72

University of Southern California

11

An Automatic Placement Tool for Rapid Prototyping of Printed Circuit Boards

John Granacki and Tauseef Kazi

ISI/RR-93-388

November, 1993

INFORMATION SCIENCES INSTITUTE

310/822-1511

4676 Admiralty Way/Marina del Rey/California 90292-6695

ISI Research Report
ISI/RR-93-388
November, 1993

//

An Automatic Placement Tool for Rapid Prototyping of Printed Circuit Boards

John Granacki and Tauseef Kazi

ISI/RR-93-388

November, 1993

University of Southern California
Information Science Institute
4676 Admiralty Way, Marina del Rey, CA 90292

Unclassified/Unlimited

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 11-93	3. REPORT TYPE AND DATES COVERED Research Report

4. TITLE AND SUBTITLE
An Automatic Placement tool for rapid prototyping of Printed Circuit Boards.

5. FUNDING NUMBERS
J-FBI-91-282

6. AUTHOR(S)
John Granacki
Tauseef kazi

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT
UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes a fully automatic placement tool for PCBs (printed circuit boards), *nap* (nonhierarchical automatic placement) that combines approaches from different placement heuristics developed both for PCB and VLSI chip placement. Although the problem of placement for a PCB can be abstractly cast into the same formalism as the problem of VLSI cell placement, a practical tool must incorporate many additional features. In this report we describe the impact of incorporating these PCB-specific features as well as other constraints imposed by the CAD environment (that is, the schematic capture system and the automatic routing tools) into an automatic placement tool. Next we discuss the selected heuristics and their associated data structures in detail. Following this discussion, we present the results of using the placement tool on two test cases along with an analysis of the tools performance. Finally, we identify the limitations of the current implementation and we propose some possible solutions and future work.

14. SUBJECT TERMS
Physical Design
Placement
Printed Circuit Board
PCB

15. NUMBER OF PAGES
20

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with [illegible] rest of the report, particularly the cover and title page. Instructions for filling in each block of [illegible] rm follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	PR	- Project
G	- Grant	TA	- Task
PE	- Program Element	WU	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities.
NASA	- See Handbook NHB 2200.2.
[illegible]	Leave blank.

Block 12b. Distribution Code.

DOD	- Leave blank.
DOE	- Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA	- Leave blank.
NTIS	- Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

An Automatic Placement Tool for the Rapid Prototyping of Printed Circuit Boards

John Granacki and Tauseef Kazi

USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292
310.822.1511

Abstract

This report describes a fully automatic placement tool for PCBs (printed circuit boards), nap (nonhierarchical automatic placement) that combines approaches from several different placement heuristics developed both for PCB and VLSI chip placement. Although the problem of placement for a PCB can be abstractly cast into the same formalism as the problem of VLSI cell placement, a practical tool must incorporate many additional features. In this report we describe the impact of incorporating these PCB-specific features as well as other constraints imposed by the CAD environment (that is, the schematic capture system and the automatic routing tools) into an automatic placement tool. Next we discuss the selected heuristics and their associated data structures in detail. Following this discussion, we present the results of using the placement tool on two test cases along with an analysis of the tools performance. Finally, we identify the limitations of the current implementation and we propose some possible solutions and future work.

1.0 Introduction

We developed the *nap* tool as part of an ARPA-sponsored project [1] to fully automate the physical design, fabrication and assembly of a PCB from a user specified netlist, parts list and board description. One of the project's objectives is to use existing commercial and university-developed software whenever possible. A commercial router that can be operated in a batch mode from a script was available; however, there was no suitable fully automatic placement tool to generate the component placement needed by the router.

Most computer-aided design tools commercially available for the physical design of printed circuit boards claim to have automatic placement capability [2]- [4]. In fact, few tools offer little more than matrix placement which is usually only useful for memory components [5]. Instead most CAD/CAE vendors emphasize interactive graphical capabilities which require the designer to place the components on the board.

The lack of emphasis on robust fully automatic placement capabilities [6] is basically driven by lack of market demand. Early attempts to automate placement caused most experienced designer to believe that they could generally produce a more compact design with less routing layers than an automatic placement tool in a reasonable amount of time. Also most automatic placement tools require the designer to modify the final placement to meet some constraints or design rules that could not be handled by the tool. Therefore, an interactive graphics editor has to be included in addition to any automatic placement capability. Since most PCB designers do not use automatic placement and prefer to use an interactive tool, the CAD vendors have had little incentive to develop a robust fully automatic placement tool. Finally, tools that did provide automatic placement capabilities for pre-1990 technology did not evolve because of lack of market demand and therefore do not meet the current PCB technology requirements.

2.0 Requirements for a Rapid Prototyping PCB Placement Tool

There are three categories of requirements: project, PCB technology and CAD environment.

The project requirements are:

1. The placement tool should be easy to interface with any CAD system for schematic capture or layout.
2. The placement tool should produce a 100% autoroutable layout.
3. The routed PCB should use the same number of signal routing layers that an experienced designer would require or in the worst case 50% more layers.
4. The placement tool should be *fully automated.*
5. The heuristics used should be proven, well documented and easy to implement.

The major requirements needed for PCB technology are:

1. arbitrary preplacement of parts,
2. two placement surfaces for parts (top and bottom),
3. special handling for decoupling capacitors and termination resistors,
4. component alignment and orientation,
5. grouping of parts,
6. assignment of additional space around a part based on package type, and
7. placement of spare components, which are usually unconnected when placed (but future connections may be known).

The project requirement for easy interfacing can also be viewed as a CAD-environment issue --- this requirement had the greatest impact on the data to be provided by the user and the data formats. To achieve the greatest flexibility for interfacing, all data required by the placement tool is stored in individual files as ASCII characters [7]. There are five required files and two optional files. The filenames and the type of data stored in each are shown in Table 1 for a design identified by the string *design-id*.If a component/part library and a package library that were CAD-system independent were available the files: *design-id*.bbx and *design-id*.ppt would not be required.

TABLE 1. Description of designer-supplied data used by *nap*

Filename	Placement Data	Required
design-id.ntl	netlist	Yes
design-id.ptl	parts list	Yes
design-id.bbx	bounding box & location of pin #1 for each package type	Yes
design-id.ppt	package type for each part type in part list	Yes
design-id.bmo	mechanical outline of board and location of special features like holes	Yes
design-id.crt	relative weighting factors to selected nets and parts (multiplies connection strength)	No
nap.ini	miscellaneous parameters and design constraints used by *nap* (e.g. grid spacing, border spacing, defaults supplied)	No

3.0 Selection of Heuristics for this Implementation

According to a recent survey article [8] heuristic algorithms for placement can generally be grouped into one of five classes: simulated annealing, force-directed placement, min-cut placement, placement by numerical optimization, and evolution-based placement. Rather than discuss each of these classes, we refer the reader to this excellent survey article and restrict our discussion to our criteria and rationale for selecting a min-cut class of heuristics.

We eliminated simulated annealing because of the long run times required, the difficulty in finding the best energy function and cooling schedule, the need for manually editing the simulated annealing placement to resolve component overlap, and our relaxed economic constraint allowing additional layers to be used in routing if necessary.

We eliminated force-directed heuristic based on previous experience trying to choose force constants that were sufficiently robust. Also, this heuristic is easier to implement for uniformly sized components and a slotted board. For the general PCB problem, component overlap usually occurs and editing is required.

We believe that numerical optimization is better suited for integrated circuit designs where the pads or exterior connections were regularly arranged on the periphery of the layout.

Most printed circuit boards that we encountered had most of the external connections at only one or two edges of the board and often had additional test connections scattered across the surface of the board. We also note that for some algorithms in this class there is no obvious approach to handling preplaced components.

Since a part/package library and detailed information concerning the location of the pins on each package were not available (at the time this tool was being developed), we selected a min-cut heuristic based on partitioning which would not require the detailed physical information for the pins on each package. The lack of package information also prohibited the implementation of gate swapping within or between packages and pin swapping on gates with multiple inputs.

Other considerations that biased our decision to use min-cut heuristics were the ability of min-cut heuristics to handle preplaced components and the short run times required for most layouts and the ability to produce 100% auto-routable designs with a reasonable number of routing layers.

4.0 Detailed Description of the Implementation

The implementation described in this report consists of three phases: an initial constructive phase based on quadrature placement using maximum conjunction/minimum disjunction [9] with additional consideration for component area included in the partitioning process, [10]. The "initial partition" *which anchors the partitions to board regions* is then used as the seed for an iterative improvement phase based on the Fiduccia-Mattheyses heuristic [11]. After the iterative improvement phase is completed, a global improvement phase based on the approach used in GORDIAN [12] is applied to the resulting partitioning. Finally, the components within the globally improved partitioning are placed as close as possible to the location assigned to each partition. The empty spaces on the plane are represented as a set of *maximum empty rectangles* and the space is managed using the method described by V. Jayakumar [13].

4.1 The Constructive Initial Partitioning Phase

In this phase, the design is bisected horizontally and vertically recursively to obtain 2^n partitions, where "n" is specified by the user. Constructive initial partitioning is employed here in order to build the initial bipartitions. The procedure for each bisection follows (the discussion of the procedure is for a vertical cut):

1. Assign a location to the partition based on whether it is a vertical or a horizontal cut (this is explained in more detail later). Based on the location of the two partitions assign the pre-placed components to the partition containing them.
2. Compute the conjunctions of all the components in the parent, with the components to the left and to the right of the cut line. Let C_{iL} be the conjunction of component i with the components to the left and C_{iR} the conjunction with the components to the right.

3. Select the child partition (either left or right) which has the smaller area of components. Let P_s be the selected partition while P_o the other one.
4. Select a component i from the parent which has the highest value for D_i, where $D_i = C_{is} - C_{io}$.
5. Move the component from the parent to the child partition P_s.
6. Re-compute C_{js} and C_{jo} for each component j in the parent which is connected to i.
7. If there are more components in the parent then go to step 3.

The partitioning phase partitions the circuit into a number of disjoint sets of components. The process involves recursive bipartitioning until the desired number of partitioning are obtained. It can be seen in Figure 1 that when the top partition (which initially contains all the components) is bipartitioned, it is cut vertically into two child partitions. These child partitions when partitioned are cut horizontally into four (2 for each) child partitions. This way the partitioning process goes on until the desired number of partitions are obtained. In general, a partition at an odd numbered level (viewing the top part as the level 1 node of the partition tree) is cut into vertical child partitions while those at even numbered levels are cut into horizontal partititions. This could be controlled by a parameter but we chose to control it by board orientation, which means the horizontal axes is the longer or the major axis of the board.

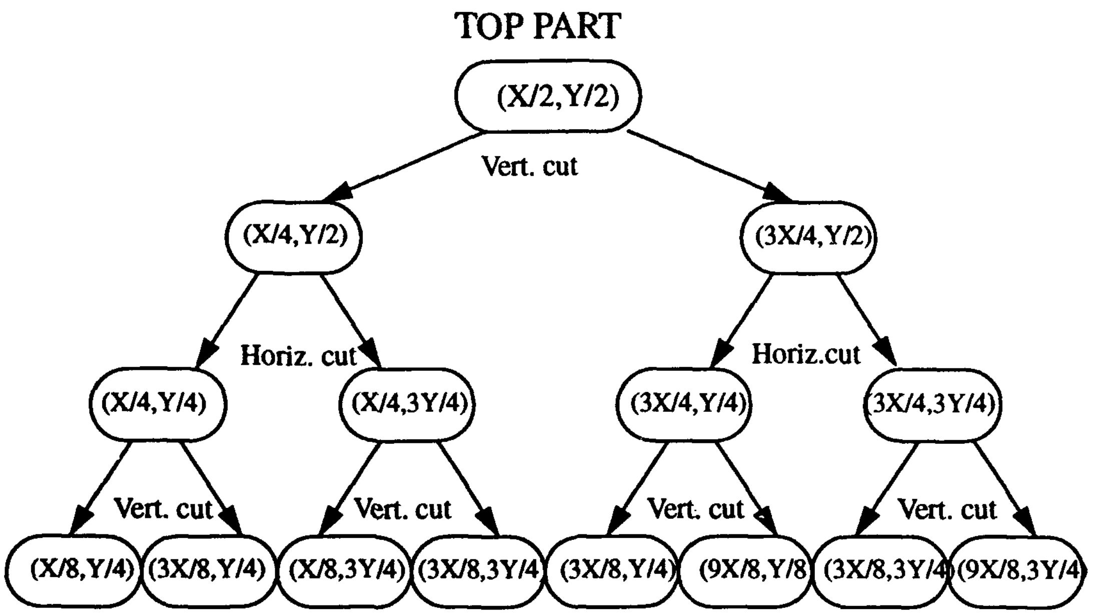

Figure 1. A graph model of the partitioned circuit.

The partitions shown in Figure 1 will occupy the regions shown in Figure 2 on the following page. It can be seen that all partitions are of a uniform size, although due to irregular sizes of components the area of components within a partition may be greater then the area assigned to it. This does not become a problem since we are not restricting the components to be placed within the boundary of their partition, rather each component is placed at a site, closest to the center of the partition, as the board gets filled, this may force the component to get placed at the opposite corner of the board. The advantages of assigning locations to partitions is that a reasonable wire length estimate can be obtained without the exact location of each component or the pins. We also know exactly which partitions contain the preplaced (fixed components like connectors, etc.) components.

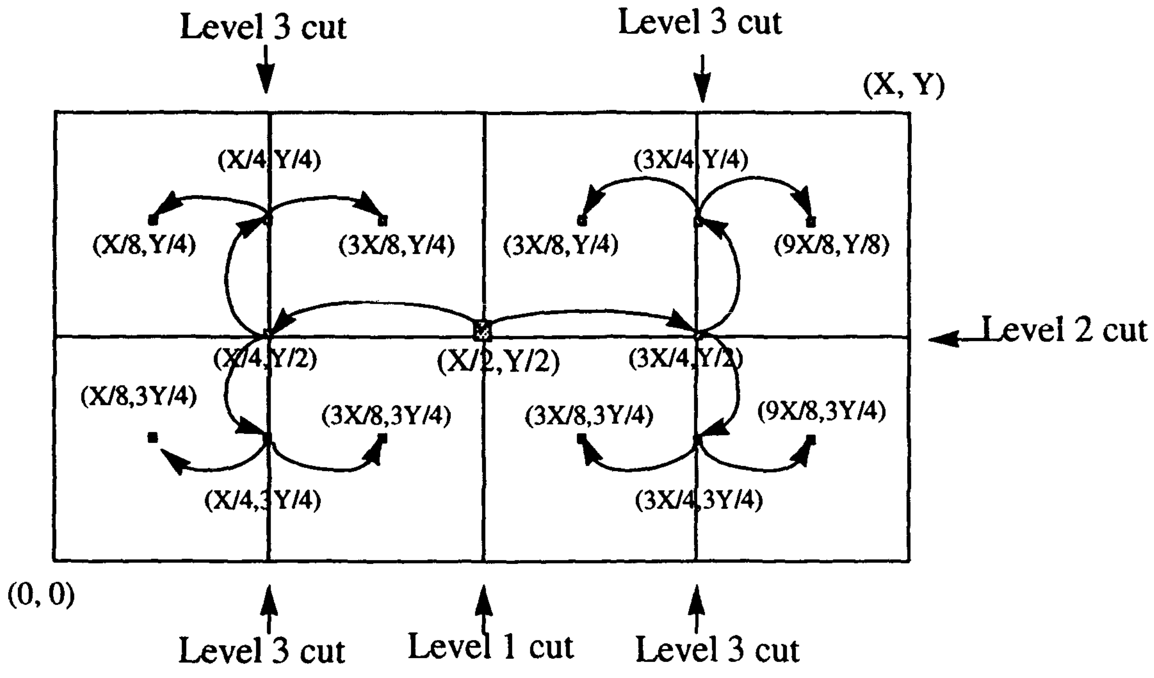

Figure 2. Relationship of the cuts and partition locations.

4.2 The Iterative Improvement Phase

Once the initial bipartitions are created they are improved using a heuristic method based on Fiduccia-Mattheyses method [11]. The method is applied on the two child partitions (left, P_L and right, P_R) obtained after partitioning the parent partition. The method used here is outlined below:

1. Compute the conjunctions of all the components in the two (left and right) partitions, with the left and right partitions. Let C_{ij} be the conjunction of a component i with partition P_j.
2. Unlock all components and save the current state of the partitions as the best state.

3. Select two components, one from each partition. The criteria for selecting a components is outlined below.
   - Select a component i from the left partition such that it is not locked and:

     $$D_i = C_{iR} - C_{iL} = MAX$$

   - Select a component j from the right partition such that it is not locked and:

     $$D_j = C_{jL} - C_{jR} = MAX$$

4. If $A_L/(A_R+A_L) < ALPHA$ (A_L and A_R are the area occupied by components in left and right partitions and $ALPHA$ is the balance factor), move the component j from P_R to P_L. If $A_R/(A_R+A_L) < ALPHA$ move the component i from P_L to P_R, otherwise move the component (i or j), which has a greater value of D.
5. Lock the component (i or j) moved. Recompute C_{kL} and C_{kR} for each component k connected to the component moved.
6. If the value of cut size (number of wires going between the two partitions) for current state of the partitions is less then the cut size for the best state, let the current state be the best state.
7. If there are any unlocked components, go to step 3.
8. Set the current state to the best state of the partitions.

The above method is applied iteratively, until there is not improvement in cutsize. This implementation does not use all of the data structures developed by Fiduccia-Mattheyses for convenience in combining the other heuristics and the heuristics for managing the empty space.

4.3 The Global Improvement Phase

Once the desired number of partitions have been obtained by the bipartitioning process, all the partitions are improved globally using a greedy procedure. Basically if moving a component from the partition it is currently in, to another partition, will reduce the total wirelength, the component is moved, provided the original partition has an area greater then the minimum required. This process is repeated for each prospective component until there is no component which improves the wirelength if moved. The process is outlined below:

1. Create an $m \times n$ matrix, where m is the number of components and n is the number of partitions. An entry w_{ij} in the matrix is the total wirelength of the board with the component i placed in partition P_j.
2. Select a component i contained in partition P_j from the design such that:
   - The area A_j of the partition P_j is greater then the minimum required.

     $A_j / A * n > ALPHA$, (A is the total board area and n the partitions.)
   - i if moved to another partition P_k would result in a reduced wirelength.

     $\exists P_k, \quad w_{ik} < w_{ij}$

- The reduction in wirelength is maximum over all the components.

  $w_{ij} - w_{ik} = MAX, \ \forall i$

3. If component i exists, move it from P_j to P_k. Re-compute w_{ix} $\forall x$ and w_{yj} and w_{yk} $\forall y$, where y is a component connected to the component i. Go to step 2.

Although the result of the global improvement phase is still a local minimum, introducing a global perspective usually produces some improvement (decrease in the estimated wirelength). This decrease has been demonstrated by running the program with different parameters on several real circuits and comparing the results obtained by running *nap* with and without the global improvement phase.

4.4 Managing the Empty Space (include as subheadings data structures?)

Placement for a single sided board is accomplished by defining a plane and associating a list of empty spaces with it. The empty spaces are represented as a set of *maximum empty rectangles* and the space is managed using the method similar to one described by V. Jayakumar [13].

The empty spaces are basically a set of overlapping rectangular regions. Initially there is one big rectangle equal to the size of the board. When placing a component, all rectangles are searched exhaustively to see which of the them can accommodate the component. From these rectangles, one is chosen that can accomodate the component at a location closest to the center of the partition containing it.

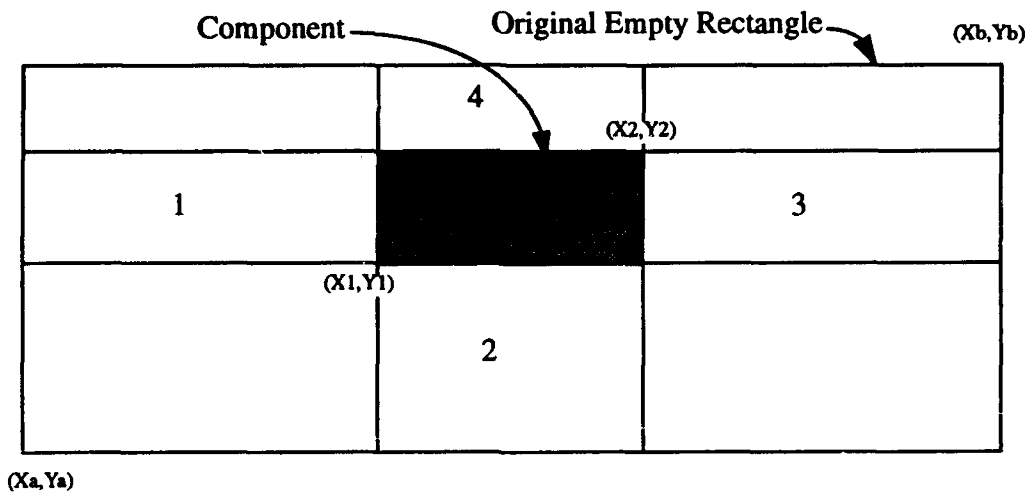

Figure 3. The empty rectangle [(Xa,Ya),(Xb,Yb)] after placement of a component, gets split into 4 smaller overlapping rectangles:

1. [(Xa,Ya),(X1,Yb)]
2. [(Xa,Ya),(Xb,Y1)]
3. [(X2,Ya),(Xb,Yb)]
4. [(Xa,Y2),(Xb,Yb)]

This location is assigned to the component and all rectangles which overlap with the component, are adjusted. Adjustment may break one rectangle into 4 smaller rectangles as shown in Figure 3. All zero dimension rectangles and those rectangles which are fully contained by others are deleted.

4.5 The Data Structures

There are two main data structures used in *nap*, one is for representing components and the other for the partitions. The data structures for components (comp_type) and for partitions (part_type) are shown in Figure 3.

```
part_type
int part_no;
int level;
int x_dim,y_dim;
int x,y;
int area;
------------------------------
comp_type *comp_list;
part_type *left,*right;
part_type *l_sib,*r_sib;
part_type *parent;
```

```
comp_type
char * name;
int locked,pre_placed;
int wire_length,;
int x,y,layer,orient;
float criticality;
int conjunction_l;
int conjunction_r;
int how_to_place_group;
comp_type*comps_in_group;
------------------------------
part_type *partition:
comp_type *next;
comp_type *next_in_part;
conn_type *connects;
```

Figure 4. The two main data structures.

The components are kept as a linear list sorted by their reference designators. Every component has a name field (which is the reference designator), fields for placement information (x, y, layer, pre_placed and orient), fields used when partitioning (locked, conjunction_l, conjunction_r, criticality, wire_length) and fields for placing components in groups (how_-to_place_groups and comps_in_groups). In addition to these fields there are a number of pointers, a pointer (partition) to the partition which contains it, a pointer (next) to the next component in the sorted linear list of all components, a pointer (connects) to the list of all the 2 pin nets containing the component and a pointer (next_in_part) to the next component that is contained by the same partition. The list of components along with their interconnections is shown in Figure 4.

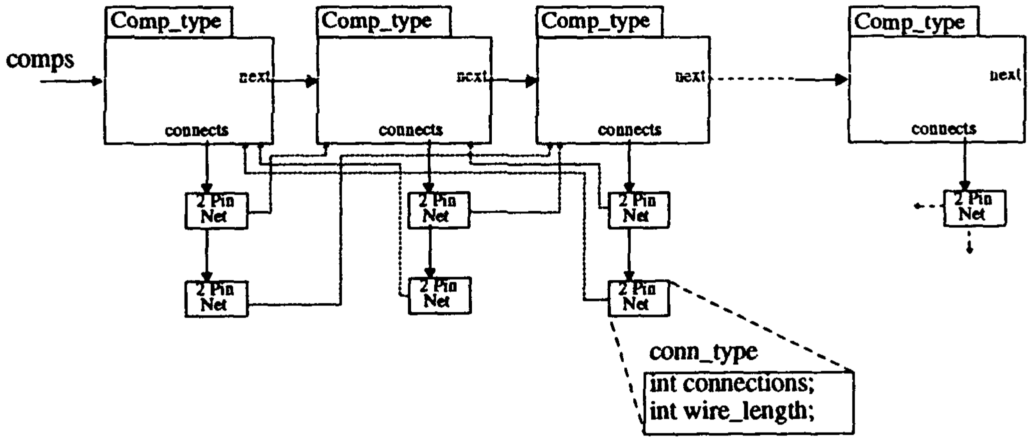

Figure 5. Linked lists of components with interconnections.

Partitions are represented in the form of a tree as shown in Figure 2. Each partition contains a part_no field (which is just a unique number assigned to identify it), a level field (which is the depth in the partition tree), x_dim and y_dim (the size of the partition), an area field (the cumulative area of the components in the partition) and fields specifying the location of the partition (x and y fields). There are 5 pointers left, right, left_sib, right_sib and parent to create the partition tree. All leaf partitions in the partition tree point to a list of components they contain. The comp_list pointer points to the first component in the list, which has its next_in_part pointer pointing to the second component and so on. This is depicted in Figure 5. Also shown in the figure are the "partition" pointers of the components pointing back to the partitions containing them.

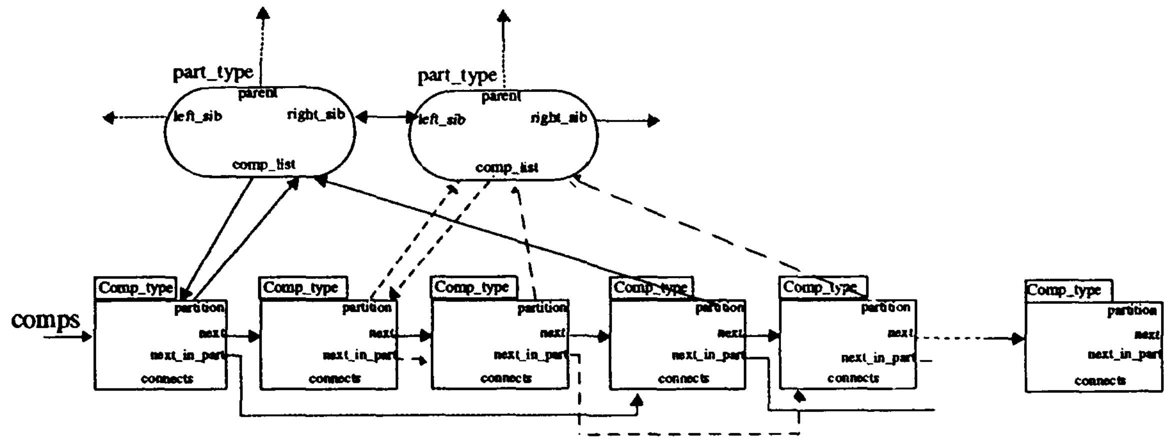

Figure 6. Relationship of partition pointers to linked list of components.

There are other data structures (shown in Figure 6) in addition to partitions and components. These include packages (pkg_type), data structure for managing empty spaces, voids, and holes on the board and data structures for managing pre-placed components. As seen in the figure these objects are arranged as linear lists of nodes containing the appropriate information. For example the node for the pre_placed_comps has a pointer pointing to the corre-

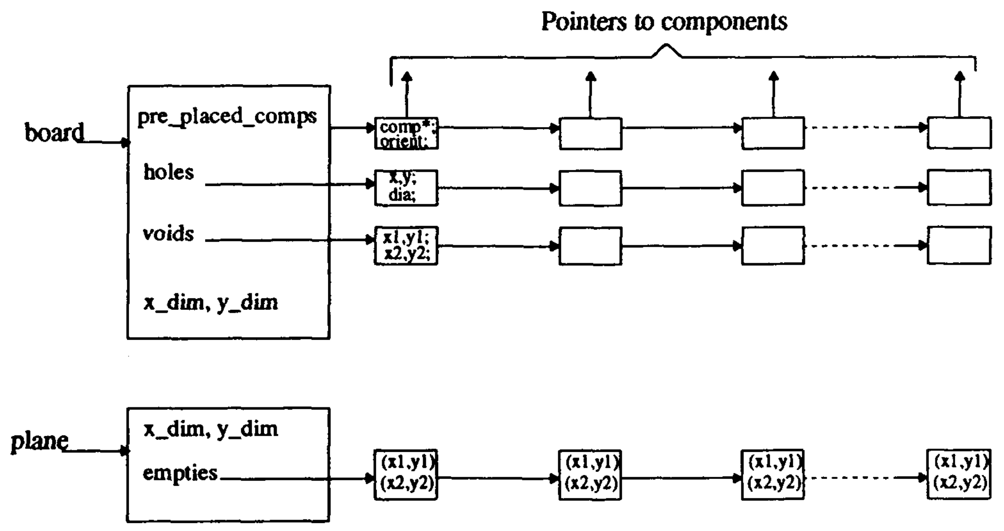

Figure 7. Data structures linking geometric objects and components to locations.

sponding comp_type object and has a field that specifies the orientation (rotation). The empty spaces (see Section 4.4 on page 8) are rectangles, represented as pairs of points (x1,y1) and (x2,y2) for the end points of the diagonal. The holes are circles with a center (x,y) and the diameter (dia). And voids are rectangles with the end points (x1,y1) and (x2,y2) of the diagonal stored in the node.

5.0 Experimental Results and Analysis

In this section, the results from a series of experiments using a single heuristic method or a combination of heuristic methods is presented. We did not randomize any arbitrary choices in our heuristics to determine the noise or scatter produced by arbitrary choices as suggested by Knapp [14].

5.1 Comparison to Breuer's mincut placement algorithm

nap uses the mincut algorithm to do the initial partitioning. It uses the constructive initial partitioning method described by Breuer [9]. *nap* does not have fixed slots but has a grid for the placement of the components. A component could be placed any where on the board and may take a number of grid cells.

Figure 7 shows the example taken from [9], containing 16 moveable components (A,B,.....,P) and two pre-placed (fixed) components (1 and 2) with zero dimension. The placement shown in Figure 7 is an optimal placement. The mincut heuristic when applied to

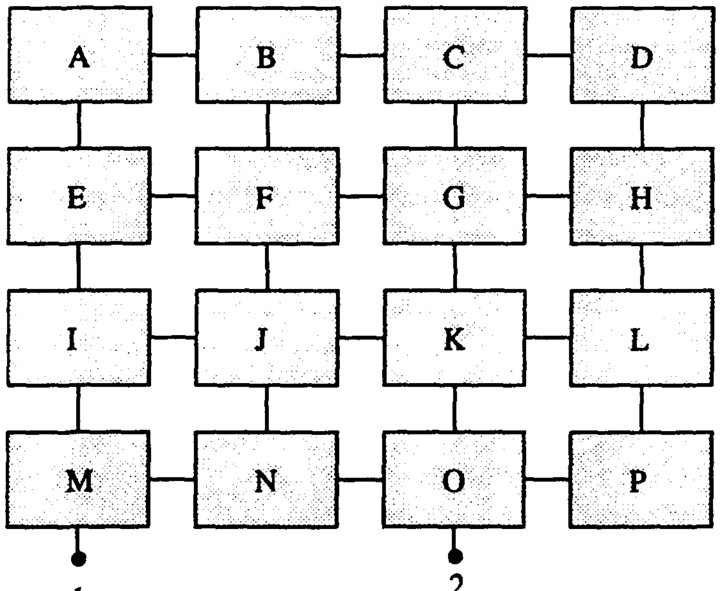

Figure 8. Placement problem with Optimal Solution.

this problem, places the components optimally and so does *nap*, provided that the connectors 1 and 2 are placed as shown above and this alphabetical ordering of the cells is used

(The actual example in Breuer's paper does not produce an optimal placement using *nap* because *nap* does not preserve the lexical ordering of the cells when it reads in the data, producing a different non-optimal initial placement).

5.2 Comparison to Fiduccia Metheyses approach

nap uses the same approach as discussed in the paper [11], but the data structures are not the same as introduced in the paper. Secondly, the notion of a net is replaced by individual connections between chips. For example a net, *N1*, connecting three components *C1, C2 and C3,* is represented as three individual connections one from *C1* to *C2*, second from *C2* to *C3* and the third from *C1* to *C3*. That is all nets are represented as two pin nets in this implementation.

The specific differences in the data structure and procedures are:

- The cells/components are represented in *nap* as a linear list sorted by their reference designators instead of an array.
- There is no array of nets, because all nets are two pin nets.
- The cell gains are re-computed every time a cell is moved from its current partition to the other partition. Gains of all the cells connected to the cell which was moved are re-computed.
- The list of cells is not sorted by their cell gains this makes *nap* less efficient than the original Fiduccia-Mattheyses heuristic.

5.3 Combining Heuristics: One Heuristic vs. Two Heuristics vs. Three Heuristics.

nap was run on two real designs. Table 2 and Table 3 give the results obtained when *nap* was run on the two designs using different parameters. The first column specifies the number of partitions the circuit was divided into, columns 2 thru 5 specify the wirelengths obtained by combining different options of *nap*. *F* specifies the use of Fiduccia-Mattheyses algorithm only, *G* signifies the use of global improvement phase only, while *FG* is the wirelength obtained by combining Fiduccia-Mattheyses with global improvement. No Options signifies only constructive initial partitioning with no improvement.

TABLE 2. Results* of running nap on "BAM"

Partitions	No Options	F	G	FG
4	1.78	1.50	1.75	1.51
8	1.59	1.32 (1.29#)	1.54	1.32 (1.37#)
16	1.36	1.13 (1.08)	1.32	1.12 (1.06)
32	1.31	1.01 (1.03)	1.22 (1.28#)	1.00 (1.00)

* Each entry represents the ratio of the wirelength for that experiment to the shortest wirelength estimated by nap. Entries in parenthesis are the ratio of the actual wirelength obtained after routing the PCB using finesse router w/ 42-16-42 routing grid to the shortest actual wirelength. circuits which routed less then 99.5% are not included here. Wirelength is estimated by Manhattan Distance for convenience. We have shown empirically that the actual wirelengths using the Finesse router track the estimated wirelength monotonically and can there for be used to select the placement with the shortest actual routed wirelength.

# The router did not finish 100%, but completed more then 99.5% connections

TABLE 3. Results of running nap on "DART"

Partitions	No Options	F	G	FG
4	1.40	1.37	1.37	1.37
8	1.21	1.13	1.20	1.13
16	1.17	1.11	1.20	1.08
32	1.13	1.05	1.11	1.05
64	1.11	1.04 (1.02)	1.00 (1.00)	1.04 (1.04)

It is clear from the tables that by applying the combination of constructive initial partitioning, a heuristic based on the Fiduccia-Mattheyses method and a global improvement heuristic, we get a better result (shorter total wirelength) than by applying each one of them individually or in combinations of two.

The wirelengths obtained from nap and the actual routing lengths obtained after running the finesse router have been plotted in Figure 8 for different placements obtained from *nap* for the BAM design. The seven points in the curve correspond to the seven entries in Table 2 which routed 100%. The figure clearly shows the monotonicity of the wirelength estimate and the actual routing length.

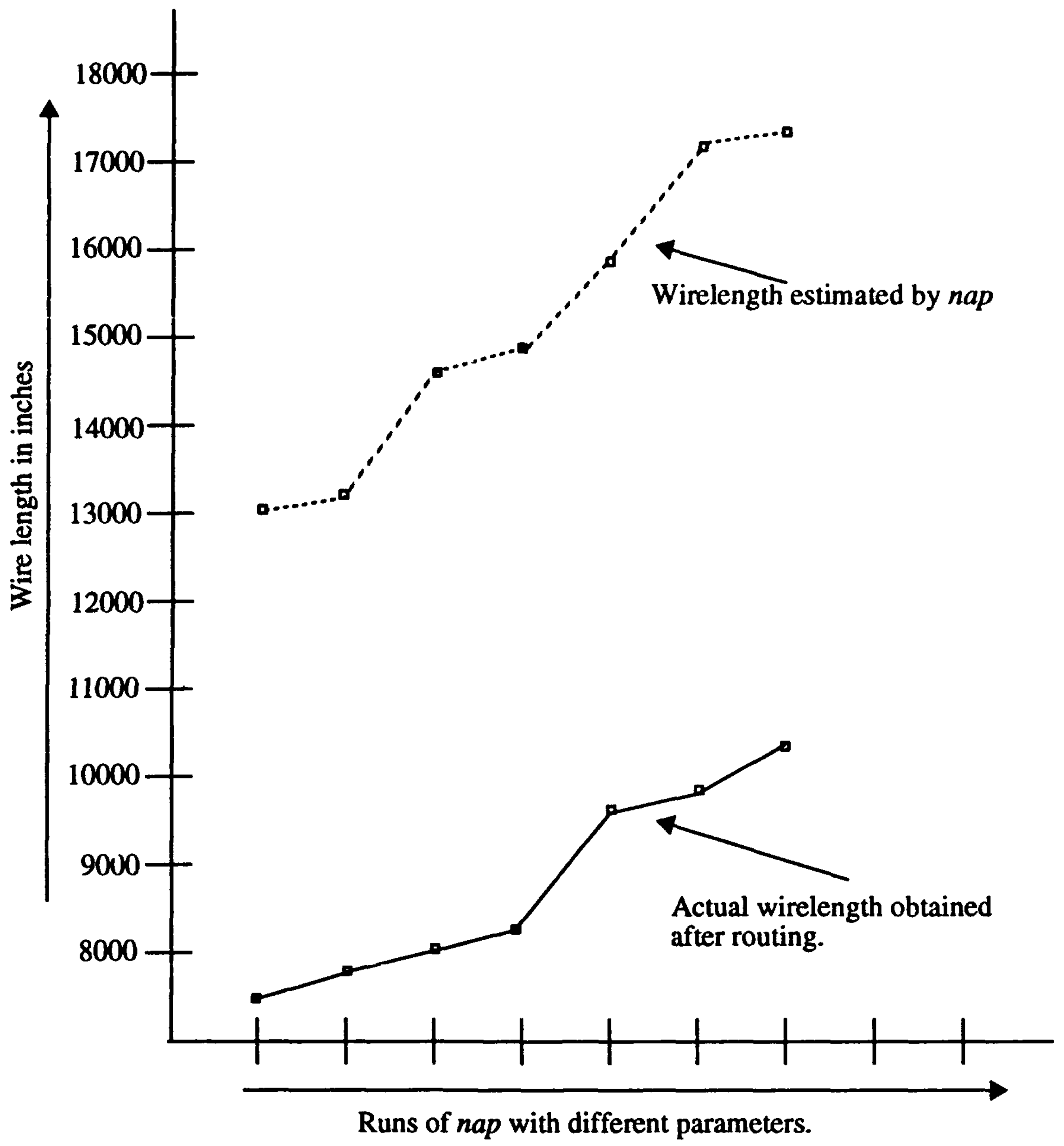

Figure 9. Comparison of actual to estimated wirelengths displaying monotonicity.

6.0 Limitations in this Implementation

1. In the current implementation, we don't have information about the pins of a component. This will have a significant affect on the quality of the layout as all pins are assumed to be located at the center of the component.
2. The wirelength prediction in the current implementation uses the manhattan distance to compute the length of wire between two components. As mentioned above the nets are broken into from-to links. The accuracy of the prediction will increase significantly once we have the pin locations. Also we can use a better method like minimum spanning tree or the stiener's tree approach once we have the net information incorporated.
3. During the partitioning phase, *nap* assumes the location of a component to be inside a partition. This may not be always true as a component may actually get placed at a location far away from the partition due to unavailability of space inside or close to the partition on the board. This will deteriorate the placement quality. This may be solved by dynamically changing the partition size during partitioning.
4. Hyperedges have been represented as cliques.

7.0 Conclusions and Future Work

The preliminary results presented here indicate that there is some decrease in estimated wirelength when two or more heuristics are used in combination. Due to the "noise" [14] caused by arbitrary choices and the limited number of experiments performed, it is not clear how the heuristics should be combined or what cost functions should be used to obtain better results, consistently over a single heuristic. Also we think that the fact that the estimated wirelength decreased when the number of partition was increased may be related to our inability to more accurately estimate wirelength based on pin location and the use of a simple Manhattan distance to estimate the wirelength. As part of the future work, we plan to try and resolve these uncertainties and to run many more experiments on different circuits.

The following subsections look at some of the open research problems that could lead to the development of a more robust placement tool for the board level-placement problem and would also leverage placement research for VLSI technologies and multi-chip modules.

7.1 Comparison of standard cell techniques, sea of gates, PCBs, etc.

A large body of the research in placement in the past has been directed at the integrated circuit problem including standard cell placement, macro placement and gate array cell/macro placement. In general, the algorithms are directly applicable but it is sometimes difficult to predict if certain characteristics of the design style will cause problems or not perform as well when used on a printed circuit board problems. For example, most standard cell designs use a row-oriented layout where the cells are abutted with each other along the row; the cells are dual-ported; there is usually no routing over the cells and the number of routing layers is limited to two or three layers --- on a PCB the row-oriented layout would map fairly well but usually the packages are not abutted; not having dual porting could result in a relatively larger overall layout and possibly would require more vias and routing layers. One of our research goals is to develop a placement framework that would allow a single algorithm to be easily applied to different technologies so the appropriateness of a given algorithm in different situations could be evaluated. This is not usually done because of the difficulty of setting up all the practical aspects of the problem that have to be included to make the comparisons/evaluations meaningful and accurate (see Section 2.0 on page 2).

7.2 Guidelines for combining cost functions/heuristics

Often a single cost function based on a single variable like wirelength is used by the placement heuristic. It would be interesting to combine together different variables in the cost function and to characterize these cost functions for circuits with different characteristics. It is possible that certain heuristics with specialized cost functions might produce better placements for certain classes of circuits. Identifying these situations would allow the layout designer to exploit this information and achieve a better placement. With a placement framework as mentioned in the previous section, it might also be possible to determine the best cost function for any given circuit empirically by trial and error.

7.3 How to best handle connectors

Connectors are always a concern in formulating the placement problem. In certain cases, for example, integrated circuit chips the location of the pads can be exploited and used as a constraint; however, the situation in the placement and location of connectors on a board is often more arbitrary, less regular and more critical. A better formulation of the general problem of how to treat fixed objects (pins) like connectors would most likely lead to better board layouts. The general problem of dealing with connectors has not been addressed in the placement literature.

7.4 Use of schematic data

There is usually a significant amount of information in a logic/circuit schematic that could be used to produce a better layout. For example, experienced circuit designers will often layout the circuit in sections, blocks, on separate schematic sheets or hierarchically. Frequently the layout will reflect the signal data flow and the signal control flow, this information could be used to orient the components [15] and place groups of components or layout the data flow path. Current PCB layout tools do not use this information. Using this information means that the placement tool must function as a floorplanning [16] tool as well as a placement tool. We plan on building a prototype tool that will use this information to place the components on a printed circuit board.

7.5 Component class information

PLEX [17] was an Expert system that attempted to mimic human designers to achieve a better placement. PLEX used a constructive approach based on reasoning but did not employ any classical CAD heuristics for placement. We think a hybrid approach would be best, that is, one that employs expert systems techniques to solve parts of the problem like grouping class of components and well known heuristics to solve the problem of where to place the group, etc.

7.6 Statistical experiments with appropriate randomization

Knapp [14] points out that due to arbitrary choices (tie breakers) most heuristics may arrive at a local minimum which is relatively high compared to other local minimums. This can be avoided by using a pseudo random number generator and a different seed to explore the range of solutions. We plan to add this to our placement framework so that it can be used routinely in testing and evaluating heuristics, combinations of heuristics and different cost functions.

7.7 Fuzzy set theory for placement

Since components are often connected to more than one additional component or groups of components, the membership in a particular group or set of components seems like it could be characterized as a fuzzy membership. Only one paper in the literature [18] has explored this approach for gate arrays. These researchers compared their results with a force-directed approach but only published limited comparisons so it is difficult to assess this approach. One interesting feature is the way the cost functions are defined. We plan on formulating a similar approach for PCBs and to run extensive experiments including Knapp's approach for examining the statistical noise.

References

[1] Granacki, John J., "Research in Information Science and Technology: Systems Assembly Core Research," Final Technical Report, USC/Information Sciences Institute, November, 1992.

[2] Spectrum staff, "Focus report: engineering software, Printed-circuit board design," *IEEE Spectrum*, vol. 27, pp. 82-85, November,1990.

[3] Losquadro, Michele (Directories Editor), "Design Guide Special: Directory of CAE Systems," *High Performance Systems*, vol. X, pp. 92-113, December, 1989.

[4] Losquadro, Michele (Directories Editor), "Design Guide Special: PCB Layou Systems," *High Performance Systems*, vol. XI, pp. 53-71, January, 1990.

[5] Khaison, Alexander and Wadland, Kenneth R., "Next Generation PCB Placement Tools," *Printed Circuit Design*, vol. 8, September, 1991

[6] Saia, Michael, "Perspectives: Placing Parts," *Printed Circuit Design*, vol. 9, March, 1992.

[7] Granacki, John J., "Printed Circuit Board Fabrication and Assembly Service: User Guide," USC/Information Sciences Institute, November, 1992.

[8] Shahookar, K., and Mazumder, P., "VLSI Cell Placement Techniques," *ACM Computing Surveys*, vol. 23, no. 2, pp. 143-220. June, 1991.

[9] Breuer, Melvin, "Min-Cut Placement," *Journal Design Automation and Fault Tolerant Computing*, vol. 1, pp. 343-362, October, 1977.

[10] Lauther, Ulrich, "A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation," *Journal of Digital Systems*, vol. IV, pp. 21-35, 1980.

[11] Fidducia, C.M. and Mattheyses, R.M., "A Linear-Time Heuristic for Improving Network Partitions," In *Proceedings 19th Design Automation Conference, pp. 175-181. ACM/IEEE June*, 1982.

[12] Klienhans, J. M., Sigl, G., Johannes, F. M., and Antreich, K. J., "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization," *IEEE Transactions on Computer-Aided Design*, vol. 10, no. 3, pp. 356-364. March, 1991.

[13] Jayakumar, V., "A Data Structure for Interactive Placement of Rectangular Objects," In *Proceedings 17th Design Automation Conference*, pp. 237-242. ACM/IEEE, June, 1980.

[14] Knapp, D.W., "Fasolt: A Program for Feedback-Driven Data Path Optimization, *IEEE Transactions on Computer-Aided Design*, vol. 11, no. 6, pp. 677-695. June, 1992.

[15] Barth, Richard, Monier, Louis and Serlet, Bertrand, "PATCHWORK: LAYOUT FROM SCHEMATIC ANNOTATIONS," In *Proceedings 25th Design Automation Conference*, pp. 250-255. ACM/IEEE, June, 1988.

[16] La Potin, David P., and Director, Stephen W., "Mason: A Global Floorplanning Approach for VLSI Design," *IEEE Transactions on Computer-Aided Design*, vol. 5, no. 4, pp. 477-489. October, 1986.

[17] Virdhagriswaran, Sankar, Levine, Sam, Fast, Scott, Pitts, Susan, "PLEX: A Knowledge Based Placement Program for Printed Wire Boards," In *Proceedings The Third Conference on Artificial Intelligence Applications*, pp. 302-305, IEEE, 1987.

[18] Razaz, M. and Gan, J., "FUZZY ALGORITHMS FOR PLACEMENT OF INTEGRATED CIRCUITS," In *Proceedings UK IT 888 Conference*, pp. 460-463, IEE, Hitchen, July 1988.