB being required for the geometric structure as pin coordinates are available from the circuit structure.

PLIST holds a list of 3-word entries, one for each node of the sub-tree currently being defined by the user. Each 3-word entry consists of the system name (see 8.1) of a display segment and the display coordinates of the node. For a component pin the display segment contains the appropriate pin marker, the user name of the segment being a pointer to the pin block in the circuit data structure in IARR. For a dummy pin the display segment contains the appropriate
inter-slot boundary, the user name being effectively zero (as no data structure representation of the dummy pin exists at this stage).

NLIST holds the data for all of the sub-trees of the current net which the user has defined completely. The format of the data in NLIST is similar to that of PLIST, with the addition of a 3-word entry at the beginning of the list of blocks of each sub-tree. These additional entries contain a count of the number of nodes in the following sub-tree, a pointer to the net block (in the circuit structure) associated with the sub-tree, and the slot number in which the sub-tree is situated. The format of NLIST is outlined in Fig. 8.2.

PAGEA holds the data for all of the manually planned nets in a format similar to that of NLIST. On completion of the planning of a particular net, the data from NLIST is added to the end of the list in PAGEA after converting the display coordinates to board coordinates and replacing the system names of segments by the corresponding user names.

On exit from the interactive procedures, PAGEA is copied into PAGEB, leaving PAGEA to be used for the geometric structure (see 7.1.1). Prior to the automatic generation of the route plans for the remaining nets, the data saved in PAGEB is extracted, one sub-tree at a time, and fed to the routines which generate a detailed route plan for a given set of geometrically fixed nodes. There is therefore no danger of a conflict arising between the manual and automatic procedures as regards the use of available dummy pin locations.
<table>
<thead>
<tr>
<th>NLIST(1)</th>
<th>SUB-TREE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net Pointer</td>
<td>Head</td>
</tr>
<tr>
<td>Slot No. of Sub-Tree</td>
<td>Block</td>
</tr>
<tr>
<td>Display Segment System Name</td>
<td>1st Node of Sub-Tree</td>
</tr>
<tr>
<td>Display X-Coord. of Node</td>
<td>First Sub-Tree of Current Net</td>
</tr>
<tr>
<td>Display Y-Coord. of Node</td>
<td>Last Node of Sub-Tree</td>
</tr>
<tr>
<td>Segment</td>
<td>Etc. for Other Sub-Trees</td>
</tr>
<tr>
<td>x</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 8.2 Data Storage for Current Net in NLIST.

Fig. 8.3 outlines the implementation of the interactive procedures, omitting the simpler features (such as KILL and DUMP) for clarity, while Fig. 8.4 outlines the subsequent processing of the interactively generated data which yields the detailed route plans.

It should be noted that at the time of interaction the program undertakes no checking — it is left to the user to ensure that all of the pins of a net are connected, that each sub-tree spans only one
INITIALISE DISPLAY FILE, ADD TITLE ETC.
ADD LIGHT BUTTONS, EACH IN UNIQUE SEGMENT.
ADD SEGMENT WITH PINs OF ALL COMPONENTS.

ADD MOVABLE SEGMENT WITH DUMMY PIN MARKER, AND SET OF SEGMENTS FOR SUPPLEMENTARY LIGHT BUTTONS
ADD SET OF SEGMENTS, ONE FOR EACH INTER-SLOT BOUNDARY.

TRANSFER DATA FROM NLIST INTO PAGEA, WITH REQUIRED CONVERSION. REMOVE PIN MARKERS AND NODE-NODE CONNECTIONS FROM DISPLAY FILE.

GET NEXT NET TO BE PLANNED IF ANY EXISTS
ADD SEGMENT WITH MARKER FOR EACH PIN OF NET. USER NAME OF SEGMENT=POINTER TO PIN BLOCK.

OPEN DISPLAY SEGMENT FOR CONNECTIONS
WAIT FOR LIGHT PEN HIT
ADD DATA TO PLIST. DRAW CONNECTION.

TRANSFER DATA FROM PLIST INTO NLIST.
JOIN LIGHT BUTTON ? yes

WAIT FOR LIGHT PEN HIT
HIT ON PIN MARKER ? no
SET DUMMY MARKER ETC. ON BOUNDARY.

COPY PAGEA INTO PAGEB

MOVE DUMMY MARKER DOWN BY VIA SIZE
MOVE DUMMY MARKER UP BY VIA SIZE.
WAIT FOR LIGHT PEN HIT
DOWN LIGHT BUTTON ? no
UP LIGHT BUTTON ? no
FIX LIGHT BUTTON ? yes

Fig. 8.3  State/flow Diagram for Interactive Route Planning.
slot, that dummy pin positions on a given boundary do not conflict, and that no circuits (loops) are created within a given net.

Fig. 8.4 Generating Detailed Route Plans from Interactive Data.
8.4 Interaction with the Program-Generated Routing.

As indicated in 4.6.1, the program-generated routing will not always be complete. It is necessary, therefore, to provide interactive facilities enabling the user to modify, and so complete, the routing. This also allows the user to make any desired improvements to the program-generated routing, and to take account of specific constraints (e.g. that a given region of the board surface should be free of tracks).

There is a problem in the provision of such facilities because of the very large volume of graphical information required to represent the details of all routes on both sides of a board, resulting in low resolution if it is all displayed and, in the case of slow refresh displays, an objectionable flicker rate. The approach adopted regards the inter-slot boundaries as "frozen" thereby reducing the problem size by a factor \( k \), where \( k \) is the number of slots on the board. The price paid for this lies in the restriction of the user's freedom to complete the routing of a board where, for example, one slot cannot be completed due to a high interconnection density in the \( y \)-axis, while an adjacent slot has space available in the \( y \)-axis. The volume of graphical information is further reduced for a given slot with the aid of a simple windowing scheme. The window is of fixed size, being half the height of the slot and as wide as the slot, the user defining the position of the window on the slot (in \( y \) only) interactively, as described in 8.4.2.

Two distinct sets of routines are available. The first, described in 8.4.1, is concerned with graphical output for the whole
board, displaying either or both sides of the board for the conflict-
free or conflicting track segments, as selected interactively. This
facility is provided primarily for the generation of "hard copy"
output, but is of some use in providing an overview of the whole
board prior to interaction with a given slot. The second set of
routines, described in 8.4.2 and 8.4.3, permits the user to
interactively modify the routing in any selected slot, the graphical
output provided being pre-determined, apart from the positioning of
the window on the slot.

8.4.1 Graphical Output for the Whole Board.

The output provided consists of the following:

(i) The pins of all components.
(ii) The routes for the whole board, consisting of the track
segments for either or both sides of the board, conflict
free or conflicting, as selected by the user (see Table
8.4).
(iii) The percentage of track segments which are conflict-free
is displayed as an aid to program evaluation.
(iv) A set of light buttons, enabling the user to control the
program, as listed in Table 8.4.

It may be noted that through-plated holes are omitted from the
above list. This assists in the reduction of the flicker-rate in
the refresh display when displaying a board of moderate size, and
also reduces the time required for plotting layouts, for a very small
loss in information content. Any production version of the program
would obviously require their inclusion.

The various light buttons, and the associated program actions are as listed in Table 8.4.

<table>
<thead>
<tr>
<th>LIGHT BUTTON</th>
<th>ACTION INITIATED BY LIGHT PEN HIT ON BUTTON</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT</td>
<td>Remove program control from the user and exit from the interactive routines.</td>
</tr>
<tr>
<td>DUMP</td>
<td>Save a copy of the current display file on disc for subsequent processing and plotting.</td>
</tr>
<tr>
<td>CONFL</td>
<td>Select CONFLICTING (unfixed) track segments to be displayed on subsequent light pen hits on DRAW.</td>
</tr>
<tr>
<td>ROUTE</td>
<td>Select ROUTED (fixed) track segments to be displayed on subsequent light pen hits on DRAW.</td>
</tr>
<tr>
<td>S/C</td>
<td>Select track segments on SOLDER and COMPONENT sides of board to be displayed (superimposed) on subsequent light pen hits on DRAW.</td>
</tr>
<tr>
<td>COMPT</td>
<td>Select track segments on COMPONENT side of board to be displayed on subsequent light pen hits on DRAW.</td>
</tr>
<tr>
<td>SOLDR</td>
<td>Select track segments on SOLDER side of board to be displayed on subsequent light pen hits on DRAW.</td>
</tr>
<tr>
<td>DRAW</td>
<td>Display view of board with track segments as selected by CONFL, ROUTE, S/C, COMPT and SOLDR.</td>
</tr>
<tr>
<td>MOD</td>
<td>Enter interactive mode permitting modification of the routing. Display set of light buttons enabling the user to initiate interaction with a selected slot, as listed in Table 8.5.</td>
</tr>
</tbody>
</table>

Table 8.4 The Light Buttons used in Controlling Graphical Output.

The implementation of the facilities listed above is depicted in outline in Fig. 8.5. The trivial features are omitted for clarity, and only one of the CONFL, ROUTE, S/C, COMPT and SOLDR light buttons is shown, as the others are similar.
INITIALISE
DISPLAY FILE
ADD TITLE
ETC.

ADD LIGHT BUTTONS, EACH IN A
UNIQUE SEGMENT.

ADD SUBPICTURES TO DISPLAY FILE
SHOWING PINS OF EACH MASTER, IN
EACH REQUIRED ORIENTATION.

SET APPROPRIATE VARIABLE TO
INDICATE TRACK SEGMENTS TO BE
DRAWN AT NEXT CALL TO DRAW.

FOREACH SLOT, I, DO:
GET DATA FOR I'th SLOT INTO
PAGEA. ADD SUBPICTURE CALLS
FOR EACH COMPONENT IN SLOT.

WAIT FOR LIGHT BUTTON HIT

ADD TRACK SEGMENTS FOR SIDE
0/1, FIXED/UNFIXED AS REQUIRED.
KEEP COUNT OF UN-/FIXED SEG.

DISPLAY % OF SEGMENTS
CONFLICT-FREE

REMOVE LIGHT BUTTONS
FROM DISPLAY

INTERACT WITH ROUTING OF SLOTS
AS DESCRIBED IN 8.4.2. ON EXIT
CONTROL RETURNS TO HERE.

EXIT

Fig. 8.5  Graphical Output of Routing for Whole Board.
8.4.2 The Facilities Provided for Routing Interaction.

In order that the routing of a selected slot may be modified in any desired manner, the program must allow the user to move or delete any specified feature block, to insert a block of any type at any point in a sub-tree, to change the type of any block, to switch any track segment from one side of the board to the other, and to change the status of any track segment from unfixed to fixed, or vice-versa.

The graphical output required in implementing these facilities is as follows:

(i) A set of light buttons enabling the user to control the program, as listed in Table 8.5.

(ii) For the current slot, the pins and conflicting track segments for both sides (superimposed) of the board.

(iii) For the current slot, the pins and conflict-free track segments for both sides (superimposed) of the board.

(iv) For the current window on the current slot, the pins, through-holes and conflict-free track segments for the COMPONENT side of the board, drawn to double the scale of (ii) and (iii) above.

(v) As for (iv), but for the SOLDER side of the board.

(vi) The outline of the window, defining (iv) and (v) above, which is of fixed height (⅓ the slot height, and hence the double scale in (iv) and (v) above) and of width equal to that of the slot.

An example of the above graphical output, prior to any interaction, is to be seen in Fig. 4.15.
The various light buttons, and the associated program actions, are listed in Table 8.5.

<table>
<thead>
<tr>
<th>LIGHT BUTTON</th>
<th>ACTION INITIATED BY LIGHT PEN HIT ON BUTTON</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXIT</td>
<td>Returns control to the interactive routines, described in 8.4.1, for graphical output (see Fig. 8.5).</td>
</tr>
<tr>
<td>DUMP</td>
<td>Save a copy of the current display file on disc for subsequent processing and plotting.</td>
</tr>
<tr>
<td>ROUTE</td>
<td>Switch the status of the selected track segment from fixed to unfixed, or vice-versa.</td>
</tr>
<tr>
<td>SWAPS</td>
<td>Switch the selected track segment to the opposite side of the board, adding or deleting through-holes as required.</td>
</tr>
<tr>
<td>INSRT</td>
<td>Add a feature block of the currently selected type to the selected track segment. The tracking cross is set at the centre of the segment and the user drags it to the desired location and loses tracking, fixing the new feature block in that position.</td>
</tr>
<tr>
<td>KILL</td>
<td>Delete the selected feature block of the currently selected type (the latter to avoid ambiguity where, for example, a through-hole and tiering are superimposed).</td>
</tr>
<tr>
<td>MOVE</td>
<td>Move the selected feature block (and any dependent tiering). The tracking cross is set on the selected block and the user drags it to the desired location and loses tracking, fixing the block in that position.</td>
</tr>
<tr>
<td>SLECT</td>
<td>SELECT the &quot;currently selected block type&quot; as for INSRT and KILL above, from a menu of all feature block types.</td>
</tr>
<tr>
<td>NEXT</td>
<td>Select the next sub-tree to be modified by getting a light pen hit on any part of the tree. Markers are superimposed on the feature blocks (except for dependent ties) of the selected tree in the two windowed views of the slot ((iv) and (v) above).</td>
</tr>
<tr>
<td>WINDO</td>
<td>Set the tracking cross on the window outline, permitting the user to position it over the desired part of the slot.</td>
</tr>
</tbody>
</table>
Table 8.5 The Light Buttons used in Routing Interaction.

By way of an example of the use of interaction, Table 8.6 lists the sequence of events in the insertion of a bend into a track segment, as seen from the user's point of view.

<table>
<thead>
<tr>
<th>USER ACTION</th>
<th>OBSERVED PROGRAM ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Hit &quot;NEXT&quot; light button.</td>
<td>None.</td>
</tr>
<tr>
<td>2. Hit any part of sub-tree to be modified.</td>
<td>Markers appear on each of the feature blocks of the sub-tree in the double-scale views.</td>
</tr>
<tr>
<td>3. Hit &quot;SLECT&quot; repeatedly until &quot;SLECT BEND&quot; message appears.</td>
<td>For each hit, steps through sequence SLECT HOLE, BEND, TIE, ETC.</td>
</tr>
<tr>
<td>4. Hit the 2 markers at the ends of the appropriate track segment.</td>
<td>The tracking cross will appear at the mid-point of the selected track segment.</td>
</tr>
<tr>
<td>5. Drag the tracking cross to the desired bend location and lose tracking.</td>
<td>A marker will appear for the bend, rounded onto the appropriate grid.</td>
</tr>
</tbody>
</table>
6. Hit "DRAW" light button.  
(Non-essential action)  
The selected window will be re-drawn, with the bend in the track segment.

Table 8.6  Adding a Bend to a Track Segment.

In the list of user actions in Table 8.6, steps 1 and 2 may be omitted where the appropriate sub-tree is already selected, and step 3 may be omitted where the appropriate block type is already selected.

8.4.3  The Implementation of Routing Interaction.

The implementation of the procedures required to perform a few of the functions listed in Table 8.5 is outlined in Fig. 8.6. Those functions not included are either trivial or similar to (parts of) the ones shown.

It should be noted that in selecting a track segment (as for INSRT, SWAPS, etc.), the user is required to get a light pen hit on the markers of the feature blocks at its two ends. Since a feature block may have no marker, as in the case of a dependent tie, or the markers of two or more blocks may be superimposed, as in the case of an independent tiering, the problem of track segment recognition is handled as follows. The currently selected sub-tree is searched, each of its track segments in turn being examined for geometric coincidence of its end points with the selected markers. The assumption is made that no sub-tree will contain two geometrically superimposed track segments, which is in general valid, though cases could arise where two superimposed segments existed, one on either side of the board.
Fig. 8.6 State/Flow Diagram for Routing Interaction.
As with the routines for placement interaction, no checking is undertaken by the program regarding the validity of user defined modifications. It is left to the user to ensure that his actions do not result in short or open circuits, etc.
Program Performance.

The evaluation of the performance of a program of this nature is difficult as no unique solution exists for any given layout problem. The evaluation will consist of two parts - a quantitative evaluation, being a measure of the success of the program, and a qualitative evaluation, being a measure of the "goodness" of the solutions, regardless of whether they are complete or not.

Before going on to consider evaluation further, it is necessary to define the category of circuit and layout technology used for program evaluation. It should be noted that the results quoted subsequently refer only to the limited category of layout problems outlined in 9.1, and that any extrapolation of these results should be made with caution.

9.1 The Selection of Test Circuits.

The layout program is designed to handle layout problems with the following specific characteristics or constraints:

(i) Circuits consisting of packaged components of similar, but not necessarily equal, size.

(ii) Double sided boards.

(iii) No restriction on the use of through-plated holes connecting tracks on the two board surfaces.

(iv) Tracks on the "solder" side of the board constrained to lie mainly in one axis, and those on the "component" side mainly in the other, the two axes being orthogonal.

It will be noted that (i) above implies a restriction in the
range of circuit types with which the program can deal. With current technology, only digital circuits are likely to fall into this category, though the increasing use of encapsulated hybrid thick- and thin-film linear and/or digital techniques is likely to widen the range included in (i) fairly rapidly. In any case proposed extensions to the program should remove this restriction altogether - see 10.1.3.

As it is desirable to assess the variation in program performance with circuit complexity (no. components, packing density, total no. of connected pins, etc.), 18 circuits of the type indicated above, ranging in size from 6 to 53 packages, are used as test cases. For each of these test circuits a layout produced by an experienced draughtsman is available, this being useful in providing a "standard" with which to compare the program-generated layouts. Table 9.1 lists the various factors related to circuit complexity for each of these test circuits. All of the Tables and Figures in this Chapter appear together at the end of the text for convenience in cross-referencing between tables, etc.

9.2 Placement Evaluation in terms of Estimated Wirelength.

As component placement and conductor routing are inter-dependent, a final evaluation of the placement procedure can only be made in the context of completed layouts (i.e. with placement and routing completed). While writing the placement program, however, it was considered useful to attempt to evaluate the "goodness" of a particular placement in a comparative rather than an absolute sense, thus allowing, for example, the effect of changes in the placement to be evaluated.
The "goodness factor" used for evaluation is simply the estimated total wirelength of all nets, where the wirelength of a net is the sum of the Manhattan distances (6x + 5y) between the connected nodes of that spanning tree of the net described in 4.1.1 (i.e. that tree which joins all of the pins of the net in any given slot in ascending order of y-coordinate, and which joins "nearest pairs" of slots containing parts of the tree by that pin to pin connection for which 6y is minimal). Though not an accurate measure of wirelength, it is likely that the above will be simply (i.e. nearly linearly) related to the actual total wirelength, in view of the algorithms used for conductor routing, assuming that all of the routes can, in fact, be completed.

To allow a comparison of the program- and draughtsman-generated placements, two "variations" of the placement program are used. The first of these generates, for a given circuit, random placements in which the components are constrained to lie on the same grid as for the "normal" placement program. This version of the program makes use of a pseudo-random number generator to select the slot and the position in the slot for each component. Any number of random placements can be generated for a given circuit simply by starting the random number generator at a different point in its sequence for each (though every start point will not necessarily yield a unique placement because of the way in which the random numbers are used to generate the placement, the probability of this, however, being very low). The second variation of the placement program accepts data describing a pre-defined placement along with the circuit input data, consisting of a list of the components together with the slot number and the
position in the slot for each. Again the components are constrained to lie on the same grid as for the "normal" placement program. The "normal" placement program (i.e. that described in Chapters 3 and 6) will be referred to simply as the placement program, while the above two versions will be referred to as the random and pre-defined placement programs, respectively.

The pre-defined placement program enables a comparison of the program- and draughtsman-generated placements to be made, as regards the total estimated wirelength. It should be noted that the procedure used in estimating total wirelength is identical for both cases, resulting in some loss of validity of the comparison due to the fact that the draughtsman would not, in general, have generated a placement designed to use the particular spanning trees on which the wirelength estimate is based, for conductor routing.

The random placement program is used to provide a base for the comparison of the program- and draughtsman-generated placements. This is done by generating a number of random placements for a given test circuit and finding the mean total wirelength for these. The wirelength for the program- or draughtsman-generated placement is then expressed relative to this mean random placement wirelength as a percentage improvement factor:

\[
\text{% IMPROVEMENT (PROGRAM)} = \frac{\text{RANDOM WIRELENGTH} - \text{PROGRAM WIRELENGTH}}{\text{RANDOM WIRELENGTH}} \times 100
\]

and similarly for the % IMPROVEMENT (DRAUGHTSMAN).
Table 9.2 lists the mean total wirelength for 4 random placements, the program- and draughtsman-generated placement wirelengths, and the corresponding improvement factors for 9 of the test circuits (labelled CIRO1, CIRO2, ..., CIRO9).

It will be noted from Table 9.2 that while the program placements are, on average, only marginally better than the draughtsman placements (in terms of the improvement factor), the program provides a more consistent improvement factor, ranging, if the first small circuit is excluded, from 24% to 32%, while that for the draughtsman ranges from 16% to 43%. No correlation is apparent between circuit size and improvement factor, and the large variation in improvement factor for the draughtsman's placements may well be due to the difference between the routing for which the draughtsman's placement was designed and the routing assumed in the estimation of total wirelength.

As the placement program is not designed to minimise wirelength on a global scale, the only minimisation being that in the allocation of clusters to slots and in the placement of components within slots, the above results appear satisfactory in themselves. It is necessary, however, to assess the wireability of the placements produced by the placement procedure before it can be deemed satisfactory.

The speed of operation and storage requirement (for the circuit structure) of the placement procedure are dealt with in the following section.
In terms of quantitative evaluation, the most frequently quoted measure of "success" in the literature describing layout programs is the number of completed connections, where by "connection" is meant the conductor path joining two component pins. The use of this measure is probably due largely to the fact that most of the programs described use maze-running techniques for conductor routing (see 2.3), for which the above measure is both meaningful and easily applied.

Where, however, as in this case, a planned routing exists for each net (or part of a net within a slot), it is more useful to measure success in terms of the specific program objectives - i.e. the degree of completion (conflict-free routing) of the planned routes. The basic element used in computing this success factor is a part (element) of a sub-tree bounded by topologically significant features (i.e. component or dummy pins, through-plated holes, or ties, but not bends), the "success factor" being the number of such sub-tree elements successfully routed expressed as a percentage of the total number of such elements for a given board. This is similar to the more common pin-to-pin measure if dummy pins and Steiner points are included along with component pins.

The above success factor is related to the component placement as well as to the conductor routing algorithms. If, for example, the success factor for a particular circuit is X%, where X<100, and interactive completion of the routing is attempted without any (further) modification of the placement, then two limiting cases are of interest:
(i) where interaction takes the success factor to 100%, and
(ii) where interaction takes the success factor to \( Y\% \), where \( X \times Y < 100 \).

In case (i), the fact that the placement is such as to permit 100% routing with interactive aid indicates that the failure to achieve 100% routing automatically lies with the routing algorithms, and so the failure factor of the routing algorithms is \((100-X)\%\). In case (ii), arguing as above, the failure factor of the routing algorithms is \((Y-X)\%\), while the remaining \((100-Y)\%\) may be attributable to bad placement or to the enforcement of the division into slots in routing interaction (no route being permitted to cross from one slot to another).

Table 9.3 lists, for each of the 18 test circuits, the c.p.u. (processor) time required for component placement, the number of words of storage occupied by the circuit structure, the c.p.u. time for automatic routing and the success factor for automatic routing (given that the power and ground nets are dealt with by the automatic program rather than interactively).

Figs. 9.1 - 9.4 depict the variation of the four factors listed in Table 9.3 with circuit size, where the "number of connected pins" (from Table 9.1) is taken as the measure of circuit size (being that factor giving the most meaningful correlation for all four of these factors). It should be noted that Table 9.1 lists the number of discrete components in each of the test circuits, though it has been assumed hitherto that these are not dealt with. The latter is not, in fact, strictly true, in that the discrete components are dealt
with by the placement program as far as the partitioning stage, thereafter being ignored. It is the relatively large number of discrete components in CIR10, CIR12 and CIR18 that accounts for the anomalously long placement c.p.u. times listed for these circuits in Table 9.3.

The relationship between circuit structure size and circuit size (number of connected pins) is seen clearly from Fig. 9.1 to be linear. The linearity follows from the circuit structure format, as described in 5.3, given that an approximately constant fraction of the total number of pins in any given circuit are actually connected. The slope of the graph follows from the sizes of the various storage structure blocks (see Appendix 3).

From the graph of placement c.p.u. time, t, against circuit size, n (the number of connected pins), drawn on logarithmic axes in Fig. 9.2, it will be noted that the straight line \( t = C n^k \) describes the relationship reasonably within the range of circuit sizes available (given that the three marked points corresponding to CIR10, CIR12, and CIR18 are abnormal as explained above). From Fig. 9.2 the value of \( k \) is found to be approximately 1.4 and that of \( C \) to be 0.017. The large spread in the plotted points is probably due to the variation in the number of iterations of the 2-way partition refinement procedure required to reach pairwise optimality.

From the graph of conductor routing c.p.u. time, t, against circuit size, n (the number of connected pins), drawn on logarithmic axes in Fig. 9.3, it will be noted that the straight line \( t = C n^k \) describes the relationship well, the values of \( k \) and \( C \) being 1.6 and 0.0074 respectively.
The graph of success factor (%) against circuit size, drawn on linear axes in Fig. 9.1, shows a clear downward trend in success factor with increasing circuit size, as is to be expected, this being largely due to increasing congestion in the vertical routes in each slot.

9.4 Interactive Route Planning for the Power and Ground Nets.

Table 9.1 lists the success factor and c.p.u. time for the automatic routing (i.e. with no routing interaction) of a number of the test circuits for the case of interactively planned power and ground nets, and by way of comparison, for the case of program-generated power and ground route plans (as in Table 9.3).

It should be noted from Table 9.1 that while the success factor is, on average only slightly improved by interactive route planning, the greatest improvement is in the larger circuits. This is to be expected in that one of the main objectives of interactive route planning is the avoidance of overcrowding of routes in the y-axis (see 4.4 and 8.3) - a problem that does not affect the smaller circuits. This may be observed more clearly from a comparison of Figs 9.5 and 9.6, showing the automatic routing for CIRO2 for the above two cases. Fig. 9.5, showing the program-generated power and ground route plan case, has a high density of interconnections in the y-axis, with 9.7% still to be routed, while Fig. 9.6, showing the interactively generated power and ground route plan case, has a more uniform distribution of conductors in the x- and y-axes, with only 8.9% still to be routed.
9.5 Overall Evaluation Including Routing Interaction.

As a result of the very long elapse time required to complete the routing of a board of moderate size, only two such attempts have been made.

The results of these two runs are listed in Table 9.5, from which the most obvious factor requiring explanation is the very long c.p.u. and elapse time involved in routing interaction. These times are largely due to the organisation of the interactive facilities which require complete re-generation of the display file after each interactive modification - a process requiring the searching of trees in the geometric data structure, and similar expensive operations. This is a temporary limitation, due in part to the limitations of the graphics system, and in part to poor design of this part of the program. An improvement by a factor of about 4 in both c.p.u. and elapse times is expected from a more efficient scheme. The total c.p.u. and elapse times quoted in Table 9.5 include the times for component placement, assuming the input data to be available in the required format in backing store (i.e. the times do not include data preparation).

The fact that CIR02 only reached 99.5% completion (leaving 3 track segments unrouted) was due to congestion in the y-axis of

1 With only 64K of core in the PDP-10 (see Appendix 4), and a time-sharing monitor occupying about 25K of that 64K, a satisfactory response time for graphical interaction can only be achieved when there are no other jobs in the system besides the 37K layout program, such that the layout program remains permanently in-core.
slot 3. This might have been avoided by the use of Steiner-minimal trees in route planning, as proposed in 11.2.1, or by the use of interactive control of slot width, as proposed in 11.3.1.

Figs. 9.7 - 9.16 show a set of pictures, drawn on the CALCOMP plotter, for CIR16, depicting the various stages of interaction and the final graphical output. The plot showing the completed solder side routing is omitted because of a failure in disc dumping, while the plots for CIR02, which would have been more useful in view of the size of the circuit, are omitted because of system software problems.

Figs. 9.17 - 9.19 show the component placement and the routing for the two sides of CIR02, as laid out by a draughtsman. It should be noted that Figs. 9.17 - 9.19 are "upsides-down" as compared with Fig. 9.6, and that Fig. 9.18, being an etching master, is mirror imaged in x as compared with the CALCOMP plots shown elsewhere.

A comparison of Fig. 9.6, which is only 91.1% routed, and Figs. 9.18 & 9.19, suggests that the program-generated routing is similar to that produced by the draughtsman in terms of the density of routes and their complexity, but that the draughtsman makes more use of those parts of the board not occupied by components than does the program.
<table>
<thead>
<tr>
<th>CIRCUIT NAME</th>
<th>NO. OF PACKAGES IN CIRCUIT</th>
<th>NO. OF DISCRETE COMPONENTS IN CIRCUIT</th>
<th>NO. OF NETS</th>
<th>NO. OF CONNECTED PINS</th>
<th>MEAN DEGREE OF NETS (PINS/NET)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIR01</td>
<td>13</td>
<td>8</td>
<td>66</td>
<td>235</td>
<td>3.6</td>
</tr>
<tr>
<td>CIR02</td>
<td>31</td>
<td>5</td>
<td>118</td>
<td>448</td>
<td>3.8</td>
</tr>
<tr>
<td>CIR03</td>
<td>35</td>
<td>6</td>
<td>105</td>
<td>454</td>
<td>4.3</td>
</tr>
<tr>
<td>CIR04</td>
<td>41</td>
<td>22</td>
<td>171</td>
<td>672</td>
<td>3.9</td>
</tr>
<tr>
<td>CIR05</td>
<td>42</td>
<td>22</td>
<td>191</td>
<td>640</td>
<td>3.4</td>
</tr>
<tr>
<td>CIR06</td>
<td>36</td>
<td>4</td>
<td>139</td>
<td>514</td>
<td>3.7</td>
</tr>
<tr>
<td>CIR07</td>
<td>44</td>
<td>33</td>
<td>209</td>
<td>707</td>
<td>3.4</td>
</tr>
<tr>
<td>CIR08</td>
<td>41</td>
<td>6</td>
<td>148</td>
<td>558</td>
<td>3.8</td>
</tr>
<tr>
<td>CIR09</td>
<td>41</td>
<td>5</td>
<td>148</td>
<td>561</td>
<td>3.8</td>
</tr>
<tr>
<td>CIR10</td>
<td>44</td>
<td>99</td>
<td>189</td>
<td>748</td>
<td>4.0</td>
</tr>
<tr>
<td>CIR11</td>
<td>41</td>
<td>11</td>
<td>145</td>
<td>587</td>
<td>4.1</td>
</tr>
<tr>
<td>CIR12</td>
<td>6</td>
<td>29</td>
<td>35</td>
<td>183</td>
<td>5.2</td>
</tr>
<tr>
<td>CIR13</td>
<td>30</td>
<td>7</td>
<td>105</td>
<td>443</td>
<td>4.2</td>
</tr>
<tr>
<td>CIR15</td>
<td>31</td>
<td>13</td>
<td>97</td>
<td>399</td>
<td>4.1</td>
</tr>
<tr>
<td>CIR16</td>
<td>21</td>
<td>24</td>
<td>67</td>
<td>316</td>
<td>4.7</td>
</tr>
<tr>
<td>CIR17</td>
<td>17</td>
<td>34</td>
<td>68</td>
<td>281</td>
<td>4.1</td>
</tr>
<tr>
<td>CIR18</td>
<td>21</td>
<td>44</td>
<td>93</td>
<td>375</td>
<td>4.0</td>
</tr>
<tr>
<td>CIR20</td>
<td>53</td>
<td>41</td>
<td>180</td>
<td>753</td>
<td>4.2</td>
</tr>
</tbody>
</table>

Table 9.1 Data Relating to the Test Circuits.
<table>
<thead>
<tr>
<th>CIRCUIT NAME</th>
<th>ESTIMATED TOTAL WIRELENGTH</th>
<th>NO. OF PACKAGES IN CIRCUIT</th>
<th>% IMPROVEMENT FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MEAN RANDOM PLACEMENT</td>
<td>DRAUGHTSMAN PLACEMENT</td>
<td>PROGRAM PLACEMENT</td>
</tr>
<tr>
<td>CIR01</td>
<td>502&quot;</td>
<td>318&quot;</td>
<td>267&quot;</td>
</tr>
<tr>
<td>CIR02</td>
<td>852&quot;</td>
<td>577&quot;</td>
<td>577&quot;</td>
</tr>
<tr>
<td>CIR03</td>
<td>822&quot;</td>
<td>465&quot;</td>
<td>556&quot;</td>
</tr>
<tr>
<td>CIR04</td>
<td>1124&quot;</td>
<td>772&quot;</td>
<td>808&quot;</td>
</tr>
<tr>
<td>CIR05</td>
<td>1137&quot;</td>
<td>938&quot;</td>
<td>783&quot;</td>
</tr>
<tr>
<td>CIR06</td>
<td>921&quot;</td>
<td>653&quot;</td>
<td>620&quot;</td>
</tr>
<tr>
<td>CIR07</td>
<td>1267&quot;</td>
<td>829&quot;</td>
<td>885&quot;</td>
</tr>
<tr>
<td>CIR08</td>
<td>1026&quot;</td>
<td>807&quot;</td>
<td>730&quot;</td>
</tr>
<tr>
<td>CIR09</td>
<td>1004&quot;</td>
<td>650&quot;</td>
<td>712&quot;</td>
</tr>
</tbody>
</table>

Table 9.2 Improvement Factor in Total Wirelength
<table>
<thead>
<tr>
<th>CIRCUIT NAME</th>
<th>c.p.u. TIME FOR COMPONENT PLACEMENT</th>
<th>CIRCUIT STRUCTURE SIZE (WORDS)</th>
<th>AUTOMATIC ROUTING WITH POWER/GROUNDHandled BY AUTOMATIC PROCEDURES</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c.p.u. TIME</td>
<td>SUCCESS FACTOR</td>
<td></td>
</tr>
<tr>
<td>CIR01</td>
<td>40s.</td>
<td>60s.</td>
<td>92.2%</td>
</tr>
<tr>
<td>CIR02</td>
<td>91s.</td>
<td>180s.</td>
<td>90.3%</td>
</tr>
<tr>
<td>CIR03</td>
<td>79s.</td>
<td>198s.</td>
<td>89.7%</td>
</tr>
<tr>
<td>CIR04</td>
<td>162s.</td>
<td>284s.</td>
<td>86.6%</td>
</tr>
<tr>
<td>CIR05</td>
<td>135s.</td>
<td>287s.</td>
<td>86.1%</td>
</tr>
<tr>
<td>CIR06</td>
<td>76s.</td>
<td>199s.</td>
<td>90.6%</td>
</tr>
<tr>
<td>CIR07</td>
<td>148s.</td>
<td>313s.</td>
<td>83.7%</td>
</tr>
<tr>
<td>CIR08</td>
<td>96s.</td>
<td>238s.</td>
<td>88.4%</td>
</tr>
<tr>
<td>CIR09</td>
<td>83s.</td>
<td>227s.</td>
<td>86.6%</td>
</tr>
<tr>
<td>CIR10</td>
<td>149s.</td>
<td>267s.</td>
<td>85.9%</td>
</tr>
<tr>
<td>CIR11</td>
<td>133s.</td>
<td>258s.</td>
<td>90.4%</td>
</tr>
<tr>
<td>CIR12</td>
<td>46s.</td>
<td>37s.</td>
<td>90.4%</td>
</tr>
<tr>
<td>CIR13</td>
<td>68s.</td>
<td>148s.</td>
<td>89.6%</td>
</tr>
<tr>
<td>CIR15</td>
<td>68s.</td>
<td>132s.</td>
<td>92.3%</td>
</tr>
<tr>
<td>CIR16</td>
<td>55s.</td>
<td>85s.</td>
<td>95.4%</td>
</tr>
<tr>
<td>CIR17</td>
<td>56s.</td>
<td>69s.</td>
<td>94.1%</td>
</tr>
<tr>
<td>CIR18</td>
<td>115s.</td>
<td>104s.</td>
<td>89.6%</td>
</tr>
<tr>
<td>CIR20</td>
<td>196s.</td>
<td>257s.</td>
<td>84.8%</td>
</tr>
</tbody>
</table>

Table 9.3 Placement and Routing with No Interaction.
<table>
<thead>
<tr>
<th>CIRCUIT</th>
<th>BY PROGRAM</th>
<th>INTERACTIVELY</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ROUTING</td>
<td>SUCCESS</td>
</tr>
<tr>
<td></td>
<td>c.p.u. TIME</td>
<td>FACTOR</td>
</tr>
<tr>
<td>CIR01</td>
<td>60s.</td>
<td>92.2%</td>
</tr>
<tr>
<td>CIR02</td>
<td>180s.</td>
<td>90.3%</td>
</tr>
<tr>
<td>CIR03</td>
<td>198s.</td>
<td>89.7%</td>
</tr>
<tr>
<td>CIR04</td>
<td>284s.</td>
<td>86.6%</td>
</tr>
<tr>
<td>CIR05</td>
<td>287s.</td>
<td>86.1%</td>
</tr>
<tr>
<td>CIR06</td>
<td>199s.</td>
<td>90.6%</td>
</tr>
<tr>
<td>CIR07</td>
<td>313s.</td>
<td>83.7%</td>
</tr>
<tr>
<td>CIR08</td>
<td>238s.</td>
<td>88.4%</td>
</tr>
<tr>
<td>CIR09</td>
<td>227s.</td>
<td>86.6%</td>
</tr>
<tr>
<td>CIR15</td>
<td>132s.</td>
<td>92.3%</td>
</tr>
<tr>
<td>CIR16</td>
<td>85s.</td>
<td>95.4%</td>
</tr>
</tbody>
</table>

Table 9.4 Interactive Power/Ground Route Planning.

<table>
<thead>
<tr>
<th>CIRCUIT</th>
<th>ELAPSE TIME</th>
<th>TOTAL</th>
<th>SUCCESS FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ROUTING INTERACTION</td>
<td>TOTAL c.p.u. TIME</td>
<td>BEFORE INTERACTION</td>
</tr>
<tr>
<td>CIR02</td>
<td>2h. 2h. 12m.</td>
<td>22m. 42s.</td>
<td>91.1%</td>
</tr>
<tr>
<td>CIR16</td>
<td>1h. 1h. 7m.</td>
<td>10m. 16s.</td>
<td>94.4%</td>
</tr>
</tbody>
</table>

Table 9.5 Interactive Completion of Routing.
Fig. 9.1 The Variation of Circuit Structure Size with Circuit Size.
No. of connected pins in circuit

High c.p.u. times due to large no. of discrete components

Fig. 9.2 The Variation of Placement c.p.u. Time with Circuit Size.
Fig. 9.3  The Variation of Routing c.p.u. time with Circuit Size.
Fig. 9.4 The Variation of Success Factor with Circuit Size.
Fig. 9.5  The Automatic Routing of CIR02, with Automatic Power/Ground Insertion, showing both sides of the board superimposed.
Fig. 9.6  The Automatic Routing of CIRO2, with Interactive Power/ Ground Insertion, showing both sides of the board superimposed.
Fig. 9.7  The Automatic Placement of CIR16.
Fig. 9.8  The Interactive Planning of the Ground Net of CIR16.
Fig. 9.9  The Automatic Routing of CIR16, showing both sides superimposed.
Fig. 9.10  The Automatic Routing of CIR16, showing the "solder" side only.
Fig. 9.11. The Automatic Routing of CIR16, showing the "component" side only.
Fig. 9.12 The Automatic Routing of CIR16, showing only the conflicting track segments.
Fig. 9.13  The 4 views of Slot 3 of CIR16 Prior to Routing Interaction.
Fig. 9.14  The 4 views of Slot 3 of CIR16 After Routing Interaction.
Fig. 9.15  The Interactively Completed Routing of CIRl6, showing both sides superimposed.
Fig. 9.16  The Interactively Completed Routing of CIR16, showing the "component" side only.
Fig. 9.17 The Draughtsman's Placement of CIRO2.
Fig. 9.18 The Draughtsman's Routing of CIR02, showing the "Solder" side only.
Fig 9.19  The Draughtsman's Routing of CIR02, showing the
"Component" side only.
The program described in this thesis performs, within the limits of its capabilities, the task of generating a layout for a given circuit, suitably coded in the required input format. It will be noted from 5.2 that the input information is that contained in a circuit, as opposed to a logic, diagram, in that specific pins of each package are assigned to specific nets - this is true of all circuits, though only relevant to digital circuits. The output information is simply that used to drive the graphic display - labelled component outlines for placement, and pin and through-hole positions together with track centre-lines for routing.

While adequate for the purpose of developing and testing a new approach to board layout, the program could be made much more useful by extending it as described in 10.1 and 10.2. The modifications which are required to overcome the known deficiencies of the layout program itself are dealt with separately in Chapter 11.

10.1 The Use of Graph Partitioning in Related Tasks.

The design of the layout for a printed wiring board is, in general, only a small part of a much larger layout problem. Given the circuit design for a large digital system, the solution of the layout problem requires that the circuit first be divided into parts allocated to specific cabinets, and so on through racks, boards and packages down to logic elements (or stopping at components/packages if not a digital system). The overall layout problem is hierarchical, its solution requiring repeated partitioning
and allocation - e.g. partition that part of the system allocated to a specific rack into a number of disjoint parts, one for each board in the rack, and allocate each partition to a specific board location in the rack, both of these being done so as to minimise the total interconnection cost, possibly with overriding restrictions imposed by system modularity or monitoring requirements.

The nature of these problems suggests the use of graph partitioning as a means of breaking the elements at one level in the hierarchy into minimally interconnected groups of elements at the next lower level. The following paragraphs outline two specific proposals for the application of a suitably modified version of the partitioning algorithm used in component placement to these problems, and also to the problem of discrete component placement.

10.1.1 The Allocation of Components to Boards.

Given a circuit, comprising a set of components of known geometry, together with their interconnections, and the requirement that the circuit be laid out on a given number of boards of given size in a given physical arrangement, the first step is that of subdividing the problem into a number of subproblems, one for each board, with defined boundary conditions. This requires the allocation of the components of the circuit to specific boards in such a manner as to minimise the number of interconnections between boards (partitioning) and the assignment of specific nets to specific edge connector pins for each board, such that the backboard/backplane/motherboard wiring problem is made as simple as possible (boundary condition definition).
It will be noted that this problem is exactly analogous to the placement problem for a given board as re-formulated for solution by graph partitioning, as described in Chapter 3. Following this analogy, the steps in the solution of the above problem would be:

(i) Partition the set of components into a number of minimally interconnected subsets, one subset for each board, using a procedure similar to that used in the allocation of the components to slots in component placement, but using component area rather than "length of slot occupied" as the measure of "size" (see 3.2 and 6.1).

(ii) Allocate each subset of the set of components to a specific board (i.e. a board in a specific location). Where the boards are arranged in a rack containing a single row of boards, this procedure is identical to the mapping of clusters into specific slots in component placement (see 3.3 and 6.2).

(iii) Assign specific nets to specific edge connector pins (or equivalent) such as to simplify the board interconnection problem as much as possible. This corresponds closely to the geometric "fixing" of the inter-slot boundaries in conductor-routing by assigning y-coordinates to dummy pins on the slot boundaries (see 4.1 and 7.2).

(iv) For each board, generate the necessary input data for the layout of that board, in the format required by the layout program (see 5.2).
It would appear, from (i) - (iii) above, that the problem of allocating components to boards, and defining the boundary conditions (edge connections) for each board, is amenable to solution by an extension of the techniques developed for the layout of a given board.

10.1.2 The Assignment of Logic Elements to Packages.

Digital circuits consist, for the most part, of basic logic elements such as inverters, AND and OR gates, and flip-flops (bistables). In terms of the packages making up a circuit (e.g. 14-pin dual-in-line package: D.I.P.), these may contain a small number of basic elements (e.g. 4 2-input gates), the number being limited primarily by the number of pins on the package, or they may contain a large number of such elements, interconnected within the package to perform some required operation. Examples of the latter category (medium- and large-scale integration: m.s.i. and l.s.i.) are counters, decoders and memories.

Where a circuit contains a number of logic elements of a given type and packages are available containing \( n \) such elements \((n>1)\), then two assignment problems arise - the assignment of circuit elements to specific packages, and the assignment of these elements to specific elements within a package (i.e. relating each circuit element assigned to a given package to a unique element within that package). The layout program described in this thesis requires that the input data contains these assignments implicitly, in that interconnections are defined as lists of package names and pin numbers. This, however, has two disadvantages, in that the designer must spend time in making the assignments, and the assignments are, of necessity, arbitrary -
particularly the assignment of circuit elements to specific elements within packages, thereby resulting in unnecessarily long and complex routes, and so degrading the overall layout - possibly to the extent of making layout impossible in certain cases.

In order to overcome these deficiencies, it is necessary to incorporate both of these assignment stages at appropriate points in the layout program. The following scheme is suggested as a means of achieving this aim:

(i) Extend the Master Library (see 5.1) to include master definitions for the basic logic elements, such definitions containing no geometric information. In addition, extend the master definitions for specific package types to include information relating the pins of the basic logic elements to the appropriate sets of pins of the packages, where applicable. Also, where 2 or more pins of a logic element are interchangeable - e.g. the 3 inputs of a 3-input gate - that information should be held in the appropriate master definition.

(ii) Extend the scope of the input data for the layout program to include logic elements in the component and interconnection lists. This "mixed-mode" input - circuit and logic diagram - would permit the designer to specify the allocation, if desired.

(iii) Add a stage to the layout program, prior to package placement, to make an initial assignment of logic elements to packages. This would be done for each type of basic logic element in turn. For a given element type, the total number of such
elements in the circuit would be found, and hence the minimum number of the appropriate packages required, each package containing n such elements. The elements would then be partitioned into minimally interconnected sets, each set containing up to n elements. At this stage an arbitrary assignment of elements within packages would be made. It will be noted that the procedure already exists to perform this partitioning, given that the size of a partition is simply the number of elements in that partition (as in the original partitioning scheme due to Kernighan and Lin (27)) – see Appendix 6.

(iv) Modify the component placement refinement stage of the layout program (see 3.5 and 6.4) to include the refinement of the allocation of logic elements within packages. This would be done using the centre of gravity technique currently used in placement refinement, with the objective of minimising overall wirelength.

(v) Add a refinement stage, immediately following component placement refinement, to select the best allocation of interchangeable pins (see (i) above), again based on the centre of gravity method, and with the objective of minimising total wirelength.

It will be noted that the above extensions fit easily into the existing layout scheme, and that the assignment procedures use, for the most part, the techniques already employed in the layout program, resulting, hopefully, in a low implementation cost for such a scheme.
The layout program uses a coarse grid for component placement, each component being allocated to a region of the board equal to the slot width in x and an integral number of 0.550" units in y (see 3.4 and 6.3). In terms of component packing density this is extremely inefficient in dealing with small discrete components. The following extension, based on graph partitioning, is suggested as a means of handling discrete components efficiently (in terms of program efficiency and board utilisation):

(i) Add a stage to the layout program, prior to package placement, to partition the circuit into minimally interconnected sets of components, such that each partition contains either:

(a) a single package,

(b) a single package and a small number of discrete components highly connected to that package (such as the R-C timing elements associated with a mononstable), or

(c) a highly interconnected set of discrete components.

The partitioning procedure would be required to yield partitions of types (a), (b) and (c) above, with the additional constraint that the size of each partition must be similar to that of a package, where by "size" is meant the board area required. Thus each partition might require 1, 2 or 3 units of 0.550" in y. A further constraint on partition size would be an experimentally derived upper
limit on the number of pins per unit board area, as this affects wireability to a considerable extent.

(ii) For partitions of types (b) and (c) above, add a new component block to the components ring (see 5.3) with as many pins as all of the components in the partition put together, and consider such "composite components" as packages for the purposes of package placement.

(iii) After package placement deal with the detailed placement of the discrete components of each composite component within the allotted board area. As these are relatively small problems, a force-field placement technique might prove adequate (see 2.2.1).

It will be noted that the partitioning procedure required is similar to that already used in component placement (see 3.2, 6.1 and Appendix 6), given that the size of an element is a measure of its use of board area, and that a given partition, in addition to having a maximum size limit, may contain only one packaged component. Thus the cost of implementing this scheme would be primarily that of handling the detailed placement of small groups of discrete components ((iii) above). Given that such groups are fairly small, an interactive approach might prove more efficient for placement, particularly with the present trend towards digital systems with few discrete components.

10.2 Post-Processing to Obtain Manufacturing Information.

The program output is currently limited to that used to drive the graphic display. In an industrial environment the following outputs would be required:
(i) Placement drawings showing the board and component outlines, with the components appropriately labelled and marked, where necessary, to indicate orientation (for physically symmetric components which are not electrically symmetric - i.e. the physically interchangeable sets of pins are not electrically interchangeable).

(ii) The output information (probably on paper or magnetic tape) to drive a numerically controlled drilling machine, defining the position and size of all holes on the board (component pins, through-plated via holes, and mechanical mounting holes).

(iii) The output information (probably on magnetic tape) required to produce the photographic masters for etching each side of the board. This would consist of an appropriately formatted description of the tracks and the lands (pads) associated with component pins and through-plated holes. In order to optimise plotter movement (i.e. minimise plotting time), all tracks of a given width in one axis would be ordered to form a partial "raster scan" of the board area (allowing plotting to take place in both directions, and so avoiding the unnecessary "flyback" of a television-type raster scan), and similarly for the other axis and for other track widths. A similar approach would be used to minimise plotting time for all lands of a given diameter.

The generation of the above output information is a straightforward matter, particularly as geometric information is held in fixed
coordinate rings (see 7.1.2), thus facilitating output in the raster-scan ordering required by (iii) above.
Conclusions.

11.1 Component Placement.

The results shown in Chapter 9 indicate that the placements produced by the program are comparable with those produced by a draughtsman, resulting in satisfactory board layouts when routing is completed. The results also show that the placement phase of the program is very fast, and that the rate of growth of placement time with circuit size is low as compared with other automatic placement methods (see (18), (51) and (53)).

The latter is, in part, a direct result of the use of graph partitioning in component placement. The c.p.u. time required for the placement of $n$ elements is generally proportional to at least $n^2$. For example, using an interchange refinement procedure in the placement of $n$ elements, the effect of exchanging each of the $n$ elements, in turn, with the remaining $n-1$ elements is considered, giving an execution time proportional to $n(n-1)$ if the interchange process itself is independent of $n$. Thus for the case of $kn$ components divided into $k$ clusters of $n$ components, the placement time without clustering is proportional to $(kn)^2$, while it is proportional to $k(n)^2$ with clustering - a saving by a factor of $k$, less, of course, the cost of partitioning itself.

11.1.1 Deficiencies in the Placement Procedures.

The following paragraphs list a number of deficiencies existing in the placement procedures described in this thesis. With the exception of the first, as noted below, these are all amenable to
straightforward solution, most of them having been omitted because of their irrelevance to the major objective - i.e. testing the overall approach in a production environment.

The first such deficiency in the placement procedure is due to the use of the "complete graph" on the set of pins of an electrical net in graph partitioning, as described in 3.1. It is possible, particularly where a given net connects more than one pin of a given component (as well as connecting to at least one pin of another component), for this to result in a non-optimal partitioning. The only means of avoiding this problem would be to generate a suitable spanning tree for each net in a given placement, and hence define the cluster inter-connections, re-generating the spanning trees after each step in the partitioning process. In view of the satisfactory nature of the placements produced by the existing algorithms (with the limitations noted below) it is doubtful if the high cost of such an extension, in terms of implementation cost and reduced speed of operation, would be justified.

The second deficiency relates to the mapping of clusters of components into slots on the board, as described in 3.3 and 6.2 This mapping is made without regard to the edge connections, the program assuming the task of making the best edge connector pin assignment on completion of the component placement phase. It is, however, rarely the case that such freedom exists, and it is therefore necessary to take account of the edge connections in the process of allocating clusters to slots. It might similarly be worth considering the application of a 2-way interchange refinement stage permitting the swapping of components between adjacent slots on completion of the
present placement procedure.

The third deficiency is the omission of the ability to handle certain of the input constraints listed in Table 5.3, such as the fixing of a given component in a predefined location. The extensions required to handle such constraints are of a fairly straightforward nature.

11.2 Conductor Routing by Program.

The results shown in Chapter 9 for the automatic routing procedures (sub-tree planning and stepping aperture conflict minimisation) indicate that the procedures are fast and that the rate of growth of routing time with circuit size is reasonable. The author is unaware of any published results relating to other layout programs handling similar problems which would enable a useful comparison to be made.

In terms of the success factor (degree of completion) of the routing algorithms, it should be noted that the set of heuristic conflict minimisation routines used in aperture routing (see 7.4) is incomplete. It is expected that the success factor achieved with automatic routing will be increased by about 5% for circuits with about 500 connected pins (see Fig. 9.4) when the conflict minimisation procedures are completed.

11.2.1 Deficiencies in the Automatic Routing Procedures.

The most notable deficiency in the automatic routing procedures relates to the planning of routes (sub-tree planning). It is
currently assumed that for any given net only one connection may
exist between any given pair of adjacent slots (except in the case
of interactively planned power and ground nets). This may result
not only in excessive total wirelength, but in congestion of routes
parallel to the y-axis. The latter is apparent, for example, in
Fig. 9.6. One solution to this problem, already noted in 4.1.1,
would be to generate a Steiner-minimal tree for each net, the
intersections of the inter-slot boundaries and the branches of the
Steiner tree defining the position and number of dummy pins required.
This solution would also obviate the need for the interactive planning
of the power and ground nets. The generation of Steiner-minimal trees
on more than 5 nodes is non-trivial, both in terms of coding effort
and execution time (see (9), (21) and (58)), but a heuristic scheme
for producing near-minimal trees might be expected to require only
a few man-weeks of programming effort and to be fast in operation.

Another deficiency in the routing procedures as implemented is
the omission of dimensional checking. This means that each item -
track or through-hole - occupies 1 grid unit (the default track-to-
track spacing), thereby allowing through-plated holes to be placed
with inadequate separation from adjacent holes, etc. The net effect
of this is small, and the incorporation of the extra code required
to perform the necessary checking during conflict minimisation
(aperture routing) is a straightforward matter.

11.3 Interactive Completion of Conductor Routing.

The results shown in Chapter 9 for the interactive completion
of conductor routing indicate that, while capable of permitting the
completion of the routing, the procedures are very slow.
As was indicated in 9.5, a 4-fold increase in speed is anticipated from a suitable re-organisation of the display file segmentation, largely due to the elimination of the need to re-generate the display file after each interactive modification. Given such an increase in speed, the layout scheme described should provide a cost-effective solution to the double sided board layout problem.

11.3.1 Deficiencies in the Routing Interaction Scheme.

Apart from the low speed of interaction mentioned above, there is one other deficiency, already noted in 8.4, in that the inter-slot boundaries are considered to be "frozen". This prevents the user from moving tracks from a slot with a high density of connections into a nearby slot with a lower density of connections. While the use of Steiner-minimal trees in route planning proposed in 11.2.1 above might obviate the need for this in most cases, it would nonetheless appear desirable to permit interactive insertion, moving and deletion of dummy pins, thus giving the user complete freedom in interaction. An alternative, or supplementary, approach would be the inclusion of interactive control of slot width, permitting the user to expand a slot with a high internal connection density and vice-versa.

11.4 Originality.

The most significant parts of the layout scheme described in this thesis which the author believes to be original contributions are the application of graph partitioning in component placement as
a means of achieving global control of placement, and the extensions made to the partitioning algorithm used (see Kernighan and Lin (27)) regarding the control of cluster size, as described in A6.2. In addition, though the stepping aperture approach used for conflict minimisation in conductor routing was propounded by Lass (32), the idea has been extended substantially, resulting in a method which is similar only in concept to that of Lass.
Acknowledgements.

The research work for this thesis was carried out in the Computer Aided Design Project of the Department of Computer Science of the University of Edinburgh. My thanks are due, in particular, to Dr. J.V. Oldfield who supervised the work and provided constant encouragement and much by way of useful advice and criticism. My thanks are also due to various other members of the Project for the provision of the graphics system software and other facilities.

In addition my thanks are due to Ferranti Ltd. (Edinburgh) which seconded me to the University for the duration of the project. In particular my thanks go to Mr. A.C. Wesley, Mr. G.A. Shippey and Mr. J.R. McIntosh of Ferranti Ltd., for many useful discussions regarding the project objectives, and for the provision of the test circuit data.

The computing facilities were provided by the UK Science Research Council, initially under grant Ref. B/SR/1718, and more recently under grant Ref. B/SR/8874.
APPENDIX 1 Data Structure Free-Storage Management

The layout programs, written in FORTRAN, use a special-purpose, ring-based data structure, built within a one-dimensional FORTRAN array. In order that the program can take blocks of storage (a block being a number of contiguous words) of variable length from this array, and return them to it, a free-storage management scheme is required. The scheme to be described is simple, reasonably fast, and satisfies the limited requirements of this program adequately.

Given an N-word array, "IARR", in which the free-storage scheme must operate, the first four words of the array, IARR(1)...IARR(4), are reserved, their functions being as indicated in Fig. Al.1. At any given time, all of the currently free blocks in IARR are threaded on a ring, the pointer to the "next" free block being the absolute value of the first word of the "current" free block. Where the "current" block is only one word long, this pointer is negative, otherwise being positive, while for a block of two or more words, the number of words in the block is held in the second word of the block, as shown in Fig. Al.1.

On requesting the allocation of a block of length L (i.e. of L words), the ring of free blocks is searched until a free block of length L is found, or the entire ring has been searched. In the former case the block is "untied" from the ring of free blocks and allocated, while in the latter, the last L words are taken from the "best" block encountered in the search, and allocated, where the "best" block is that one whose length is greater than L by the least amount. Allocating from the end of a block in this manner has the advantage of requiring fewer expensive pointer operations than the more
common method in which the head of the free block would be allocated. The allocated block is cleared (set to zero) before being passed to the program. Fig. A1.2 outlines the storage allocation scheme.

On returning a block to free store, each block in the ring of free blocks is checked for adjacency with the block being returned, and if any free block is found to be adjacent with the returned block, the free block is untied from the free ring and the two blocks are merged to form a composite "returned" block. As there are two possible adjacencies, the checking must be carried on until either two adjacencies are found, or all free blocks have been checked. The returned block is then added to the beginning of the free ring. Though more expensive in c.p.u. time than other schemes which only "garbage-collect" on running out of free store, this scheme avoids potential free store fragmentation, in which sufficient free store may be available to fulfill a given allocation request, but is unusable because it is not in a contiguous block. Fig. A1.3 outlines the scheme for returning blocks to free store.

A further, machine dependent, allocation routine has also been implemented, which allocates a block of given length at a given index in the free-storage array, if available. This is used by the "paging" scheme in conjunction with FORTRAN EQUIVALENCE statements to circumvent certain deficiencies in the random access file scheme available (see 7.1), although it is inconsistent, in concept, with a "free-storage" scheme.
<table>
<thead>
<tr>
<th>User Array Identifier</th>
<th>Total Array Length</th>
<th>Pointer to 1st Free Block</th>
<th>Current No. of Free Words</th>
<th>Pointer to Next Free Block</th>
<th>Length of This Free Block</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Reserved by free-storage scheme.
- Allocated parts of IARR are shown hatched.
- Pointers on "Free Ring".

Fig. A1.1 A Simple "Free Ring" in an N-word Array.
Fig. A1.2 Flowchart: Allocation of L-word Block from Free Store.

Fig. A1.3 Flowchart: Return of L-word Block to Free Store.
APPENDIX 2  The Use of Macro Processing Techniques.

A2.1  The STAGE2 Macroprocessor.

A macro processor may be considered as a character string manipulator which analyses an input string according to a set of rules, producing another string as output. The "rules" used in converting between input and output correspond to the macro definitions.

The general form of a macro definition may be considered as:

\[
\text{NAME}(P1,P2,P3,\ldots,PN) \\
\text{CODE BODY} \\
\text{END}
\]

where \(\text{NAME}(\ ,\ ,\ ,\ ,\ ,\ )\) is either the macro name \(\text{NAME}\) together with a delimiter structure \((\ ,\ ,\ ,\ ,\ ,\ ))\) defining the parameter positions (as in ML/I (5), (6)) or is the distributed macro name (as in STAGE2 (16)), \(P1,P2,\ldots,PN\) are the formal parameters of the definition, \(\text{CODE BODY}\) is the replacement code which may include functions of any or all of the formal parameters, and \(\text{END}\) marks the end of the definition. A call on such a macro would be of the form:

\[
\text{NAME}(A1,A2,A3,\ldots,AN)
\]

where \(A1,A2,\ldots,AN\) are the actual parameter strings which are substituted for the formal parameters in the code body of the definition in the process of generating the replacement text (output string) for the call.

STAGE2 is the macroprocessor used. A brief description follows of those features of STAGE2 used in connection with the layout program, but for a detailed description of STAGE2 the reader is referred to
"The STAGE2 Macroprocessor user reference manual" (46).

STAGE2 deals with input text on a line-by-line basis, a special character (end-of-line flag) being used to mark the end of each input line. Similarly, a special character (parameter flag) is used to mark the formal parameter positions in a macro definition. Thus, using the characters ";" and "'" (semicolon and prime) for the end-of-line and parameter flags respectively, the first line of any macro definition, specifying the distributed name and parameter positions, might have the form:

\[ \text{'NAME(', , , , , , );} \]

where the parameters are numbered 1, 2, ..., N from left to right. The n'th parameter (from the left) is referred to in the code body of the macro definition as "'nk" where k is an integer (conversion code) defining an operation on the actual parameter string, such as direct copying or evaluation as an arithmetic expression. For copying, the most commonly used operation, k=0, and so the macro definition:

\[ \text{'EXAMPLE(' , , , , , ) = ';} \]

\[ \text{10EXAMPLE('20('30 + 2)) = '40;} \]

would convert the macro call:

\[ \text{EXAMPLE(ARRAY,POINT) = VALUE;} \]

into the output line:

\[ \text{EXAMPLE(ARRAY(POINT + 2)) = VALUE.} \]

1 Separate characters are, in fact, used for the end-of-line flag and for the parameter flag, depending on whether they belong to either the code body of a macro definition (target flags) or to the input string or distributed name of a macro definition (source flags). For simplicity the source and target end-of-line flags are assumed to be identical, and likewise for the source and target parameter flags.
One special parameter conversion is used, of the form "0j". It is not a genuine parameter conversion as parameter 0 does not exist (only 1,2,...,N exist), but causes a symbol generator to output a symbol with subscript "j". Thus 6'00,6'01 and 6'02 would, within a given macro call, yield the symbols 60,61 and 62. A subsequent call of the same macro would yield the symbols 63,64 and 65 respectively, with an obvious use in creating unique labels local to a given macro call.

Processor functions, of the form "Fk", are available to handle such functions as termination of processing, where k is an integer (function code) defining the processor function. The one most used is that which causes a given line to be output without (further) evaluation, for which k = 1. This is useful in causing the processor to skip FORTRAN format and comment lines, thus saving processing time and avoiding the expansion of any macro appearing (accidentally) in such lines. Thus the macro definition:

'FORMAT(');
'10FORMAT('20)F1;

would cause FORTRAN format statements appearing in the input string to be copied to the output string without evaluation.

A2.2 Macro Processing as a General Programming Aid.

Where a particular text string (which may occupy several lines) is duplicated at several points in a program it is convenient to define a macro with a relatively simple name with the appropriate string as its replacement text. The COMMON statements in the layout program fall into this category, a copy appearing in each subroutine
of the program. By inserting an appropriate macro call in the source code instead of a list of COMMON statements, the probability of errors occurring is virtually removed, and the time taken in writing the abbreviated source code is also reduced. In addition, any major change in the organisation of the common blocks requires only the re-definition of the appropriate macro(s) and the re-processing of the source code for each subroutine affected, rather than the editing of individual subroutines otherwise required.

A similar, but simpler, use of macros is that in which macro definitions relate meaningful macro names with constant values. This is used in the layout program as a means of allowing easy recognition of the various data structure block type identifiers (see Appendix 3). Thus, for example, the macro definition:

'TYPETIE';
'106'20;

allows the use of "TYPETIE" instead of the virtually meaningless constant, "6", being the TYPE (identifier) of a geometric TIE block (see Table A3.9). The advantage of this trivial substitution lies in the improved readability of the resulting source code.

A2.3 Macro Processing as a Data Structuring Aid.

The layout program uses a ring-based data structure, as described in Chapter 5 (the circuit structure) and in 7.1 (the geometric structure). Associations between elements (blocks) are expressed by pointers which link associated blocks into rings. A detailed description of the various block types is to be found in Appendix 3. It will be noted from Appendix 3 that the various field of each block type are packed
together so that the blocks are of minimal length (number of words), subject to the proviso that the fields holding the pointers of a given ring must occupy the same position relative to the head of each type of block on that ring. The "head" of a block is that word of the block used to address it - any pointer to a block points to the head of the block. Thus, for example, the ring linking a net with its associated pins uses similar pointer positions in both the pin and the net blocks (see Tables A3.3 and A3.4).

Data is inserted into a given field of a block using the FORTRAN subroutine IPAC, and extracted from it using the FORTRAN function IUNPAC, called as follows:

\[ \text{CALL IPAC(NAME(HEAD+OFFSET),LBIT,RBIT,IVALUE)}, \text{ and} \]

\[ \text{IVALUE = IUNPAC(NAME(HEAD+OFFSET),LBIT,RBIT)}, \]

where \( \text{NAME} \) is the data structure array, \( \text{HEAD} \) is the index of the head of the block in array \( \text{NAME} \), \( \text{OFFSET} \) is the offset of the word holding the field relative to the head of the block, \( \text{LBIT} \) is the position of the left-most bit of the field in that word (the most significant bit), \( \text{RBIT} \) is the position of the right-most bit of the field, giving a field width \( N \), where \( N=\text{RBIT}-\text{LBIT}+1 \), and \( \text{IVALUE} \) is a variable containing in its least significant \( N \) bits the value being written into the field by \text{IPAC} or extracted from it by \text{IUNPAC} (the remaining bits, if any, of \( \text{IVALUE} \) being zero).

The use of expressions of the above form for each data structure operation (reading from or writing to the structure) in the layout program is undesirable in that the substitution of the appropriate constants for the three parameters \( \text{OFFSET}, \text{LBIT} \) and \( \text{RBIT} \) is an error-
prone process, and the resulting code is virtually unreadable (in terms of conveying a meaningful description of the operation, that is). Further, the process of altering the program code to incorporate any amendments made to the format (field arrangement) of any of the block types, as, for example, in moving the program to a machine of different word length, requires the modification of many, if not all, of these parameters throughout the program - a tedious, time-consuming, and error-prone process.

The above three deficiencies are overcome with the aid of macro processing. For each field with a given function, a macro is defined which has a meaningful name in relation to that function and which includes, in its replacement text, the three parameters defining the field position relative to the head of the block containing it, namely OFFSET, LBIT and RBIT. For example, consider the ring linking a fixed-X ringhead with each geometric feature block at the x-coordinate of that ringhead (see Fig. 7.2). The pointer from a block ICURR to the next in the fixed-X ring, INEXT, would be set by the following macro call:

```fortran
NEXXRING(PAGEA,ICURR)=INEXT;
```

which, on processing by the macro definition:

```fortran
'NEXXRING' = \'
10CALL IPAC(20('30+1),12,23,'40);
```

would yield the FORTRAN expression:

```fortran
CALL IPAC(PAGEA(ICURR+1),12,23,INEXT).
```
Similarly to retrieve that pointer, the macro call:

\[
\text{INEXT} = \text{NEXXRING}(\text{PAGEA}, \text{ICURR});
\]

after processing by the macro definition:

\[
'\text{INEXXRING('',')}';
'10=\text{IUNPAC('20('30+1),12,23)}'40;
;
\]

would yield the FORTRAN expression:

\[
\text{INEXT} = \text{IUNPAC}(\text{PAGEA}(\text{ICURR}+1),12,23).
\]

Thus the layout program source code consists of FORTRAN statements intermixed with macro calls, many of which are of the above form. On macro processing this source code the macro calls are expanded in accordance with the appropriate macro definitions, yielding pure FORTRAN code which may then be compiled and loaded in the normal manner.

An operation which is required at many points in the layout program is the searching of a given ring for a block satisfying a particular requirement - for example, searching for the fixed-X ringhead on the fixed-X ring to which a given geometric feature block is tied. As the operation required in moving from one element of a ring to the next varies according to the type of ring, a FORTRAN subroutine cannot easily be used to handle all operations of this class. Consider, however, the following macro definition:

\[
'\text{FIND('',',',',')}';
'10\text{ISAVE}'=20;
60'00 \text{ '20}' =30;
\text{ITEMP}' =40;
\text{IF('20.EQ.ISAVE)GO TO 60'01;}
\text{IF(ITEMP.NE.'50)GO TO 60'00;}
60'01 \text{ IF(ITEMP.NE.'50)GO TO '60'70;}
;
In a call of this macro, some of the actual parameters are themselves macro calls, as for example in the call:

\[ \text{FIND}(\text{NEXT, NEXXRING(PAGEA, NEXT), TYPE(PAGEA, NEXT), TYPEXHD, 50}); \]

which would be expanded initially (ignoring the further expansion of the macros NEXXRING, TYPE and TYPEXHD) as:

\[
\begin{align*}
\text{ISAVE} &= \text{NEXT} \\
600 \quad \text{NEXT} &= \text{NEXXRING(PAGEA, NEXT)} \\
\text{ITEMP} &= \text{TYPE(PAGEA, NEXT)} \\
\text{IF}(\text{NEXT}, \text{EQ}, \text{ISAVE}) &\text{GO TO 601} \\
\text{IF}(\text{ITEMP}, \text{NE}, \text{TYPEXHD}) &\text{GO TO 600} \\
601 \quad \text{IF}(\text{ITEMP}, \text{NE}, \text{TYPEXHD}) &\text{GO TO 50}
\end{align*}
\]

It will be noted that the resulting code performs the function mentioned above, namely, the searching of the fixed-X ring, to which the block "NEXT" is tied, for the fixed-X ringhead, which has a block identifier of value "TYPEXHD". If the search fails, control is passed to label "50" of the program. In more general form, a call of this macro would be:

\[ \text{FIND}(\text{START, GET NEXT, EVALUATE NEXT, TARGET, FAIL EXIT}); \]

which, on evaluation, yields code which searches a ring, starting on the block with pointer in "START", and moving from one block to the next with the "GET NEXT" mechanism, for a block for which the "EVALUATE NEXT" function is equal to the value of "TARGET". If successful, control is passed to the next statement in sequence, the variable "START" holding a pointer to the required block, and if unsuccessful, control is passed to the label "FAIL EXIT".

It will be seen from the above description that the use of macro processing permits the use of meaningful and sometimes abbreviated source code, and, as a result of the once-only definition of the
parameters relating field positions to the appropriate block heads, assists in achieving program transportability and ease of storage structure modification.

The "transportability" has, in fact, been tested for the placement program (about one-third of the total code). The placement program was developed on an ICL 4130 (24 bit words) and transferred to a DEC PDP-10 (36 bit words). The time taken for the transfer, including the re-design of the storage structure and the corresponding re-definition of the macros, was just over one man-week.
APPENDIX 3  Detailed Description of Storage Structure Blocks.

The following description of each of the block types used in the layout program data structure corresponds to the program implementation on a PDP-10, a machine with 36-bit words. All of the "fields" - individually addressable sets of adjacent bits - of the blocks are defined relative to the block address (not necessarily the first word of a block) by means of macro definitions (see Appendix 2). Thus the re-definition of the storage structure for a machine with, for example, a different word length, requires only the re-design of the blocks and the re-definition of the appropriate macros. The term "storage structure" is used to denote a particular implementation of the more general "data structure", the data structure being that on which the program operates, by way of macro calls.

A field of any block is defined by three parameters, being the offset of the word containing the field relative to the block address, the block address being considered as word zero, and the left-most and right-most bit positions of the field. Thus a field $F(1,21,35)$ defines a 15-bit wide field in word 1 of a block, occupying the least significant 15 bit positions, while $F(2,20,20)$ defines a 1-bit field in bit position 20 of word 2 of a block. Fig. A3.1 illustrates these two fields diagrammatically, labelling them A and B respectively.
The following tables describe the fields, and their functions, for each of the block types used. The macro name (or at least the most recognisable part of the distributed name - see Appendix 2) is listed for each field, but it should be noted that some discrepancies exist between the macro names, which are intended to be meaningful, and the program description. This arises from the fact that, historically, a planar-graph approach to the problem was considered, and so a different circuit graph was envisaged. Hence nets are referred to as nodes, and component pins as links (link branches), and so on.
### Table A3.1 The Fields of a MASTER Block

<table>
<thead>
<tr>
<th>FIELD</th>
<th>MACRO NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-4,0,35)</td>
<td>MASPIC</td>
<td>Contain pointer(s) to display file sub-picture definitions of master, at orientations 3,2,1 and 0, respectively.</td>
</tr>
<tr>
<td>(-3,0,35)</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>(-2,0,35)</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>(-1,0,35)</td>
<td>&quot;</td>
<td></td>
</tr>
<tr>
<td>(-1,23,23)</td>
<td>MASTOR</td>
<td>Preferred orientation of master.</td>
</tr>
<tr>
<td>(-1,24,29)</td>
<td>MASTRANX</td>
<td>X-transparency of master.</td>
</tr>
<tr>
<td>(-1,30,35)</td>
<td>MASTRANY</td>
<td>Y-transparency of master.</td>
</tr>
<tr>
<td>(0,0,5)</td>
<td>TYPE</td>
<td>Master block identifier. Value = 1.</td>
</tr>
<tr>
<td>(0,6,13)</td>
<td>PINSIZE</td>
<td>Diameter of pins of master.</td>
</tr>
<tr>
<td>(0,14,20)</td>
<td>PINMAST</td>
<td>Number of pins of master.</td>
</tr>
<tr>
<td>(0,21,35)</td>
<td>NEXT</td>
<td>Pointer to next master block in ring.</td>
</tr>
<tr>
<td>(1,0,35)</td>
<td>NAMMAST</td>
<td>User name of master, 5 ASCII characters.</td>
</tr>
<tr>
<td>(2,0,35)</td>
<td>SGNMAST</td>
<td>Normally 0, but -1 for board master.</td>
</tr>
<tr>
<td>(3,0,5)</td>
<td>XPITMAS</td>
<td>X-pitch of master.</td>
</tr>
<tr>
<td>(3,6,11)</td>
<td>YPITMAS</td>
<td>Y-pitch of master.</td>
</tr>
<tr>
<td>(3,12,23)</td>
<td>XOUTMAS</td>
<td>X-outline of master.</td>
</tr>
<tr>
<td>(3,24,35)</td>
<td>YOUTMAS</td>
<td>Y-outline of master.</td>
</tr>
<tr>
<td>(4,18,26)</td>
<td>XPINMAS</td>
<td>X-coordinate of pin 1 of master.</td>
</tr>
<tr>
<td>(4,27,35)</td>
<td>YPINMAS</td>
<td>Y-coordinate of pin 1 of master.</td>
</tr>
</tbody>
</table>

Subsequent pin coordinates are stored in successive \(\frac{1}{4}\) words of the master block, two pairs per word.
### The Fields of a COMPONENT Block

<table>
<thead>
<tr>
<th>FIELD</th>
<th>MACRO NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-1,0,35)</td>
<td>COMSUB</td>
<td>Component Sub-block pointer (in partitioning).</td>
</tr>
<tr>
<td>(-1,0,35)</td>
<td>COMPIC</td>
<td>Display file sub-picture pointer (in interaction).</td>
</tr>
<tr>
<td>(0,0,5)</td>
<td>TYPE</td>
<td>Component block identifier. Value = 4.</td>
</tr>
<tr>
<td>(0,6,20)</td>
<td>MASTPTR</td>
<td>Pointer to master block of component.</td>
</tr>
<tr>
<td>(0,21,35)</td>
<td>NEXT</td>
<td>Pointer to next component block in ring.</td>
</tr>
<tr>
<td>(1,0,35)</td>
<td>COMPNAM</td>
<td>User name of component, 5 ASCII characters.</td>
</tr>
<tr>
<td>(2,9,10)</td>
<td>ORIENT</td>
<td>Orientation of component.</td>
</tr>
<tr>
<td>(2,11,11)</td>
<td>ORIENTF</td>
<td>Bit = 1 iff orientation fixed by constraint.</td>
</tr>
<tr>
<td>(2,12,23)</td>
<td>XDATUM</td>
<td>X-coordinate of component datum.</td>
</tr>
<tr>
<td>(2,24,35)</td>
<td>YDATUM</td>
<td>Y-coordinate of component datum.</td>
</tr>
<tr>
<td>(3,0,3)</td>
<td>COMPNO</td>
<td>Sequence number of component in slot.</td>
</tr>
<tr>
<td>(3,6,20)</td>
<td>COMSEG</td>
<td>Pointer on ring of components of cluster.</td>
</tr>
<tr>
<td>(4,-,-)</td>
<td>-</td>
<td>Subsequent words contain the pin blocks of the component, in order.</td>
</tr>
</tbody>
</table>

Table A3.2  The Fields of a COMPONENT Block

### The Fields of a PIN Block

<table>
<thead>
<tr>
<th>FIELD</th>
<th>MACRO NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0,5)</td>
<td>TYPE</td>
<td>Pin Block identifier. Value = 10.</td>
</tr>
<tr>
<td>(0,6,6)</td>
<td>TAGLINK</td>
<td>Used as marker in partitioning, etc.</td>
</tr>
<tr>
<td>(0,7,10)</td>
<td>SEGLINK</td>
<td>No. of slot in which pin is placed.</td>
</tr>
<tr>
<td>(0,11,12)</td>
<td>LINKSID</td>
<td>Board side(s) on which pin exists.</td>
</tr>
<tr>
<td>(0,13,13)</td>
<td>LINKSIDF</td>
<td>Bit = 1 iff pin on fixed side of board.</td>
</tr>
<tr>
<td>(0,14,20)</td>
<td>TRAKWID</td>
<td>Width of track associated with pin.</td>
</tr>
<tr>
<td>(0,21,35)</td>
<td>NEXLINK</td>
<td>Pointer to component head block.</td>
</tr>
<tr>
<td>(1,9,20)</td>
<td>LINKOPY</td>
<td>Pointer to &quot;copy&quot; of pin block in segment &quot;SEGLINK&quot; of geometric structure.</td>
</tr>
<tr>
<td>(1,21,35)</td>
<td>NODBPTR</td>
<td>Pointer on ring of pins of net.</td>
</tr>
</tbody>
</table>

Table A3.3  The Fields of a PIN Block.
### Table A3.4 The Fields of a NET Block.

<table>
<thead>
<tr>
<th>FIELD</th>
<th>MACRO NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0,5)</td>
<td>TYPE</td>
<td>Net Block identifier. Value = 2.</td>
</tr>
<tr>
<td>(0,6,13)</td>
<td>TRAKSEP</td>
<td>Track separation for net.</td>
</tr>
<tr>
<td>(0,14,20)</td>
<td>TRAKWID</td>
<td>Track width for net.</td>
</tr>
<tr>
<td>(0,21,35)</td>
<td>NEXT</td>
<td>Pointer to next net block in ring.</td>
</tr>
<tr>
<td>(1,0,20)</td>
<td>XYNODE</td>
<td>Y-coordinate of centre of gravity of net.</td>
</tr>
<tr>
<td>(1,21,35)</td>
<td>NODBPTR</td>
<td>Pointer to ring of pins of net.</td>
</tr>
<tr>
<td>(2,0,9)</td>
<td>NODEG</td>
<td>Degree (no. of pins) of net.</td>
</tr>
<tr>
<td>(2,10,13)</td>
<td>CRITWT</td>
<td>Weight associated with CRITICAL length net.</td>
</tr>
<tr>
<td>(2,14,14)</td>
<td>CRITNOD</td>
<td>Bit = 1 iff net of CRITICAL length.</td>
</tr>
<tr>
<td>(2,15,15)</td>
<td>TAGBIT</td>
<td>Used as marker in partitioning, etc.</td>
</tr>
<tr>
<td>(2,17,17)</td>
<td>TRAKSID</td>
<td>Board side on which net is routed.</td>
</tr>
<tr>
<td>(2,18,18)</td>
<td>TRAKSIDF</td>
<td>Bit = 1 iff net on fixed side of board.</td>
</tr>
<tr>
<td>(2,19,19)</td>
<td>POWRBIT</td>
<td>Bit = 1 iff net is power net.</td>
</tr>
<tr>
<td>(2,20,20)</td>
<td>GRNDBIT</td>
<td>Bit = 1 iff net is ground net.</td>
</tr>
<tr>
<td>(2,21,35)</td>
<td>NAMNODE</td>
<td>User name of net, integer value.</td>
</tr>
</tbody>
</table>

### Table A3.5 The Fields of a Cluster Block.

<table>
<thead>
<tr>
<th>FIELD</th>
<th>MACRO NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0,5)</td>
<td>TYPE</td>
<td>Cluster block identifier. Value = 31.</td>
</tr>
<tr>
<td>(0,21,35)</td>
<td>NEXT</td>
<td>Pointer to next cluster block in ring.</td>
</tr>
<tr>
<td>(1,0,35)</td>
<td>SUBNEX</td>
<td>Pointer to ring of sub-blocks of cluster.</td>
</tr>
<tr>
<td>(2,0,35)</td>
<td>NAMSEG</td>
<td>No. of slot in which cluster is placed.</td>
</tr>
<tr>
<td>(3,0,5)</td>
<td>ICSEG</td>
<td>No. of components in cluster.</td>
</tr>
<tr>
<td>(3,6,20)</td>
<td>COMSEG</td>
<td>Pointer to ring of components of cluster.</td>
</tr>
<tr>
<td>(3,21,35)</td>
<td>LSTRIP</td>
<td>Total component length in cluster (in y).</td>
</tr>
</tbody>
</table>
A3.2 The Geometric Structure Blocks.

The following tables list the fields of the geometric data structure blocks. It should be remembered that the geometric structure is segmented, one segment (page) corresponding to each slot on the board. Each segment is, in fact, a self-contained data structure, built in a 1-dimensional FORTRAN ARRAY using the free-storage scheme described in Appendix 1. There are no cross-page references between segments, but pointers do exist from each segment to the circuit structure, which is always in core, so posing no problems.

Because of the similarity of the geometric feature blocks - namely PIN, THROUGH-HOLE, DUMMY PIN, BEND and TIE blocks - these are dealt with together in Table A3.9, any differences being noted in the description. It should be remembered that a sub-tree is, in fact a directed tree, and so such statements as "the track following a feature block" are valid, and, in the context of a specific tree structure, defined.

<table>
<thead>
<tr>
<th>FIELD</th>
<th>MACRO NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0,8)</td>
<td>TYPEX</td>
<td>No. of pointers to other sub-blocks.</td>
</tr>
<tr>
<td>(0,21,35)</td>
<td>SUBCOM</td>
<td>Pointer to component of sub-block.</td>
</tr>
<tr>
<td>(1,0,35)</td>
<td>SUBNEX</td>
<td>Pointer on ring of sub-blocks of cluster.</td>
</tr>
<tr>
<td>(2,0,35)</td>
<td>LENCOM</td>
<td>Length of component of sub-block (in y).</td>
</tr>
<tr>
<td>(3,0,35)</td>
<td>SUBDEL</td>
<td>Workspace: holds D-value or swap gain.</td>
</tr>
<tr>
<td>(4,0,8)</td>
<td>WEIGHT</td>
<td>Weight of connections to SUB2PTR.</td>
</tr>
<tr>
<td>(4,9,35)</td>
<td>SUB2PTR</td>
<td>Pointer to 1st connected sub-block.</td>
</tr>
</tbody>
</table>

*The above two are repeated, in subsequent words, for each of the TYPEX connected component sub-blocks.

Table A3.6 The Fields of a Component SUB-BLOCK.
Table A3.7  **The Fields of a ROOT-POINTER Block.**

<table>
<thead>
<tr>
<th>FIELD</th>
<th>MACRO NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0,5)</td>
<td>TYPE</td>
<td>Root-pointer block identifier. Value = 9.</td>
</tr>
<tr>
<td>(0,12,23)</td>
<td>ROOTPTR</td>
<td>Pointer to root block of sub-tree.</td>
</tr>
<tr>
<td>(0,24,35)</td>
<td>NEXROOT</td>
<td>Pointer to next root-pointer block in ring.</td>
</tr>
<tr>
<td>(1,21,35)</td>
<td>NODROT</td>
<td>Pointer to net in circuit structure to which the sub-tree on ROOTPTR belongs.</td>
</tr>
</tbody>
</table>

Table A3.8  **The Fields of a FIXED-X or FIXED-Y RINGHEAD Block.**

<table>
<thead>
<tr>
<th>FIELD</th>
<th>MACRO NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0,5)</td>
<td>TYPE</td>
<td>Fixed-X ringhead identifier. Value = 29.</td>
</tr>
<tr>
<td>(0,12,23)</td>
<td>LAXYHEAD</td>
<td>Fixed-Y ringhead identifier. Value = 30.</td>
</tr>
<tr>
<td>(0,24,35)</td>
<td>NEXYHEAD</td>
<td>Pointer to last fixed-X/fixed-Y ringhead.</td>
</tr>
<tr>
<td>(1,0,11)</td>
<td>XYCOORD</td>
<td>Pointer to next fixed-X/fixed-Y ringhead.</td>
</tr>
<tr>
<td>(1,12,23)</td>
<td>NexusRIng</td>
<td>Fixed-X/fixed-Y coordinate.</td>
</tr>
<tr>
<td>(1,24,35)</td>
<td>NEXYRING</td>
<td>Pointer to ring of geometric feature blocks at fixed-X (for fixed-X ringhead only).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Pointer to ring of geometric feature blocks at fixed-Y (for fixed-Y ringhead only).</td>
</tr>
<tr>
<td>FIELD</td>
<td>MACRO NAME</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| (0,0,5) | TYPE       | PIN block identifier. Value = 10.  
            |            | HOLE block identifier. Value = 7.  
            |            | DUMMY block identifier. Value = 8.  
            |            | BEND block identifier. Value = 5.  
| (0,6,6) | TAGLINK    | Board side of track following block.  |
| (0,7,10)| SEGLINK    | No. of slot containing block.  |
| (0,11,12)| LINKSID   | Board side(s) occupied by feature block.  |
| (0,13,13)| LINKSIDF  | Bit = 1 iff track following block is allocated to one side of board.  |
| (0,14,20)| TRAKWID  | Width of track following feature block.  |
| (0,21,35)| NEXLINK  | Pointer to duplicate pin in circuit structure, for PIN blocks only.  |
| (0,21,35)| NEXNODE  | Pointer to associated net block in circuit structure, for DUMMY blocks only.  |
| (0,21,23)| NOTIES   | No. of ties in tie-ring. TIE Blocks only.  |
| (0,24,35)| TIEERING | Pointer to next tie in tie-ring, for TIE blocks only.  |
| (1,0,11)| NODETREE  | Pointer to next block in sub-tree.  |
| (1,12,23)| NEXXRING | Pointer to next block in fixed-X ring.  |
| (1,24,35)| NEXYRING | Pointer to next block in fixed-Y ring.  |
| (2,0,11)| BROOT     | Pointer to root of tree of block.  |
| (2,12,23)| LASNOT   | Pointer to last block in sub-tree. If block is root, pointer to ROOT POINTER.  |
| (2,24,24)| TIEDEP   | Bit = 1 iff dependent block (see 7.1).  |
| (2,26,26)| GETTAG   | Bit = 1 iff track following block is conflict-free.  |
| (2,27,35)| LANDIAM  | Diameter of land of PIN or HOLE.  |

Table A3.9 The Fields of a GEOMETRIC FEATURE Block.
<table>
<thead>
<tr>
<th>FIELD</th>
<th>MACRO NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0,5)</td>
<td>TYPE</td>
<td>TEMPORARY block identifier. Value = 12. IMPEDIMENT block identifier. Value = 13.</td>
</tr>
<tr>
<td>(0,6,6)</td>
<td>TAGLINK</td>
<td>Board side of track following block*.</td>
</tr>
<tr>
<td>(0,11,12)</td>
<td>LINKSID</td>
<td>Board side(s) occupied by block*.</td>
</tr>
<tr>
<td>(0,13,13)</td>
<td>LINKSIDF</td>
<td>Bit = 1 iff track following block* is allocated to one side of the board.</td>
</tr>
<tr>
<td>(0,14,20)</td>
<td>TRAKWID</td>
<td>Width of track following block*.</td>
</tr>
<tr>
<td>(0,24,35)</td>
<td>NEXROOT</td>
<td>Pointer to next temporary/impediment block.</td>
</tr>
<tr>
<td>(1,0,11)</td>
<td>ALLOX</td>
<td>Workspace : index in array &quot;NEXUS&quot;.</td>
</tr>
<tr>
<td>(1,12,23)</td>
<td>NEXXRING</td>
<td>X-coordinate of block* (not pointer).</td>
</tr>
<tr>
<td>(1,24,35)</td>
<td>NEXYRING</td>
<td>Y-coordinate of block* (not pointer).</td>
</tr>
<tr>
<td>(2,0,11)</td>
<td>BROOT</td>
<td>Pointer to root of sub-tree of block*.</td>
</tr>
<tr>
<td>(2,27,35)</td>
<td>LANDIAM</td>
<td>Diameter of land if block* is PIN or HOLE. Width of track if block* is DUMMY, TIE or BEND.</td>
</tr>
<tr>
<td>(3,0,11)</td>
<td>NEXTY</td>
<td>Y-coordinate of next block in tree.</td>
</tr>
<tr>
<td>(3,12,23)</td>
<td>NESEX</td>
<td>X-coordinate of next block in tree.</td>
</tr>
<tr>
<td>(3,24,35)</td>
<td>LASTREE</td>
<td>Pointer to associated feature block of temporary/impediment block.</td>
</tr>
<tr>
<td>(4,0,11)</td>
<td>TARGY</td>
<td>Y-coordinate of target of this tree branch.</td>
</tr>
<tr>
<td>(4,12,23)</td>
<td>TARGX</td>
<td>X-coordinate of target of this tree branch.</td>
</tr>
</tbody>
</table>

* blocks marked thus are the associated feature blocks of the temporary/impediment blocks, as from LASTREE.

Table A3.10. The Fields of a TEMPORARY or IMPEDIMENT Block.
APPENDIX 4. The System Hardware Configuration.

Fig. A4.1 shows those parts of the hardware configuration used by the layout program, the arrows representing the data paths between the two machines and their peripherals.
Because of the total size of the layout program and the need to run it in a machine of moderate size, both for program development and any future industrial implementation, the program is overlayed. The overlaying scheme used on the PDP-10 allows one overlay (the resident overlay) to remain permanently¹ in core, together with one other overlay (a temporary overlay) selected (under program control) from the set of temporary overlays. The resident overlay is used to hold the data structure array and common blocks, together with those subroutines which are called from many points in the program (i.e. from most or all of the temporary overlays). The division of the remainder of the program into temporary overlays is arranged to minimise the total program size while avoiding excessive time overheads in changing the temporary overlay. Table A5.1 outlines this division, giving an indication of the size of each of the overlays.

From Table A5.1 it will be noted that the total program size, being the sum of the sizes of the resident overlay and the largest temporary overlay, is $15700 + 14700 = 30400$ words. This compares with a total code size of $63300$ words, for the whole program. The program, of course, requires more than $30400$ words of core to run in, as space is required for the FORTRAN run-time system and for input/output device buffers. In the PDP-10, about $37K$ of core is required, in total.

¹ "Permanent" is used here only with respect to the user program, the user program being swapped between core and disc under the control of the time-sharing monitor.
<table>
<thead>
<tr>
<th>OVERLAY</th>
<th>CONTENT</th>
<th>SIZE (WORDS)</th>
<th>OVERLAY SIZE (WORDS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESIDENT</td>
<td>Common blocks</td>
<td>600</td>
<td>15700</td>
</tr>
<tr>
<td></td>
<td>Data Structure array</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Routines called from</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>many points in program</td>
<td>3200</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Graphics routines</td>
<td>800</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FORTRAN library routines</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>1st</td>
<td>Placement routines</td>
<td>13300</td>
<td>13300</td>
</tr>
<tr>
<td>2nd</td>
<td>Placement interaction</td>
<td></td>
<td>12200</td>
</tr>
<tr>
<td></td>
<td>routines</td>
<td>2700</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Graphics routines</td>
<td>5500</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Display file</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>3rd</td>
<td>Routines to initialise</td>
<td></td>
<td>700</td>
</tr>
<tr>
<td></td>
<td>geometric data structure</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4th</td>
<td>Interactive route planning</td>
<td></td>
<td>10900</td>
</tr>
<tr>
<td></td>
<td>routines</td>
<td>1900</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Graphics routines</td>
<td>5000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Display file</td>
<td>4000</td>
<td></td>
</tr>
<tr>
<td>5th</td>
<td>Route planning routines</td>
<td>3700</td>
<td>3700</td>
</tr>
<tr>
<td>6th</td>
<td>Aperture routing routines</td>
<td>10600</td>
<td>10600</td>
</tr>
<tr>
<td>7th</td>
<td>Routines for interaction with automatic routing</td>
<td>14700</td>
<td></td>
</tr>
<tr>
<td></td>
<td>with automatic routing</td>
<td>4600</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Graphics routines</td>
<td>6100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Display file</td>
<td>4000</td>
<td></td>
</tr>
</tbody>
</table>

Table A5.1  Content and Size of Program Overlays.
APPENDIX 6  2-Way Partitioning

The 2-way partitioning procedure, as described by Kernighan and Lin (27), is considered in A6.1, and thereafter the additions required for the solution of the problem, as outlined in 3.2, in A6.2.

A6.1  The Procedure due to Kernighan and Lin

This section is, in essence, a slightly summarised reiteration of the description of 2-way partitioning contained in the quoted paper of Kernighan and Lin (27), which source the author gratefully acknowledges (Copyright 1970, American Telephone and Telegraph Company).

Given a set S of 2n nodes, with associated cost matrix (weighted connectivity matrix) \( C(c_{ij}) \), \( i,j=1...2n \), where \( c_{ii}=0 \) for all \( i \), and \( c_{ij}=c_{ji} \) for all \( i,j \), we require to find a partition into two sets, A and B, each with n nodes, such that the total interconnection cost between A and B is minimal. We shall designate this total cost \( T \), where

\[
T = \sum_{A \times B} c_{ab}.
\]

For each node \( a \in A \) the external cost \( E_a \) is given by:

\[
E_a = \sum_{y \in B} c_{ay},
\]

while the internal cost \( I_a \) is given by:

\[
I_a = \sum_{x \in A} c_{ax}.
\]

Similarly for a node \( b \in B \) we will have \( E_b \) and \( I_b \). Defining \( D_z \) as the difference between external and internal cost we have

\[
D_z = E_z - I_z \text{ for all } z \in S.
\]
If $z$ is the total cost due to all interconnections between $A$ and $B$ which do not involve $a$ or $b$, then the cost $T$ is given by:

$$T = z + E_a + E_b - c_{ab}.$$ 

Now, if we consider the interchange of $a$ and $b$ between $A$ and $B$, then the new total cost, $T'$, will be:

$$T' = z + I_a + I_b + c_{ab}.$$ 

The "gain" (i.e. reduction in total cost) from this swap is thus:

$$\text{GAIN} = \text{OLD COST} - \text{NEW COST} = T - T' = E_a - I_a + E_b - I_b - 2c_{ab} = D_a + D_b - 2c_{ab} \ldots \ldots \ldots \ldots \quad (A6.1)$$

In order to partition efficiently for minimal total cost, we wish to select $a_i \in A$ and $b_j \in B$ for maximal gain. Specifically, we wish to select a subset $X \subset A$ and a subset $Y \subset B$, the interchange of which will yield the minimal cost partition, and so we require to sequentially identify the elements of $X$ and $Y$.

Since $c_{ij} > 0$, we have from Eqn. A6.1 the following: if we compute $D_a$ for all $a \in A$ and $D_b$ for all $b \in B$, and order these in two lists, with the highest values at the top, then in scanning these lists in order, if a pair $D_{a_i} \ , \ D_{b_j}$ is found whose sum does not exceed the maximum improvement encountered so far in this scan, then there will not be a pair lower in the lists which can give a greater gain, and so scanning can be terminated. In general the greatest gain will correspond to those values nearest the top of the lists, and so it is adequate to save only the first three $D_a$ and $D_b$ values (i.e. the highest three).
Thus we partition as follows. First compute the D values for all of the nodes of S, saving only the three largest of the D_a and of the D_b values. Then, considering only these elements of A and B corresponding to the saved D_a and D_b values, find a_1 \epsilon A and b_1 \epsilon B so as to maximise the gain g_1 as in Eqn. A6.1. These elements, a_1 and b_1 are removed from A and B, and the D values recomputed for A-a_1 and B-b_1 using:

\[ D'_x = D_x + 2c_{x_1} a_1 - 2c_{x_1} b_1, \quad x \in A-a_1, \]

and \[ D'_y = D_y + 2c_{y_1} b_1 - 2c_{y_1} a_1, \quad y \in B-b_1, \]

again saving only the largest three of the D_a and D_b values. We then select a_2 \epsilon A-a_1 and b_2 \epsilon B-b_1 for maximum gain g_2, remove them from A-a_1 and B-b_1, re-compute the D values, and so on until all the nodes have been removed from A and B. Consider then the swapping of the first k of the selected pairs, taken in order. The total gain will be \[ g_1 + g_2 + \ldots + g_k, \]
and to maximise total gain, we simply choose k for maximal gain. If there exists a value of k for which the total gain is both maximal and positive, then the first k pairs, a_1, b_1, a_2, b_2 \ldots a_k, b_k, are swapped, and the procedure re-started. If for all k the total gain is zero or negative, then a locally optimum partition has been achieved.

This procedure is probably best summarised by the outline flowchart of Fig. A6.1.

A6.2 The Modified Partitioning Procedure.

The problem may be re-stated as that of partitioning the set S of n+m nodes, with cost matrix C as in A6.1, and where the size of node i is l_i, into two sets A and B of n and m nodes respectively.
FOREACH \( aeA \) find \( E_a, I_a \).
FOREACH \( beB \) find \( E_b, I_b \).
HENCE \( D_a, D_b \). SET \( p=1, n=\text{No.} \)
OF NODES, \( A=A, B=B \).

CONSIDERING ONLY THE 3 LARGEST
\( D_a \) and \( D_b \) VALUES, FIND \( a_1, b_j \)
SO AS TO MAXIMISE GAIN, \( g_i \), WHERE
\[
g_i = D_a + D_b - 2c_{a_i b_j}
\]

SET \( a = a_1, \)
\( b = b_j, \)
\( A_{p+1} = A_p - a_1, \)
\( B_{p+1} = B_p - b_j. \)

\( p=n? \)
\( yes \)

CHOSE \( k \) TO
MAXIMISE TOTAL
GAIN, \( G \), WHERE
\[
k \quad G = \sum_{i=1}^{k} g_i
\]
SWAP 1st \( k \) ELEMENTS
SELECTED FROM \( A \) and \( B \)
a_1, \ldots, a_k \to B \) and
b_1, \ldots, b_k \to A.

\( G > 0? \)
\( yes \)
\( no \)
EXIT

Fig. A6.1 Flowchart of Kernighan and Lin's Partitioning Procedure
(i.e. of cardinality \( n \) and \( m \)), such that the total interconnection cost, \( T \), between \( A \) and \( B \) is minimal, subject to the overriding constraint that the sizes of the sets be similar (the degree of similarity required is specified functionally, as below), where the size of \( A \) is:

\[
L_A = \sum_{a=1}^{n} 1_a , \quad a \in A
\]

and the size of \( B \) is:

\[
L_B = \sum_{b=1}^{m} 1_b , \quad b \in B
\]

This is achieved by including a size-dependent COST factor in the determination of the gain, \( g_1 \), resulting from a given swap, the behaviour of this factor being such as to favour any interchange resulting in more nearly equal partition sizes, and vice-versa. Also, the factor behaves in a non-linear manner, such that it only becomes significant where the partition sizes are grossly different (either before or after the swap being considered). Thus, as indicated in 3.2.3, we use the following function in deriving the net gain, \( g_1 \), resulting from the swapping of \( a_1 \in A \) and \( b_1 \in B \), and similarly for \( g_2, g_3 \ldots g_{\text{max}} \):

\[
ge_1 = D_{a_1} + D_{b_1} - 2c_{a_1}b_1 - \text{COST} ,
\]

\[
\text{COST} = c_1 \times \text{RATIO} + c_2 \times \text{RATIO}^3 ,
\]

\[
\text{RATIO} = \frac{\left[ \frac{L_A}{L_B} - \frac{L_A-1_{a_1}+1_{b_1}}{L_B} \right]}{\left[ \frac{L_A-1_{a_1}+1_{b_1}}{L_B} \right]}
\]

where the coefficients \( c_1 \) and \( c_2 \) are linear functions of \( T = \sum_{A \times B} c_{ab} \), in order that the relative "weight" attached to this geometric factor should be independent of cluster connectivity. After the calculation.
of each $g_i$, the total cluster sizes are modified appropriately:

$$L'_A = L_A - l_{a_i} + l_{b_j},$$

$$L'_B = L_B - l_{b_j} + l_{a_i}.$$

It should be noted that, as compared with A6.1, partitions are no longer required to be of equal cardinality, and so in establishing the sequence of step-wise maximal gains, $g_1 \cdots g_{\text{max}}$, the process terminates when either $A$ or $B$ is empty, and we no longer have the guarantee that

$$\max_{i=1}^{\text{max}} g_i = 0,$$

although this is unimportant.

The process otherwise remains as outlined in A6.1.
This Appendix is a re-print of an article published by the author in "Electronics Letters" (23).

COMPONENT PLACEMENT THROUGH GRAPH PARTITIONING IN COMPUTER-AIDED PRINTED-WIRING-BOARD DESIGN

Indexing terms: Computer-aided circuit design, Printed circuits, Graph theory

The partitioning of the abstract graph representing an electronic circuit has been applied to the placement of components on a printed wiring board. Results obtained from a computer program exhibit a marked degree of similarity to placements produced by a skilled draughtsman, for a very low processing cost.

The interconnections of any circuit can be represented by the branches of an abstract graph, components being considered as nodes. In turn, for the design of printed wiring boards by computer, such a graph may be represented by an associative data structure. The author has implemented a ring-based associative structure designed for this application.

The use of the partitioning of the nodes of a graph for the allocation of components to particular circuit boards has already been suggested, together with an efficient heuristic method of partitioning, by Kernighan and Lin. To the best of the author's knowledge, component placement for a given board has not yet been attempted in a similar manner. By the use of partitioning, it is possible to use the more significant features of the electrical circuit (i.e. connectivity) in generating a placement at the global level. The partitioning procedure due to Kernighan and Lin solves the combinatorial problem of dividing the nodes of a graph, with costs on its branches, into subsets no larger than a given maximum size. In implementing their procedure, a number of changes have been made to handle the 'real' problem, as opposed to an abstract combinatorial one. The major amendments are as follows:

(a) The original concept of 'size' was simply the 'number of nodes', whereas a literal interpretation of size as 'total component area' is required. To this end, a cost function is added in evaluating the desirability of a given exchange, being negative for an exchange resulting in partition areas nearer to equality, and vice versa.

(b) The redefinition of size in (a) suggests that 'slack' (unconnected nodes of zero size) be introduced, so that unidirectional exchanges may be made, but without any increase in the number of partitions (cf. Kernighan and Lin).

(c) The original program used the $N \times N$ symmetric cost matrix (weighted adjacency matrix) of the graph on $N$ nodes as its data base. For most circuits, this matrix will be both large ($N^2$) and sparse, and thus it was considered wise to implement the algorithm using a ring-based associative structure as its data base. This has proved to be economical in storage and fairly fast (see Table 1).
Although the placement of components on a board and the routing of conductors by a draughtsman are parts of a unified process, heavily dependent on the excellent human eye–brain pattern-recognition abilities, one property of a 'good' placement (i.e. one of the attributes of 'goodness' in this context), which can be easily defined, is that highly connected (in the graph-theoretic sense) components should be as close together as possible, in the interest of ease of wiring (i.e. short, simple interconnections). Considerable help in achieving this can be obtained by partitioning the circuit graph so that the nodes (components) lie in maximally internally connected (and therefore minimally interconnected) clusters. The remaining problem is then to associate clusters with regions of the board, and components with regions of the clusters. Assuming a technology using at least two surfaces for conductor routes, it seems reasonable to suppose that a conductor pattern favouring orthogonal routes on the two (sets of) surfaces will yield the highest potential packing density. Thus a given cluster is associated with a narrow strip of the board, so that the relatively high internal (to the cluster) connection density corresponds to the use of mainly one of the two available axes for routing, leaving the other largely free for the (probably) more complex cluster interconnections.

A FORTRAN program has been written to perform these tasks, and the results compared with those of a skilled draughtsman for a number of digital circuits (from 11 to 44 components, each with 14, 16 or 24 pins). It should be noted that the program does not guarantee a globally optimum solution, either in partitioning or in placement. For comparative purposes, a modified version of the program was produced which generates random placements, and also a version which accepts data describing the placement produced by a draughtsman in an industrial production environment. In the latter two cases, the method of estimation of total route length remained identical to that of the automatic program.

From drawings showing interconnections as straight pin-to-pin connections (available in an internal report, but not printed here owing to technical difficulties), it can be seen that the program-generated solutions are very similar to those produced by a draughtsman in terms of interconnection density (conductor length per unit area of board), while both of these are considerably better than any of the random solutions produced, both in terms of the mean interconnection density and the uniformity of the density distribution over the area of the board.

For purposes of assessment, an estimate of total conductor length was obtained from the program for all three cases. Although no great accuracy is claimed for the absolute values produced, the estimate does provide a useful basis for comparison, being one of the major 'goodness' criteria applicable to component placement. The mean total conductor length, from a number of random placements of a given circuit, was used as a base against which a percentage improvement was calculated for the program and draughtsman-generated placements. The results are tabulated, along with the c.p.u. time (for reading the circuit data, building the data structure, partitioning the circuit, placing clusters on the board and placing components in clusters) for a DEC PDP-10 (64K 1 μs core, but without fast registers), in Table 1.

*HOPE, A. K.: 'Component placement through graph partitioning in computer-aided printed-wiring-board design' (Internal report CAD-R-143)
Table 1  PERCENTAGE IMPROVEMENT IN TOTAL CONDUCTOR LENGTH

<table>
<thead>
<tr>
<th>Circuit number</th>
<th>Number of packages</th>
<th>Improvement in wire length over random placement</th>
<th>C.P.U. time for program placement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>of draughtsman placement of program placement</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>%</td>
<td>%</td>
</tr>
<tr>
<td>2</td>
<td>31</td>
<td>37</td>
<td>47</td>
</tr>
<tr>
<td>3</td>
<td>35</td>
<td>43</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>36</td>
<td>29</td>
<td>33</td>
</tr>
<tr>
<td>5</td>
<td>41</td>
<td>21</td>
<td>29</td>
</tr>
<tr>
<td>6</td>
<td>41</td>
<td>35</td>
<td>29</td>
</tr>
<tr>
<td>7</td>
<td>41</td>
<td>27</td>
<td>25</td>
</tr>
<tr>
<td>8</td>
<td>42</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>9</td>
<td>44</td>
<td>35</td>
<td>30</td>
</tr>
</tbody>
</table>

It will be noted from Table 1 that, on average, the improvement by program is marginally better than that achieved by a draughtsman, and that, moreover, the program results are more consistent. It can thus reasonably be claimed that the particular aspects of pattern recognition relating to the extraction of clusters from a graph can be adequately simulated by an automatic program as outlined, producing a placement which has a reasonable total wire length, and hopefully will have high 'wireability' (although the latter remains to be proven).

Thanks are due to Ferranti Ltd., Edinburgh, which is supporting this work, and to the UK Science Research Council for the provision of computing facilities under grant B/SR/8874.

A. K. HOPE

28th January 1972

Computer-Aided Design Project
James Clerk Maxwell Building
University of Edinburgh
King's Buildings
Edinburgh EM9 3JJ, Scotland

References

Appendix 8  A Planar Graph Drawing Program.

This Appendix is a re-print of an article published by the author in "Software - practice and experience" (24). Though of no direct relevance in the approach finally adopted to layout, as described in this thesis, it is included as being of potential interest in the approach to layout using planar graph theory, as outlined in 2.4.
A Planar Graph Drawing Program

ALEXANDER K. HOPE
Computer-aided Design Project
University of Edinburgh, Forrest Hill, Edinburgh EH1 2QL

SUMMARY
This paper describes a program, written in FORTRAN, for a 64K I.C.L. 4130 which will accept data describing the ordering of nodes and branches within the regions of a planar graph, and generate a two-dimensional representation, without crossovers, from these data. The program was written to provide one of the basic 'tools' required in the development of a printed wiring board layout program for the electronics industry.

The program generates display code which is sent to a satellite PDP-7 computer driving a 340 display. On completion of the automatic drawing phase an interactive phase is entered in which the user can 'tidy' and label the drawing, by means of keyboard and light-pen commands.

Brief notes are included on the hardware, the data structures package (MINIJASP, derived from ASP) and the graphics package, employed.

KEY WORDS Abstract-graph Planar No-crossings Ring-structure

BACKGROUND
For purposes of computer-aided layout of printed wiring boards, an electronic circuit may be represented by an abstract graph, thus:

(1) \( n \) terminal devices are represented by subgraphs consisting of a ring of \( n \) branches, the \( n \) nodes of which have one outgoing branch each.

(2) \( n \) branch conductor trees (i.e. the common electrical connection of \( n \) component terminals) are represented by nodes of degree \( n \), with \( n \) outgoing branches.

(3) The edge-connector and perimeter of the board are represented by pseudo-branches which serve to restrain components and conductors to lie 'on' the board, and ensure that the abstract graph is connected (i.e. a path always exists between any two given nodes).

It is clear that for a given circuit to be physically realizable, the partitions of this complete abstract graph, corresponding to individual layers of the printed wiring board, must be planar, since components may not cross each other, and conductors may only meet where an electrical connection is intended. It is, of course, permissible for certain components to 'bridge' a fixed number of conductors, thus introducing a class of permissible non-planarities.

In general, the abstract graph representing an electronic circuit will not be planar, and it is necessary to extract one or more planar partitions of the whole graph in the first instance. For purposes of program development, and also in the hope of being able to introduce interaction at the graph-theoretic stage of a printed wiring board layout program, the present program for drawing planar graphs was developed.

Received 12 August 1970
Revised 9 September 1970
HARDWARE AND SOFTWARE PACKAGES EMPLOYED

Hardware
The system consists of a 64K (2 μsec, 24-bit word) I.C.L. 4130 tied via a high-speed link (300 kilobaud) to an 8K (18-bit word) PDP-7 with a 340 display.

Graphics software
The system software for the interactive graphic display consists of two major parts, one for each machine. On the 4130 side a set of FORTRAN subprograms is available which enables the user to build and modify a display file under control of his FORTRAN program. This display file is transmitted via the link to the PDP-7 where it is interpreted by an executive program which drives the display. The user's program is thus written entirely in FORTRAN, the details of the display file generation, link handling and display file interpretation being transparent to the user.

A paper describing this software in some detail is to be published shortly in this journal.

ASP and MINIJASP
ASP² is a ring-implemented data structure package, which allows the user to store and manipulate data items, or the relationships between those items.

An ASP structure consists of elements (containing an arbitrary, user-defined, amount of data) and rings, ringstarts and associators which define the relationships between the elements.

For example, ELEMENTS have 'upper' and 'lower' RINGS. The upper rings are rings of associators, these belonging to associative rings which link elements into 'sets' (e.g. in Figure 3b the set of points \( \langle C_2, NP_1, C_1 \rangle \) is the set of points on edge 1, and so lies on one associative ring). The lower rings are rings of ringstarts, each ringstart beginning an associative ring (e.g. the ringstart on the lower ring of edge 1 in Figure 3b).

Each element of a ring could usefully contain pointers to the next element, the previous element and the head of the ring, in the interests of speed. This, however, costs a lot in terms of storage requirements for the extra pointers.

MINIJASP is a somewhat restricted version of ASP, written as a set of FORTRAN subroutines for the I.C.L. 4130. MINIJASP rings contain only forward pointers, thus sacrificing speed for storage.

APPROACH
A ring-structure package (MINIJASP, see previous Section) is used to hold the topological data, the layout data being added to the structure as the layout proceeds.

The layout is developed, in a rectangular geometry for ease of display, working radially outwards from a starting region. The terms 'radial' and 'tangential' will be used for brevity in an obvious, though not strict, sense. We thus define a 'working rectangle', being the perimeter of the graph as drawn, at any given stage of its development.

PROGRAM DESCRIPTION
An outline flow diagram is shown in Figure 1.

Building the data structure representing the graph
The classes of regions, branches and nodes are each assigned a separate associative ring in the structure (see Figure 2), all nodes, for example, lying on the node ring.
Start

Read data. Build data - structure for abstract graph

Select and draw START region

GROW unplaced branches of nodes radially outwards

Any branches left?

n

MERGE adjacent common branches tangentially

Any branches left?

n

INSERT target NODES of degree > 2 on outgoing branches

Further enclosing of current regions possible?

n

INSERT NODES of degree 2, if any, on outgoing branches

EXTEND branches radially to avoid geometric overlap

More merging or inserting?

n

JOIN adjacent groups of branches in common target nodes

EXTEND unjoined branches radially to clear the joined groups

Interactive phase

Finish

Figure 1. Outline flow diagram
Regions, and their ordered branches and nodes, are read in from paper-tape and a structure is built up, as outlined in Figure 2. Unfortunately the MINIJASP package only allows addition of new items (specifically associators and ringstarts) at the beginning of any given ring. It is thus impossible, for example, to generate the structure taking account of the ordering of nodes and branches within regions as well as the ordering of branches.

**Figure 2. Graphical data storage in the MINIJASP data-structure.** (a) Simple graph; (b) Data structure corresponding to graph of Figure 2(a)
about nodes. Consequently it is necessary to delete some of the structure following the initial building, and recreate it with the required ordering. It may be reasonably argued that this is a direct consequence of using a structure in which all relations and their inverses are held explicitly, but this is essential in terms of speed at the layout stage.

Adding layout data to the data-structure
Two further classes are added to the MINIJASP structure, being:

(1) GEOMETRIC POINTS (3 word blocks: normalized X and Y co-ordinates, and a pointer to the branch segment or node associated with the point).

(2) WORKING RECTANGLE EDGES, from which the branches and nodes currently being dealt with by the layout program can be accessed, in appropriate (spatial) order.

A fragment of the structure is illustrated in Figure 3.

The layout routines increment the working rectangle appropriately, creating new points associated with the appropriate nodes or branch segments. (Since a branch may have two spatially isolated segments prior to its completion by merging, it is necessary to specify 'branch segment' rather than 'branch'.)

The user may specify (by keyboard) which region is required to be the infinite region (i.e. the 'outside' of the board). The program will then select a starting region separated from the infinite region by as many other regions as possible. (For this purpose, regions are defined as being adjacent if they share a common branch or a common node—the normal definition only considers common branches.) The starting region is drawn as a rectangle with the required number of nodes and branches, uniformly spaced as far as is possible.

It should be noted that the ability to specify the infinite region allows the user to draw the stereographic projection in the topological sense) of the graph, based on any of its regions. A trivial modification would allow the user to specify both the infinite region and the starting region.

By a process of growing and radially extending branches, and attempting to enclose regions by merging and joining tangentially, as indicated in Figure 1, the program will eventually enclose all but the infinite region. This process is probably best described by the series of frames, showing a layout after successive application of the appropriate algorithms, in Figure 4.

Interaction
After completion of the automatic drawing phase, the user may interact, at present at a fairly low level (by which is meant geometric rather than topological level), with the program in order to 'tidy' and label the drawing. This is done by means of keyboard and light-pen commands which allow:

(1) Inserting, moving and deleting geometric construction points (i.e. 'bending' points).

(2) Moving nodes.

(3) Labelling regions and nodes with the names specified in the input data to the program.

These routines update the appropriate co-ordinates, etc. in the MINIJASP structure held in the 4130, and transmit the required amendments to the display file via the high-speed link to the PDP-7 and 340 display. The speed of interaction is very high, giving
almost instantaneous response for simple operations (e.g. deletion of bending points), and of the order of 1 sec. delay in moving a node, where all of the branches associated with that node have to be deleted and redrawn.

* Figure 3. Layout data storage in the MINI/ASP data-structure.* (a) Simple layout; (b) Fragment of data structure corresponding to layout of Figure 3(a)

* Only one edge ring and one branch ring are shown, for clarity.
CONCLUSIONS AND COMMENTS

The program: its operation and uses

The program (written in S/C FORTRAN) uses 48K (24-bit words) of core, including a 4K display file and a 12K MINIJASP array for the data structure. The time required to

Figure 4. Program output. (a) After drawing the start region; (b) after application of the following algorithms: GROW, EXTEND, MERGE, INSERT NODES, EXTEND;* (c) on completion of the automatic phase; (d) after geometric interaction to 'tidy' the graph and label the regions

* See Figure 1 for interpretation of names.
draw a graph of some 20 regions, 20 nodes and 38 branches is about 90 sec. The time required for interaction depends on the use to be made of the graph, but is typically 5 min. simply to tidy and label such a graph.

It is hoped that the interactive phase will be extended, in conjunction with the printed wiring board layout program, to allow interaction at the graphical stage of this program (e.g. to allow re-routing of non-planar branches). The author would be interested to hear of any other possible application areas.

**A criticism of the MINIJASP associative structures package**

It has already been mentioned in the section 'Building the data structure representing the graph' that, where strict ordering of elements is required, MINIJASP imposes certain restrictions, since an associator or ringstart may only be inserted at the beginning of a ring. This restriction does not apply in ASP, which allows insertion of items at any defined point in a ring.

In both ASP and MINIJASP there is a storage penalty for having a highly connected structure: in MINIJASP, for example, each ringstart, associator or element incurs a 2-word overhead for pointers and labels.

In MINIJASP there is, moreover, the problem due to unidirectional pointers: for example, to transfer the \( n \) elements of a ring, preserving order, to another ring, it is necessary to traverse the original ring \( n \) times, rather than once.

In both ASP and MINIJASP there is a problem due to the language level. In providing a general-purpose package the routines are necessarily basic which involves the user in a great deal of coding when dealing with a structure of any complexity.

In order to avoid a number of these problems, a hybrid structure has been employed in which some of the data words of elements are used as pointers, rather than using ringstarts and associators. Where, as in this case, the hierarchical nature of the structure is rigidly defined, this technique will lead to:

1. A significant reduction in the amount of coding required for given operations.
2. An increase in the speed of operation, since some accesses can be made in 1 or 2 FORTRAN statements, rather than the 10 or 20 of the corresponding MINIJASP routines.
3. A considerable reduction in storage requirements in terms of both data structure and program code.

The fact that this hybrid structure is both faster, and smaller than a 'pure' MINIJASP structure simply reflects the inefficiency of a general purpose, ring structure package as compared with an ad hoc structure designed for a specific application. The latter would appear to be preferable when dealing with large applications programs and/or large structures where (and only where) the associative nature of the data can be simply expressed (i.e. where the full generality, in terms of associations expressed, of an ASP-like package is not required).

**ACKNOWLEDGEMENT**

My thanks are due to Ferranti Limited, Edinburgh, who are supporting this work, and to the departments of Computer Science and Machine Intelligence and Perception of Edinburgh University for the provision of computing facilities.
REFERENCES
Glossary of Terms.

The following list of terms is not intended to be comprehensive, but rather to list those terms used throughout the thesis with a unique or unusual meaning. In addition, a number of terms are listed where ambiguity exists in the literature regarding their meaning.

Abstract Graph  A non-empty set of nodes, a set of branches, and a mapping of the branches into the set of unordered pairs of nodes.

Branch (Edge)  An element of an abstract graph joining two nodes.

Circuit Graph  As used in partitioning, is the graph in which nodes represent components and branches their interconnections. Any pair of nodes is joined by at most one branch, that branch having an associated weight indicating the number of nets with a pin belonging to each of the components (nodes) of the branch.

Circuit Structure  The associative data structure holding the descriptions of the components and their interconnections.

Cluster  A highly interconnected set of components.

Complete Graph  An abstract Graph in which each distinct pair of nodes is joined by a branch, and in which there are no loops and no parallel branches.

Dependent Tiering  A tiering which is geometrically coincident with a through-plated hole or component pin.
Directed Tree  A tree (graph) in which a direction is associated with each branch (i.e. the graph mapping is of branches into the set of ordered pairs of nodes).

Dummy Component  Unconnected component occupying 1 grid unit (0.550") in Y, used in component placement refinement.

Dummy Pin  The geometric point on the "component" side of a board on an inter-slot boundary at which a branch of a particular net crosses the boundary.

Fixed-x/y Ring  A ring of geometric feature blocks in a slot all at the same x/y coordinate. The blocks in a given fixed-x/y ring are arranged in order of ascending y/x coordinate. The fixed-x/y rings start on fixed-x/y ringheads.

Fixed-x/y Ringhead  The block starting a fixed-x/y ring, and holding the x/y coordinate value. The fixed-x/y ringheads themselves are arranged in a ring with forward and backward pointers corresponding to decreasing/increasing coordinate values.

Geometric Feature  Any geometrically significant part of a sub-tree (i.e. the end of a sub-tree branch). Feature types are component pin, dummy pin, through-plated hole, bend and tie, the latter permitting the formation of junctions of more than 2 branches.

Impediment Ring  As for the Temporary Ring, but for the bottom of the aperture.

Inter-Slot Boundary  The mutual boundary of 2 adjacent slots on the board.
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manhattan Distance</td>
<td>In a 2-D Cartesian coordinate system, the Manhattan distance between two points is the sum of their separations in each axis independently - i.e. ((\delta x + \delta y)).</td>
</tr>
<tr>
<td>Master Definition</td>
<td>The data structure representation of the geometric description of a component, including its body outline dimensions and pin positions and sizes. One such master definition is required for each physically distinct type of component used.</td>
</tr>
<tr>
<td>Master Library</td>
<td>File (on backing store) containing master definitions.</td>
</tr>
<tr>
<td>Net (Node)</td>
<td>A set of electrically interconnected component pins.</td>
</tr>
<tr>
<td>Node (Vertex)</td>
<td>An element of an abstract graph.</td>
</tr>
<tr>
<td>Pad (Land)</td>
<td>An annular region of copper on each board surface, concentric with a through-hole (for double-sided boards).</td>
</tr>
<tr>
<td>Partition</td>
<td>A subset of the set of nodes of a graph, arranged such that the sum of the costs on the branches cut in forming partitions is minimal.</td>
</tr>
<tr>
<td>Root (sub-tree root)</td>
<td>That component or dummy pin of a sub-tree which is nearest to the top of the slot which the sub-tree occupies.</td>
</tr>
<tr>
<td>Root Pointer</td>
<td>A block with a pointer to a sub-tree root block.</td>
</tr>
<tr>
<td></td>
<td>The root-pointer block also points to the net block in the circuit structure to which the associated sub-tree belongs, thereby linking the circuit and geometric structures.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Route</td>
<td>The geometric description of the path followed by a track.</td>
</tr>
<tr>
<td>Route Plan</td>
<td>That Steiner tree which spans the set of nodes of a net in a slot, regardless of conflicts with other such trees. Formed in the first stage of routing and subsequently modified in the conflict minimisation stage to yield the final routing.</td>
</tr>
<tr>
<td>Segment</td>
<td>A self-contained data structure containing the topological and geometric description of the sub-trees in a particular slot.</td>
</tr>
<tr>
<td>Slack</td>
<td>The addition of unconnected nodes (components) of zero size to each partition to permit unidirectional swaps in partition refinement is referred to as the addition of &quot;slack&quot; to the problem.</td>
</tr>
<tr>
<td>Slot</td>
<td>A rectangular region of the board bounded by the board edges in y and by hypothetical inter-slot boundaries in x. A slot is wide enough to hold a single column of packages and the appropriate inter-package wiring.</td>
</tr>
<tr>
<td>Spanning Tree</td>
<td>A set of branches joining each of a given set of nodes with no loops and no parallel branches.</td>
</tr>
<tr>
<td>Steiner Tree</td>
<td>A special case of a spanning tree in which extra nodes, referred to as Steiner Points, are included to modify the topology and geometry of the tree. In general the object of adding Steiner Points is to minimise the total branch length, or to avoid conflicts with other such trees.</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Steiner-minimal Tree</td>
<td>A Steiner Tree of minimal total branch length, spanning a given set of nodes.</td>
</tr>
<tr>
<td>Stepping Aperture</td>
<td>A rectangular region of a slot, bounded in x by the inter-slot boundaries, and of variable height, the height being such that the top and bottom of the aperture correspond to adjacent (in y) features in the slot, with no feature at any intermediate y coordinate.</td>
</tr>
<tr>
<td>Sub-Block</td>
<td>A data structure block associated with a component, holding only the connectivity information and the component dimensions required for partitioning.</td>
</tr>
<tr>
<td>Sub-Tree</td>
<td>A directed Steiner tree spanning that subset of the set of nodes of a net within the geometric limits of a particular slot, including any dummy pins on the relevant inter-slot boundaries.</td>
</tr>
<tr>
<td>Temporary Ring</td>
<td>A ring of blocks in the geometric data structure, corresponding to the top of the stepping aperture, with one block for each feature at the top of the aperture and one block for each track segment crossing the top of the aperture (i.e. with its start point above the aperture and its end point below the top of the aperture).</td>
</tr>
<tr>
<td>Through-plated Hole</td>
<td>A hole in the board, plated internally with copper and tin-lead/gold, electrically connecting two pads, one on either side of the board and concentric with the hole. Used either for &quot;stitching&quot; a track from one side of the board to the other, or, in appropriately placed sets, for mounting components.</td>
</tr>
</tbody>
</table>
Tiering  A ring of tie blocks in the geometric data structure. The junction of n track segments is represented by a ring of (n-1) ties, one tie block being associated with each track segment incident on the junction and directed away from it.

Track  The conducting copper path, left on the board surface after etching, joining component pins, through-holes, etc.

Track Segment  A straight track joining two points (two feature blocks in the sub-tree representation).

Transparency  The transparency of a component in a given axis is the number of tracks parallel to that axis which can pass within the limits of the component body, unhindered by the pads associated with the component pins, expressed fractionally - i.e. as the number of tracks per unit length of component body.

Tree  A connected abstract graph with no circuits (a circuit being a sequence of branches, both ends of which are incident on the same node).

Via  The "channel" in which a track runs. The via size is the centre-to-centre track spacing.

x/y-Outline  The x- and y-outline dimensions of that rectangle "containing" a component (master) body.

x/y-Pitch  The x- and y-pitch of a component are the minimum board dimensions required by that component, including space for the offset axial leads of 2-pin components as appropriate, etc.
BIBLIOGRAPHY

The numbers in parenthesis are used to reference specific entries in the bibliography from the text of the thesis. A number of entries exist which, though not referred to explicitly, are included because of their relevance to the subject of the thesis.


(3) BEINEKE, L.W., "The decomposition of complete graphs into planar subgraphs", in "Graph Theory and Theoretical Physics", Academic Press, 1967, pp. 139-153.


(8) CASE, P.W., LeCLERCQ, A.R., GRAFF, H.H., MURLEY, W.G.,
GRIFFITH, L.E., and SPENCE, T.M., "Solid Logic Design
8, No. 2, April 1964, pp. 127-140.

(9) CHANG, S-K., "The generation of minimal trees with a
Steiner topology", Journal ACM, Vol. 19, No. 4, Oct. 1972,
pp. 699-711.

(10) CHILDS, D.L., "Feasibility of a set-theoretic data structure",
Concomp Project Technical Report No. 6, University of
Michigan, Aug. 1968.

(11) CHILDS, D.L., "Description of a set-theoretic data structure",

(12) CROCKER, N.R., McGuFFIN, R.W., and NAYLOR, R., "Computer-
aided placement for high-density chip-interconnection
systems", Electron. Lett. (IEE), Vol. 8 No. 20, Oct. 1972,
pp. 503-504.

(13) CULLYER, W.J., TUBBS, S., and STOCKTON, A.P. "Computer-
aided layout of microcircuits", Radio and Electronic
Engineer, May 1968, pp. 251-261.

(14) DUNNE, G.V., "The design of printed circuit layouts by computer",
Proc. 3rd Australian Computer Conference, Canberra, May 1966,
pp. 419-423.


