CIRCUIT PLACEMENT METHODS USING MULTIPLE EIGENVECTORS AND LINEAR PROBE TECHNIQUES

by

Jonathan Alexander Frankle

Memorandum No. UCB/ERL M87/32

13 May 1987

CIRCUIT PLACEMENT METHODS USING MULTIPLE EIGENVECTORS AND LINEAR PROBE TECHNIQUES

by

Jonathan Alexander Frankle

Memorandum No. UCB/ERL M87/32

13 May 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

CIRCUIT PLACEMENT METHODS USING MULTIPLE EIGENVECTORS AND LINEAR PROBE TECHNIQUES

by

Jonathan Alexander Frankle

Memorandum No. UCB/ERL M87/32

13 May 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

CIRCUIT PLACEMENT METHODS USING MULTIPLE EIGENVECTORS AND LINEAR PROBE TECHNIQUES

Jonathan Alexander Frankle

Ph.D. **Computer Science**

ABSTRACT

Efficient wiring of electronic circuits depends on good component placement. One formulation of the placement problem is as follows: given n legal positions, n components, and an n-by-n symmetric matrix of connections between components, assign the components to the legal positions so that the sum of squared connection distances is minimized. We transform this to an equivalent problem in which every feasible placement is represented by a point in $n-1$ dimensions, and the object is to find the point furthest from the origin. This is accomplished by expressing each placement as a weighted sum of certain eigenvectors, which contribute independently to placement cost.

It is possible to find the feasible point with maximum projection on any given direction in the transformed problem space. We call this operation a "probe". Individual probes can be used to produce good points, and iterated probes can be used to produce *sequences* of points at increasing distance from the origin. Iterated probes can work in projected spaces with increasing numbers of dimensions, with early stages focusing on the most valuable dimensions.

By upper-bounding the distance of the furthest point from the origin in k dimensions, for $k<n$, we prove better lower bounds on placement cost than those given by previous techniques. Some of our proofs use $O(\sqrt{k}(c_\varphi)^{k-1})$ random probes, where the desired tightness of the bound determines c_φ. We also describe an adaptive algorithm that precisely

determines the furthest point in a k-dimensional projection, using $O(n^{2(k-1)})$ probes in the worst case.

We generalize our approach to handle fixed components. Finally, we test several placement algorithms that use probes against an exhaustive pairwise interchange heuristic, on randomly generated test cases. For one-dimensional placement problems, our techniques are faster and produce slightly better solutions. For two-dimensional problems, the results depend on problem size. With up to about 64 components, an exact implementation of iterated probes gives the best results, but it is substantially slower than pairwise interchange. With 256 or more components, an approximate iterated probes heuristic is faster than pairwise interchange and gives equally good results. For intermediate cases we obtain the best results by using probes to generate initial placements and pairwise interchange to improve them.

Supported by Semiconductor Research Corporation,
grant SRC 82-11-008.

Richard M. Karp
April 15, 1987

Chairman of Committee: Richard M. Karp.

Acknowledgments

I am pleased to acknowledge help from many sources. My greatest debt is to my research adviser, Dick Karp. I first knew him only as "Professor Karp", in the four courses I took with him. The first of these, an introductory course on the analysis of algorithms, was tremendously inspiring. Dick's lectures combined clarity, rigor, beauty, and what is probably most important in sustaining a graduate student's interest in research: fun. The other courses offered more of the same.

Dick's talent for focusing on the essence of a problem, which makes his lectures so effective, has also made him a steady guiding force as a research adviser. He has pushed me forward patiently yet forcefully, encouraging potentially fruitful work and helping me discard bad leads. He has offered important insights at every step. Along the way he has stirred my interest in many fields that out of ignorance or fear I would otherwise never have explored.

Dick's critical readings of this work have made me a better critic of my own expository style. I hope his advice stays with me. I have learned to avoid burdening my listener with descriptions of what I am not doing. Someday I may learn when to get out of the way and let mathematics speak for itself.

Finally, I must acknowledge Dick's role in arranging funding to support this work. I am also grateful to the Semiconductor Research Corporation, which provided that funding.

Two other strong influences have been John McCann and Manuel Blum. John was my supervisor at Polaroid's Vision Research Laboratory in the four years before I began graduate study. He taught me the value of forming hypotheses and testing them quickly. The experimental side of the current work has benefited greatly from this spirit. Manuel Blum, in hiring me as his teaching assistant for introductory algorithms just after I had taken the course myself, gave me a terrific opportunity to solidify my grasp of the field.

To John Blanks I am grateful for providing the turning point in the progress of this research. On learning about his dissertation work, I realized that all the algorithms we had developed for partitioning could be adapted to the broader problem of placement. He kindly sent me a copy of his dissertation and engaged me in some brief but stimulating exchanges of ideas.

For help in producing this document I am grateful for various software tools of the local UNIX* environment: especially *emacs*, *pc*, *make*, *rdist*, *eqn*, and *troff*; and for hardware manufactured by Digital Equipment Corporation and Xerox. Mary Edmunds turned my rough sketches into attractive figures. The data plots were produced using Ed Hunter's program *ggraph*.

I thank Beresford Parlett and Michael Klass for serving on my committee and for reading versions of the thesis. Both had more suggestions than I have so far been able to pursue. Malgorzata Marek-Sadowska also read the thesis and helped keep me in touch with the practical side of current placement problems. The advice of Lorraine Hirsch helped me improve the

*UNIX is a trademark of AT&T Bell Labs.

readability of several sections.

Writing a dissertation also requires more general "life-support" and advice. I am grateful to everyone who persuaded me that working at home with a terminal and modem was essential, and to Peter Danzig and Ken Krieg who helped make the workplace at home a reality. I thank the City of San Leandro for providing a very pleasant home in the last few months. I thank David Glueck for his no-nonsense credos of positive thinking, such as "Why *shouldn't* we win this chess tournament?", and, most importantly: "Just write it up!"

I appreciate the patience and support of all my friends, and of my parents, during the times I have withdrawn from regular activities for the sake of this work. Finally, I thank my ever-supportive, loving wife, Lorraine Hirsch: her faith fed my own.

Table of Contents

Chapter 1

1. Circuit Placement: Background

This work presents new methods for placement in the physical design of electronic circuits. The main object of physical design is to transform a functional design into a physical layout. A functional design specifies components and interconnections that will behave as desired; the layout assigns positions to the components and interconnecting wires.

Layout is traditionally conducted in two phases. In *placement*, positions of the components are determined. In *routing*, paths are then specified for the wiring that interconnects the components. Both phases must conform with certain design rules prescribed by the technology being used.

A poor placement may require several times more wire than a good one; this consumes layout area and leads to wire congestion that can make routing difficult or impossible. Because good placement is critical to good layout, much effort has been invested in developing placement algorithms.

This chapter is an introduction to the placement problem. Section 1.1 summarizes the elements and goals of circuit placement, including a precise statement of the objective that we shall employ. In section 1.2 we survey the most widely used algorithmic strategies for circuit placement. Section 1.3 outlines the new approach developed in this work and describes the organization of the remaining chapters.

Sections 1.1 and 1.2 are not meant to be exhaustive reviews of the literature on circuit placement. [Preas86] gives a more complete survey of placement techniques, including 89 references. A general overview of placement and routing issues and an annotated bibliography appear in [Soukup81].

1.1. The placement problem

A mathematical formulation of the placement problem requires a measure of placement quality, and abstractions to represent components, connections, and the layout surface. Our particular formulation, and a discussion of the technologies for which it is appropriate, will be examined in section 2.1; here we review some abstractions common to many approaches. Some procedures consider technology-specific details that these abstractions suppress. But most placement algorithms, *e.g.*, those to be discussed in section 1.2, do not depend on such details.

In some layout styles, the surface where components are placed can be represented as a fixed set of discrete *slots*. A placement assigns each component to a distinct slot. For a component to function, its internal circuitry must connect with external wiring at locations on the component known as *ports*. For simplicity the problem is often formulated as if all of a component's ports were located at the center point of its assigned slot. Placements are then modeled as assignments of components to given *points*, which we call legal positions.

The functional design specifies subsets of ports, known as signal sets, that the wiring must connect into electrically common interconnection *nets*. Placements are evaluated according to how effective a wiring of these nets they allow. Defining a corresponding cost measure is the key step in treating placement as a mathematical optimization problem.

Choosing an appropriate cost function is difficult for two reasons.

1) In practice there are many conflicting goals. [Preas86, Hanan72] mention easy routability, small layout area, low crosstalk among signals, uniform heat dissipation, and maximum circuit performance.

2) Cost should be fast to compute. For example, the ideal way to compare

proposed placements might be to perform and evaluate a finished routing for each one, but this is impractically slow.

Consequently, simple functions that estimate routability are typically used to evaluate placement quality. Some are based on estimating the length of wire required to connect the nets. One commonly used approximation to the wire length of a net is half the perimeter of the smallest rectangle that contains the components of the net. The minimum length of a tree that connects the component positions is a more accurate approximation.

The weighted sum of *squared* connection distances has also been used as a measure of placement cost [Cheng84, Blanks85b]. We adopt this measure in the current work. Specifically, we assume that we are given entries c_{ij} representing the number of connections between components i and j, and our goal is to minimize

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}((x[i]-x[j])^2+(y[i]-y[j])^2), \tag{1.1}$$

where $(x[i],y[i])$ is the position given to component i. The squared-distance metric penalizes long connections; it also approximates signal propagation time along wires. These and other of its properties are elaborated in section 2.1.3.

Measures based on wire length do not account for interactions among the nets. They therefore do little to discourage unbalanced distributions of wires, which can make routing difficult. Excessive demand for wiring in specific areas is known as *congestion*. An example of a metric that is sensitive to congestion is the number of nets that cross a given straight line through the layout (a "cutline"). One version of the placement problem aims to minimize the maximum number of nets that cross any cutline.

There are thus many ways to define the placement problem. Even the simplest formulations lead to very difficult problems. For instance, we show in section 2.1 that to minimize cost (1.1) is an *NP-hard* problem; [Sahni80] proves that several other formulations of placement also belong to this class. NP-hardness implies that no known algorithm can solve these problems in time bounded by any polynomial function of the input size; and that it is highly unlikely that an algorithm that does so exists. (See [Karp75], [Garey79] for detailed discussions of *NP* and related problem classes.) Because modern circuits can have thousands of components, it is thus not feasible to solve most placement problems exactly.

Circuit designers must therefore rely on heuristic procedures for placement. Such procedures do not guarantee optimal solutions; instead, they generate one or more fairly good placements, reasonably quickly. The next section reviews some of the most widely used placement heuristics.

It is difficult to compare the effectiveness of these algorithms. They are often developed and tested on different sets of problem instances. Furthermore, different algorithms often aim to minimize different objective functions; obviously it makes little sense to evaluate an algorithm according to an objective other than the one that it was designed to optimize.

In this respect, cost measure (1.1) offers an important advantage: [Blanks85a,b] observed that it facilitates proofs of good lower bounds on placement cost for given circuits. We can use lower bounds to show in some cases that heuristic placements are close to optimal, with respect to the chosen cost measure.

1.2. Review of placement techniques

Placement algorithms are generally divided into *constructive* and *iterative*. Constructive algorithms start with few if any components in assigned positions, and produce an initial placement. Iterative algorithms take a complete placement as input and produce a new, improved placement. Not every algorithm is purely constructive or purely iterative, and most placement systems use both kinds of procedures. Still, the distinction is reasonable, as most procedures emphasize one mode or the other.

1.2.1. Constructive algorithms

We review constructive algorithms based on three ideas: cluster growth, partitioning, and constraint relaxation.

1.2.1.1. Cluster growth

Cluster growth begins with only a few components placed; these may be thought of as "seeds". New components are introduced into the placement, one at a time, at positions determined by their relation to the components already placed. Various rules have been used to control the order in which new components are selected: a few of these are summarized in [Hanan72].

Although clustering methods are easy to implement, they are currently losing favor. One reason is that the final placements are heavily influenced by the first components that are placed. Since the early decisions are based on incomplete information, they can easily lead to placements that are far from optimal.

1.2.1.2. Partitioning

From the bottom-up approach represented by cluster growth, we now turn to a top-down approach that uses successive partitioning of the components. The idea is to divide the components into groups that will be assigned to separate regions of the layout, so that the number of connections between different groups is kept small. The most common application of this idea works by recursive *bisections*, *i.e.*, cuts into two groups. For example, the first cut might decide which components to place on the left and right sides of the layout; then the components on each side would be separated into top and bottom halves, etc.

The motivation for partitioning is clear. Since routing trouble from excessive wire crossings is most likely to be associated with cutlines through the center, it makes sense to try to reduce connections across the middle. Ever since the partitioning heuristic in [Kernighan70] became popular, many placement algorithms have employed partitioning.

Before describing this and other techniques, we note that partitioning is itself a difficult problem.* In fact, no practical algorithm is known to find a balanced 2-way partition in which the number of connections between the two sides is guaranteed to be within any constant factor of the minimum achievable number. Thus while it is possible for partitioning techniques to perform well in various situations, the practice of calling them "min-cut" heuristics is misleading, since none of them reliably achieves this goal.

We now outline the procedure of [Kernighan70] for improving a partition. The number of connections between each pair of n nodes is given. An

*Section 2.1.4 states a particularly simple form of the problem ("graph partition") that is NP-complete.

initial partition is given into two sets, each with $n/2$ nodes. The cost is the total number of connections between the two sets. The procedure performs $n/2$ exchanges of nodes between the sets, in the course of which all nodes will switch sides. For each exchange, the pair of nodes not previously moved whose exchange will produce the lowest-cost partition is selected. (Some exchanges can increase cost.) Among the $n/2$ configurations in this sequence, the procedure returns to the one that achieved the minimum cost. If the new partition is better than the initial one, then a new pass is begun with the new one.

[Fiduccia82] extends the above technique to partition circuits that include multi-component nets. Cost is evaluated as the number of nets that have components on different sides of the partition. At each step they find the best component to move instead of the best pair to exchange. By using efficient data structures to avoid unnecessary updates of the payoff function for component moves, a pass that moves all components can be performed in time that is a linear function of the size of the circuit description.

[Dunlop85] shows how to produce better placements when using the approach of recursive partitioning into halves, quarters, eighths, etc. The idea, called *terminal propagation*, allows choices made in partitioning components of a given subregion to influence subsequent partitioning in adjacent subregions. Suppose each component has been committed to the left or right side, and components on the left have been divided into a top and bottom half. The right side is not partitioned independently: if a component on the right belongs to a net that includes components on the top left, a penalty is introduced for placing that component in the bottom half of the right side. In a small experimental study, [Hartoog86] reported that a simplified form of terminal propagation was the most effective of several

placement procedures.

[Barnes82a] considers the problem of partitioning an n-node graph into k blocks of specified sizes. Let A be the adjacency matrix of the graph, and represent each partition by an n-by-k matrix X in which x_{ij} equals 1 if node i belongs to block j and 0 otherwise. Barnes shows that the problem has an equivalent restatement: Find the partition X for which XX^T is the closest approximation to A.

If X could be chosen so that its k columns were proportional (respectively) to the first k eigenvectors of A, it would represent an optimal partition. This motivates Barnes to search for a partition in which the columns of X approximate these eigenvectors. The desired approximation is provided by a solution to a linear programming transportation problem. It is worth noting that this solution optimizes closeness in approximating the eigenvectors, whereas the partition cost is actually a function of the closeness in approximating A.

1.2.1.3. Constraint relaxation

We now review a class of algorithms that operate in two characteristic phases. The first phase relaxes the feasibility constraints associated with discrete legal positions: *i.e.*, components are allowed to occupy other positions. Cost is minimized among all placements in a set that includes the feasible ones. Usually placements in the extended set satisfy certain basic properties of feasible placements, *e.g.*, have the same *sum* of component positions.

Once the legal-position constraints are relaxed, the resulting problem can be solved efficiently with optimization techniques such as gradient projection. This yields what we call a "relaxed placement", which typically

features an unacceptable distribution of components. The second phase maps the output of the first phase to a legal placement.

This approach appears in many forms. [Quinn79] suggests that the components and connections be modeled as particles and springs, with the optimum configuration characterized by a net force of zero on every particle. A constant repulsive force between unconnected components is introduced to avoid the equilibrium configuration in which all components collapse to a single point. The paper states that this force "is the single most critical parameter of the system." In the second phase their procedure moves the components onto legal positions by solving a linear assignment problem to minimize the total of the squared distances traveled by all components.

The first phase in [Cheng84] finds the configuration that minimizes cost among all placements that give the components positions whose sum equals the sum of the legal positions. The result would be a placement in which all components are placed at the center of gravity of the legal positions, except for the effects of explicit constraints that fix some components at predetermined locations. Thus if the problem statement does not include such constraints, this approach must introduce some arbitrarily.

The second phase produces an assignment of components to legal positions by a sequence of partitions. First the $n/2$ left-half components and $n/2$ right-half components are committed to the corresponding halves of the layout; then the upper $n/4$ components on each side are committed to upper quadrants, etc.

Before each partitioning step, three repositioning passes along the partitioning direction are applied to the components of the region. Each pass rescales the positions of a small fraction of the region's components, locks these components into their new locations, and then reoptimizes the

positions of the other components in the region. *Rescaling* applies an affine map to the affected components so that the spacing proportions between them are preserved, while the center and spread are adjusted to match those of corresponding legal positions. *Reoptimizing* mimics the first-phase relaxation step, except that only the nonfixed components of the current region are allowed to move.

In all cases, components outside the region are kept fixed. For definiteness, our summary of the three passes will assume that the partitioning direction is left to right, and that the fraction of "locked" components (a user-specified parameter in the range [0,.5]) equals 15%.

1) Positions of the leftmost 15% components are rescaled and fixed, and the other component positions are reoptimized.

2) Positions of the rightmost 15% components are rescaled and fixed, and the others reoptimized.

3) Positions of the leftmost components are rescaled once more, the leftmost and rightmost 15% components are fixed, and the remainder are reoptimized.

[Blanks85b] also uses metric (1.1). As in the above approaches, the first phase yields a placement that does not assign components to legal positions. Instead, cost is minimized among placements that agree with the legal positions in the sums and sums-of-squares of the x coordinates and y coordinates, and in the inner product of x and y. As is shown in [Hall70], the optimal x and y vectors under these constraints are given by two eigenvectors of a slightly modified connection matrix. In his second phase, Blanks uses component interchanges (an iterative technique discussed in the next section). Interchanges are first used to convert an arbitrary placement on the legal positions into a legal placement that is as similar as possible to

the eigenvector solution; additional interchanges are used to reduce cost.

The relaxed placement phase in [Just86] uses Lagrange multipliers to enforce the correct sum and sum-of-squares of component positions. In the second phase, components are transported from the relaxed positions to "buckets" defined within columns of the placement plane. The approach allows components of different heights; it associates each component with a number of units of supply according to its height. The cost of transporting each unit is given by the squared distance from the component to the bucket, and the objective is to minimize total cost.

There is no guarantee that the optimal solution will transport all units of a component to the same bucket, so some post-processing is required. First each component is assigned entirely to the bucket that contains the largest number of its units. Finally, the components in each bucket are ordered according to their original vertical positions in the relaxed placement.

[Otten82] uses a two-phase approach for the initial allocation of high-level modules in "floorplanning". Each module can contain several components. The modules can have different sizes, and the designer has some control over the shape of each one. Otten's goal is to arrange the modules into a compact layout. Since the formalism of legal positions is not so suitable here, he takes another approach.

The first stage depends only on interconnection nets, not module sizes. From the net lists, Otten derives *desired* distances between center points for each pair of modules; the goal of his relaxed placement is to place the modules in the plane so that these distances are approximated as closely as possible. The matrix of desired distances determines what he calls the *Schoenberg* matrix S.

If S has rank r, there is a configuration of the modules in r-dimensional Euclidean space that realizes all the desired distances: in each dimension, the module coordinates are prescribed by one of the r eigenvectors of S. Generally r is much greater than 2. Since the modules must actually be placed in 2 dimensions, not all the distances can equal the goals. The relaxed placement sets the two-dimensional coordinates according to the two leading eigenvectors of S; this minimizes the discrepancy between the original matrix S and the Schoenberg matrix given by the actual distances.

In the second phase, a rectangle with area at least equal to the sum of the module areas is carved into regions by a sequence of vertical and horizontal "slicing lines". At each step the modules of a region are partitioned among child regions that share the shorter dimension of their parent. Along the other dimension, lengths of the child regions are made proportional to the total areas of their modules. Eventually the modules are all assigned to distinct regions. The basic approach assumes that each module is flexible enough to be fit into any rectangle that has the required area, although more realistic models are also mentioned.

The relative positions of modules in the relaxed placement determine which partitions are allowed in the second phase. For example, $m-1$ vertical slicing lines are possible in a region with m modules; each line corresponds to one breakpoint in the horizontal ordering of these modules in the relaxed placement.

The process for deciding which partitioning lines in a region to accept at a given stage also uses the relaxed positions. The region's modules are represented as squares centered at their assigned positions in the relaxed placement. With all ratios between module areas preserved, the squares are first made large enough so that every pair overlaps, and then

simultaneously shrunk. At some point in the shrinking process, the modules become separable into two sets by a line that intersects no squares. The slicing line corresponding to that partition of modules is accepted first; further lines are considered in the order in which such partitions become possible during continued shrinking. A region is allowed to accept a limited number of parallel slicing lines in any stage. Otherwise, the likely result would be modules with shapes that deviate too far from being square.

We have examined five different approaches to placement that use constraint relaxation. These represent only a small sample of the possible algorithms of this type. Any technique for defining and solving a relaxed placement problem could be followed by any of the second-phase heuristics. Combinations of heuristics, that apply one after another or mix features of different approaches, are also possible.

1.2.2. Iterative algorithms

In iterative placement the basic step generates a proposed placement that is slightly different from the current one, and accepts the proposed change under specified conditions. The most common change considered is an *exchange*, in which two components trade places. We divide iterative algorithms into two classes, depending on whether changes that increase cost are ever accepted.

1.2.2.1. Pure improvement procedures

We first consider the general form of improvement algorithms based on exchanges. An exchange of components is proposed, and if it decreases cost it is accepted. When no further exchanges can decrease cost, we say that a local optimum has been reached. At this point the algorithm stops.

If the effect of an exchange on the cost function is time-consuming to compute, time constraints may make it necessary to select proposed exchanges carefully, and to stop before a local optimum is reached. [Blanks85b] showed that cost metric (1.1) makes it easy to compute the incremental cost of any proposed exchange and to perform updates when exchanges are accepted.

It is thus feasible to perform what Blanks refers to as "exhaustive pairwise interchange" on very large circuits. In each pass of this procedure, every pair of components is considered for exchange, and all exchanges that decrease cost are accepted. Passes continue until one in which no pair is accepted.

[Goto81] is representative of techniques that allow more complicated improvement steps, such as the following: component A takes B's place, B takes C's place, C takes D's place, and D takes A's place. The length of cycle allowed, sometimes denoted λ, determines the number of available changes in a given placement. (In the above example, $\lambda=4$.) This number, which we can think of as the size of the "neighborhood" of possible moves, is roughly proportional to n^{λ}. It is almost never practical to use $\lambda>2$ in exhaustive improvement procedures.

Goto makes effective use of longer cycles by selecting moves from a very restricted set. In each cycle, every component but the last must move near to the location that would minimize cost, assuming all other components stay fixed. Complicated moves may be useful, even if the resulting neighborhoods become too large in practice for moves to continue until a local optimum is reached.

1.2.2.2. Simulated annealing

Among iterative algorithms in which steps that increase cost are allowed, the most popular use an approach known as simulated annealing. The idea, as introduced in [Kirkpatrick83], exploits an analogy between the problems of minimizing cost in combinatorial problems and of producing low-energy states in materials. Accepting only those changes that decrease cost is analogized to rapid quenching from high to low temperature. Pure improvement algorithms get stuck at local optima, just as quenching yields suboptimal material structures. The lowest energies are achieved only if temperature is decreased gradually according to an annealing schedule that keeps the material in equilibrium.

An algorithm in [Metropolis53] simulates the equilibrium behavior of a collection of atoms by proposing at each step a small move that would result in a change in energy of ΔE. Moves with $\Delta E \leq 0$ are always accepted; moves with $\Delta E > 0$ are accepted with probability $e^{-\Delta E/T}$, where T is the temperature. Combinatorial optimization by annealing proceeds in a similar fashion. Given an initial configuration, specified numbers of moves are attempted at several decreasing values of a control parameter T: cost is substituted for energy in the above acceptance rule.

Annealing offers a general algorithmic template that has been applied to diverse optimization problems, including circuit placement. An implementation that has been tested on numerous industrial circuits is reported in [Sechen86]. This package employs annealing algorithms that consider an unusual range of details. Proposed moves include component displacements, exchanges, and orientation changes. Costs assess net perimeter lengths, component overlaps, row imbalance, and wiring track requirements.

Although simulated annealing has achieved encouraging results in placement and related design problems, the approach has drawbacks. Improved results often come at the expense of very long computation times. Furthermore, there is no sound theoretical justification for applying the specific annealing formulation. In placement and partitioning experiments, [Nahar85] and [Nahar86] found that simulated annealing performed no better than other methods that allow cost-increasing moves. They favor a procedure that accepts bad moves only when the last several attempts to find a good one have failed. Annealing may prove more important for indicating the value of accepting bad moves to escape local optima than for the details of the physical analogy.

1.3. Outline of new approach

The current work takes a new approach to the placement problem. We use linear algebra to express the problem in a transformed geometric setting. Each possible placement is represented by a point in a multi-dimensional space, in such a way that the optimal placement is transformed to the furthest point from the origin.*

This transformation allows us to take advantage of a new tool, called the probe, for finding good placements and for proving lower bounds on placement cost. The probe procedure takes any given direction in the transformed space and produces the feasible placement whose point has the maximum projection on that direction. We present a variety of algorithms and lower-bound methods based on probes.

---

*This approach was first reported in [Frankle86].

The value of each dimension of the transformed problem space can be precisely quantified in a natural way according to its potential to contribute to the objective function. The best use of probes is to concentrate them in directions spanned by the most valuable dimensions. The projections of the transformed points in the best dimensions supply a systematically derived approximation to the placement problem, in any fixed number of dimensions. Such approximations enable us to apply multi-dimensional search techniques to the placement problem.

In some respects the work on partitioning in [Barnes82a,b; Barnes84] is a precursor of our approach to placement. Each approach uses eigenvectors and eigenvalues to produce solutions and lower bounds on cost, and each depends on linear optimization techniques.

There are significant differences. Our approach uses more general combinations of eigenvectors. We consider entire *placements* (assignments of n components to n legal positions), not just partitions into k sets. Note finally that our placement problem can represent k-way partitioning as a special case, by specifying multiple legal positions at each of k locations.*

It is also interesting to compare our approach with the algorithms based on constraint relaxation (section 1.2.1.3). The relaxed placement phase effectively produces a two-dimensional approximation to our transformed problem. The mapping to a feasible placement is like a single probe. (Some methods perform exactly the same function as a probe; others

---

*Components assigned to the same location compose one set of the partition. No cost is charged for connections within such a set, since the squared distance is zero. We can choose k locations in k-1 −dimensional space so that the Euclidean distance between the locations of any two sets is the same. Higher-dimensional placement is discussed in section 6.5.

differ slightly.)

Constraint relaxation produces a problem formulation for which an optimal *relaxed* placement can be found. The trouble is that this solution can be very far from the *feasible* placement that it leads to in phase 2, which consequently can be far from optimal among feasible placements.

If we had to stake everything on a single probe, the direction corresponding to an optimal relaxed placement would be the right choice. But we face no such limitation. In fact the span of the most valuable dimensions comprises a wealth of probe directions, each of which corresponds to a promising relaxed placement. Although these relaxed placements cost more than the optimal one, many are typically more nearly feasible. It is quite possible for these relaxed placements to lead to feasible placements that cost less than the one determined by the "optimal" relaxed placement.

The organization of the remaining chapters is as follows. The next four chapters develop our methods for the special case of one-dimensional placement (in which all legal positions lie on a straight line). Chapter 2 gives the furthest-point transformation and the probe procedure. In chapter 3 we present placement heuristics that use sequences of probes. Chapter 4 describes how to use probes to derive lower bounds on the placement cost of a given circuit. In the course of chapter 4 we develop and analyze a polynomial-time algorithm to find the furthest point from the origin in projections of our transformed problem into any fixed number of dimensions. In chapter 5 we treat multi-component nets and specially constrained components.

Chapter 6 shows how the techniques of chapters 2 through 5 are generalized to handle two-dimensional placement. Experimental results appear

in chapter 7 and conclusions in chapter 8.

Chapter 2

2. Transformation of the Circuit Placement Problem

Our problem is to assign the components of a given circuit to a fixed set of positions so that the total of squared connection distances is minimized. This chapter presents a problem transformation and develops our main procedure for taking advantage of the transformation. In section 2.1 we present the precise problem formulation and examine several of its features in detail. Section 2.2 shows how to restate any instance of the problem in terms of the eigenvectors and eigenvalues of a certain matrix. In section 2.3 we recast the problem in a transformed geometric setting, as a search in a discrete set of points for the one at maximum distance from the origin. Section 2.4 describes the "probe", a procedure for locating good points in the transformed setting. Most of our new methods are based on probes.

2.1. Problem statement

A problem instance consists of n circuit components and n legal positions in the (x,y) plane. The only information about the components is a symmetric matrix C in which c_{ij} is the number of connections between component i and component j. We seek a one-to-one assignment of components to legal positions that minimizes

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}((x[i]-x[j])^2+(y[i]-y[j])^2),$$

where $(x[i],y[i])$ represents the position given to component i.

To clarify the strengths and weaknesses of this model of the circuit placement problem, we examine the assumptions imposed by adopting this formulation of connection matrix, legal positions, and cost function.

2.1.1. The connection matrix

A circuit is defined by listing its components and specifying the necessary connections between them. For a given circuit, our goal is to find a feasible placement of the components that makes routing easy. Routability is difficult to measure, so instead we take an approach that designers have found works reasonably well: we define a cost by summing contributions from each connection. Reasonable abstractions of the placement problem must make this sort of simplification.

It is convenient to express cost as a sum of contributions from all pairs of components, with contributions of zero from pairs that are not connected. This motivates us to use n-by-n matrices C to represent circuits with n components. Entry c_{ij} in C represents the connection between components i and j, and each c_{ij} will be a multiplicative factor in one term of the cost summation.

In the simplest matrix representation of a circuit, entry c_{ij} equals 1 if components i and j are connected and 0 if they are not. Some features of matrix representations of circuits are apparent even in this simple case. Most entries in the connection matrix tend to be zeros, because in typical circuits most pairs of components are not connected. The diagonal entries c_{ii} equal zero, since we are not concerned with internal connections.

An obvious refinement of the above scheme is to let different types of connections be represented by matrix entries with different values. A value may have a precise physical correlate; *e.g.*, the number of wires connecting two components on a printed circuit board. In general, the value c_{ij} indicates the importance of the connections between components i and j. Each value acts as a weighting factor for the contribution that the associated pair adds to the total cost. We can assign larger values to special pairs, *e.g.*,

those with timing-critical connections, to give them greater weight.

The signals directed from i to j and from j to i might have different weights, say w_{ij} and w_{ji}. We assume that in the placement cost summation, these values should both be multiplied by a function of the positions given to components i and j. Total cost would be the same if instead of using w_{ij} and w_{ji} in C, we set both c_{ij} and c_{ji} to $\frac{1}{2}(w_{ij}+w_{ji})$. We can thus restrict attention to *symmetric* connection matrices.

In typical circuits, some signals (or "nets") are passed among more than two components. Such nets offer us a choice of connections, where a "connection" is a pair of components that a signal can pass between without going through other components.

It would be straightforward to represent a net in the C matrix if we knew its connections: we would add weight to precisely the entries of C that correspond to these pairs. But usually a net will function properly with any connections that provide a path between any two of its components. For example, a net with s components has $s\cdot(s-1)/2$ pairs; any $s-1$ connections that form a spanning tree are typically sufficient.*

In practice, connection choices are postponed so they can suit the placement. Choosing particular connections in advance can bias the evaluation of placement costs (see Figure 2-1).

---

*In reality, a list of $s-1$ connections would still be only an approximation, because wiring is typically accomplished by *Steiner trees*, which can make use of additional connection points to reduce total wire length.

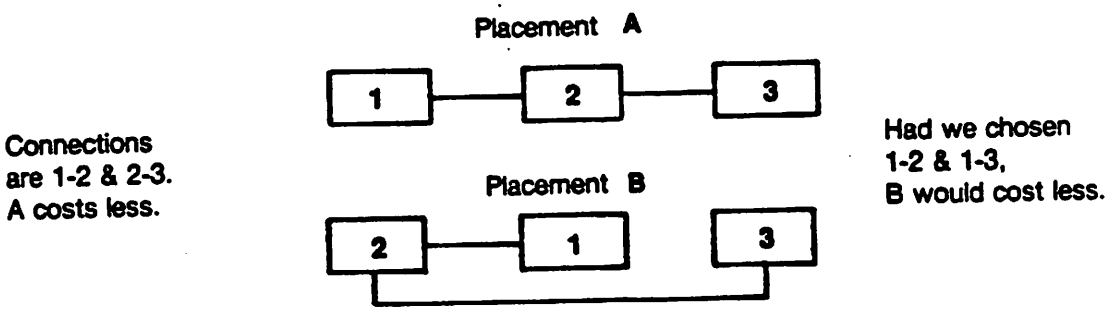


Figure 2-1. Choosing connections biases placement cost evaluation.

We want to represent nets without biasing the placements. For every pair of components i, j in a net, we therefore add the same value to c_{ij}. In nets with more components, a smaller fraction of the pairs will actually become connections, so the value we use ought to be a decreasing function of the number of components in the net. In section 5.1 we determine an appropriate function.

2.1.2. Legal positions and feasible placements

We define feasible placements in terms of n (x,y) points, termed "legal positions", which must be specified in any instance of the problem. In the basic problem, a feasible placement is one that assigns each of the n components to a different legal position.

We represent a placement as an n-by-2 matrix X, in which the entries in row i are the x and y coordinates selected for the ith component. We refer to the columns of X as vectors x and y, so that $(x[i],y[i])$ is the position of component i. If L is an n-by-2 matrix whose rows are the legal positions in some arbitrary order, then a *feasible* placement X is any permutation of the n rows of L. We can thus write the feasibility requirement as follows:

$$X \in \{\Pi L: \Pi \text{ is a permutation matrix}\}$$

(A permutation matrix is one in which every row and every column has

$n-1$ zeros and one 1.)

We view the interconnections as our primary concern and the individual features of specific components as secondary complications. Thus we take the position of a component to be the position of its center, and neglect such geometric aspects as its size, shape, orientation, and where on its perimeter the various signals appear.

A usable circuit placement will ultimately specify an arrangement of the components in complete detail. Permutations of legal positions are intended to capture the essential combinatorial difficulty of the arrangement problem in an easy to manipulate form. It seems necessary to omit details such as possible compaction of component and wire positions. A formulation that took into account detailed component shapes, routing area variations, and the corresponding constraints on placement would be unworkably complicated mathematically.

How useful is our model of feasibility? It adequately represents the core of most placement problems, but for some technologies it does not do the whole job of specifying usable placements. In the rest of this section we indicate situations for which the basic model is well suited, and discuss ways to extend the range of its practical applications.

A formulation based on fixed legal positions is most appropriate when component sizes and shapes are uniform. In this case, rearranging components among the legal positions can never cause them to overlap. Of present-day technologies, gate arrays come closest to having uniform components.

Gate-array designs employ clusters of universal logic elements, or "gates", that are laid out in an array on a regular grid. A few standard sizes of array are mass-produced. To implement a particular circuit, one

selects an array of adequate size and maps the required components onto its gates, the positions of which are literally fixed in advance. A masking step in the manufacturing process adapts the array to a given design by enabling the specific connections required. The representation of feasible placements as permutations appears well suited to gate arrays.

In one respect our model is too restrictive. We ask for exactly n legal positions, when in practice more than n are usually available. The formulation could allow a finite number of surplus positions. However, the convenience for the model of having exactly n positions specified seems worth the price of arbitrarily requiring that certain positions be selected and others ignored. We assume that any reasonable choice of n close-packed positions (*e.g.*, filling a shape as close to square as possible) will allow solutions nearly as good as the best we could obtain with an optimal selection.

In most technologies, the components are not strictly uniform in size and shape. For instance, in printed-circuit-board layout, the components are packaged in a few different sizes. If the packages are placed close together, small packages can occupy positions where larger ones might not fit. We could insure the feasibility of arbitrary component permutations by spacing the legal positions far apart, but the resulting spread between components will be unacceptable if the variation in sizes is too great.

To adapt our definition of feasible placements to a wider range of real circuits, later chapters generalize the notion of feasible placements by allowing extra constraints for special components. In particular, a problem instance can divide the components into two or more classes and specify which legal positions will be occupied by each class. A group of components that are incompatible with most legal positions can be relegated to a group of legal positions that can accommodate them. Or, by specifying classes that

have only one member, individual components can be "pre-placed".

The division of components into classes is useful when a few components are much larger than the rest, *e.g.*, macro cells in standard-cell design. Another important application of classes is the special treatment of external ports or I-O pads. These components are the interfaces between the given circuit and the rest of the world. Since external signals are delivered at boundaries of the layout surface, such components must almost always be placed at a boundary; and individual pads must sometimes occupy particular positions.

Permutations of legal positions, together with the above extension for special components, constitute a set of convenient models of feasible placements that adequately represent several types of practical problems. These models are best suited to problems that involve many components of similar size and shape. They are less appropriate for problems that involve arranging components with highly diverse contours to fit close together. Although it might be possible to extend the permutation approach, *e.g.*, by modeling components of arbitrary shape as tightly-connected collections of more uniform smaller cells, we will not explore such approaches here.

2.1.3. The quadratic cost function

We have described connection matrices and feasible placements. For any function that attaches costs to circuit placements, we can now study the problem of finding a minimum-cost feasible placement. But it is not obvious what cost function is appropriate.

In practice we evaluate a placement in terms of how well it translates into a finished design, *i.e.*, a layout with the connections routed. Does a routing for the placement exist that obeys the rules of the technology? Into

how small an area can the layout be compacted? To answer such questions we must attempt a complete routing, which would be an unreasonable prerequisite for computing placement cost: it would be inefficient; it would make cost depend on the particular routing strategies used; and it would not provide a closed formula for cost. We want placement cost to correlate well with routing difficulty. But "difficulty" is hard to express in a usable formula because routing procedures are complicated, and they differ from each other in fundamental respects (*e.g.*, how they handle congested regions).

In any case, connections over long distances tend to be hard to route. This observation underlies the common definition of placement cost as the sum over all component pairs i, j of $c_{ij} \cdot ((x[i]-x[j])^2 + (y[i]-y[j])^2)$. Each term is the product of a connection weight c_{ij} and the squared distance ("d^2") between components i and j.

Before discussing why we use the particular function d^2, we note some consequences of using *any* function $f(d)$ that is "superadditive", *i.e.*, that satisfies $f(a+b) > f(a) + f(b)$ for positive distances a and b. Such a function penalizes long connections; a single large distance costs more than two smaller ones of the same total length. Since overall speed and reliability tend to deteriorate when a circuit uses long connections, this form of penalty is reasonable.

On the other hand, simply taking cost proportional to distance would give a more robust measure in one situation: suppose we introduce extra components (*e.g.*, amplifiers or "repeaters") along a long wire. Total cost of the connections does not change if cost is proportional to distance, but it decreases if we use a superadditive function. For example, with $f(d)=d^2$, the evaluated cost to connect two components would be cut in half if a repeater were placed at their midpoint, since $f(d/2)+f(d/2)=\frac{1}{2}f(d)$. We

elect to give up the property of connection cost invariance with added repeaters, since the function d^2 has some very useful qualities.

Signal propagation time on a wire grows as the square of the length traveled. The dependence is quadratic because propagation time is proportional to the product of resistance and capacitance, both of which grow linearly with wire length.* Our measure of placement cost is thus a weighted total of estimated propagation times in all connections of the circuit. The estimates are low because connecting wires do not generally run in straight paths; wire lengths tend to exceed the distances between components. Although total propagation time is not an ideal measure of circuit cost, it is interesting that squared distances are good estimates of propagation times.

Thanks to the Pythagorean theorem, squared distances offer a significant convenience; each d^2 equals the squared distance in x plus the squared distance in y. As a result, the contributions to placement cost from the x and y vectors can be computed separately and added together:

$$\text{cost}(x,y)=\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}((x[i]-x[j])^2+(y[i]-y[j])^2)=\text{cost}(x)+\text{cost}(y).$$

The most important advantage of using squared distances is that it enables us to write cost(x) as a quadratic form $\sum_{i=1}^{n}\sum_{j=1}^{n}b_{ij}x[i]x[j]$. In section 2.2 we exploit the properties of quadratic forms to develop a useful restatement of the circuit placement problem.

*See [Ullman84, page 3].

2.1.4. Computational complexity of the problem

Many critical problems in circuit layout are NP-hard. An NP-hard problem is one that is as difficult as any problem in the class "NP", in the following sense: an algorithm capable of solving the given problem efficiently could be used to solve any problem in NP efficiently. In this context, an efficient solution is one whose worst-case running time is bounded by a polynomial function of the length of data needed to represent the problem instance. (See [Karp75, Garey79].)

To prove that a problem is NP-hard, one supplies a polynomial-time reduction that transforms some known NP-complete problem (one already proven to be difficult in the above sense) into the form of the given problem. The following proof that our formulation of circuit placement is NP-hard shows that despite the simplifications we have made, the problem is still difficult.

We transform graph partition (which is NP-complete*) into the circuit placement problem.

> An *INSTANCE* of graph partition consists of a graph (V,E) and an integer k. V is a set of nodes 1 to n, with n even; E is a set of edges, *i.e.*, pairs of nodes of V. The *QUESTION* is to decide if there is a partition of V into disjoint sets S and T, with $n/2$ nodes each, such that no more than k edges in E have nodes in both S and T. Given an instance of graph partition, produce an instance of circuit placement as follows: Form an n-by-n matrix C with $c_{ij}=1$ when $<i,j> \in E$ and $c_{ij}=0$ otherwise. Specify $n/2$ legal positions at (0,0) and $n/2$ at (1,0).**

*[Garey76] calls this "minimum cut into equal-sized subsets." Their statement also specifies two nodes that the partition must separate, but the proof given works as well with no distinguished nodes.

**Although it is counterintuitive to ask that multiple components be placed at the same location, our problem definition does not forbid it. In fact, partitioning is so important in practice that we specialize several of our methods for exactly this problem.

In the resulting problem, placement cost equals the number of edges whose nodes are assigned to different locations. An algorithm capable of determining optimal placements could thus decide the graph partitioning question by answering "yes" when the minimum cost in the transformed problem was less than or equal to k.

An efficient method to solve our circuit placement problem exactly would therefore imply an efficient solution to any problem in NP, including dozens of problems that are presumed to be intractable. Since such a result is highly unlikely, we seek efficient heuristic methods for finding placements that are close to optimal. We cannot guarantee that our approach will find a solution within a specified factor of the minimal cost in every instance. However, it does provide good placements and lower bounds on placement cost for any given circuit.

Placement cost may be calculated by summing the results of two applications of the same cost function: once to the x vector and once to the y vector. It is easiest to explain our approach if we focus initially on problems that involve only one vector. Accordingly, we restrict attention in chapters 2 through 5 to problems in which the y coordinates of all legal positions equal 0. All of our methods have natural generalizations for full two-dimensional problems, and we present these in chapters 6 and 7. These extensions are treated separately because substantial technical complications arise in two dimensions. Our perspective on the two-dimensional techniques will be clearer if we first develop the fundamental concepts in the one-dimensional setting.

In the prototypical example of one-dimensional placement, the legal positions are spaced at uniform intervals along the x axis. We call this case "UIP", for uniform-interval placement. In chapter 7 we use UIP as a test

case for the placement of randomly generated graphs. We prove in appendix 1 that UIP is NP-hard.

2.2. Restatement using eigenvectors

In this section and the next, we develop the transformation to a furthest-point problem. Readers who are conversant with linear algebra are likely to find the careful explanation of every step belabored. They are invited to bypass the next five pages and simply read the condensed derivation, which appears at the end of section 2.3.

For a one-dimensional problem, we represent circuit placements with a single vector x, in which component $x[i]$ of the vector identifies the x position given to the ith component of the circuit. Our measure of placement cost for the vector x is

$$\text{cost}(x) = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}(x[i]-x[j])^2.$$

Expanding the terms $(x[i]-x[j])^2$, we obtain

$$\text{cost}(x) = -\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}\, x[i]x[j] + \frac{1}{2}\left(\sum_{i=1}^{n} x^2[i]\sum_{j=1}^{n} c_{ij} + \sum_{j=1}^{n} x^2[j]\sum_{i=1}^{n} c_{ij}\right).$$

Because C is symmetric, the kth row-sum $\sum_{j=1}^{n} c_{kj}$ equals the kth column-sum $\sum_{i=1}^{n} c_{ik}$, so the parenthesized double sums are equal. Using matrix notation to represent the quadratic form $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x[i]x[j]$, we have

$$\text{cost}(x) = -x^T Cx + \sum_{i=1}^{n} x^2[i]\cdot(i\text{th row-sum of } C).$$

Define a matrix B:

$$B = -C + D, \tag{2.0}$$

where D is a diagonal matrix with entries equal to the row-sums of C. We

can then write simply

$$\text{cost}(x)=x^T Bx. \tag{2.1}$$

We can restate the problem in a convenient form using the eigenvectors of B. Because B is symmetric, it has n orthonormal eigenvectors u_r.* Each eigenvector has an associated cost: $\text{cost}(u_r)=u_r{}^T Bu_r=\lambda_r$. *Any* vector x has a unique expansion $x=\sum_r \alpha_r(x)u_r$ with coefficients $\alpha_r(x)=x^T u_r$.

The above expansion is convenient because we can use it to write placement cost as a sum of independent contributions from the different eigenvectors:

$$\text{cost}(x) = \sum_r \alpha_r{}^2(x)\lambda_r . \tag{2.2}$$

In one dimension, the feasibility requirement reduces to $x \in \{\Pi l$: Π is a permutation matrix$\}$, where l is a vector whose components are the legal x positions in arbitrary order. Less formally, x must be a permutation of the legal positions. We thus have the following problem restatement:

> Choose x to minimize $\sum_r \alpha_r{}^2(x)\lambda_r$
>
> *where* x is a permutation of the legal positions
>
> and $\alpha_r(x)=x^T u_r$

We have chosen to express x as a weighted combination $\sum_r \alpha_r(x)u_r$; *i.e.*, we have selected the eigenvectors of B as a "basis". Since *any* n linearly independent vectors form a basis, it is worthwhile to explain what makes

* An eigenvector of B is simply a nonzero vector u_r that satisfies $Bu_r=\lambda_r u_r$ for a constant λ_r (the eigenvalue). The λ_r's are real when B is symmetric. To be "orthonormal" is to satisfy $u_r{}^T u_s=0$ for $r \neq s$ ("orthogonality"), and $u_r{}^T u_r=1$. A good reference on eigenvectors is [Parlett80].

the eigenvectors uniquely useful in this role. We have already mentioned one important property: the u_r's may be chosen to be orthonormal. In general we might have to solve a system of linear equations to determine the expansion coefficients $\alpha_r(x)$; but with orthonormal basis vectors u_r, each $\alpha_r(x)$ is given by the simple inner product $x^T u_r$. Another advantage of orthonormal bases is that they preserve inner products: $p^T q = \alpha(p)^T \alpha(q)$. We will soon make use of a special case of this property: $\sum_i x^2[i] = x^T x = \alpha(x)^T \alpha(x) = \sum_r \alpha_r{}^2(x)$.

The property of the eigenvector basis that is crucial to our problem restatement is its "cost independence". Briefly, we want $\text{cost}(u_r + u_s) = \text{cost}(u_r) + \text{cost}(u_s)$ for any pair of vectors u_r, u_s in the basis. By equation (2.1), this amounts to requiring

$$u_r{}^T B u_r + u_s{}^T B u_s + 2 u_r{}^T B u_s = u_r{}^T B u_r + u_s{}^T B u_s,$$

or simply

$$u_r{}^T B u_s = 0. \tag{2.3}$$

Orthogonality implies $u_r{}^T u_s = 0$; it does not insure $u_r{}^T B u_s = 0$. But when the u's are orthogonal *eigenvectors* of B, $B u_s = \lambda_s u_s$, and requirement (2.3) is satisfied.

It is instructive to compare the eigenvector basis to another orthonormal basis that appears to be a more natural choice. Most placement algorithms work with the vector x by manipulating individual components. In effect they expand x in terms of the orthonormal basis $\{e_1, e_2, \ldots, e_n\}$, where e_i has 1 for its ith component and zeros for all the others: $x = \sum_i x[i] e_i$. The problem with this basis is that the contributions of the vectors e_i to placement cost are not independent. Because they interact, we cannot attach fixed costs to choices of individual coefficients $x[i]$. By contrast, if we know

a coefficient $\alpha_r(x)$ of an individual u_r in the eigenvector expansion, we know that it will contribute exactly $\alpha_r^2(x)\lambda_r$ to placement cost, regardless of the values of the other coefficients in the expansion. Eigenvectors of B are the only orthonormal bases with cost independence.

Discussing the advantages of expressing x as a combination of eigenvectors of B has set the stage for our problem transformation. Before proceeding, we establish two helpful conventions for the legal positions. Note that if we alter the legal positions by a "shift" (adding a constant to all of them) or a "scaling" (multiplying all by a nonzero factor), nothing essential about the problem is changed. A shift does not change placement costs, which depend entirely upon differences between component positions; and scaling by k multiplies the cost of every feasible placement by k^2. We can therefore assume without loss of generality that the legal positions are centered at the origin ($\sum l[i]=0$) and have unit variance ($\sum l^2[i]=1$).

The above conventions slightly simplify the eigenvector expansion. Since by definition (2.0) every row of B sums to 0, the vector $(\sqrt{1/n},\sqrt{1/n},\cdots\sqrt{1/n})^T$ is an eigenvector, which we label u_0. The corresponding eigenvalue λ_0 equals 0 ($Bu_0=0u_0$; intuitively, when all components are put at the same place, the connection cost is zero). Centering the legal positions allows us to ignore u_0, because its contribution $x^T u_0$ equals zero for every feasible x. We label the other eigenvectors u_1 through u_{n-1}, in order of increasing eigenvalue. Scaling so that $\sum l^2[i]=1$ insures (by orthonormality) that

$$\sum_{r=1}^{n-1}\alpha_r^2(x)=1 \text{ for every feasible placement.} \qquad (2.4)$$

The observation (2.4) allows us to interpret equation (2.2) as follows: the cost of a vector is the weighted average of its constituent eigenvector

costs λ_r. The bounds $\lambda_1 \leq \text{cost}(x) \leq \lambda_{n-1}$ follow. Thus (as shown in [Blanks85b]) if the cost of a heuristic solution approaches λ_1, a proof of near-optimality is immediate. Simply setting $x = u_1$ would be optimal ([Hall70]), but the components of u_1 are unlikely to coincide with the legal positions. We therefore seek heuristics to find feasible placement vectors whose expansions are dominated by the lowest-cost eigenvectors. In the next section we describe a problem transformation that facilitates this search.

2.3. Transformation to a furthest-point problem

For any real constant H, minimizing $\text{cost}(x) = \sum_{r=1}^{n-1} \alpha_r^2(x)\lambda_r$ is equivalent to maximizing

$$H - \text{cost}(x) = \sum_{r=1}^{n-1} \alpha_r^2(x)[H - \lambda_r] \; * \tag{2.5}$$

Pick an arbitrary $H \geq \lambda_{n-1}$ and define a matrix V whose columns are the eigenvectors $v_r = u_r\sqrt{H - \lambda_r}$; this scaling has the effect of associating the greatest lengths with the lowest-cost eigenvectors. Then the expression (2.5) to be maximized equals $\sum_{r=1}^{n-1}(x^T v_r)^2$, which we write as $\|x^T V\|^2$. By representing the feasible placements x as points $x^T V$ (with coordinates $x^T v_r$), the problem becomes a search for the point furthest from 0.

What is the transformation all about? The overall idea is to represent the problem in a domain that allows us to use geometric search tools. The usual way to interpret vectors in R^n geometrically is to take each component $x[i]$ as the coordinate on an axis e_i. Our transformation involves a

* because $\sum_{r=1}^{n-1} \alpha_r^2(x) = 1$ for all placements.

"renaming" stage (we adopt new axes) and a "relocation" stage (we multiply the coordinates on each new axis by a different scale factor).

When we change reference axes from the directions e_i to the directions u_r, the coordinates change from $x^T e_i$ to $x^T u_r$. In effect, each placement x is simply given a new name: $x^T U$. The feasible points, initially equidistant from the origin ($\Sigma x^2[i]=1$), all remain at distance 1 under the change of coordinates.

Now multiply every coordinate $x^T u_r$ by $\sqrt{H-\lambda_r}$: the coordinates of the relocated points equal $x^T v_r$, where $v_r = u_r\sqrt{H-\lambda_r}$. Because of eigenvector cost independence, the distance from the origin to each relocated point $x^T V$ is a function of its cost:

$$\|x^T V\|^2 = H - \text{cost}(x) \tag{2.6}$$

By representing feasible points as rows $x^T V$ rather than columns $V^T x$, we make it easier to distinguish vectors in the transformed domain from those in the original domain. Columns belong to the original "placement" domain, and we associate each of their n components (*e.g.*, $x[i]$ or $u_r[i]$) with the physical location of a particular circuit component. Rows belong to the transformed "point" domain, and we identify each of their $n-1$ components (*e.g.*, α_r, d_r, or $x^T v_r$) with a particular eigenvector of B and its associated axis.

In the next section we take advantage of the chosen transformation: we describe a simple tool for discovering points that are far from the origin.

Condensed derivation of furthest-point problem

$$\text{cost}(x)=\tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}(x[i]-x[j])^2.$$

Define

$$B=-C+D, \tag{2.0}$$

where D is diagonal with $d_{ii}=\sum_j c_{ij}$.

$$\text{cost}(x)=x^TBx. \tag{2.1}$$

Let u_r's be orthonormal eigenvectors of B, with eigenvalues $0=\lambda_0\le\lambda_1\le\cdots\lambda_{n-1}$. Let $\alpha_r(x)=x^Tu_r$.

$$\text{cost}(x)=\sum_r \alpha_r{}^2(x)\lambda_r\,. \tag{2.2}$$

Choose legal positions with $\Sigma l[i]=0$, $\Sigma l^2[i]=1$; so $\alpha_0(x)=0$ and

$$\sum_{r=1}^{n-1}\alpha_r{}^2(x)=1 \text{ for every feasible placement.} \tag{2.4}$$

$$H-\text{cost}(x)=\sum_{r=1}^{n-1}\alpha_r{}^2(x)[H-\lambda_r] \tag{2.5}$$

Pick $H\ge\lambda_{n-1}$. Let V comprise columns $v_r=u_r\sqrt{H-\lambda_r}$ for $r=1$ to $n-1$.

$$\|x^TV\|^2=H-\text{cost}(x) \tag{2.6}$$

2.4. Probes for good points

We have seen that the problem transformation allows us to evaluate $H-\text{cost}(x)$ in terms of the distance $\|x^TV\|$ from the origin to the transformed point. The transformation pays off because we can efficiently produce the point that is furthest out along any given direction. We use the word "probe" to stand for both the direction and for the operation that finds the point with maximum projection in that direction. Any probe $d^T\in R^{n-1}$ delivers a special x – one that maximizes x^TVd. The diagram illustrates the idea for a case in which only d_1 and d_2 are nonzero. Each

dot plots the first two components $(x^T v_1, x^T v_2)$ of a point $x^T V$ corresponding to some feasible placement x.

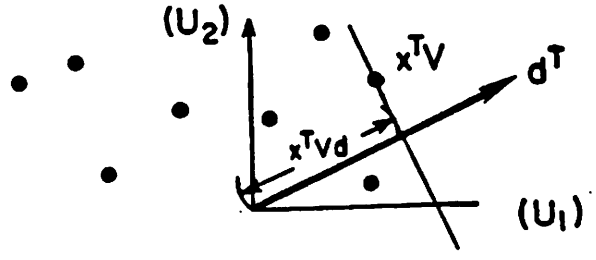


Figure 2-2. A probe locates the point $x^T V$ with maximum projection on d^T.

Every probe yields both a placement and a proof that no point in the entire set of feasible solutions has a greater projection in the probe direction.

To perform a probe, we first compute Vd, whose n elements $(Vd)[i]$ are the projections of the rows $V[i]$ onto the probe. (This takes time $O(nk)$, where k is the number of nonzero elements of d.) The objective is then to determine x, the permutation of legal positions that maximizes $x^T(Vd)$. The solution is to order x to match the ordering of the components of Vd; if $(Vd)[m]$ is smallest then assign $x[m]$ the smallest legal position, etc. Otherwise, for some pair of nodes i, j, $(x[i]-x[j])\cdot((Vd)[i]-(Vd)[j])<0$; and swapping i with j would increase the inner product. To sort (Vd) requires time $O(n \log n)$.

We can view our whole problem as a search for the best probe direction, because there is always some probe direction that would produce the optimal point. To see this, let f^T be the point furthest from the origin in an arbitrary set of points; and consider the unit vector d^T aimed at f^T, that is, $d^T = f^T / \|f^T\|$. Using d^T as a probe direction would produce the point f^T, since it is necessarily the one with maximum projection on d^T.*

*Write the squared distance to any point p^T as $\|p^T\|^2 = (p^T d)^2 + \|r^T\|^2$, where r^T (the residual portion of p^T orthogonal to d^T) equals $p^T - (p^T d)d^T$. Since f^T has zero

There is also a direct proof that the furthest point and the best probe are equivalent search objectives. The furthest point objective is $\underset{\text{feasible } x}{Max} \|x^T V\|$, which we can rewrite as $\underset{\text{feasible } x}{Max} (\underset{|d|=1}{Max} x^T V d)$.** Interchanging the maximizations, our objective is

$$\underset{|d|=1}{Max} (\underset{\text{feasible } x}{Max} x^T V d), \tag{2.7}$$

which precisely defines the search for a best probe.

When we map placements to points, cost becomes represented by distance from the origin. If we transformed to points with coordinates $x^T u_r \sqrt{\lambda_r}$ instead of $x^T u_r \sqrt{H-\lambda_r}$, the squared distances would equal cost(x) instead of $H - \text{cost}(x)$. Thus, depending on which formula we choose, the transformed optimum can be the point with minimum magnitude or the point with maximum magnitude. We convert the objective from cost minimization to distance maximization because transforming to a furthest-point problem enables us to use probes. That the application of probes depends on a simple change of objective from min to max is a curious fact that calls for some explanation.

The idea of probes is to attack the global problem by searching locally in one direction at a time. We have seen that the point furthest from the origin must also be the one with maximum projection on a probe aimed at that point. But there is not necessarily any direction on which the point closest to the origin has the minimum projection. Minimizing the projection on d^T selects the point *furthest* from 0 in the direction $-d^T$.

---

residual, $p^T d > f^T d$ would give $\|p^T\| > \|f^T\|$, contradicting the assumption that f^T is the furthest point.

**The Euclidean length $\|p^T\|$ equals the maximum value of $p^T d$ over all vectors d of unit length.

In other words, probes are linear programs and thus return *extrema*, *i.e.*, points from the convex hull of the feasible solutions. If we did not change our goal from convex minimization to convex maximization, the best points would be the most "interior" solutions.

The other possible goal to associate with a direction d^T in the magnitude-minimizing formulation would be to minimize $|p^T d|$, but this too is ill-conceived. To have a large projection, a point must have a large magnitude. However, for *any* point p^T there is an n-1$-$dimensional space of directions d^T for which $p^T d = 0$. Knowing that a point has 0 projection in a given direction tells us nothing about the magnitude of that point. Furthermore, to find the feasible x that minimizes $|x^T(Vd)|$ for fixed Vd is NP-hard, even though the corresponding maximization is easy. (Appendix 2 proves this by a reduction from set partition.)

The transformation to a furthest-point problem gives us the opportunity to use probes. But probes are only a tool. The challenge is to devise strategies for selecting useful directions.

Probes with only a few nonzero components have important practical advantages. They are more efficient than random probes, since calculating Vd requires a number of operations roughly proportional to the number of nonzeros in d. They also allow us to choose nonzero components that improve our chance of finding a good point: for example, we can select the k eigenvectors with the smallest λ_r's. Since the point-scaling factors $\sqrt{H-\lambda_r}$ are greatest for these axes, we expect such probes to discover points further out than points found by average probes.

We can view our transformed placement problem as a search for a probe in R^{n-1} that produces the point furthest from the origin. It is useful to study variants of this problem that restrict the set of probe components

that we allow to be nonzero. With the other components fixed at zero, we can explore the range of possible contributions to $H-\text{cost}(x)$ from k selected eigenvectors. Limiting the probe set reduces the original search in R^{n-1} to a k-dimensional search problem. Each dimension in R^k corresponds to an active column-vector in V. Since all but k dimensions are ignored, we are essentially projecting the points into R^k and measuring distances in this subspace.

How do k-dimensional search problems relate to the full problem? First, some solutions may not be discoverable by any probe in R^k.* Second, distances to points in k dimensions will in general differ from the corresponding distances in the original space.

In certain fortuitous situations, all the original lengths can be perfectly preserved in a projected space of fewer than n-1 dimensions. Suppose eigenvalues λ_{k+1} through λ_{n-1} were identical. (For a concrete application of this thought experiment, we can take $k=n-2$ for any problem.) Now break the summation in (2.5) into two pieces:

$$H-\text{cost}(x)=\sum_{r=1}^{k}\alpha_r{}^2(x)[H-\lambda_r]+\sum_{r=k+1}^{n-1}\alpha_r{}^2(x)[H-\lambda_r] \qquad (2.8)$$

If we set $H=\lambda_{n-1}$ in this hypothetical case, then we observe the following desirable results: the terms in the second summation all equal zero; costs may be evaluated in R^k; and the optimal point must be detectable by a probe in R^k.

Although λ_{k+1} through λ_{n-1} are usually not equal, the thought experiment suggests guidelines for choosing H in more general circumstances. Any k-dimensional search intentionally neglects a sum like the second one

*In chapter 4 we present an exact formula for the number of points discoverable by probes, as a function of n and k.

above. For any particular x there will be some H in the range of the neglected eigenvalues that makes the neglected terms sum to zero. In general, however, no single choice of H can provide lengths in R^k that equal the full lengths for all the solutions.

The next two chapters both exploit the idea of reducing the number of active dimensions. Chapter 3 gives methods to find good placements. In this case we want k-dimensional distances to match the full distances as closely as possible. Our choice of H should therefore be an estimate of an effective *average* of the neglected eigenvalues; it might be an overestimate for some placements and an underestimate for others. Chapter 4 gives methods that take any given circuit and prove lower bounds on its placement cost. In proving a lower bound, we must guarantee that any estimate that we use for the cost contributed by neglected eigenvectors is an underestimate. Thus a k-dimensional search with u_1 through u_k must use $H \leq \lambda_{k+1}$ in this case.

Chapter 3

3. One-dimensional Placements by Iterated Probes

In chapter 2 we introduced the problem of minimizing $x^T B x$, where x must be some permutation of a given vector of legal positions. The scaled eigenvectors v_r $(= u_r\sqrt{H-\lambda_r})$ of B define a transformation from placements x to points $x^T V$. The minimum-cost feasible placement (call it "x_*") is mapped to the point furthest from the origin, with $H - \text{cost}(x_*) = \|x_*^T V\|^2$.

Some bounds on $\text{cost}(x_*)$ are easy to obtain. We observed that $\text{cost}(x_*) \geq \lambda_1$, where λ_1 is the least positive eigenvalue of B. Also, any probe in the transformed space yields a feasible point p^T whose distance from the origin cannot exceed that of the optimal point. Combining these observations, we have

$$\|p^T\|^2 \leq \|x_*^T V\|^2 \leq H - \lambda_1 \qquad (3.1)$$

We will give different methods that use probes to tighten the inequalities in (3.1). To decrease the right-hand bound from $H - \lambda_1$ to some new value M, we must prove that *every* feasible point $x^T V$ satisfies $\|x^T V\|^2 \leq M$. Techniques for such proofs, which provide improved lower bounds on placement cost for given circuits, are developed in chapter 4. It is easier to work at sharpening the left-hand inequality: we can use any heuristic that discovers feasible points that are far from the origin. In this chapter we examine placement heuristics that obtain distant points by using sequences of probes.

3.1. The idea of iterated probes

The common idea of progressively improving an initial solution can be adapted to the furthest-point problem. Recall that a probe operation locates the point with maximum projection in a given direction. Our basic improvement step is to direct a new probe toward the maximum-projection point discovered by the previous one. Let p^T be the point at which we aim. On the new probe, the projection of p^T equals its magnitude; so when we find the point with *maximum* projection, its magnitude must at least equal that of p^T. By iterating the process, we step through a sequence of points, each further from the origin than the last, until the same point is detected by two successive probes. We say that the last point in a sequence is "stable". In Figure 3-1, we illustrate a sequence of three probes; the second point found is stable.

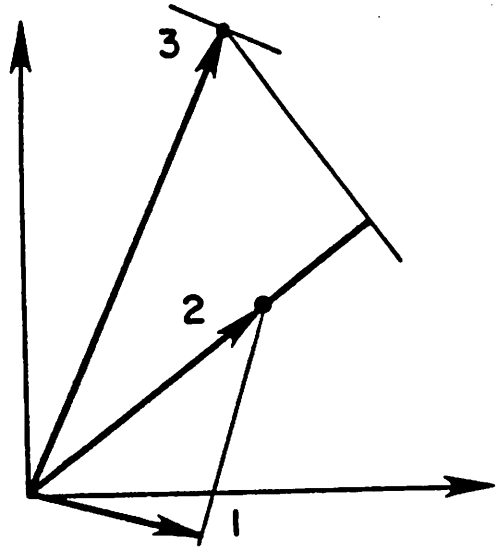


Figure 3-1. Illustration of iterated probes.

The steady increase in magnitude from point to point implies that each placement that we encounter costs less than the previous one, when we iterate on the full transformed problem in R^{n-1}. If instead we iterate with

projections of the transformed points in a k-dimensional space (with k possibly much less than $n-1$), we can isolate the most valuable eigenvectors and improve efficiency. We assume that improvements in k dimensions will tend to improve placement cost. This assumption is based on equation (2.8), reproduced here,

$$H-\text{cost}(x)=\sum_{r=1}^{k}\alpha_r^2(x)[H-\lambda_r]+\sum_{r=k+1}^{n-1}\alpha_r^2(x)[H-\lambda_r]$$

and a "principle of ignorance": that maximizing the first sum will not systematically worsen the distribution of terms in the second, uncontrolled sum. While it is possible for an improvement in k dimensions to be associated with a larger placement cost, such cases should not be too frequent.

The distributions of coefficients α_r^2 observed in good placements provide the best evidence that it is reasonable to work with k-dimensional approximations. In chapter 7 we examine, for heuristic solutions to placement problems, the fractions of $\sum_{r=1}^{n-1}\alpha_r^2$ contained in different terms. Typically, the first 5 to 10% of the terms make up 95% or more of the sum. We know that the dimensions with the smallest eigenvalues add the most to our objective function, per α_r^2. The distributions tell us that in practice, these dimensions can make a substantial contribution to good solutions. This observation is the main rationale for considering projections of the problem into relatively few dimensions.

We conclude this section with a detailed statement of the basic iterated probes method, using a mix of *PASCAL* syntax and high-level description.

```
procedure IterateProbes ( (*INPUTS*) V, n, L, k, dIn, doSteps;
                          (*OUTPUT*) x);
  {INPUTS:
  V is a matrix. Typically, its columns v_r are scaled eigenvectors.
  n is the number of entries in each column v_r;
       also the length of column vectors in general.
  L is a column vector: the legal positions in nondecreasing order.
  k is the number of active columns in V;
       also the length of row vectors ("directions") in general.
       This procedure treats V as an n-by-k matrix, with
        columns v_1 through v_k active.
  dIn is a row vector that defines the first probe direction.
  doSteps is an integer: the maximum number of probes to perform.

  OUTPUT:
  x is a column vector that returns the final permutation of L.
       A sequence of directions d^T is used; at each step, x is set equal to
   the permutation of L for which x^T V d is maximum. The direction x^T V
   is used next; the process continues until x stops changing or doSteps
   iterations have been executed.

  LOCAL VARIABLES:}
  i,r,step: integer;    rank: array[1..n] of integer;    stable: boolean;
  d: row (* with elements d_r *);    projection,previousX: column;

  begin (* IterateProbes *)
     step := 0;
     d := dIn;
     repeat
        for i := 1 to n do projection[i] := k-dim. inner product of
              ith row of V with d;
        (* To maximize x^T V d, fill x with the permutation of legal positions
              that matches the ordering of the projection components. *)
        Determine rank, an array of indices such that projection[rank[1]] ≤
              projection[rank[2]] ≤ · · · ≤ projection[rank[n]];
        for i := 1 to n do x[rank[i]] := L[i];
        (* Prepare next probe. *)
        for r := 1 to k do d_r := n-dim. inner product x^T v_r.
        (* Check if done. *)
        step := step + 1;
        if step=1 then stable := FALSE else stable := (x=previousX);
        previousX := x;
     until (step=doSteps) or stable;
  end; (* IterateProbes *)
```

Each improvement step (pass through the repeat loop) takes less than twice the computation of a single probe. (The k n-dimensional inner products that determine the new probe direction are roughly as expensive as the n k-dimensional inner products needed for the probe itself.)

The rest of the chapter concerns how to use this procedure. How do we pick starting directions? How many dimensions should we use? The guidelines that we will give for these and other parameters are based on experiments: these experiments and the computational results are described in detail in chapter 7.

3.2. Using iterated probes

We can apply *IterateProbes* to try to improve the solution that any probe produces. By trying many different start directions, we increase the chance of discovering valuable points.

We could try a greater number of independent directions if we performed just one probe per trial instead of iterating; but in the search for distant points, iteration pays off. That is, we typically get better results from a small sample of iterated-probe solutions than from a large sample of independent probes.

3.2.1. Start directions

How should we pick start directions? Rather than generating directions at random, we want to target the dimensions with the greatest potential. A simple strategy is to stay in the span of the s dimensions with the smallest λ_r's. Once we adopt this restriction, we assume for simplicity that random initial directions in R^s are good enough. Selecting a value for s (the dimensionality of the start-space) then becomes the main choice in determining

start directions.

If s is too large, solution quality deteriorates. If s is too small, many independent start directions may converge to only a few independent solutions.* In most experiments, taking s on the order of $\sqrt{n}$ has worked best.

For a specific problem instance we might take the best preliminary placement cost obtained by some heuristic and pick the value of s for which λ_s is closest to this cost. (Only the dimensions with smaller eigenvalues can help reduce cost further.) Or, we can try a quick experiment: perform a fixed number of independent random probes in s dimensions for each of several values of s, and see which value gives the best solutions.

These heuristics provide a range of possible values. We can let s vary over this range in different trials; as the results accumulate, we learn which values work best.

To get started we must set the scale factors $\sqrt{H-\lambda_r}$ for the s eigenvectors v_r $(=u_r\sqrt{H-\lambda_r})$, *i.e.* we must assign a value to H. Recall that H estimates an effective average of eigenvalues associated with the *other* eigenvectors. The starting value of H is not critical: one possibility is the average of eigenvalues λ_{s+1} through λ_{n-1}.

To perform the starting probe in a sequence, we invoke *IterateProbes* with $k=s$ and $doSteps=1$.

*We could reduce the likelihood of repeat solutions by injecting randomness throughout the sequence, *e.g.*, by adding perturbations to the probes instead of aiming directly at points.

3.2.2. Number of dimensions for iterations

Once we have performed a first probe, we are ready to iterate. We must set a value for k, the number of active vectors in V. We assume from now on that the eigenvectors with the smallest associated eigenvalues will be used; any deviations from this rule will be noted.

Although large values of k require proportionally more computation in the inner products, the extra work can lead to higher-quality solutions. We have observed improvements in solution quality with increases in k up to a substantial fraction of n (the total number of dimensions), *e.g.*, $\frac{n}{3}$ to $\frac{n}{2}$. Iterating with this many dimensions requires a quadratic number of operations per step. Fortunately, there is no reason that every probe in a sequence must use the same number of dimensions.

3.3. Iterating in stages

Having observed that iterations can take advantage of many more dimensions than are useful for start directions, it is natural to consider increasing the number of dimensions in the course of the sequence of probes. We can iterate in several stages, with the number of dimensions increasing from one stage to the next.

To generalize iterated probes, we provide for different numbers of dimensions and steps in the different stages. Transitions are straightforward: the solution x at the end of a stage determines the initial probe direction for the next. We simply extend the computation of components $x^T v_r$ to the newly activated eigenvectors. The following is a prototypical example of the use of stages.

```
procedure Stages ( (*INPUTS*) U, n, λ, L, s, dIn, last, k, steps, epsilon;
                   (*OUTPUT*) x);
  {INPUTS:
  U is a matrix. Its columns u_r are eigenvectors, with u_r^T u_r = 1.
  n is the number of entries in each column u_r;
      columns in general have length n; rows have length up to n-1.
  λ is a row vector: the eigenvalues λ_r associated with the u_r's.
  L is a column vector: the legal positions in nondecreasing order.
  s is the number of active entries in the start direction.
  dIn is a row vector that defines the start direction.
  last is an integer: the maximum number of stages.
  k is an array of last integers: the ath entry
      is the number of active columns in stage a.
  steps is an array of last integers: the ath entry
      is the maximum number of probes to perform in stage a.
  epsilon is a real number. If the fraction of Σ(x^T u_r)^2 in the active dimensions
      exceeds 1-epsilon at the end of a stage, no more stages are begun.

  OUTPUT:
  x is a column vector that returns the final permutation of L.

  LOCAL VARIABLES:}
  r,a: integer;  H: real;  d: row (* with elements d_r *);
  V: matrix (* with columns v_r *);
  begin (* Stages *)
    H := 1/(n-1-s) Σ_{r=s+1}^{n-1} λ_r;
    for r := 1 to s do v_r := u_r * sqrt (H-λ_r);
    IterateProbes (V,n,L,s,dIn,1,x);
    a := 0;
    repeat
      a := a+1;
      H := NextH (* defined later in text *) (k[a],U,n,x,λ);
      for r := 1 to k[a] do v_r := u_r * sqrt (H-λ_r);
      for r := 1 to k[a] do d_r := x^T v_r;
      IterateProbes (V,n,L,k[a],d,steps[a],x);
    until (a=last) or (Σ_{r=1}^{k[a]} (x^T u_r)^2 > 1-epsilon);
  end; (* Stages *)
```

Except for the function *NextH*, which we will discuss shortly, the operation of *Stages* is clear. We have obtained the best results by conducting the first stage with as many dimensions as are used for the starting probe, and multiplying the number of dimensions by a constant in each successive stage. A multiplier of 2 works well. With a smaller multiplier, there are

more stages; we sometimes find better solutions, at the cost of increased computation time.

Another parameter for each stage is the maximum number of probes (*steps*). We can stop early stages before they reach a stable point: this saves time without necessarily affecting solution quality. A probe in R^k requires time $O(n[k+log(n)])$. Thus we can divide the total time fairly evenly among the stages by choosing a parameter p and setting *steps* for R^k to the integer nearest to $p/(k+log(n))$.

The resulting time per stage would be $O(pn)$. When k increases by a constant multiple with each stage, there are $O(log(n))$ stages, for a total time of $O(pn\,log(n))$. To insure a positive number of steps in the last stage, we need $p>k[last]/2$. Since this is $O(n)$, the overall running time is $O(n^2log(n))$.

To complete the specification of *Stages*, we must define the function *NextH*, which fixes H at the beginning of each stage. What is at stake when we choose H? The set of placements that arbitrary probes can generate with given eigenvectors is identical for any H greater than the associated eigenvalues. This is because these placements correspond to all the component orderings of vectors in the span of the eigenvectors $u_r\sqrt{H-\lambda_r}$; the span is the same for any positive values $\sqrt{H-\lambda_r}$.

However, the relative magnitudes of points in R^k do depend on H. Suppose $\lambda_1=.04$ and $\lambda_2=1$: for $H=2$ the ratio $\sqrt{H-\lambda_1}/\sqrt{H-\lambda_2}$ is only 1.4, but for $H=1.04$ the ratio equals 5. A consequence of this kind of change is that it is possible to have two solutions whose magnitude ordering in R^2 is reversed by changing H. The solution with larger α_2 may have the greater magnitude when H is large, while the one with larger α_1 dominates it when H is small. (Compare a and c in Figure 3-2.)

Magnitude ordering of projected points depends on H.
$\lambda_1 = 0.04$, $\lambda_2 = 1$. Plot $(x^T V = (X^T U_1 \sqrt{H-.04},\ x^T U_2 \sqrt{H-1})$.

$(a^T U_1,\ a^T U_2)$	$(b^T U_1, b^T U_2)$	$(c^T U_1,\ C^T U_2)$
(0, 0.2)	(0.05, 0.11)	(0.08, 0)

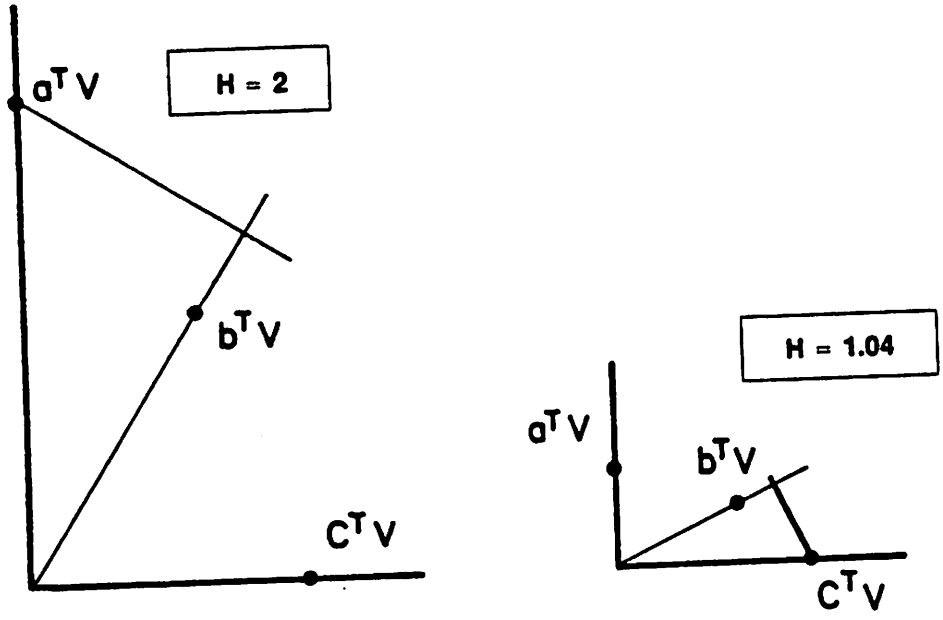


Figure 3-2.

Because the choice of H can alter the values associated with different solutions, it can also affect iteration sequences. Thus in Figure 3-2, a probe directed at point $b^T V$ leads to solution a when $H=2$, but to solution c when $H=1.04$. For certain intermediate values (*e.g.*, $H=1.25$), $b^T V$ is stable.

In summary, different values of H may change our evaluations of placements in R^k. Which placements are stable, and even which is furthest from the origin, may change depending on H. These changes arise from the differences between the squared magnitudes of feasible points and of their projections in R^k. Squared magnitudes in R^{n-1} *equal* $H-\text{cost}(x)$; in R^k they are approximations. The difference between the exact and approximate values is $\sum_{r=k+1}^{n-1} \alpha_r^2(x)[H-\lambda_r]$. For any particular x, we can choose H so that this difference equals 0:

$$\sum_{r=k+1}^{n-1} \alpha_r^2(x)[H-\lambda_r]=0$$

$$[\sum_{r=k+1}^{n-1} \alpha_r^2(x)]\, H=\sum_{r=k+1}^{n-1} \alpha_r^2(x)\lambda_r$$

$$H=\frac{\sum_{r=k+1}^{n-1} \alpha_r^2(x)\lambda_r}{\sum_{r=k+1}^{n-1} \alpha_r^2(x)} \tag{3.2}$$

For other placements, the difference may then be more or less than zero. But the variation is not too great among the best solutions, since for them the k active terms $\alpha_r^2(x)$ tend to dominate.

We can use *NextH*(k,U,n,x,λ) to set H according to (3.2). If U contains all the eigenvectors, then for each $r>k$, we can evaluate $\alpha_r(x) = x^T u_r$ directly. But when $k<n-k$, it is possible to compute H more efficiently. From (2.4), we have $\sum_{r=k+1}^{n-1} \alpha_r^2(x)=1-\sum_{r=1}^{k}\alpha_r^2(x)$. And from (2.2), $\sum_{r=k+1}^{n-1} \alpha_r^2(x)\lambda_r = \text{cost}(x)-\sum_{r=1}^{k}\alpha_r^2(x)\lambda_r$. Since we can evaluate cost(x) using the original C matrix,* all references to eigenvectors $k+1$ through $n-1$ can be eliminated. This may save a great deal of space. More importantly, eigenvectors that are never activated need not even be computed; this may significantly reduce the computational demands associated with determining eigenvectors.

We have experimented with variations of *IterateProbes* that do extra work between probes. For example, we can update H after every probe instead of only between stages; but this yields no significant improvement.

---

* With a reasonable data structure for C, *e.g.*, lists of the connections for each circuit component, matrix storage space and cost-evaluation time are proportional to the number of nonzero entries c_{ij}, which is typically much less than n^2.

Another possibility is to evaluate cost(x) after every probe, and at the end of each stage restore the lowest-cost solution encountered. (Because of the discrepancy between R^k and R^{n-1}, the final point in a stage is not always the best one.) Beginning each stage with the best point from the previous stage gives better final solutions in some trials and worse ones in others. We prefer the version of *IterateProbes* described earlier because it is simpler and slightly faster.

We saw at the end of chapter 2 that $n-2$ dimensions are enough to represent the transformed problem perfectly, so we might typically use this many dimensions in the last stage. Because the later stages seldom introduce major changes, we provide a test that can stop the procedure before then. Think of $\sum_{r=1}^{k} \alpha_r^2(x)$ as the "power" of x that is concentrated in its k-dimensional projection. We begin new stages only until this sum exceeds $1-epsilon$. For example, with $epsilon=0.005$, we forgo refinements in higher dimensions if the current projection already has 99.5% of the power.

In summary, the crucial idea of *Stages* is to work with problem approximations that progress from coarse to refined. This general idea should be applicable to a wide range of problems. The approach shares some qualities with simulated annealing, in its gradual commitment to structural features of the solution. For optimization problems that involve a quadratic form, the ordering of eigenvalues provides valuable guidance. We can design algorithms that concentrate in their early stages on the dimensions with the greatest potential for contributing to the objective function.

3.4. Sparse iteration

In this section we examine our basic iterative improvement step from a different viewpoint, which leads us to a much more efficient way to perform iterations on the full transformed problem in R^{n-1}.

Consider iteration in k dimensions; *i.e.*, take V to be an n-by-k matrix with the active eigenvectors as its columns. Let x be the current solution vector. The next probe direction d^T is $x^T V$; we pick the next solution vector to align with Vd, which equals $VV^T x$. Define an n-by-n matrix of rank k:

$$A = VV^T$$

The squared distance $\|x^T V\|^2$ to the projection of a feasible point in R^k is $x^T V V^T x$, or simply $x^T A x$.

From the definition $v_r = u_r \sqrt{H - \lambda_r}$, we have $A = VV^T = U(H - \Lambda)U^T$, where $H - \Lambda$ is a diagonal k-by-k matrix with entries $H - \lambda_r$. That is, the eigenvalues of A are $H - \lambda_r$.

Given a solution vector x, our iteration step selects the feasible solution vector w that maximizes $w^T VV^T x$. If we had to use explicit matrix multiplication, the association $(VV^T)x$ would be much worse than computing $V(V^T x)$ as in *IterateProbes*. To compute $A = VV^T$ would require $O(n^2 k)$ work, versus $O(nk)$ *total* work previously. But for the special case $k = n-1$, the A matrix is available without performing any multiplications. This is because when all $n-1$ eigenvectors are active, A is identical to the original connection matrix C, except on its diagonal.

To see this, consider the $n-1$-by-$n-1$ diagonal matrices H (with entries equal to the scalar H) and Λ (with entries λ_r). We have $A = U(H - \Lambda)U^T$, and with $k = n-1$ we also have $B = U \Lambda U^T$; thus

$$A = H - B.$$

From definition (2.0) we can substitute $B = D - C$, where D is a diagonal matrix with entries equal to the row sums of C:

$$A = C + (H - D).$$

Thus when $k = n - 1$, we can compute Ax in time proportional to the number of nonzero entries of C. Because C is usually sparse, we obtain an efficient algorithm to iterate in $n - 1$ dimensions, using the A matrix just defined:

```
procedure SparseIteration ( (* INPUTS *) A, n, L; (* INPUT&OUTPUT *) x);
  {INPUTS:
  A is a square matrix, with only the nonzero entries stored.
  n is the order of A, and the length of column vectors in general.
  L is a column vector: the legal positions in nondecreasing order.
  INPUT&OUTPUT:
  x is the initial placement vector (permutation of L);
       it is modified in a sequence of iterations, until it stabilizes.
  LOCAL VARIABLES:}
  i: integer;    p,previousX: column;    rank: array[1..n] of integer;
  begin (* SparseIteration *)
    repeat
      previousX := x;
      p := Ax;
      Obtain array rank such that p[rank[1]]≤p[rank[2]]≤ · · · ≤p[rank[n]];
      for i := 1 to n do x[rank[i]] := L[i];
    until x=previousX;
  end; (* SparseIteration *)
```

If $|E|$ is the number of connections in the C matrix, the running time is $O(|E| + n\,log(n))$ per pass. A typical use of *SparseIteration* is to improve an output x from *Stages*. The effect is identical to that of a stage with $k = n - 2$ and $steps = \infty$.

The operation of multiplying a solution vector by the A matrix produces an effect similar to that of a relaxation step in other placement heuristics. Writing A as $H + (C - D)$, we get the following expression for the typical component of Ax:

$$(Ax)[i] = H \cdot x[i] + \sum_j c_{ij}(x[j] - x[i]) \tag{3.3}$$

In words, each component is ranked according to a constant multiple of its current position plus a vector sum of "forces" exerted by the other components. The magnitude of each force equals the product of the distance to the other component and the weight of the connection with that component.

Our scheme differs from past relaxation techniques. Other heuristics ignore the constraints associated with legal positions during relaxation. At best, they introduce fictitious repulsive forces to discourage components from being placed on top of each other. By contrast, we have a feasible placement at every step. Our procedure moves the components from the positions $(Ax)[i]$ onto the legal positions, preserving their relative order.

In general this process need not converge (the placement may oscillate between configurations). But if we choose $H \geq \lambda_{n-1}$, this problem cannot arise in *SparseIteration*. We show this by proving that the step from x to w can only increase the magnitude of the corresponding point ($w^T A w \geq x^T A x$). Because the eigenvalues of A are $H - \lambda_r$, A is positive semidefinite. Thus we have $(w^T - x^T)A(w - x) \geq 0$, or

$$w^T A w + x^T A x \geq 2 w^T A x .$$

Add the probe property

$$2 w^T A x \geq 2 x^T A x .$$

Cancelling like terms, we have

$$w^T A w \geq x^T A x .$$

This proof applies as well to iterated probes in k dimensions; when H is as large as the λ_r of every active eigenvector, VV^T is positive semidefinite, and the magnitudes of solution points in R^k increase monotonically. Although this guarantees that every iteration will converge, it may be

unnecesarily conservative to require monotone improvement. For example, in sparse iteration the current x contributes with a factor of H to Ax (equation 3.3): steps with $H \leq \lambda_{n-1}$ would be less dominated by the current solution, and thus freer to change significantly.

We can compute (3.3) for any real value of H. But in k dimensions, it is not obvious how to carry out probes when the restriction $H \geq \lambda_r$ is violated. Suppose that some active eigenvalues are smaller than H and others are larger.* The transformed components $x^T u_r \sqrt{H-\lambda_r}$ would then divide into the purely real and purely imaginary, respectively. The problem would no longer map into R^k; but we could use an algebraic representation with a total of k axes, some real and some imaginary. Restricting the corresponding components of probe directions d^T to purely real or purely imaginary values would insure that Vd remains a real vector, since the imaginary vectors of V would all be multiplied by imaginary components of d. When iterating, the appropriate restrictions arise naturally from $d_r = x^T v_r$.

We can thus extend the application of probes by experimenting with $H < \lambda_{n-1}$ in sparse iteration; and in k dimensions, by using imaginary vectors v_r to represent terms with $H - \lambda_r < 0$. In each case VV^T has negative eigenvalues, which makes steps that decrease $x^T VV^T x$ possible. Still, in practice this function may increase as much or more under these conditions as when we enforce $H \geq \lambda_r$: when it does not, it is easy enough to stop iterating or to increase H.

---

*When the active eigenvectors are not those with the smallest λ_r's, we may want to represent the other dimensions with an H that falls between the active eigenvalues. Chapter 5 discusses some situations where this may occur.

3.5. Summary

In this chapter we developed improvement heuristics that use iterated sequences of probes for one-dimensional placement. All running times per trial are at most $O(n^2 log(n))$. If we discount the one-time cost for a given circuit of computing the eigenvectors of B**, this is competitive with the run-times of traditional approaches such as exhaustive pairwise interchange. Chapters 5 and 6 will extend our methods to handle constrained components and two-dimensional placements.

Computational experience with these heuristics has been encouraging. In chapter 7, for a variety of problem instances we obtain lower-cost placements than are obtained by other approaches.

These heuristics generate good placements on their own; in addition, these solutions can serve as starting points for other improvement procedures. Our problem transformation also stimulates the development of new improvement heuristics. For example, to escape from local optima, we can take a good point discovered by any probe and perform new probe sequences starting with various directions in its vicinity. Another heuristic takes two or more directions associated with good points and solves for the furthest point in a projected space spanned by these directions. (Given two directions, we can do so in time $O(n^2 log(n))$.) These ideas will be explored in future work.

**As discussed in chapter 7, most procedures to compute all eigenvectors are $O(n^3)$, but significant speedups are possible for sparse matrices.

Chapter 4

4. Proving Lower Bounds on Placement Cost of a Given Circuit

In chapter 2 we transformed the circuit placement problem into a search for the furthest point from the origin in a multidimensional space. For the resulting furthest-point problem we introduced a search tool, the "probe", which finds the point with maximum projection on any given direction. In chapter 3 we used sequences of probes to discover points with large magnitude, which correspond to low-cost placements. In this chapter we will describe how probes can serve the complementary purpose of proving lower bounds on placement cost of a given circuit.

We will be more satisfied with our heuristic placements if we can show that much better ones are not possible. This is our motivation to seek lower bounds on cost. In the transformed problem, these correspond to upper limits on how far points can lie from the origin. By working with certain k-dimensional projections, we can rigorously derive such limits for the full-dimensional furthest-point problem.

If we set H to λ_{k+1} in equation (2.5), we have:

$$\lambda_{k+1} - \text{cost}(x) = \sum_{r=1}^{n-1} \alpha_r^2(x)[\lambda_{k+1} - \lambda_r].$$

Terms with $\lambda_r \geq \lambda_{k+1}$ are never positive, so only the first k terms can make a positive contribution to the sum. Therefore,

$$\lambda_{k+1} - \text{cost}(x) \leq \sum_{r=1}^{k} \alpha_r^2(x)[\lambda_{k+1} - \lambda_r]. \tag{4.1}$$

Letting $v_r = u_r\sqrt{\lambda_{k+1} - \lambda_r}$, the sum equals the squared magnitude in k dimensions of the point $x^T V$. For any given circuit, we thus obtain lower bounds on its placement cost by deriving upper bounds on $\|x^T V\|^2$ in R^k.

To apply probes to prove upper bounds on the possible magnitude of points in R^k requires different strategies than when our aim was to find distant points. Each probe returns a point with maximum projection in a given direction. That point now interests us less than the guarantee that no other point lies past it. We want to collect enough such guarantees to prove that no point can be found in *any* direction beyond a certain distance from the origin.

To do so we must cover the search space globally. Rather than exploring isolated regions in depth (as in iterated probes), we need broad, provably thorough coverage. We can frame this problem in terms of general point sets p^T and probe directions D_i^T in R^k. Most of the analysis depends neither on where the points come from nor on how probes are conducted.* Each probe direction D_i^T yields M_i, the smallest real number that satisfies $M_i \geq p^T D_i$ for all points p^T. We want to select collections of probes D_i^T so that the resulting inequalities imply that every $\|p^T\|$ is at most B, with B as small as possible.

In section 4.1 we consider nonadaptive strategies, in which the set of probe directions does not depend on the results of any probes. In section 4.2 we discuss adaptive methods, in which the results of earlier probes may be considered when selecting the directions to use for later ones. Finally, in section 4.3 we analyze specific features of furthest-point problems that arise via our transformation from circuit-placement problems. In particular, we show that in a k-dimensional projection, the furthest point from the origin can be found with a number of probes that grows as a polynomial function

---

*e.g., our *derivation*, in which $p^T = x^T V$ for x a permutation of L, is irrelevant.

of n, the number of circuit components.

4.1. Nonadaptive probe sets

4.1.1. Axis probes

The simplest way to use probes to obtain an upper bound on $\|x^T V\|^2$ is to find the maximum possible value of each term $(x^T v_r)^2$ independently. The maximum of $\|x^T V\|^2$ is certainly no more than the sum of the maximum values attainable in each component. We probe in both directions along each axis u_r. Since Vd then equals a single eigenvector $\pm v_r$, the placement x that maximizes $x^T Vd$ has the $x[i]$'s rank-ordered to match the ordering of eigenvector components $u_r[i]$.

For general point sets, probes in opposite directions give independent information. But suppose we start with a symmetric placement problem, in which l is a legal position only if $-l$ is also. In this case the set of transformed points is symmetric about the origin. Let x be the feasible vector whose component ordering aligns with Vd. Then $-x$ is feasible, aligns with $-Vd$, and yields the same inner product $x^T Vd$. In other words, a single probe effectively covers directions d^T and $-d^T$ simultaneously. Symmetric problems thus require half as many probes; in particular, we can obtain a tight upper bound on $(x^T v_r)^2$ from one probe on axis r.

For some circuits, the use of a few axis probes provides a lower bound on cost that is far better than λ_1. The lower bound improves with more axes – to a point. Starting with inequality (4.1), we will derive the condition for when an additional axis improves the bound. Let f_r be the maximum value of ${\alpha_r}^2(x)$, which we obtain by probing along axis r. Using k axes, we have

$$\lambda_{k+1} - \text{cost}(x) \le \sum_{r=1}^{k} f_r[\lambda_{k+1} - \lambda_r], \text{ or}$$

$$\text{cost}(x) \ge \lambda_{k+1} - \sum_{r=1}^{k} f_r[\lambda_{k+1} - \lambda_r] = \sum_{r=1}^{k} f_r\lambda_r + (1 - \sum_{r=1}^{k} f_r)\lambda_{k+1} \equiv "A_k". \qquad (4.2)$$

The k-axis lower bound (A_k) is a weighted average of λ_1 through λ_{k+1}. Now $A_k - A_{k-1} = (1 - \sum_{r=1}^{k} f_r)(\lambda_{k+1} - \lambda_k)$. Since $\lambda_{k+1} \ge \lambda_k$, the lower bound increases with additional axes only so long as $\sum_{r=1}^{k} f_r < 1$.

4.1.2. Regular coverage of R^k

The upper bound on $\|x^T V\|^2$ derived from axis probes is seldom very tight, since we give so much away in assuming that every component $x^T v_r$ might simultaneously reach its maximum value. The furthest point in R^k typically has much smaller magnitude than this bound allows. To prove closer bounds, we use more probes.

Extra probes extend the portions of R^k known to be devoid of points. Our upper bound is associated with the furthest place from the origin that probes have not excluded. We seek a fixed set of probes that will, for any problem, yield an upper bound that is close to the magnitude of the furthest point p_*^T. To do so, we must be sure that every direction has a probe nearby.

The idea is to cover R^k with enough unit-length probe directions D_i^T that one of them must come near p_*^T. Each probe yields M_i, the maximum value of $p^T D_i$ over all points p^T. The Euclidean length $\|p_*^T\|$ equals the maximum value of $p_*^T d$ over all vectors d^T of unit length. Thus $Max\{M_i\} \le \|p_*^T\|$; we obtain an *upper* bound by proving that the gap in this inequality is small.

By simple geometry, if D_i^T falls within an angle φ of p_*^T, the corresponding projection satisfies $M_i \geq \|p_*^T\| \cos\varphi$. For a concrete example, suppose we probe R^2 at 1° intervals. *Some* probe D_i^T must come within $^\circ$ of p_*^T. Then

$$\|p_*^T\|^2 \leq M_i^2[1+\tan^2(^\circ)]: \tag{4.3}$$

the upper bound is within 2.5% of $\|p_*^T\|^2$.

4.1.2.1. Asymptotic number of probes required

We are thus led to the following problem: how many probes do we need in R^k and how should we arrange them, so that every direction is within φ radians of some probe? In R^2 the optimal solution is trivial; lay out $\lceil \pi/\varphi \rceil$ probes at intervals of 2φ radians. The general problem in k dimensions is harder. Let the minimum number of probes needed be $P_k(\varphi)$.

We first show that for fixed φ, the number of probes increases exponentially with the number of dimensions: in particular, for all $k \geq 2$,

$$P_k(\varphi) > 2\sqrt{k}\ (\frac{1}{\varphi})^{k-1}. \tag{4.4}$$

To prove (4.4), observe that any probe covers those directions within φ radians of itself. Let $a_k(\varphi)$ denote the area of a spherical cap with half-angle φ on the unit sphere in R^k. Then if μ_k is the area of the entire unit sphere, we cannot possibly cover it without using at least $\mu_k / a_k(\varphi)$ probes. Now

$$a_k(\varphi) = \mu_{k-1} \int_0^\varphi \sin^{k-2} x\, dx. \tag{4.5}$$

The areas μ_k are given by $\mu_1 = 2$, $\mu_2 = 2\pi$, and a recurrence for $k \geq 2$: $\mu_k = \frac{2\pi}{k-2} \mu_{k-2}$. We have

$$P_k(\varphi) \geq \mu_k / \left[\mu_{k-1} \int_0^\varphi \sin^{k-2} x\, dx\right].$$

We get a lower bound on this ratio from a simple upper bound on the integral: $x > \sin x$ for $x > 0$. This implies

$$P_k(\varphi) > \frac{\mu_k}{\mu_{k-1}} \cdot \frac{1}{\varphi^{k-1}/(k-1)}.$$

Since $\frac{\mu_k}{\mu_{k-1}} > \frac{2\sqrt{k}}{k-1}$ for all $k \geq 2$, inequality (4.4) follows.

To cover the unit sphere in R^k requires more than $\mu_k / a_k(\varphi)$ caps, because caps necessarily overlap. We can see this from the smaller asymptotic growth rate of $N_k(\varphi)$, the maximum number of mutually disjoint caps that may be placed on the surface of the unit sphere in R^k. For example, for $\varphi < \pi/4$, [Rankin55] gave an upper bound on $N_k(\varphi)$ that is of the order of

$$k^{3/2}\left(\frac{1}{\sqrt{2}\sin\varphi}\right)^{k-1},$$

and [Sidel'nikov73] lowered this bound slightly.

We now give a method to construct explicit probe sets in R^k. The lattice Z^k comprises the points in R^k whose coordinates are integers. Let the m-sphere be those points whose squared distance from the origin equals m. Suppose we direct a probe at any point of Z^k if the unit cube centered there intersects the m-sphere. Thus, consider the probe set

$$S = \{x \in Z^k : \sum_{r=1}^{k} (x_r - \tfrac{1}{2})^2 \leq m \leq \sum_{r=1}^{k} (x_r + \tfrac{1}{2})^2\}.$$

We can choose m to guarantee any given φ. Every point on the m-sphere lies within a distance $\sqrt{k}/2$ of some point in S. It thus suffices to pick m so that $\arcsin\left[\sqrt{k}/(2\sqrt{m})\right] \leq \varphi$, or

$$\sqrt{m} \geq \sqrt{k}/(2\sin\varphi). \tag{4.6}$$

Let us estimate the number of probes required. We first count the number of spherical "shells" that can contribute lattice points to S. A t-sphere can have points in S only if

$$(\sqrt{m} - \frac{\sqrt{k}}{2})^2 \leq t \leq (\sqrt{m} + \frac{\sqrt{k}}{2})^2,$$

for a total of $2\sqrt{mk}$ shells, asymptotically. There are $\Theta(\sqrt{m}^{k-2})$ points per shell. Substituting the value of $\sqrt{m}$ from (4.6), we have on the order of $\left[\frac{\sqrt{k}}{2\sin\varphi}\right]^{k-2}$ points per shell, times $\frac{k}{\sin\varphi}$ shells, for

$$\text{a total of } O\left[\frac{\sqrt{k}}{2\sin\varphi}\right]^{k} \text{ probes.} \tag{4.7}$$

We might hope to achieve the same coverage with smaller sets of probes. To improve the above construction, one approach would be to use lattices other than Z^k. In general a lattice is defined by any set of linearly independent vectors in R^k; it comprises all sums of integer multiples of these vectors. The use of lattices to cover all directions in R^k does not seem to have been investigated, although the literature treats several related problems. For example, to find dense packings of spheres or sparse coverings of space by overlapping spheres, a common idea is to center the spheres at points of a lattice.

The idea of using a general lattice to supply useful sets of directions *has* been exploited in packing problems. In particular, [Sloane81] shows that to obtain a k-dimensional spherical code (a set of directions, no two of which form a small angle), one can take the lattice points that lie on an appropriate m-sphere.

Our use of cube center-points in Z^k was similar. To cover arbitrary directions, we took points from several shells to be probes. Generalizing

that construction, consider any lattice covering of space. We could take as probes the center-points of a set of bodies that together cover the surface of a sphere. Unfortunately, using a different lattice is unlikely to give us much more efficient probe sets.

The problem is that for common families of lattices, the minimum distance between points is the same for all k, but the maximum distance of any point in space from the lattice is proportional to $\sqrt{k}$. As a result, the number of probes needed by lattice constructions tends to grow as $(\sqrt{k}/c_\varphi)^k$, as in (4.7). (c_φ is a real number that depends on φ.) Some lattices may achieve larger values of c_φ than others, but as yet there is no prospect of avoiding the factor of $\sqrt{k}^k$.

4.1.2.2. Random probes

The size of explicit probe sets based on lattices grows much faster as a function of k than the lower bound (4.4) might allow. It is natural to ask: do satisfactory probe sets exist with sizes closer to this bound? Adapting a result of [Rogers63], we can answer this question affirmatively:

THEOREM (Rogers): If $\varphi < \arcsin(1)$ and $k \geq 9$, then there exists a covering of the unit sphere in R^k by P spherical caps of half-angle φ if P is at least

$$\frac{\mu_k}{a_k(\varphi)}\left[k\log k + k\log\log k + k\log\left(\frac{1}{\sin\varphi}\right) + \tfrac{1}{2}\log(16k)\right]\left[1-\frac{2}{\log k}\right]^{-1}.$$

Rogers also gives an upper bound on $\frac{\mu_k}{a_k(\varphi)}$ that closely matches (4.4):

$$\frac{\mu_k}{a_k(\varphi)} < 4\sqrt{k}\left(\frac{1}{\sin\varphi}\right)^{k-1}. \tag{4.8}$$

Together these observations imply, for fixed φ and large k,

$$P_k(\varphi) = O\left(k^{3/2}\log k\left(\frac{1}{\sin\varphi}\right)^{k-1}\right).$$

This encouraging result is nonconstructive. That is, Rogers does not supply efficient probe sets; he only proves that they must exist. His reasoning nicely illustrates the probabilistic method, which often tells us that desirable objects exist without helping us construct them. He first defines an $\varepsilon>0$ that depends on φ and k. He shows that, averaged over the sample space of all possible probe sets, the mean value of surface area left uncovered by P caps of $\varphi-\varepsilon$ radians is less than the area of a single cap of ε radians. This can only be true if at least one of the probe sets, not identified explicitly, has this property. For that set, it follows that every point on the sphere must lie within φ radians of a probe.

By a similar argument, relatively small sets of independently chosen random probes provide usable guarantees with high probability. Our object is to insure that, wherever the furthest point lies, some probe falls within φ radians of it. The probability that every one of P independent random probes will miss the furthest point by more than φ radians equals $\left[1-\frac{a_k(\varphi)}{\mu_k}\right]^P$. Since $1+x\le e^x$ for any real x, we can upper-bound this failure probability by $e^{-P\frac{a_k(\varphi)}{\mu_k}}$. For example: for any k, conducting $5\mu_k/a_k(\varphi)$ probes reduces the probability of an unsatisfactory set to less than e^{-5}. To increase the certainty of success, simply pick a constant greater than 5.

It remains only to show how to generate random directions on the k-dimensional unit sphere. [Muller59] gives a simple method:

1) Generate k independent normal deviates x_r, for r = 1 to k.
2) Use the direction d^T given by $d_r=x_r/(\sum_{r=1}^{k}x_r^2)^{\frac{1}{2}}$.

Normal deviates are numbers gotten by sampling a standard normal distribution. To simulate this process on a computer, it suffices to have two commonly available functions: one to give pseudo-random numbers, and one to

invert the normal distribution function Φ. We first generate a number p from the uniform distribution on [0,1]. We then obtain the deviate, x, for which

$$\Phi(x)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} e^{-u^2/2}du = p$$

Muller's method gives points distributed uniformly on the unit sphere because the probability density function of the vectors x generated in step 1 (k-dimensional normal) is constant at any fixed distance from the origin, independent of direction.

4.1.3. Practical shortcuts

In the previous section we saw that the number of probes needed to cover R^k is an exponential function of k. The computational demands of covering k dimensions may therefore be excessive even for small values of k, say 10 or 20. While this analysis fundamentally limits the potential of fixed probe sets for multidimensional furthest-point problems, we are still interested in making probes as efficient as possible. We now present several techniques designed to optimize special classes of probe sets.

4.1.3.1. Partitioning

The object of graph partitioning is to split n components into two equal-sized groups, minimizing connections between the groups. We represent this as a circuit placement problem by using an L vector of legal positions with just two values, each repeated $\frac{n}{2}$ times. Recall that each probe computes Vd for a given direction d^T, and arranges the legal positions into a vector whose order matches that of Vd. To permute the legal positions appropriately, we in general need a total ordering of the

components of Vd, but for partitioning it suffices to separate the largest $\frac{n}{2}$ components from the smallest. (This division fully characterizes a feasible placement, since there are no distinctions among components within each half.) To determine the partition x that maximizes $x^T Vd$ we thus compute the median value of Vd, which requires $O(n)$ time rather than the $O(n \log n)$ required to sort the components.

When we seek the maximum value of $x^T Vd$ over a large set of probe directions (as in upper-bounding $\|x^T V\|$ in R^k), still greater savings are possible. Let l,s be the values of the legal positions ($l > s$). For direction d^T, the maximizing inner product satisfies

$$x^T Vd = l \sum \left[\frac{n}{2} \textit{ largest components of } Vd \right] + s \sum \left[\frac{n}{2} \textit{ smallest components} \right].$$

For $r \neq 0$, the components of v_r sum to 0 ($v_r{}^T u_0 = 0$); since Vd is a linear combination of v_r's, its components also sum to 0. Thus

$$x^T Vd = (l-s) \sum (\tfrac{n}{2} \textit{ largest components }).$$

Now $(l-s)\sum(\textit{positive components of } Vd)$ is a good upper bound on $x^T Vd$. It is an upper bound because the positive elements in a set of real numbers always constitute the subset whose sum is largest. It is a *good* bound because we expect Vd to have roughly $\frac{n}{2}$ positive components, since its component sum equals 0. Comparing components with zero instead of the precise median element makes the computation trivial.

Over a *set* of probes, the maximum $(l-s)\sum(\textit{positive components of } Vd)$ is an upper bound on $x^T Vd$ for the set. However, a little extra work gives the exact maximum ("m"). The idea is best expressed as a fragment of code that computes m.

m := 0;
for each probe d^T **do begin**
  up := $(l-s)\sum$(positive components of Vd)
  if up > m **then** m := Max [m, $(l-s)\sum(\frac{n}{2}$ largest components of $Vd)$]
end; (* for *)

We compute x^TVd exactly (by finding the median component) only when the computation has a chance of yielding the maximum value for the set.

4.1.3.2. Sets of related probes

For a special case of the placement problem, the previous section shows how to speed up or bypass some probe steps that use the vector Vd. We now turn to the computation of Vd itself, namely, the linear combination of k vectors v_r. We count vector operations, bearing in mind that for vectors in R^n, a scalar multiplication or addition requires n real arithmetic operations. Each vector v_r is multiplied by a scalar d_r, and the k results are summed. Thus we normally use k multiplies and $k-1$ adds to compute Vd.

Because we can use any scalar multiple of d^T, $k-1$ multiplies actually suffice. Let c equal some nonzero d_r. Suppose that at the start, we reset each d_r to d_r/c. Our use of Vd is to take its inner product with a legal vector. If we multiply this inner product by c, we recover the result we would have obtained with the original d_r's; this device saves one vector multiply, since it sets one $d_r=1$.

For general probe sets, and in particular for sets of random probes, the computation of Vd must proceed independently for each direction d^T. But if we use a set of related directions, the same arithmetic operations may appear in the computations required by several probes. With careful organization, we can re-use intermediate results common to different probes instead of recomputing them.

The easiest way to understand related probes is to consider an example. We shall study an algorithm that computes Vd for every probe in a specific set of $\frac{1}{2}(3^k-1)$ vectors d^T in R^k, for any $k \geq 1$. This illustrates the basic ideas in a form that is practical to implement. Compared to general probes, the algorithm saves a factor of $2(k-1)$ operations in computing each Vd. (It uses no multiplications and at most one addition per probe.)

We consider the 3^k probes in R^k in which each component is either 0 or ± 1, and discard the case with k 0's. It is convenient to identify these probes with the first 3^k-1 positive integers in base 3, which require up to k ternary digits. Places 0 through $k-1$ of the integer represent the probe components: in each place we let the digit 2 stand for a component of -1, 0 stand for 0, and 1 stand for 1.

The probe set is symmetric about the origin: if we compute Vd, there is no reason to compute $V(-d)$ separately. We thus omit half the integers, namely those whose leading digit is a 2. Our algorithm steps a control variable c through the remaining integers. We refer to the digit in the pth place of c as $Dit(p,c)$; that is,

$$Dit(p,c) = \lfloor (c \bmod 3^{p+1})/3^p \rfloor .$$

The inputs (*e.g.*, vectors) appear in the array $v[0..k-1]$. At each step we compute the combination of inputs determined by the probe associated with c, and place it in the output array $sum[0..k-1]$. Each new combination can be computed by adding one input (or its negative) to an already computed combination that is available in the array of outputs.

```
for j := 0 to k-1 do begin              (* j-digit integers, base 3 *)
   for c := 3^j to (2* 3^j)-1 do begin (* with lead digit = 1 *)
      (* "place" to update is smallest integer ≥0 for which Dit(place,c)≠0 *)
      place := 0; while Dit(place,c)=0 do place := place+1;
      if 3^place=c then sum[place] := v[place]
      else begin
         (* Use sum stored in next highest place with nonzero digit *)
         old := place+1; while Dit(old,c)=0 do old := old+1;
         case Dit(place,c) of
            1: sum[place] := sum[old]+v[place];
            2: sum[place] := sum[old]-v[place];
         end; (* case *)
      end; (* else *)
      (* sum[place] now has new combination of v's *)
   end; (* for c *)
end; (* for j *)
```

How well do these probes cover R^k? The coverage of a probe set S may be defined by

$$\cos\varphi(S) = \underset{x}{Min}[\underset{p\in S}{Max}\cos(x,p)], \tag{4.9}$$

where the minimum is over all nonzero vectors $x \in R^k$. For the furthest-point problem, this gives the upper bound

$$\|p_*^T\|^2 \le M_i^2[1+\tan^2\varphi(S)]\,, \tag{4.10}$$

as in (4.3): here M_i is the largest projection obtained by a probe. Thus $\tan^2\varphi(S)$ represents the worst-case relative error associated with (4.10), since $M_i \le \|p_*^T\|$. Typically we will observe that the magnitudes of some points returned by probes exceed M_i, and thus obtain a larger lower bound for $\|p_*^T\|$. In these cases the *a posteriori* relative error in (4.10) is reduced accordingly.

For comparison, we compute $\tan^2\varphi(S)$ for the set of $2k$ axis probes. The worst-covered x's have the same absolute value in each component: *e.g.*, $(\pm1,\pm1,\pm1)$ in R^3. Cosine $\varphi(S)=1/\sqrt{k}$; $\tan^2\varphi(S)=k-1$.

Suppose we add only the 2^k worst-covered vectors (with ± 1 in each component) to the set S. Let k be a perfect square; consider vectors with $\sqrt{k}$ components equal to 1 and the rest equal to zero. The maximum cosine that such a vector makes with any probe equals $1/k^{\frac{1}{4}}$; thus $\tan^2\varphi(S) \geq \sqrt{k} - 1$.

In Appendix 3 we prove that for the example set of $3^k - 1$ directions considered above,

$$\tan^2\varphi(S) < \frac{.256 + ln(k)}{4}. \tag{4.11}$$

This error bound grows relatively slowly with k: $\tan^2\varphi$ does not reach 1.0 until k=43. However, for k>11, random probes provide better coverage more efficiently, even if we allow for the speedup factor of $2(k-1)$ achieved by this set of related probes.

In a wide range of probe sets, we can use incremental computing as above to share operations among probes. For example, it is easy to generalize the above sequence to sets of c^k probes, in which each component assumes c values. Sharing operations can speed up *any* set of probes drawn from lattice points.

Speedup is not much justification for working with related probes. For one thing, saving a factor of $2(k-1)$ is the best we can hope for. And the possibility of having to design new probe sets and algorithms when k and φ change seems like an unnecessary inconvenience, since simple random probes require no such work.

Still, clever designs of organized probe sets may pay off, by letting us dispense with unnecessary probes. For example, on the basis of earlier unproductive probes in the neighborhood, it may be possible to prove, without trying them, that no probe in a whole cluster of directions can improve on the current best point.

4.1.3.3. Parallelism and probes

New computers make it feasible to use many processors that work in parallel. Sequential processing may remain the standard technology; but it is interesting to study the extent to which different computations lend themselves to parallel processing. Probes definitely have the potential to make efficient use of multiple processors.

Each probe computes Vd by combining k vectors v_r. This entails an independent inner product computation of length k in each of the n components. All nk scalar multiplications involved could be performed in parallel, as could the n subsequent component sums. Any of several known strategies for parallel sorting could then be applied to sort Vd. Finally, the inner product computation $x^T(Vd)$ could use up to n processors.

An obvious way to exploit even more processors is to perform multiple probes simultaneously. The requirement for large probe sets established in (4.4) means essentially that we can find work for as many parallel processors as are available.

4.1.3.4. Choosing k wisely

To obtain close upper bounds on the summation in (4.1), the simplest nonadaptive probing strategy is to use randomly generated directions in R^k. We now show how to estimate which k will result in the best lower bound on placement cost. The goal is to avoid time-consuming probe sets that use sub-optimal values of k.

For each k, let ${B_k}^2$ be the upper bound on $\sum_{r=1}^{k}(x^T u_r)^2[\lambda_{k+1}-\lambda_r]$ that we expect to derive from probes in R^k; write (4.1) as

$$\text{cost}(x) \geq \lambda_{k+1} - {B_k}^2. \tag{4.12}$$

The λ's are known; good approximate values for ${B_k}^2$ allow us to make an informed guess as to which k will maximize (4.12). For each k, to estimate

$$ {B_k}^2 = M_i^2(1+\tan^2\varphi), \tag{4.13} $$

we need to know two things: 1) φ, the angular coverage in R^k; and 2) an estimate of M_i, the maximum projection we expect to find.

1) To compute the angle φ, we first specify the number of probes ("P") that we could perform in R^k, by dividing the time allowed for the whole set by the time per probe in R^k. We can then easily find φ for which $5\frac{\mu_k}{a_k(\varphi)} = P$ (here we assume that the final bound from the probe set should be correct with probability at least $1-e^{-5}$).

2) We now give a heuristic to estimate M_i for a given k in (4.13) without performing the whole set of probes. Recall that this is only a "guessing" stage, designed to tell us which k to select: the actual lower bound will be proven in a "verifying" stage, using the recommended value of k. We borrow an idea from the last chapter. From a small sample of iterated probe sequences, take the magnitude of the furthest point; it approximates the maximum projection that will arise if we cover R^k with probes.

In conclusion, we can use the observed distribution of λ's and the computed dependence of coverage angle on k, together with a small sample of outputs from furthest-point heuristics, to determine in which search space R^k random probes are likely to provide the best lower bound.

It is also sometimes possible to *prove* that other choices of k are bad. To do so, we want to prove limits on how large the lower bound (4.12) in an alternative search space R^k could become if we flooded it with probes. As $\varphi \to 0$ in (4.13), ${B_k}^2 \to M_i^2 = \|p_*^T\|^2$, where p_*^T is the furthest point in R^k. We

can thus define an "asymptotic" value of (4.12):

$$L_k = \lambda_{k+1} - \|p_*^T\|^2. \tag{4.14}$$

Any point p^T (*e.g.*, from iterated probes) provides the upper limit

$$\lambda_{k+1} - \|p^T\|^2 \geq L_k. \tag{4.15}$$

4.2. Adaptive Methods

The approach in section 4.1 was to use a fixed probe set for any problem instance. To guarantee that some probe would always come near the furthest point, we distributed probes uniformly throughout the search space. We now consider adaptive probing strategies, which use results of early probes to help decide where to allocate later probes. In most cases, we can learn more from probes chosen adaptively than we would from a fixed probe set of comparable size.

Because nonadaptive techniques allow no probe choice to be influenced by any others, the probes may yield much redundant information. This is an obvious weakness of random probe sets. With an adaptive approach, we can avoid unnecessary probes: what is more, we can develop strategies to find the optimal (furthest) point in any finite set using only a finite number of probes.

4.2.1. Probing for the convex hull

Consider a finite point set S in R^k. Suppose we could blanket R^k with an infinite number of probes, covering all possible directions: what would we learn about S? Each probe returns a perpendicular hyperplane* that

*For a nonzero $d \in R^k$ and real c, the set of points p^T that satisfy $p^T d = c$ is a *hyperplane*; hyperplanes are the generalization of lines in R^2 and planes in R^3.

contains a point of S and is the boundary of a halfspace that includes all of S. The intersection of all such halfspaces is the smallest convex set that contains S, and is known as the convex hull of S, or $conv(S)$. The convex hull of a finite set of points is called a polytope.

We will see that to find a point in S at maximum distance from the origin, it is sufficient (though not always necessary) to fully determine the polytope $P = conv(S)$. Polytopes are specified in terms of their boundaries. The hyperplanes mentioned above, which contact and bound S, also contact and bound P; they are said to *support* P. The intersection of P with a supporting hyperplane is called a face. In R^k, two extreme cases are especially important: faces of dimension 0 ("*vertices*"), when the intersection is a single point; and faces of dimension $k-1$ ("*facets*"), when the intersection is contained in no other hyperplane.

Our motivation to determine the polytope $P = conv(S)$ is that some vertex of P maximizes distance from the origin among all points of S. This follows from three elementary facts: 1) squared distance from the origin is a convex function; 2) the maximum of a convex function on a polytope is attained at one of its vertices; and 3) the vertices of $conv(S)$ are all points of S. We can thus solve the furthest-point problem on any set for which we can construct the convex hull, because once the vertices are computed it is easy to find one that is furthest from the origin.

The conclusion of the thought experiment with an infinite number of probes is that determining the vertices of $conv(S)$ is enough. This gives us some reason to be hopeful. For instance, when we transform a circuit placement problem into R^k for k much less than n, S comprises $n!$ points but $conv(S)$ has far fewer than $n!$ vertices. (See section 4.3.) But the question remains: is it possible to determine the convex hull from a reasonable

number of probes?

Our problem differs from traditional formulations of the convex hull problem. Usually one is given m points and asked to enumerate all facets of their convex hull, or given m halfspaces and asked to enumerate all vertices of their intersection. In each case the best known algorithms in R^k for $k \geq 4$ are due to [Seidel86]. They require the solution of m linear programs with $m-1$ constraints, plus time $O(k^3|F|\log m)$ to enumerate $|F|$ facets or $O(k^3|V|\log m)$ to enumerate $|V|$ vertices. (From now on we use F to denote sets of facets and V to denote sets of vertices, with $|F|$ and $|V|$ denoting the respective sizes of these sets.)

In our problem, neither points nor halfspaces are provided at the outset. Instead we must determine the polytope P by performing probes in directions that we select; each one finds the perpendicular hyperplane h that supports P, and a point in $h \bigcap P$. Probing can be thought of as consulting an "oracle" that solves linear programs of the form $\left[Max\ v^T d\colon v^T \in P\right]$ for any requested $d \in R^k$. Our primary concern is to bound the worst-case number of probes necessary to determine a polytope.

[Dobkin86] studies the problem of determining polytopes with probes, using a variety of probe models: of these, the moving hyperplane or "hand probe" is the model that most resembles our own. A hand probe returns the supporting hyperplane h perpendicular to a given direction, without telling which point or points of h contact the polytope. Under this model Dobkin *et al.* show that $|V|+|F|$ probes are necessary and $(k+2)|V|+|F|$ probes are sufficient to determine a polytope.

Compared to hand probes, our probes are more powerful (each one returns a polytope vertex) and thus easier to use. We now show how to find all the vertices and facets of a polytope using only $|V|+|F|$ probes.

The idea is simple: find $k+1$ vertices, compute *their* convex hull, and "conjecture" that $|V|=k+1$. Try to verify this conjecture by directing probes normal to the facets of the current convex hull. If a probe returns a supporting hyperplane that contains a conjectured facet, that *facet* is verified. Otherwise the probe necessarily discovers a new vertex beyond the conjectured facet. In that case add 1 to the conjectured size of V, update the convex hull to include the new vertex, and begin testing the new conjecture. The conjecture is proven when all facets are verified.

We now present the algorithm in more detail. The algorithm assumes that a procedure to perform probes is available; we first specify the behavior of this procedure.

{**procedure** Probe ((*INPUT*) d; (*OUTPUTS*) v, H);
INPUTS:
d is a vector in R^k.

OUTPUTS:
v is a polytope vertex.
H is the halfspace $\{x: x^T d \leq v^T d\}$;
(H contains the polytope, and its boundary contains v.)
}

```
procedure ProbeForHull ( (*INPUT*) k; (*OUTPUTS*) V, F);
  {Use probes to determine a polytope; assume that the origin is in its interior.
  INPUT:
  k, an integer, is the dimension of the search space.

  OUTPUTS:
  V is the set of polytope vertices.
  F is the set of polytope facets.}

  begin (* ProbeForHull *)
    (* Find k+1 distinct vertices. *)
    Probe (<1,0,0,...,0>, v1, H1);
    Probe (<-1,0,0,...,0>, v2, H2);
    V := {v1,v2}
    for j := 2 to k do begin
      (* To avoid rediscovering vertices in V, choose d: d^T v ≤ 0 for v ∈ V. *)
      Solve for direction d such that dr = 0 for r>j and
        d^T v is constant for v ∈ V. (* j-1 constraints in R^j. *)
      if d^T v > 0 then d := -d;
      Probe (d, v_{j+1}, H_{j+1});
      V := V ⋃ {v_{j+1}}
    end; (* for *)
    m := k+1; (* m will be the number of probes performed. *)
    (* See if any points lie beyond conv(V), by testing conjectured facets. *)
    F := facets (conv(V));
    while some facet of F is unverified do begin
      f := an unverified facet of F;
      Compute direction d ∈ R^k normal to f
      (* For all vertices v ∈ f, d^T v is the same positive constant. *)
      m := m+1;
      Probe (d, v_m, H_m);
      if d^T v_m > d^T v for v ∈ f then
        begin (* update *)
          F := facets (conv(V ⋃ {v_m}));
          (* The update of F involves deleting f and any other
          unverified facets that v_m lies beyond, and adding
          new facets (unverified) that contain v_m. *)
          V := V ⋃ {v_m};
        end; (* update *)
      else mark f verified.
    end; (* while *)
  end; (* ProbeForHull *)
```

The essence of our algorithm is an operational definition for facets: Let d be the normal vector of the hyperplane that contains k given points of V. These points are the vertices of a facet of $conv(V)$ *iff* their projection on d is larger than that of any other point in V.*

An unverified facet satisfies this condition among points of V that the algorithm has already discovered. By expanding the test to <u>all</u> of V, a probe along d always either

1) produces a new vertex, or

2) establishes that the tested facet is part of the final hull.

Consequently, when the algorithm terminates it has performed exactly $|V|+|F|$ probes, as claimed.

To see that the algorithm correctly determines the polytope P, observe that after each pass through the *while* loop,

$$conv(V) \subseteq P \subseteq \bigcap_{i=1,m} H_i \,. \tag{4.16}$$

When every facet is verified, $conv(V) = \bigcap_{i=1,m} H_i$; thus P has been completely determined.

We now argue that *ProbeForHull* is optimal, in the sense that no algorithm can guarantee to determine an arbitrary polytope using fewer than $|V|+|F|$ probes. A correct algorithm must report every vertex and verify every facet: otherwise there is a proper inclusion in (4.16). (If a vertex is

---

*For simplicity, the text assumes non-degeneracy (no hyperplane contains $k+1$ points of V), so each facet of V's hull is determined by exactly k points. In general, k points on a hyperplane normal to d lie on a facet iff no other point has a *greater* projection on d. When more than k points lie on a facet, they all tie for greatest projection. It is reasonable to assume that a probe can return all the points in a tie. (Section 4.3 treats the case of ties in transformed placement problems.) On this assumption, *ProbeForHull* requires <u>at most</u> $|V|+|F|$ probes in degenerate cases.

missed, $conv(V)$ is too small; if a facet is unverified, $\bigcap H$ is too large.)

We show that no algorithm can insure that any probe does more than find one new vertex or verify one facet. Clearly, at most one facet can be verified by a probe. Thus we must only rule out the possibility that a probe discovers multiple new vertices, or establishes a facet at the same time it finds a vertex. In either case, some *new* vertex has a projection on the probe that equals another point's projection exactly. Intuitively, this is unlikely; an algorithm would have to be clairvoyant to select a probe direction that matches an unknown point so perfectly.

To substantiate this intuition, we take the role of an adversary who responds to probes. We want to eliminate "fortunate accidents" of the above type. In proving a lower bound, an adversary is entitled to pick a hard case: we fix on any *simplicial* polytope (one with k vertices per facet), reserving the right to adjust its vertex positions slightly as we reveal them to the algorithm. For this purpose we observe that there is an $\varepsilon>0$ such that we can simultaneously move every vertex anywhere within ε of its initial position without altering the structure of the convex hull (composition of polytope faces).

We respond to most probes according to the original vertex positions. But suppose a probe is about to find a new vertex v whose initial position has a projection identical to that of another undiscovered vertex or an entire facet. In this case we reposition v by introducing a slight perturbation that increases its projection and thus "breaks the tie" in v's favor. By induction no previous bounding hyperplane intersects v, so it is easy to make this perturbation consistent with information from earlier probes. Then no probe detects multiple vertices, and no facet is verified until all k of its vertices have been found: the algorithm must perform $|V|+|F|$ probes.

4.2.2. Probing for the furthest point

For specific point sets it will often be possible for a probing algorithm to determine our actual objective, the *furthest* vertex on the convex hull, without completely determining the polytope. Figure 4-1 illustrates how this can happen. In this example, three probes have produced only two of the polygon's six vertices; yet they give enough information to conclude that point C is the furthest vertex from the origin.

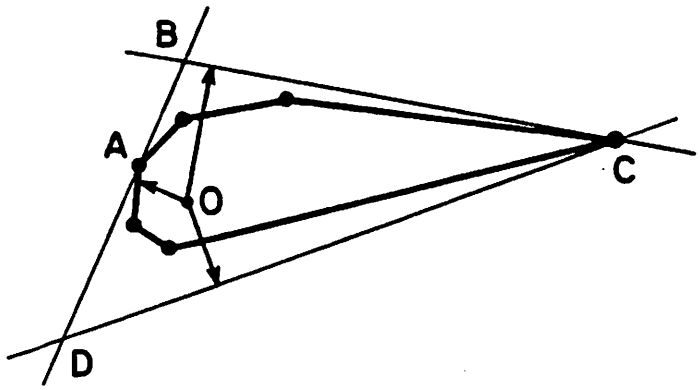


Figure 4-1. The furthest point is determined, but not the complete hull.

The polygon must lie in the intersection of the three halfplanes determined by the probes; by inspection, C is the furthest point from the origin in this containing region (triangle BCD). Because C is also a polygon vertex, it must be the point we seek.

The possibility of finding the furthest point without fully mapping out the set's convex hull is very important in practice. We can find the furthest point on some polytopes while ignoring a great deal of their structure. But there are still unfavorable cases. For instance, we now show that in R^2, any algorithm to determine the furthest point of a polygon requires as many probes in the worst case as are needed to completely determine a polygon: $|V|+|F|\ (=2m)$.

We follow an adversary strategy, which first selects an arbitrary $m \geq 3$ (not revealed to the algorithm) and a circle centered at the origin. Our basic idea is to put vertices where the algorithm's first m probes meet the circle. Typically this will yield m vertices whose hull contains the origin; we study this case first. Consider any two neighboring vertices, *e.g.*, A and B in Figure 4-2.

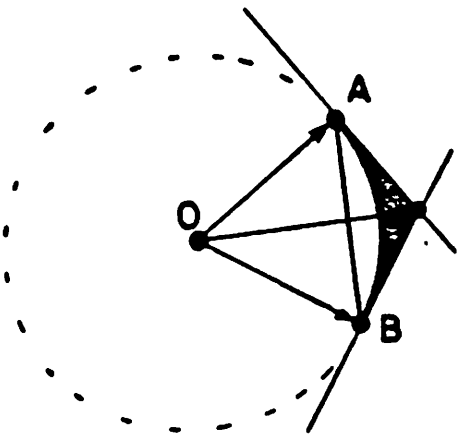


Figure 4-2.

After m probes, any point beyond arc AB and bounded toward the origin by the tangent lines through A and through B is still eligible to be a vertex. Since all these points (the shaded area in the figure) are further from the origin than the known vertices, the algorithm must rule out this area. Only a probe directed between A and B can exclude any of this area.* (On any external probe, A or B has a greater projection than the shaded points.) Therefore, the algorithm needs at least m more probes (one in each area), for a total of $2m$ probes in all.

We now handle cases in which the first m probes are atypical. If the algorithm repeats any probe, it discovers only p distinct vertices, with

*For a *single* probe to exclude the whole area, it must aim at the midpoint of AB.

$p<m$. In this case we "give away" $m-p$ probes at uniform-angle spacing in the biggest gap, and tell the algorithm (for free) that each such probe meets a vertex at the circle. This returns us to the situation with m vertices.

Finally, suppose the first m probes are distinct, but all lie in one half-plane. If our last vertex followed the basic strategy, the polygon would not contain the origin. So instead we find the most separated pair of vertices u,w among the first $m-1$, and place a vertex v on the circle so that v bisects the external angle ($\geq 180°$) between u and w. This insures that triangle uvw contains the origin. Each of the $m-2$ sections *internal* to uw requires a probe, just as in the main case. As for the external section, two probes suffice only if the algorithm is "fortunate": vertex v and either u or w must have identical projections on the first probe in this section. As before, we can perturb v (this time along the circle) in response to such a probe. Thus a total of $(m-1)+(m-2)+3 = 2m$ probes are needed, which concludes the proof.

This proof of a $|V|+|F|$ lower bound for furthest-point problems applies only in R^2: in R^k for $k\geq 3$, it becomes possible to use probes that lie outside a cone spanning k vertices to exclude regions of that cone. But it is easy to see that finding the furthest vertex of a polytope in R^k requires at least $|V|$ probes in the worst case. Consider any simplicial polytope whose vertices are equidistant from the origin. A furthest-point algorithm must find all the vertices, and in the worst case no probe will find more than one. (This can be proven formally by another adversary argument involving perturbations of the points.)

The consideration of polytopes in which the vertices are equidistant from the origin has demonstrated that finding the furthest vertex can

sometimes require as many probes as determining the whole polytope. Nevertheless, a suitably modified *ProbeForHull* can solve *some* furthest-point problems for which it would be intolerably slow to compute the entire hull. The basic idea is to keep track of the furthest point in R^k that is still eligible to be a vertex.

Figure 4-1 makes it clear that the furthest eligible point is determined by the bounding halfspaces H_i that probes return. As we presented it, *ProbeForHull* makes no explicit use of these halfspaces. To handle the furthest-point problem, *ProbeForHull* should maintain an up-to-date specification of $Q = \bigcap_{i=1,m} H_i$, just as it does for $conv(V)$. A few initializing probes can insure that Q is bounded.*

Among the vertices of Q and of $conv(V)$, let q and v, respectively, be at maximum distance from the origin. We change *ProbeForHull*'s termination condition so that it continues probing to verify facets only while $\|q\|^2 > (1+\varepsilon)\|v\|^2$, where ε is the acceptable error in determining the furthest point.

The furthest-point objective also provides a reasonable criterion for selecting which unverified facet to test at each step; until now the choice of facet has been completely arbitrary. We propose to choose a facet that lies beneath q, *i.e*, one contained in a hyperplane that separates $conv(V)$ from q. This method insures that every probe has the possibility of concluding the search, since it might discover a vertex (q) that is provably furthest from the origin.

---

*$E.g.$, $2k$ axis probes.

By leaving the basic structure of *ProbeForHull* intact, we preserve the guarantee that every probe either finds a new vertex or verifies a facet. The refinements focus the early exploration on polytope regions likely to be most productive in our search for the furthest point. They allow us to gather useful information even when |V|+|F| is so large that we cannot hope to determine the polytope completely.

Of course, the necessary background work between probes essentially doubles if we must maintain the polytope $\bigcap H_i$ in addition to $conv(V)$. Convex hull computation is a solved problem, but as we noted earlier, the solutions are expensive. Fortunately, it is much easier to solve the set of problems encountered in the course of a probe sequence than it would be to solve the same number of independent problems.

An efficient implementation of *ProbeForHull* is beyond the scope of this work. It is worth mentioning, however, that the required updates are similar to problems that have been studied in inductive approaches to computing the convex hull. They are also related to "on-line" versions of the problem, where the hull is updated to include one arbitrary point after another. Our situation enjoys two advantages, compared to the general on-line problem. First, each point returned by a probe is a vertex, and it *remains* a vertex through all updates of the polytope. Second, each new point is known to lie beyond a particular facet; this is a useful head start in the update process.

4.3. The sizes of V and F for placement polytopes

In section 4.2 we gave an algorithm, *ProbeForHull*, which finds the furthest of a set of points in R^k by computing the convex hull of the set. We showed that |V|+|F| probes suffice to determine the hull completely. In this section we investigate the implications of this result for the class of

point sets derived from n-component circuit placement problems. We refer to the associated convex hulls as "placement polytopes". Our main finding is that for any fixed number of dimensions, *ProbeForHull* produces the optimal solution to these furthest-point problems in time that grows as a polynomial function of n. To prove this we show that for placement problems transformed into R^k, $|V|+|F|=O(n^{2(k-1)})$.

We know that the transformed placement problem has $n!$ points, one for each ordering of the components. Our first concern is to determine how many of these points are "visible" to probes in R^k: only these points can be vertices of the convex hull. The visible solutions correspond to all possible component orderings of vectors in the span of v_1 through v_k: as our probe directions d range over R^k, we produce vectors Vd in this span.

The solution to the following equivalent problem (viewing rows of V as points) appears in [Cover67].

Consider n points $V[1], V[2], \ldots, V[n]$ in R^k, which are to be ordered by orthogonal projection onto a reference vector $d \in R^k$. Say that a permutation π of the integers 1,2,...,n is "linearly inducible" if there is a d such that

$$d^T V[\pi(1)] < d^T V[\pi(2)] < \cdots < d^T V[\pi(n)].$$

We would like to know the *number* of linearly inducible orderings of n points in pairwise general position* in R^k, which we denote $Q(n,k)$.

---

*Given a set of points, let S be a subset of the connecting line segments defined by pairs of the points. If S includes a path of three or more disjoint segments that starts and ends at the same point, say that S has a *cycle*. Points in R^k are in "pairwise general position" if the only sets of k connecting segments with a common perpendicular are those that have a cycle.

Recall that points in R^k are *degenerate* if some $k+1$ belong to a hyperplane; degenerate points are never in pairwise general position, since any k segments that span the $k+1$ points in a hyperplane will have a common perpendicular but no cycle. Pairwise general position is a strictly stronger condition than nondegeneracy. For example, in R^2 it disallows not only three collinear points, but also two parallel connecting segments.

The problem is thus slightly mis-stated in Cover's paper, which required only that the points be non-degenerate. [Goodman86] pointed out that the stronger condition must be satisfied to obtain the maximum number of orderings.

Cover showed that

$$Q(n,k)=2\sum_{m=0}^{k-1}{}_nS_m=2\left[1+\sum_{2\le i\le n-1} i+\sum_{2\le i<j\le n-1} ij+\cdots\right] (k \text{ terms}),$$

where ${}_nS_m$ is the sum of the $\binom{n-2}{m}$ possible products of m numbers taken without repetition from $\{2,3,\ldots,n-1\}$. The last sum includes $\binom{n-2}{k-1}$ products, each of which is less than n^{k-1}. It is easy to see that

$$Q(n,k)=\Theta(n^{2(k-1)}) . \tag{4.17}$$

Also of note is Cover's solution to a related problem. Let $C(n,k+1)$ be the number of linearly separable dichotomies of n non-degenerate points in R^k.* He showed that

$$C(n,k+1)=2\sum_{m=0}^{k}\binom{n-1}{m}=\Theta(n^k).$$

$C(n,k+1)$ is an upper bound on how many of the $\binom{n}{n/2}$ solutions to an n-component *partition* problem can be discovered by probes in R^k.

From Cover's analysis, we conclude that placement polytopes have at most $Q(n,k)$ vertices. Our goal now is to determine the number of *facets* of placement polytopes. We do so in two stages. First we find a necessary and sufficient condition for a direction d to be the normal vector of a facet. Then we calculate the number of distinct directions that can satisfy the condition.

Given *any* set of points S and direction d, let $M(d)$ denote the set of points in S with maximum projection on d. The supporting hyperplane of $conv(S)$ normal to d intersects $conv(S)$ in a face whose dimension equals that of the affine hull of $M(d)$ ("$aff(M(d))$").** Thus in R^k, d determines a

*that is, distinct partitions of the points according to whether their projections on a given vector are greater or less than a specified constant.

**The affine hull of a point set W is the intersection of all hyperplanes that contain

facet if and only if the dimension of $aff(M(d))$ equals $k-1$. We will prove that for point sets derived from placement problems, the dimension of $aff(M(d))$ is easily characterized in terms of the components of Vd.

We require the following

LEMMA: Assume that the n legal positions are distinct. If $(Vd)[i]<(Vd)[j]$, then $x[i]<x[j]$ for all points x^TV that maximize x^TVd.

PROOF: Otherwise $x[i]>x[j]$ (by distinctness of the legal positions), and swapping the positions of components i and j would strictly increase x^TVd.

COROLLARY: The x vectors of points x^TV in $M(d)$ are identical except for possible rearrangements of positions among components that are identical in Vd.

The corollary allows us to compute the possible numbers of points in $M(d)$. The number is always of the form $\prod_i(c_i!)$. Each term represents a c_i-way tie among certain elements of Vd; permutations of the corresponding components of x do not change x^TVd. We will now show that the dimension of the face determined by d equals $\sum_i(c_i-1)$. This sum has a natural interpretation: it is the number of pairs of components of Vd that must be equated to specify all the ties. The notion that a certain number of pairs is necessary rests on the following definition: a set of equalities is *nonredundant* if none is implied by the others.

THEOREM: Suppose l is an n-vector with all components distinct, and $V[1]$ through $V[n]$ are points in pairwise general position in R^k. Define S (the set of solution points) by

$$S=\{x^TV: x=\Pi l \text{ for some permutation matrix } \Pi\}.$$

For an arbitrary direction $d\in R^k$, define $M(d)$ (the points discovered by a probe in direction d) as

$$M(d)=\{m\in S: m^Td\geq s^Td \text{ for all } s\in S\}.$$

Then the dimension of the affine hull of $M(d)$ equals the maximum number

---

W. Equivalently, the affine hull of points w_i equals $\{\sum_i\alpha_iw_i: \sum_i\alpha_i=1\}$

of nonredundant equalities between components in Vd.

PROOF: Let j be the maximum number of nonredundant equalities in Vd. The case $j=0$ is trivial; with no equalities, $M(d)$ comprises one point. So assume $1 \le j \le k-1$, and let (a_1,b_1) through (a_j,b_j) be nonredundant index pairs of equated elements in Vd. To prove the theorem we find points p_0 through p_j in $M(d)$ and demonstrate that

1) these $j+1$ points are nondegenerate, and
2) every point in $M(d)$ is in their span.

Let $p_0 = x_0{}^T V$ be any of the points discovered. For each (a,b) pair, consider the placement obtained by swapping the corresponding legal positions in x_0. For $1 \le i \le j$, we obtain points

$$p_i = p_0 + \Delta_i (V[a_i] - V[b_i])$$

in $M(d)$, where $\Delta_i = x_0[b_i] - x_0[a_i] \neq 0$.

1) The points p_0 through p_j span j dimensions. Otherwise, consider the j segments $V[a]$-to-$V[b]$ for the given index pairs and any other $k-j$ segments between points in $V[1..n]$ that do not introduce a cycle. These k segments have a common perpendicular, contradicting our assumption that $V[1..n]$ are in pairwise general position.
2) We claim that any point p in $M(d)$ can be written as

$$p = p_0 + \sum_{i=1}^{j} c_i (p_i - p_0) ,$$

for suitable constants c_i. The above corollary tells us that the points in $M(d)$ all result from simple rearrangements of the positions associated with identical components in Vd. Any such rearrangement can be accomplished by a sequence of swaps (of $x[i]$,$x[j]$) within "tied" sets of components. Each tied set is connected by its representative (a,b) pairs. We can therefore move from p_0 to p by a sequence of (a,b) swaps. By the definition of the points p_i, each such swap adds some constant multiple of $(p_i - p_0)$ to the current position, proving our claim.

The theorem implies that we can identify each facet with a set of $k-1$ nonredundant equations of elements of Vd. Computing the number of facets of a placement polytope in R^k thus reduces to the following problem: Given n items, we must equate $k-1$ pairs of them so that no equation is implied by the others. How many different choices are possible? (If some c items are equated, the particular pairs used to link them together does not matter.) Denote this number by $D(n,k)$.*

The maximum possible number of normal vectors to facets is $D(n,k)$. If the points $V[1..n]$ are not in pairwise general position, there are fewer.

Each group of equated items defines an equivalence class. We begin with n classes. Every nonredundant equation merges two classes and thus reduces the number of classes by one. At the end we have $n-k+1$ classes; thus our problem is simply to count the possible partitions of n items into $n-k+1$ nonempty subsets. The solution is known as the Stirling number of the second kind, $S(n,n-k+1)$. These numbers obey the recurrence relation $S(n,t)=S(n-1,t-1)+t\cdot S(n-1,t)$. To compute $D(n,k)=S(n,n-k+1)$ for small k, it is easier to use the equivalent relation

$$D(n,k) = D(n-1,k)+(n-k+1)\cdot D(n-1,k-1),$$

with the boundary conditions $D(n,1)=D(n,n)=1$ for $n\geq 1$. Finally, the problem formulation provides a crude upper bound on $D(n,k)$: From $\binom{n}{2}$ possible equations, $k-1$ are selected: therefore $D(n,k)\leq\binom{\binom{n}{2}}{k-1}$.

Since the probe vector in either direction along a normal line will find a facet, there are twice as many facets as normal vectors. We conclude that for the convex hull of the points of an n-component placement problem transformed into R^k, the number of facets satisfies

$$|F|\leq 2D(n,k)=2S(n,n-k+1)\leq 2\binom{\binom{n}{2}}{k-1}=\Theta(n^{2(k-1)}). \qquad (4.18)$$

We have just seen that the normal vectors to all facets of placement polytopes are determined by the underlying points $V[i]$. By computing these normal directions before probing, we can insure that every probe verifies a facet! We can thus determine the whole hull with |F| nonadaptive probes. Of course, for practical values of n, the range of k for which it is feasible to consider performing $\Theta(n^{2(k-1)})$ probes is severely limited.

From the theoretical standpoint, (4.17) and (4.18) are important because they establish that for fixed k, the problem of determining the placement polytope and thus of finding its furthest point is solvable in polynomial time. From the practical standpoint, the ideas in section 4.2.2, which highlight the *differences* between finding a furthest point and determining a polytope, are more important. By showing us how to choose probes that are especially informative about the furthest point, these ideas illustrate the real power of adaptive probing.

Chapter 5

5. Application To Circuits

We have concentrated on the problem

$$\text{minimize } x^T B x : x \in \{\Pi L\} \text{ , } \Pi \text{ a permutation,}$$

which resulted from our formulation of circuit placement. Before our techniques can be applied, we need to consider some aspects of real circuits that our abstractions have suppressed. This chapter addresses two practical concerns. In section 5.1, we propose a new weighting function to represent multi-component nets in the n-by-n connection matrix. In sections 5.2 and 5.3, we study two methods that our approach can use to handle components that must lie in specially constrained positions.

5.1. Weights for component pairs in large nets

In our general placement problem, cost is defined as the sum of terms c_{ij}·(squared distance between components i and j). How should the values c_{ij} be set up to represent circuits that have nets with more than two components? In section 2.1 we argued that for every pair of components i,j in a net, the same weight should be added to c_{ij}. This approach is commonly known as the "clique model" of nets.

Care must be taken when using the clique model, so that nets with many components do not overwhelm the cost function. Let $w(s)$ be the weight added per entry for nets with s components. If we used $w(s)=1$, a net with s components would contribute $s(s-1)/2$ times as much total weight as a two-component net ([Schweikert72]).

Past work has compensated for this effect by using $w(s)=\frac{1}{s-1}$ or $w(s)=\frac{2}{s}$ ([Charney68, Cheng84]). The latter choice gives a total weight of $s-1$ for s components, which equals the number of connections needed to link them. But it is not clear that the total weight of a net should match the number of connections it requires.

It makes more sense to determine the weights $w(s)$ by the effect they will have in cost summations. Ideally, the total cost contributed by each net ($w(s)$ times the sum of squared distances of all its component pairs) would equal the square of its span,* for any s, as it does for 2-component nets with weight 1. But this cannot be guaranteed since for nets with $s>2$, total cost depends not only on the span but also on the detailed configuration of components.

We <u>can</u> choose $w(s)$ so that the biggest discrepancy (among all configurations) between total cost and span^2 will be as small as possible. It suffices to consider the extreme costs among configurations of nets with span$^2=1$. Cost can range from $\frac{s}{2}\cdot w(s)$ (when all but two components are midway between the extremes) to $(\frac{s}{2})^2\cdot w(s)$ (when half the components are at each extreme). We propose to set

$$w(s)=(\frac{2}{s})^{\frac{3}{2}}\ , \tag{5.0}$$

since this minimizes the worst-case deviation of cost from span^2. Cost is then a factor of $\sqrt{s/2}$ below span^2 in one extreme case, and a factor of $\sqrt{s/2}$ above it in the other.

*The (x) "span" of a net is the maximum distance (in x) between any two of its components.

5.2. Classes of components

In applications, differences between components are often so pronounced that it is inappropriate to assume that every permutation of the legal positions is feasible. The positions of certain components may be restricted by functional considerations. For example, the components responsible for external connections (I-O pads) typically must lie near an edge of the layout. To be useful in practice, our problem formulation must explicitly accommodate restrictions on specific components.

While the initial problem statement in section 2.1 allows arbitrary permutations of legal positions, a way to handle restricted components was also mentioned. In addition to specifying the connection matrix and legal positions, the input can divide the components into two or more classes (where class l has m_l components, $\sum m_l = n$) and specify m_l positions to be occupied by each class.

Performing a probe to maximize $x^T(Vd)$ among the $\prod(m_l!)$ vectors x that are feasible under these constraints is as easy as when all $n!$ permutations are possible. This is because the inner product to be maximized equals the sum of separate inner products of length m_l for the different classes, and arbitrary component permutations are permitted *within* each class. Thus the solution requires no new techniques: we simply order the x positions in each class independently to match the order of the corresponding portion of Vd. This straightforward generalization of probes enables us to produce the feasible solution point with maximum projection in any direction for problems with different classes of components.

Our original probes precisely matched the ranking of each component in x to its rank in the ordering of Vd. When we place a component among specially designated legal positions according to its ranking in a class, its

overall ranking in x may be very different from the one determined by Vd.

This effect would not be too serious if the legal positions for each class were well distributed over the range of positions. For instance, suppose two classes of $n/2$ components occupied alternating positions along the x axis. A component placed by rank within its class would tend to be close to its overall ranking as well.

But a typical class of I-O pads comprises a small fraction of the components, with all the associated positions at one side of the circuit. As the ordering of x is constrained, $x^T Vd$ becomes less than it would be if arbitrary permutations were allowed. This reduces the effectiveness of our heuristics, which work best when the proportion of $\sum \alpha_r^2(x)$ found in the active eigenvectors is large.

To summarize: when disjoint sets of components are to be arranged within separate sets of positions, using probes to order the components separately in each class is an appropriate method. However, class constraints tend to undermine the power of probe directions spanned by low-cost eigenvectors.

5.3. Fixed components

If each special component must occupy a predetermined position, we could define a "class" for each one. But there is little reason to work with a solution space in which the positions of these components vary. In this case it makes more sense to fix the special components explicitly, and consider the reduced problem on the remaining components. That is the approach in this section.

We first examine the new form that the objective function takes, and show that we can still transform to a furthest-point problem that enables us

to use probes. We then study features that distinguish the transformation with fixed components from the general problem. We conclude by presenting a variation of the given method.

5.3.1. New form of objective function and probes

Number the fixed components $m+1$ through n, and let $x[m+1]$ through $x[n]$ be their respective positions. Assume that components 1 through m may be freely permuted among m legal positions. We now reconsider the original expression for placement cost,

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}(x[i]-x[j])^2,$$

with the aim of separating out terms that involve the fixed components. The double sum can be broken into parts, [1]+]2]+[3], where

$$[1] = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}c_{ij}(x[i]-x[j])^2$$

$$[2] = \sum_{i=1}^{m}\sum_{k=m+1}^{n}c_{ik}(x[i]-x[k])^2$$

$$[3] = \frac{1}{2}\sum_{k=m+1}^{n}\sum_{l=m+1}^{n}c_{kl}(x[k]-x[l])^2\ ;$$

and part [2] can in turn be broken into [2.1]+[2.2]+[2.3], where

$$[2.1] = \sum_{i=1}^{m}x^2[i]\left(\sum_{k=m+1}^{n}c_{ik}\right)$$

$$[2.2] = \sum_{i=1}^{m}x[i]\left(-2\sum_{k=m+1}^{n}c_{ik}x[k]\right)$$

$$[2.3] = \sum_{k=m+1}^{n}x^2[k]\left(\sum_{i=1}^{m}c_{ik}\right).$$

We shall treat terms [2.3] and [3] as constants, since they do not depend on the positions of components 1 through m. Define $Q=[2.3]+[3]$. For $i=1$ to m, define $g[i]=\left(-2\sum_{k=m+1}^{n}c_{ik}x[k]\right)$. Define $\tilde{C}$, $\tilde{B}$, and $\tilde{D}$ as the upper left m-

by-m portions of the C, B, and D matrices from chapter 2 (2.0), respectively.

Using these definitions, we can write cost concisely as a function of the vector $x[1..m]$. As in chapter 2, $[1]=-x^T\tilde{C}x+\sum_{i=1}^{m}x^2[i](\sum_{i=1}^{m}c_{ik})$; and since $d_{ii}=\sum_{i=1}^{n}c_{ik}$,

$$[1]+[2.1] = -x^T\tilde{C}x+x^T\tilde{D}x = x^T\tilde{B}x\ .*$$

Therefore, $\text{cost}(x)=[1]+[2.1]+[2.2]+[2.3]+[3]$ implies

$$\text{cost}(x) = x^T\tilde{B}x+x^Tg+Q\ . \tag{5.1}$$

Our goal is to revise the furthest-point transformation to incorporate the linear term in x, namely x^Tg. For the matrix $\tilde{B}$, we denote the eigenvectors u_r and associated eigenvalues λ_r, as when the objective was a pure quadratic form. We note first that

$$x^Tg=[\sum_{r=1}^{m}(\alpha_r(x))u_r{}^T]g\ ,$$

where $\alpha_r(x)=x^Tu_r$. If we choose some $H\geq Max\{\lambda_r\}$, we can convert to a maximization problem with the objective function

$$\sum_{r=1}^{m}\alpha_r{}^2(x)[H-\lambda_r]\ -\ \alpha_r(x)(u_r{}^Tg)\ . \tag{5.2}$$

We can rewrite the objective in an equivalent form,

$$\text{maximize}\ \sum_{r=1}^{m}(x^Tv_r+z_r)^2\ , \tag{5.3}$$

---

*Note that since d_{ii} is the row-sum of n elements, while the rows of $\tilde{C}$ have only m elements, the row-sums of $\tilde{B}$ are in general not zero. As a result, several convenient properties of B do not hold for $\tilde{B}$: the least eigenvalue is not zero, the components of the associated eigenvector are not constant, and the components of other eigenvectors do not sum to zero.

by defining $v_r = u_r\sqrt{H-\lambda_r}$ as before and choosing each constant z_r to "complete the square".

Before we proceed, let us determine z_r. For each r we equate the linear terms in (5.2) and (5.3):

$$-\alpha_r(x)(u_r{}^T g) = 2(x^T u_r\sqrt{H-\lambda_r})z_r \ ;$$

thus

$$z_r = -u_r{}^T g/(2\sqrt{H-\lambda_r}) \ . \tag{5.4}$$

The advantage of the form (5.3) is that we can think of each $x^T v_r + z_r$ as a coordinate of a point, $x^T V + z^T$. After the transformation from placements x to points $x^T V + z^T$, our objective is again to find the furthest point.

It is as easy to perform a probe in this formulation as before. Given a probe vector d, the point with the greatest projection on d is the one that maximizes $[x^T V + z^T]d$. Since z is a constant vector (defined in (5.4)), $z^T d$ does not affect the probe operation. The desired placement is determined by filling x with the m legal positions in the order that matches Vd.

Starting with any point, we also can *iterate* with probes in this setting to produce a sequence of points at increasing distance from the origin. At each step, the probe direction d^T is set to the current point, $x^T V + z^T$. The new point is determined by the component ordering of

$$Vd = VV^T x + Vz \ . \tag{5.5}$$

Suppose a point with greater projection is found. Because the projection of the new point is greater, in the direction of the old point, magnitude can only increase at each step until convergence.

5.3.2. Features of the transformation with fixed components

In the transformation to points with coordinates $x^T v_r + z_r$, there is no longer an obvious criterion to decide which dimensions are most valuable. We can still work with low-dimensional projections of the points. We can carry out iterated probes in stages, with the number of dimensions increasing from stage to stage. But which dimensions should come first?

This uncertainty is due to the form of the function (5.2). While large numbers $H-\lambda_r$ indicate potential value as before, the numbers $u_r{}^T g$ are also factors of terms in this objective function. Because for each r, the relative contribution to (5.2) from the terms associated with these two numbers is a function of $\alpha_r(x)$, the balance changes from one placement to another.

Thus large numbers $H-\lambda_r$ and $|u_r{}^T g|$ are both valuable, and the tradeoff between them depends on x *and* r. The observation that $H-\lambda_r$ enters with a factor of $\alpha_r{}^2$ and $u_r{}^T g$ enters with a factor of α_r has led us to formulate an *ad hoc* heuristic for ranking the dimensions. We consider the u_r's in decreasing order of

$$A(H-\lambda_r)+|u_r{}^T g| \tag{5.6}$$

for some A (*e.g.*, $\sqrt{1/m}$) chosen to represent the typical α.

Because of the new objective function, we must also rethink how to compute H. Assume that we have a placement in hand, as is true in iterated probes. The preceding discussion tells us that we can no longer count on having an index set of active eigenvectors of the form {1..k}. Let K denote the current set of indices. The obvious generalization of (3.2) is to set

$$H=(\sum_{r\in K}\alpha_r{}^2(x)\lambda_r)/(\sum_{r\in K}\alpha_r{}^2(x)) \ . \tag{5.7}$$

The next question is how to compute (5.7) efficiently when $|K|$ is much less than m. (As in chapter 3, the goal is to avoid computing and storing inactive eigenvectors.) To efficiently compute the denominator of (5.7) is easy. If the norm of the legal-position vector is $\|L\|$, then $\sum_{r \notin K} \alpha_r^2(x) = \|L\|^2 - \sum_{r \in K} \alpha_r^2(x)$. For the numerator, we use $\sum_{r \notin K} \alpha_r^2(x)\lambda_r = x^T \tilde{B} x - \sum_{r \in K} \alpha_r^2(x)\lambda_r$. Observe that from (5.1), we can obtain $x^T \tilde{B} x = \text{cost}(x) - x^T g - Q$.

Because the active indices need not correspond to the smallest eigenvalues, the value of H from equation (5.7) may be smaller than some active eigenvalues. This produces scale factors $v_r/u_r = \sqrt{H - \lambda_r}$ that are imaginary numbers. We then have two options.

We can use vectors v_r whose components are purely imaginary, as at the end of section 3.4. Because the associated values of z_r also become imaginary, the iterated probe computation of Vd according to (5.5), $V(V^T x + z)$, produces a real vector result. The drawback is that monotonic improvement in the iteration is no longer guaranteed. To see why, we analyze what must happen for an iterative probe step to decrease the objective function:

Let $A = VV^T$. For a step from x to w, a decrease of the objective function means that

$$x^T Ax + 2x^T Vz > w^T Aw + 2w^T Vz .$$

The probe insures that

$$2w^T Ax + 2w^T Vz \geq 2x^T Ax + 2x^T Vz .$$

Summing these two conditions, we obtain

$$2w^T Ax > x^T Ax + w^T Aw .$$

If we define $\Delta = w - x$, the latter condition is equivalent to $\Delta A \Delta < 0$. This

possibility can only be ruled out if A's eigenvalues $H-\lambda_r$ are all nonnegative.

The other option is to abandon (5.7), and select a value of H that is greater than λ_r for all $r \in K$. For example, if k is the largest index in K, we can set $H=\lambda_{k+1}$. We have just shown that this will restore the monotonic improvement guarantee, but it gives a less satisfactory approximation for the missing eigenvalues. Which approach is better is an experimental question that is still open.

5.3.3. A "mixed representation" method for fixed components

The global objective of the furthest-point problem in the presence of fixed components is to maximize

$$\|x^TV+z^T\|^2 = [x^TV+z^T][V^Tx+z] = x^TVV^Tx+2x^TVz+z^Tz\ . \quad (5.8)$$

In the complete representation of the transformed space (in which V includes all m eigenvectors), $Vz=-.5g$. This can be verified from the definition of z (5.4); or we can substitute $-.5g$ for Vz in the final expression of (5.8) and obtain the correct dependence on x in the objective function, namely $x^TVV^Tx-x^Tg$.

According to (5.5), the basic iterated probe operation is to find the feasible vector w that maximizes $w^TVd=w^T[VV^Tx+Vz]$. When using stages, we increase the number of vectors in V in a sequence of steps: we work with projections of the solution points that are progressively more accurate representations of the complete problem.

We now describe an approach using "mixed" representations: the quadratic term x^TVV^Tx goes through a sequence of stages, but the linear term $2x^TVz$ is complete ($=-x^Tg$) from the start. Although this approach resists

natural interpretations involving projections of the points $x^T V + z^T$, it has favorable properties in both theory and practice.

Let A represent VV^T in each stage. The "mixed representation" step moves from a placement x to the placement w that maximizes $w^T(Ax - .5g)$. As with probes, to prove monotonic improvement we must require that A be positive semidefinite. This can be insured for any choice of active eigenvectors, by choosing H greater than the largest associated eigenvalue.* We will prove that

$$w^T Aw - w^T g \geq x^T Ax - x^T g \ . \tag{5.9}$$

PROOF: From A being positive semidefinite we have

$$w^T Aw + x^T Ax \geq 2w^T Ax \ ,$$

and from the maximizing property of w,

$$2w^T Ax - w^T g \geq 2x^T Ax - x^T g \ .$$

Adding these together and canceling like terms gives (5.9).

As the number of columns in V increases, so does the likelihood that (5.9) signifies an improvement in the true objective function. And for any number of dimensions, the completeness of the linear term yields a stronger correspondence to the full problem than that provided by iterated probes.

In experiments on problems with fixed components, mixed iteration has performed better than iterated probes. Convergence is much quicker, and placements of lower cost are obtained.

Even with the improvement from mixed iteration, results applying the furthest-point transformation to examples with fixed components have been

---

*Furthermore, mixed iteration in effect introduces all the terms $u_r{}^T g$ from the start, so it is less important to consider the dimensions in a special order such as that prescribed by (5.6). Experiments have confirmed that using dimensions with the smallest λ_r's works better in mixed iteration than with ordinary iterated probes; with this choice of dimensions, A is always positive semidefinite.

less favorable (compared to traditional approaches such as pairwise interchange) than cases without fixed components.

Chapter 6

6. Two-dimensional Placement

The previous chapters have dealt with the restricted placement problem in which all y positions are equal. The possible applications of our methods are greatly extended when we consider the original problem in its general form, which allows n legal positions anywhere in the (x,y) plane.

In this chapter we generalize the furthest-point transformation and associated probe techniques from one-dimensional to two-dimensional problems. As one might expect, the number of required dimensions in the transformed problem also doubles. We shall find that the function of a probe does not change, but that significantly more computation is needed to perform it.

In section 6.1 we present the furthest-point transformation for problems involving (x,y) positions. Section 6.2 describes the function of probes in this setting. In section 6.3 we show that standard algorithms for the linear assignment problem implement the probe operation exactly; we also give a faster technique for approximating the result of a probe. Section 6.4 treats techniques for proving lower bounds on two-dimensional placement cost.

To simplify the development, in sections 6.1 through 6.4 we assume that no components have specially constrained positions. Section 6.5 considers extensions to constrained components in two dimensions, and to placements in three dimensions.

6.1. The furthest-point transformation

A two-dimensional placement is specified by vectors x and y, where $(x[i],y[i])$ is the position of component i. Represent any placement by adjoining vectors x and y in an n-by-2 matrix X. Let L be an n-by-2 matrix whose rows are the n legal positions. We recall from section 2.1.2 that a placement X is feasible if

$$X \in \{\Pi L\colon \Pi \text{ is a permutation matrix}\} . \tag{6.1}$$

Thus while x and y are permutations of two separate vectors (the columns of L), the permutations applied to these vectors must be identical.

Because cost is the sum of squared connection distances, it separates into two double sums:

$$\tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}(x[i]-x[j])^2+\tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}(y[i]-y[j])^2 .$$

Defining B as in (2.0), our goal is:

$$\text{minimize cost } (x,y) = x^TBx + y^TBy . \tag{6.2}$$

Using the orthonormal eigenvectors u_r of B and the matrix V with columns $v_r = u_r\sqrt{H-\lambda_r}$ as in (2.5) and (2.6), we see that (6.2) is equivalent to

$$\text{maximize } 2H - \text{cost}(x,y) = \|x^TV\|^2 + \|y^TV\|^2 . \tag{6.3}$$

If we transform each placement x,y into a point with $n-1$ coordinates x^Tv_r and $n-1$ coordinates y^Tv_r, the optimal placement is mapped to the furthest point from the origin.

Let k be the number of columns of the transformation matrix V.*

---

*In the *complete* furthest-point transformation, $k = n-1$. As in chapters 4 and 5, it is often useful to consider projections of the points into fewer dimensions; this effectively truncates V, giving $k < n-1$.

Rather than representing the transformed points in R^{2k} as row vectors, it is convenient to use 2-by-k matrices. The transformation maps placements X to points X^TV.

When 2-by-k matrices represent points, a corresponding notation for squared distances is helpful. The appropriate choice is the square of the Frobenius norm: $\|A\|_F^2$ is defined as the sum of squares of A's elements, or equivalently as the trace of A^TA. This allows us to condense (6.1) and (6.3):

$$\text{Maximize } \|X^TV\|_F^2 \text{ , } X \in \{\Pi L\text{: } \Pi \text{ a permutation}\} \tag{6.4}$$

6.2. Probes

The function of a probe is to find the feasible point with maximum projection on a given direction of the search space. For placement problems involving (x,y) positions, the search space has $2k$ dimensions when k eigenvectors are active. We pick $2k$ probe coordinates $d_1..d_k$ and $e_1..e_k$, implicitly defining vectors d and e. The aim of a probe is to produce a feasible placement x,y that will

$$\text{maximize } \sum_{r=1}^{k}(x^Tv_r)d_r + \sum_{r=1}^{k}(y^Tv_r)e_r = x^TVd + y^TVe \text{ .} \tag{6.5}$$

Searching for the best probe d,e is equivalent to pursuing the furthest-point objective (6.3). To prove this we apply the reasoning that led to (2.7), with the number of dimensions doubled:

$$\|X^TV\|_F = (\|x^TV\|^2 + \|y^TV\|^2)^{\frac{1}{2}} = \max_{\sum_r d_r^2 + e_r^2 = 1} (x^TVd + y^TVe) \text{ .}$$

For any probe, form a k-by-2 matrix D having d and e for its columns. Writing $x^TVd + y^TVe$ as $trace[X^TVD]$, we find

$$\underset{\text{feasible } X}{Max} \|X^T V\|_F = \underset{\text{feasible } X}{Max} \left\{ \underset{\|D\|_F=1}{Max} trace[(X^T V)D] \right\}$$

$$= \underset{\|D\|_F=1}{Max} \left\{ \underset{\text{feasible } X}{Max} trace[X^T (VD)] \right\}, \qquad (6.6)$$

which concludes the proof.

Probes thus play as useful a role in two-dimensional placement as they do in one-dimensional placement. But because X and VD are both n-by-2, performing a probe is now more complicated. Our goal is:

$$\text{Find an } X \in \{\Pi L\} \text{ that maximizes } trace[X^T (VD)], \qquad (6.7)$$

for a fixed VD. Section 6.3 discusses how to attack this problem.

The probe technique for one-dimensional placement, which is based on sorting the components of a vector, could be used to maximize $x^T Vd$ *or* $y^T Ve$. But a permutation that optimizes x alone or y alone will in general be suboptimal for objective (6.7), which is the sum of these terms.

Although it cannot solve (6.7), the one-dimensional technique can be used to improve two-dimensional placements. The idea is to improve $\|x^T V\|^2$ without altering the y vector, and to improve $\|y^T V\|^2$ without altering the x vector. For example, given a placement $x,y = \pi l, \pi m$, we can set $d^T = x^T V$ and seek a new permutation π' that maximizes $(\pi' l)^T Vd$, subject to the "y-preserving" constraint $\pi' m = \pi m$.

When all components of m are distinct, no $\pi' \neq \pi$ satisfies this constraint. But suppose the legal positions were given by the intersection points of a few horizontal and vertical lines. Then π' is y-preserving so long as it maps each component to another one on the same horizontal line.

We can conduct probes that enforce this kind of restriction by special use of the "class" constraints introduced in section 5.2. For x probes we divide the components into classes according to their current horizontal

lines: for y probes, according to their vertical lines. Of course, probes are less powerful under class constraints such as these.

6.3. The (x,y) probe operation

6.3.1. Exact solution by linear assignment

We can view (x,y) probing as a special case of the following "linear assignment" problem:

n jobs must be assigned to n people. The cost of assigning person i to job j is f_{ij}. Find the permutation π that minimizes the total cost $\sum_{i=1}^{n} f_{i,\pi(i)}$.

Let us express (6.7) in this form. The column of Π in which row i gets a 1 is denoted $\pi(i)$. (Thus $X=\Pi L$ means $X[i]=L[\pi(i)]$.) We associate cost f_{ij} with the assignment of component i to legal position j, where

$$f_{ij} = -L^T[j](VD)[i] \ .* \tag{6.8}$$

Known methods to solve the linear assignment problem (*e.g.*, the "Hungarian" algorithm given by [Munkres57]) have $O(n^3)$ running time in the worst case. The Hungarian algorithm is treated in standard texts on combinatorial optimization, *e.g.* [Lawler76, Papadimitriou82]. While we can conclude that (x,y) probing is tractable, the $O(n^3)$ running time may be unsatisfactory for large placement problems, especially since our methods make such heavy use of probes.

One response to the bad news about linear assignment is to study the *expected* running time of algorithms for its exact solution. For example, [Karp80] gives an algorithm whose expected running time is $O(n^2 \log n)$, for

*The minus sign converts the maximization problem (6.7) into a minimization problem.

a class of linear assignment problems with independent identically distributed f_{ij}'s. But since this sample space is not representative of problems with costs determined by (6.8), such an analysis is not very reassuring.

6.3.2. Restatement as Euclidean^2 blue-green matching

The n-by-n cost matrices specified by (6.8), which are fully determined by $4n$ numbers, constitute a highly restricted class of problem instances. We might hope to design algorithms for problems in this class that are more efficient than algorithms for the general linear assignment problem. To encourage the development of such algorithms, we show that for this class of problems, there is a simple geometric interpretation of assignment cost.

Let $(l[j],m[j]) = L^T[j]$ and $(a[i],b[i]) = (VD)^T[i]$ represent points in the (x,y) plane. Then from (6.8), the total cost associated with a permutation π equals

$$\sum_{i=1}^{n}(-l[\pi(i)]a[i] - m[\pi(i)]b[i]) . \tag{6.9}$$

Nothing essential changes if we double this objective and add

$$\sum_{i=1}^{n}\left[(l[\pi(i)])^2 + (a[i])^2 + (m[\pi(i)])^2 + (b[i])^2\right] ,$$

since this sum is constant for all permutations. This gives

$$\text{cost}(\pi) = \sum_{i=1}^{n}\left[(l[\pi(i)] - a[i])^2 + (m[\pi(i)] - b[i])^2\right] . \tag{6.10}$$

In other words, our problem is to assign n blue points (a,b) to n green points (l,m) so that the sum of squared distances between paired points is minimized. We refer to this class of linear assignment problem as Euclidean^2 blue-green matching.

6.3.3. Approximate solution to Euclidean^2 blue-green matching

We have not yet found a way to compute exact solutions for Euclidean^2 blue-green matching problems that is faster than known algorithms for the linear assignment problem. However, in many applications of probes, approximate solutions are almost as useful as exact solutions. The special form of Euclidean^2 blue-green matching lends itself to fast heuristics for good *approximate* solutions.

[Avis83] reviews several heuristics for a closely related problem, in which there is no distinction between blue and green points, and the cost function is the sum of distances. Most of these heuristics are based on divide-and-conquer strategies. The problem region is partitioned into small subregions; an efficient algorithm then pairs up as many points as possible in each subregion, and a "cleanup" procedure is used to match leftover points.

[Ajtai84] uses a similar idea to study C_n, the expected optimal cost for problem (6.10) when n blue and n green points are distributed uniformly on the unit square. They prove that $C_n = \Theta(\log n)$, by showing that the typical squared distance between paired points is of order $(\log n)/n$. Their upper bound on C_n is derived from the analysis of a matching algorithm based on separating point sets into halves according to their positions.

Applying the separation procedure recursively, the algorithm constructs a binary tree for the blue points. Each node of the tree corresponds to a subset of the blue points, and the branching at each node represents a partition of that subset determined by a vertical or horizontal line: the partition direction alternates from level to level. The leaves of the tree correspond to individual points. A separate application constructs a similar tree for the green points. The two trees determine a matching, in which blue and green

points assigned to corresponding leaves are paired with each other.

For definiteness, consider the beginning levels for points of either color. The first level selects a vertical line such that half the points are to the left of (or on) the line, and the other half to the right of it (or on it).* The second level picks one horizontal line to separate the left set of points into halves, and another horizontal line to divide the right set of points. Further levels recursively split the point sets associated with each quadrant, until none of the resulting sets contains more than one point.

The scheme of separate recursive partitions is illustrated in Figure 6-1.

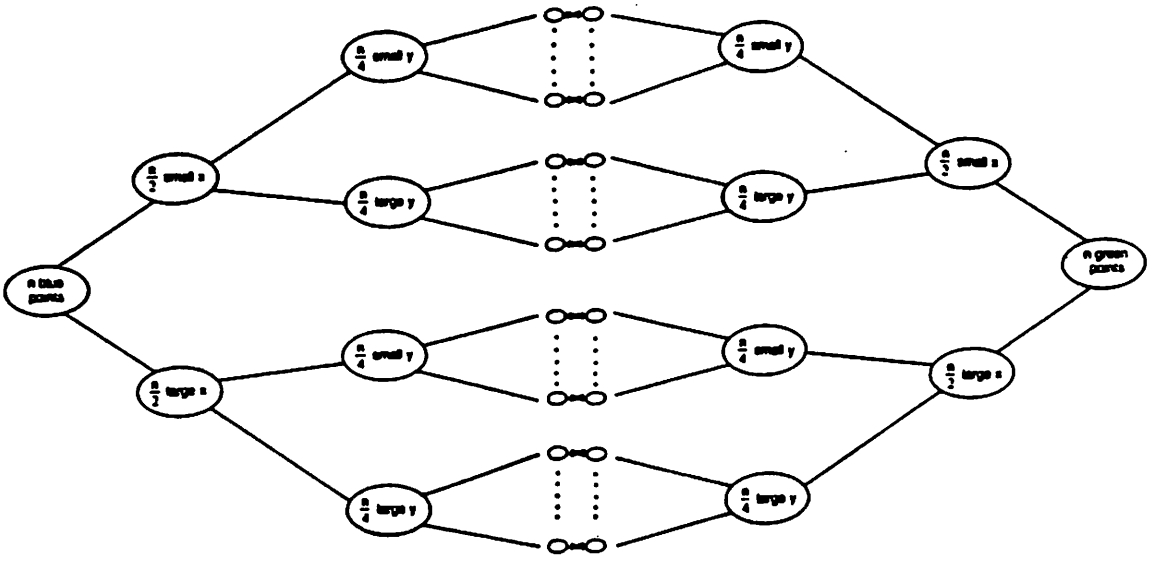


Figure 6-1. Blue and green points are partitioned separately, then matched.

For completeness we now implement the above algorithm as a procedure (*TwoDimSplits*) that can be used to find a permutation that approximates the minimum cost (6.10). We assume the existence of the following

*Whenever n is not a power of 2, some levels must partition sets with odd numbers of points. In these cases an arbitrary rule is applied: *e.g.*, the left side gets one more point than the right.

partitioning procedure:

```
{procedure Partition ( (*INPUTS*) data, lo, mid, hi; (*INPUT&OUTPUT*) index);
   INPUTS:
   data is a real array.
   lo,mid,hi are integers, with lo ≤ mid ≤ hi.
   index is an integer array:
        data[index[i]] is a valid entry of the data array for i ∈ [lo,hi].

   OUTPUT:
   The numbers in index[lo:hi] are rearranged so that
   for all integers i ∈ [lo,mid] and j ∈ [mid+1,hi], data[index[i]] ≤ data[index[j]].
   (The indices are reordered, not the data values themselves.)
}

procedure TwoDimSplits ( (*INPUTS*) x, y, lo, hi, stage;
     (*INPUT&OUTPUT*) index);
   {Recursive procedure to split the points (x[index[i]],y[index[i]]), i=lo..hi.
   INPUTS:
   x,y (real arrays): coordinates of points in the plane.
   lo,hi (integers): range delimiters of portion of index array to split.
   stage (integer): recursion depth. If stage is odd, split x; if even, split y.
   index (integer array): index[i], i=lo..hi, are recursively separated, by
         the values x[index[i]] or y[index[i]], in alternating stages.

   OUTPUT:
   Integers index are returned in the order that results from the above splits.

   begin (* TwoDimSplits *)
      if lo<hi then begin
         mid := ⌊ (lo+hi)/2 ⌋;
         if odd(stage) then Partition (x, lo, mid, hi, index)
                       else Partition (y, lo, mid, hi, index);
         TwoDimSplits (x, y, lo, mid, stage+1, index);
         TwoDimSplits (x, y, mid+1, hi, stage+1, index);
      end; (* if *)
   end; (* TwoDimSplits *)
```

To apply *TwoDimSplits* to (6.10), we load the integers 1..n into two arrays, *indexL*[1:n] and *indexA*[1:n], and execute the following code:

```
TwoDimSplits (l, m, 1, n, 1, indexL);
TwoDimSplits (a, b, 1, n, 1, indexA);
for i := 1 to n do π[indexA[i]] := indexL[i];
```

For a sample component ($indexA[1]$) of the resulting placement, the overall effect is

$$X[indexA[1]] = L[\pi(indexA[1])] = L[indexL[1]] .$$

We now compute $T(n)$, the running time of *TwoDimSplits* on n points. If $P(n)$ represents the time in a call to *Partition* with n points, we have the recurrence $T(n)=P(n)+2T(n/2)$. Expanding, we obtain

$$T(n)=P(n)+2P(n/2)+4P(n/4)+ \cdots \tag{6.11}$$

The running time of *Partition* is dominated by the time needed to compute a median element; thus $P(n)=cn$, where c is a constant. Each of the $\log n$ terms in (6.11) equals cn, so the total running time of *TwoDimSplits* is $O(n \log n)$.

In summary, the assignment procedure based on *TwoDimSplits* is both straightforward and fast: it is an eminently practical way to perform "approximate" (x,y) probes. We can use this method in any probing heuristic, *e.g.*, iterated probes.

While approximate probing can help us locate good placements, it does not provide the guarantees that exact probing can. For instance, we have no assurance that the points obtained by an iterative sequence of approximate probes will consistently improve or converge. And this kind of approximate probe is unsuitable for lower-bound techniques. The next section addresses the problem of how to prove lower bounds on two-dimensional placement cost.

6.4. Lower bounds on two-dimensional placement cost

There is a trivial way to prove lower bounds on the cost of a two-dimensional placement problem. Compute a lower bound L_x on the one-dimensional placement problem obtained by ignoring the y coordinates. Compute a lower bound L_y, ignoring the x coordinates. Then

$$\text{cost}(x,y)=\text{cost}(x)+\text{cost}(y)\geq L_x+Ly. \tag{6.12}$$

Chapter 4 gives a wide range of lower-bound methods for one-dimensional placements, from "axis probes" to *ProbeForHull*; any of these may be substituted in (6.12).

However, lower bounds of the form L_x+L_y fail to assess a charge for the central difficulty of two-dimensional placement; namely, that x and y cannot be permuted independently. In this section we study two lower-bound methods that do exploit the link between x and y. These methods are the natural generalizations to (x,y) placement of axis probes and random probes, respectively.

6.4.1. Generalized axis probes

In two-dimensional placement, (4.1) generalizes to

$$2\lambda_{k+1}-\text{cost}(x,y) \leq \sum_{r=1}^{k}(\lambda_{k+1}-\lambda_r)[(x^T u_r)^2+(y^T u_r)^2] \,. \tag{6.13}$$

Thus any upper bound on the right-hand side implies a lower bound on cost. Using the definition $v_r=u_r\sqrt{\lambda_{k+1}-\lambda_r}$, the function on the right equals

$$\sum_{r=1}^{k}[(x^T v_r)^2+(y^T v_r)^2] \,. \tag{6.14}$$

If we had upper bounds on the contributions for each r, their sum would upper-bound (6.14). In chapter 4, we were able to upper-bound the

contribution from u_r by probing along the associated axis. Now there are two such axes, and we want to upper-bound their joint contribution. For each feasible x,y we view the projection of X^TV onto the plane spanned by the two u_r axes as a point (x^Tv_r,y^Tv_r). We want to upper-bound the squared distance from the origin to such points.

In principle it is easy to compute an upper bound within a factor of $1+\varepsilon$ of the maximum squared distance for any $\varepsilon>0$. Let $\varphi=arctan(\sqrt{\varepsilon})$, and perform unit-magnitude probes at intervals of 2φ radians in the plane. Then $(1+\tan^2\varphi)$ times the square of the largest detected projection is an upper bound of the desired quality.

This procedure is actually quite practical. First of all, not many probes are required. For instance, (4.3) showed that for $\varepsilon=.025$, twenty probes suffice. And in contrast to general (x,y) probes, which require $O(n^3)$ time, probes in the u_r plane can be performed in time $O(n\log n)$. This is because we can apply the probing technique for one-dimensional placement in a new way.

In chapter 4, projections of points in R^2 were of the form (x^Tv_r,x^Tv_s): here they are of the form (x^Tv_r,y^Tv_r). (Before, we sought the optimal combination of two eigenvectors: now we want the optimal distribution of a single eigenvector between the x and y dimensions.) Before, the permutation π was chosen to maximize the projection $(\pi l)^T[v_rd_r+v_sd_s]$: the analogous projection is now $(\pi l)^Td_rv_r+(\pi m)^Te_rv_r$. Because here Vd is a scalar multiple of Ve, we can rewrite this expression as a single inner product,

$$(\pi[d_rl+e_rm])^Tv_r\ .$$

Thus the optimal permutation can be found by sorting components, as before.

In summary, we can compute as tight an upper bound as desired on each term $(x^T v_r)^2 + (y^T v_r)^2$ in (6.14), using only a constant factor more computation than was required to maximize $(x^T v_r)^2$ in one-dimensional placement.

6.4.2. Generalized random probes

Blanketing the search space with random probes is the simplest way to derive tighter lower bounds than those provided by axis probes. When (6.14) includes k eigenvectors, the corresponding probes lie in R^{2k}. We know from (4.8) that we can get φ-radian coverage of R^{2k} using $O\left\lceil \sqrt{k}\left(\frac{1}{\sin\varphi}\right)^{2k-1} \right\rceil$ probes.

From a practical standpoint, the major problem with random (x,y) probing is the $O(n^3)$ running time needed to perform each probe using an exact algorithm for linear assignment. Fortunately, we may be able to prove that a certain probe is the best one in a set without computing *all* the probes exactly. As with the partitioning problem in section 4.1.3.1, if we can quickly prove that a probe would give a result less than another one already found, there is no reason to proceed with the new one.

Thus we need fast ways to prove upper bounds on the result of a probe. Many are available: the following is one of the simplest.

$$\text{Max}[x^T V d] + \text{Max}[y^T V e] \geq \text{Max}[x^T V d + y^T V e] .$$

We can compute the left-hand side in time $O(n \log n)$. The fraction by which this side overestimates the other is clearly no greater than the ratio of the smaller to the larger term in the left-hand sum.

In summary, the available worst-case time bounds for random (x,y) probing are discouraging. But clever heuristics can produce fast upper bounds that may allow us to bypass the exact computation for many probes. Random probe sets may thus be more manageable in practice than the worst-case analysis would indicate.

6.5. Extensions for special components and higher dimensions

Although our treatment of (x,y) placement has thus far ignored the possibility of special components, the ideas from chapter 5 are easily adapted to two-dimensional placement. In this section we briefly review how this is accomplished.

The specification of component classes does not make (x,y) probing any harder. We need only solve a separate linear assignment problem within each class.

Neither do fixed components introduce new complications. The object of a probe d,e will be to find the feasible placement x,y that maximizes a function of the form $x^TVd+z^Td+y^TVe+s^Te$. Since z^Td and s^Te are constant, they have no impact on the assignment problem.

The only extra difficulty when fixed components appear in (x,y) problems has to do with an aspect of special components that was already complicated in one-dimensional placement: which eigenvectors are most important? We can expect a heuristic ordering of the type suggested in (5.6) to turn out differently in x and y. Consequently, an order based on some form of averaging between the two dimensions is called for.

Finally, we note that with trivial modifications, the approach of this chapter can handle three-dimensional placement problems. We add a new column (z) to the placement matrix X, making it n-by-3. Placements still

transform to points $X^T V$, which are now 3-by-k. Performing an exact probe does not become any harder. Our goal is still to maximize *trace* $X^T(VD)$. This is again equivalent to a problem of matching blue and green points to minimize the sum of squared distances between paired points. While the z dimension adds a new term to each entry f_{ij} in the cost matrix, the problem is still one of linear assignment.

Chapter 7

7. Experimental Results

In this chapter we present computational results using the eigenvector transformation and probe techniques.

In section 7.1 we examine the eigenvalue distributions of B matrices from various problem instances, and the distribution of "power" (α_r^2) contributed by the associated eigenvectors in good solutions. We exhibit feasible placements in which the spectral decomposition $x = \sum \alpha_r(x) u_r$ is dominated by a few eigenvectors with the lowest cost. The availability of such placements is the main rationale behind methods that project the solution points into spaces of low dimension.

Section 7.2 gives results for one-dimensional placement problems. We use iterated probes to produce slightly better placements than are obtained by exhaustive pairwise interchange. For these problems, previously known lower bounds on placement cost are a factor of two or three below the costs of the best placements produced by our algorithms. Using random probes, we prove lower bounds that narrow the cost gap to less than 20%.

In section 7.3 we study probe techniques for two-dimensional placement. The *TwoDimSplits* heuristic of section 6.3.3 gives projections that approximate the exact results for individual probe computations, with the typical error decreasing as the number of components is increased. We experiment with both iterated *exact* probes (computing the optimal linear assignment for each probe in the sequence) and iterated *approximate* probes (applying the heuristic for each probe). Iterated approximate probing is faster than exhaustive pairwise interchange, and for large problems it yields placements that are just as good. Iterated exact probing gives consistently

better placements, but it is substantially slower. We also present preliminary results with some real printed-circuit-board examples.

Our main test cases are randomly generated graphs. [Blanks85b] tested examples in which each of the n components is connected to exactly d others ("d-regular" graphs). For appropriately scaled $\sqrt{n}$-by-$\sqrt{n}$ grids, $\lambda_1+\lambda_2$ is an absolute lower bound on placement cost: Blanks found that for these graphs, pairwise interchange achieves costs within a few percent of this lower bound.

We first tested regular graphs on problems in which the nodes must be assigned to positions spaced evenly along a line. (Appendix 1 proved that uniform interval placement (UIP) is NP-hard for general graphs.) For d-regular graphs with d=5, pairwise interchange produced results analogous to those of Blanks: *i.e.*, placements with costs within a few percent of λ_1.

For UIP, the regular graphs we studied enjoy the unusual property that a single probe along u_1 produces a provably near-optimal placement x. We observed that $\alpha_1^2(x)$ by itself typically exceeds 98% of the total power in all eigenvectors. While this is certainly a success story for the eigenvector approach, the proper conclusion is that regular graphs are very special.

Real circuits are obviously not regular. A first step toward generating more realistic examples is thus to let components have different numbers of connections. We generated n-node graphs in which each edge appears independently with probability d/n; thus the *expected* degree of each node is roughly d, but the actual number of connections varies from one node to the next. We refer to this sample space of graphs as $G_{n,d/n}$. It is far more difficult to find good placements and prove good lower bounds on placement cost for graphs drawn from $G_{n,d/n}$ than it is for regular graphs.

In sparse cases (*e.g.*, $G_{n,3/n}$), it is possible for the nodes to fall into separate connected components. Usually the largest of these contains over 90% of the nodes. We restrict consideration to the nodes in the largest connected component, and the associated connection matrix.

For each graph, we compute all the eigenvalues and eigenvectors of B using the *IMSL FORTRAN* library routine *eigrs*, which has a running time of $(n/22)^3$ seconds for n-by-n matrices on a *DEC VAX* 11/785. We perform this computation once and store the eigenvectors and eigenvalues on disk for use in our algorithms and lower-bound proofs.

For typical circuits the B matrix tends to be sparse, and it would be significantly more efficient to use operator methods (*e.g.*, the method of Lanczos) to compute eigenvectors when n is large. The fundamental step in the Lanczos method is to calculate Bx for a vector x. If the average number of connections per component is d, this step requires approximately dn operations. Roughly $6n$ steps are required. Since the number of necessary auxiliary operations is quadratic, about $(6d+c)n^2$ operations are needed in all, where c is roughly 80.*

7.1. Distributions of λ_r and α_r^2

According to (2.2), placement cost in each dimension equals $\sum \alpha_r^2(x)\lambda_r$, where the λ_r's are the eigenvalues of the B matrix, and each $\alpha_r^2(x)$ is a coefficient indicating the contribution of eigenvector u_r to the placement. In a sense, the distribution of λ_r's sets the groundwork that defines the range of possible costs for *any* set of legal positions. The extent to which eigenvectors with the most favorable costs in this range can contribute to legal

*[Parlett, personal communication].

placements is determined by the specific positions.

We first consider the distribution of eigenvalues. Some theorems on random matrices are suggestive of what we might expect when sampling graphs from $G_{n,d/n}$. Specifically, a result of [Furedi81] applies to random graphs $G_{n,p}$, when p is fixed as n goes to infinity. In this case, the largest eigenvalue is approximately normally distributed with mean $(n-1)p+1$ and variance $2p$. The other eigenvalues obey a "semi-circle" distribution (eigenvalue density(x) proportional to $\sqrt{1-(x/r)^2}$ where r is a constant) between $-2\sqrt{np}$ and $+2\sqrt{np}$. The probability that even one of these eigenvalues has magnitude greater than $2\sqrt{np}+O(n^{1/3}\log n)$ tends to zero.

Several obstacles prevent us from applying these results to our situation. Our graphs are finite, and the theorems apply as n goes to infinity. More important, in the relevant *family* of graphs $G_{n,d/n}$, p does not stay fixed as n increases. To extend these results to sparse graphs is a nontrivial problem. [Boppana87] has obtained a similar bound on the magnitudes of eigenvalues for sample spaces in which p is allowed to decrease. But it does not apply to graphs in which the expected degree, d, is constant, since one of his conditions reduces to $2\sqrt{d} \geq (5\log n)^3$.

We are free to ask what the results of [Furedi81] would imply if they *did* extend to our graphs. The C matrix would have one eigenvalue near $d+1$ and the rest distributed between $-2\sqrt{d}$ and $+2\sqrt{d}$. For d-regular graphs, each eigenvalue of $B=D-C$ equals d minus an eigenvalue of C. In other words, the spectrum is reflected and shifted so that the smallest eigenvalue of B equals 0, and the gaps between eigenvalues are preserved. Suppose (without justification) that a similar relation holds for graphs drawn from $G_{n,d/n}$. The nonzero eigenvalues would then range between $d+1-2\sqrt{d}$ and $d+1+2\sqrt{d}$.

The above steps are mere speculation, but they provide a rough idea of what to expect from spectra of graphs drawn from $G_{n,d/n}$. Measured eigenvalue distributions for $n=256$ in the cases $d=32$, 8, and 3 are presented in Figures 7-1a, b, and c, respectively. Each plot is obtained by averaging data from three example graphs. The ordinate of each horizontal segment represents the number of eigenvalues observed between the abscissa values of its endpoints. In each case the nonzero eigenvalues span a somewhat greater range than our guess of $4\sqrt{d}$. The distribution for $d=32$ is roughly semi-circular. In the sparser cases, the distributions are increasingly skewed toward smaller eigenvalues.

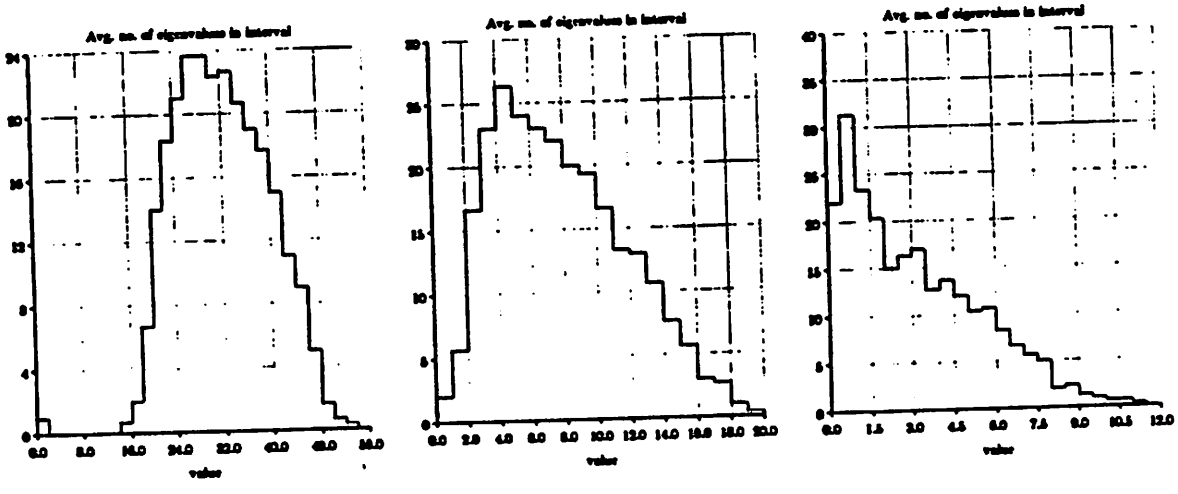


Figure 7-1a,b, and c. Distributions of λ_i's for (left to right:) $G_{256,32/256}$, $G_{256,8/256}$, and $G_{256,3/256}$.

The success of the eigenvector approach depends on the existence of placements whose spectral decompositions are dominated by eigenvectors with small associated eigenvalues. To demonstrate that such placements are available, we compare the spectral content of random legal placements and placements found by heuristics. We consider one sample graph from $G_{256,8/256}$ and one from $G_{256,3/256}$. Each plot in Figure 7-2b and c is the average of power spectra from five uniform-interval placements. The upper plots

correspond to random placements, and the lower plots to placements found by iterated probes.

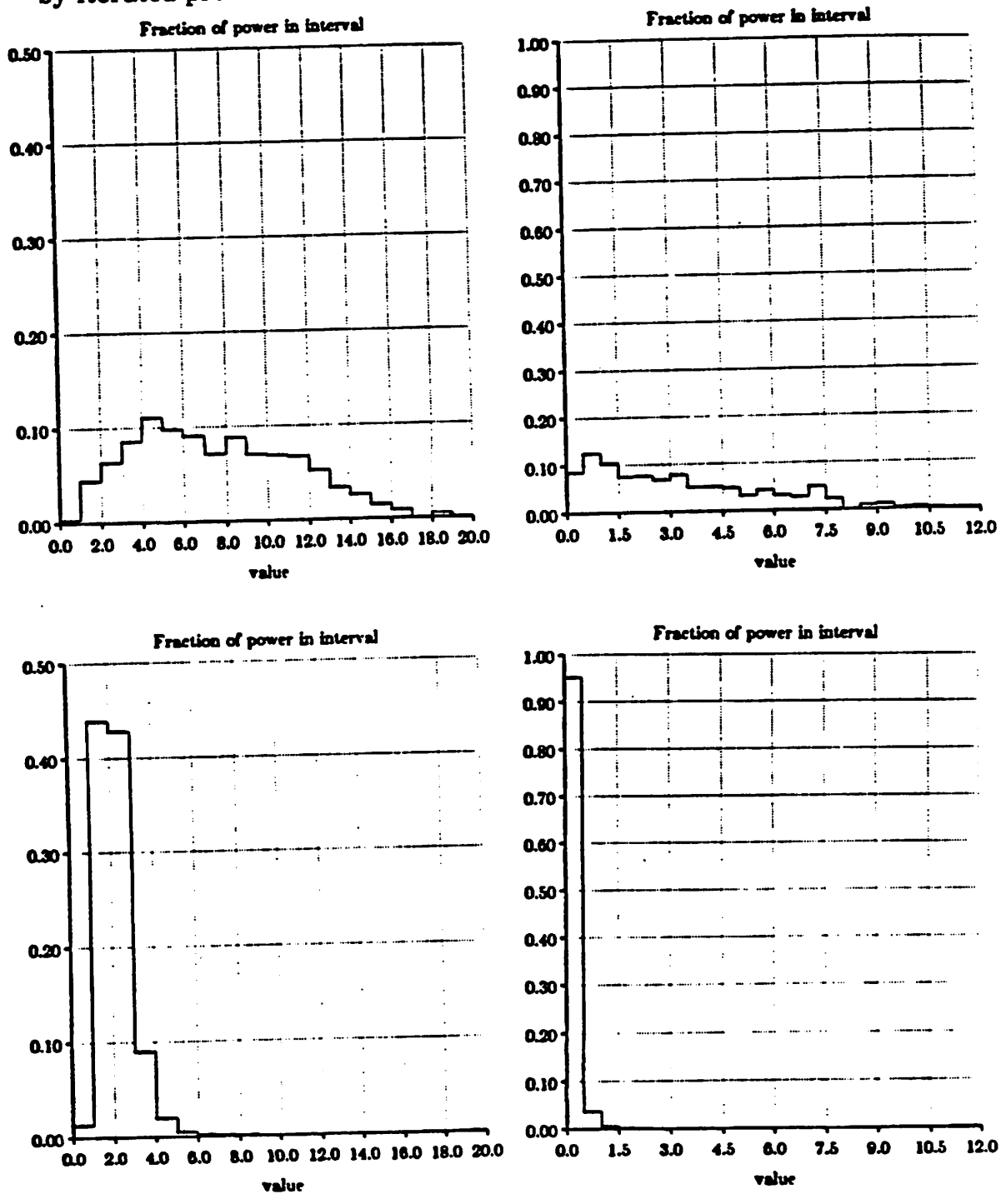


Figure 7-2b and c.
Power spectra of random placements (above) and good placements (below).
The plots 7-2b (at left) correspond to a graph from $G_{256,8/256}$.
The plots 7-2c (at right) correspond to a graph from $G_{256,3/256}$.

The lower spectra show, at least for these graphs, that good placements are characterized by a high concentration of power in the lowest-cost eigenvectors. The iterated probes algorithm was not fine-tuned for these placements. Any decent heuristic could produce placements with spectra essentially indistinguishable from the ones shown, at this level of detail.

7.2. Uniform interval placement of random graphs

We consider the problem of placing nodes at uniformly-spaced positions (UIP), for graphs drawn from $G_{n,d/n}$. Our test cases are six of the graphs for which the distributions of eigenvalues are plotted in Figure 7-1: three examples each for $d=3$ and $d=8$.

7.2.1. Low-cost placements

We compare the iterated-probes approach to the pairwise-interchange method. Using techniques suggested by [Blanks85b], a Pascal program was written that considers every pairwise interchange of n nodes and performs those that decrease cost (and the necessary updates) in roughly 5 seconds for $n=256$. (All run times refer to the *DEC VAX* 11/785.) Passes are repeated until one in which no exchange improves the cost. These examples require an average of 45 passes (240 seconds).

A Pascal implementation of the iterated-probes method described in section 3.3 performs all the stages in 40 seconds. (Appendix 4 gives the specific parameters used.) The resulting average costs were lower for every example. The average improvement was 8% for $d=3$ and 2% for $d=8$.

7.2.2. Lower bounds on cost

For each of the 6 graphs, we compare lower bounds (LB's) derived three ways: (1) by computing λ_1; (2) by summing component maxima $(x^T v_r)^2$ obtained from axis probes; and (3) by covering a space R^k with 10,000 random probes. For this application k ranges from 5 to 10, and each probe takes roughly 0.1 seconds. To evaluate the lower bounds, we take the cost of the best solution produced by iterated probes ("*IP*") as a fixed reference. The best lower bound is the one that most closely approaches this algorithmic cost. Hence our measure of tightness for $LB(i)$ is the residual percentage excess: $100 \cdot (\frac{IP}{LB(i)} - 1)$.

	Tightness of lower bound	
	average(6 examples)	[range]
LB(1)	124%	[48%,242%]
LB(2)	38%	[28%, 48%]
LB(3)	17%	[12%, 20%]

Joint consideration of several eigenvectors yields dramatic payoff in closing the cost gap between lower bounds and heuristic solutions.

7.3. Two-dimensional placement

For two-dimensional placement, the utility of probing is limited by the efficiency of algorithms for the linear assignment problem. As noted in section 6.3.1, the standard algorithm for the exact solution of this problem has $O(n^3)$ running time in the worst case. While 40 seconds suffice to run an entire iterated-probe sequence for a one-dimensional problem with 256 components, a single probe for a *two*-dimensional problem with half as many components takes roughly 90 seconds. Since an iterated-probe sequence can involve dozens of probes, this technique for *exact* probing seems impractical,

at least for large problems.

This is our motivation for comparing the performance of the linear assignment algorithm with that of the much faster heuristic discussed in section 6.3.3, which computes *approximate* probes. We first study the typical error associated with individual probes using the approximate technique. We then compare the performance of the exact and approximate techniques in iterated probes.

For legal positions we use the smallest nearly-square array (*e.g.*, 4×6, 10×12, 15×17) with enough grid points to accommodate the components. The precise method of selecting the legal grid points is described in appendix 4.

We compute approximate probes using *TwoDimSplits* (section 6.3.3), and exact probes using the assignment algorithm distributed by IMSL, Inc., as the routine *assct* [IMSL87].*

7.3.1. The error in approximating individual probes

We compare approximate with exact probes for two-dimensional placement of graphs drawn from $G_{n,3/n}$, for n ranging from 24 to 64. With fewer components, the data for approximate probes are not consistent enough to be meaningful. For example, when $n=8$ the technique finds the optimal solution in half the trials, but produces solutions with up to twice the optimal cost in others. (For problems this small, the optimal assignment can be computed quickly, anyway.)

---

* The method used to represent the real numbers of the cost matrix as an array of integers for this routine is given in appendix 4.

We generate three graphs each for $n = 24$, 36, 48, and 64, and for each graph we test 20 probes. We let V comprise the six eigenvectors with the smallest λ_r's: recall that the columns of V are given by $v_r = u_r\sqrt{H-\lambda_r}$.** Each probe uses a random direction in R^{12} as a pair of vectors d,e of length 6, and computes $a = Vd$ and $b = Ve$. The expressions (6.9) and (6.10) give two measures for the results of a probe. If x,y is the placement obtained, we can measure the *projection* $\sum_{i=1}^{n}(x[i]a[i] + y[i]b[i])$, which is to be maximized, and the *total squared distance* $\sum_{i=1}^{n}[(x[i]-a[i])^2 + (y[i]-b[i])^2]$, which is to be minimized.

We compare the median results of 20 exact probes and 20 approximate probes for each graph, according to both measures. For each ratio (exact/approximate projection, and approximate/exact total squared distance) there is a corresponding percentage error ($100\times(ratio-1)$). Figure 7-3 plots these percentage errors on separate curves: each data point is an average over three graphs with the same number of components.

**As suggested in section 3.2.1, H is set to the average of all eigenvalues besides λ_1 through λ_6.

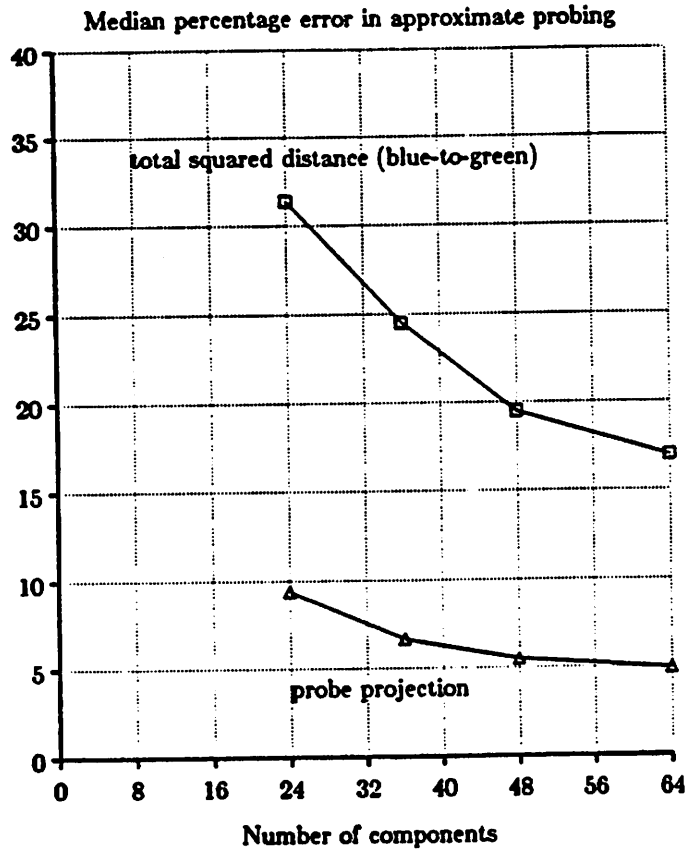


Figure 7-3.

The percentage errors in total squared distance are larger. One reason is that the difference between $\sum_{i=1}^{n}(x[i]-a[i])^2$ in the approximate and exact solutions is twice the difference between the optimal and approximate sums $\sum_{i=1}^{n}x[i]a[i]$. Another reason is that the former sums tend to be smaller.

It is hard to know whether the observed errors of 5 or 10 percent in the projections of approximate probes would seriously compromise techniques that substitute these for exact probes. We investigate this question for one technique in the next section. However, the results in Figure 7-3 are encouraging. The errors seem fairly small, and they decrease as problem size increases.

7.3.2. Iterated probes for the placement of random graphs

We now describe experiments with the method of iterated probes for two-dimensional placement. As in section 7.2, we use results with exhaustive pairwise interchange as a standard of comparison. We test the graphs discussed in the last section; in addition, we consider three graphs drawn from $G_{n,3/n}$ for each of the cases $n = 128$, 256, and 512. For these larger test problems we restrict the tests to pairwise interchange and iterated approximate probes, since iterated exact probes is too time-consuming.

In experiments with pairwise interchange on one-dimensional problems, we used random placements for our starting configurations. We justified this decision by verifying that the use of "good" starting placements, *e.g.*, derived by matching the component orderings of low-cost eigenvectors, did not improve the results.

For two-dimensional placement, [Blanks85b] reports better results using initial placements that attempt to minimize

$$\sum_{i=1}^{n}[(x[i]-u_1[i])^2+(y[i]-u_2[i])^2] \tag{7.1}$$

than with random initial placements. This starting placement is what would result from a special probe in which d_1 and e_2 are equal, and all other probe components are zero.

Our first experiments confirmed the results of Blanks; *i.e.*, we found that for two-dimensional problems, good initial placements do improve the results obtained by pairwise interchange. This led us to design a controlled experiment for comparing iterated probes with exhaustive pairwise interchange. We test five methods for each graph.

The first two methods correspond to those tested by Blanks. Method A, which serves as our baseline, runs exhaustive pairwise interchange from 20

randomly generated initial placements. Method B uses a single starting placement derived to match u_1, u_2. With $n \leq 128$, we compute this placement using the linear assignment algorithm to minimize (7.1). For $n > 128$ we use an approximate probe.

We then generate a set of 20 placements from probes using linear combinations Vd, Ve, where V consists of the eigenvectors $v_1 = u_1\sqrt{H - \lambda_1}$, $v_2 = u_2\sqrt{H - \lambda_2}$; and d, e are obtained from random directions in R^4. (In all cases except $n > 128$ we use the exact method for these probes.) Each of these 20 placements is used as an input to methods C (exhaustive pairwise interchange), D (iterated exact probes), and E (iterated approximate probes). Because method D is slow, we test it only for $n \leq 64$.

Both iterated-probe methods use stages, as in section 3.3. For two-dimensional placement problems, the iteration in each stage takes place in a space with two dimensions for each active eigenvector. In the first stage, 2 eigenvectors are active. Each succeeding stage doubles the number of active eigenvectors; the last stage is reached when this number is at least half the number of components.

The probes in every stage continue as long as the distance of the points from the origin increases. With iterated *approximate* probes, it is possible for this distance to decrease at some steps. When such a step is encountered, we reject it and terminate the stage. The placement cost at the end of each stage is recorded, and the lowest one is taken as the output of the sequence.

We plot the results of the experiment in Figure 7-4a. For each graph we take the median cost of 20 trials, and each data point is the average of the medians of three graphs with the same number of components. Since each median is normalized with respect to method A (pairwise interchange

with random starts), all points for this method are assigned the value 100%.

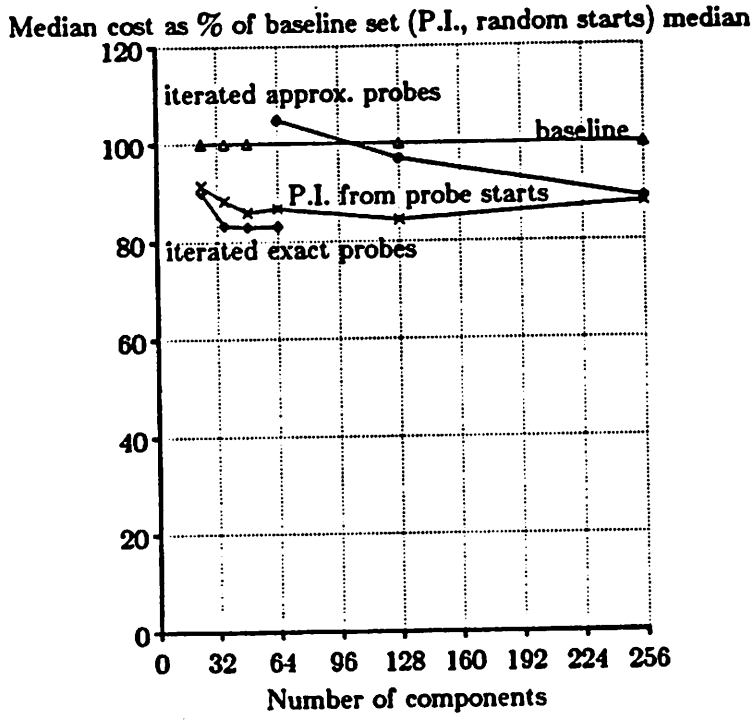


Figure 7-4a. Comparison of median placement costs obtained by iterated exact probes, iterated approximate probes, and pairwise interchange from probe starts.

For $n \leq 64$, method D (iterated exact probes) gives the lowest-cost placements. Method C (pairwise interchange from probe starts) is only slightly worse; it consistently produces placements that cost about 14% less than those obtained from the same algorithm with random starts.

For $n = 24$, 36, and 48, method E (iterated approximate probes) is essentially worthless. In the majority of these cases, the best placement that it produces is no better than the placement it is given to start with. Fortunately, just as the problem size becomes too big for the linear assignment algorithm to manage in reasonable running time (around $n > 64$), method E starts to become competitive. By $n = 256$, method E is essentially as good as method C.

We extended the comparison to $n=512$ to see if E (iterated approximate probes) would overtake C (pairwise interchange). E was about 5% better in one example, but 1% and 8% worse in the other two. The overall percentages (in terms of a baseline set with method A) were 93.7 for method E and 92.4 for method C. Method E took roughly 100 seconds per trial, which was five to six times faster than method C.

In Figure 7-4b we plot results in which the *best* placements from each method are compared to the best of method A. These results are similar to those for the medians, although the differences between the best costs of the different methods are not as great as the differences between the medians.

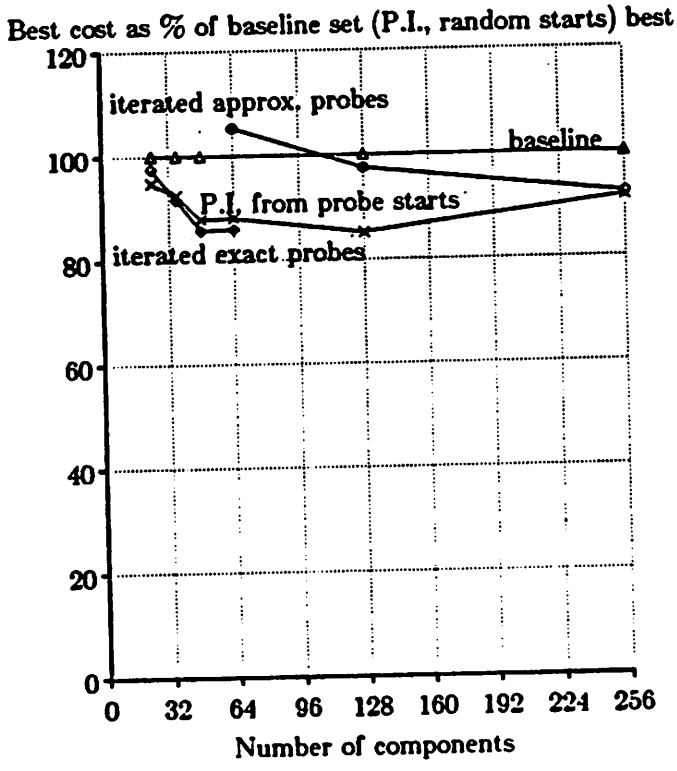


Figure 7-4b

This experiment is important for two reasons. First, by giving methods C,D, and E identical starting placements, we obtain a fairer test of the relative merits of these improvement heuristics than would be possible

otherwise. Second, we show that the eigenvector approach can contribute to placement algorithms in two distinct ways:

1) generating initial placements, and

2) improving given placements.

In methods D and E the approach serves both these functions. In method C, a probe is used only to generate the initial placement: this by itself improves the results obtained by pairwise interchange.

To complete the story of this experiment, we need to describe the results of method B: pairwise interchange from the (u_1,u_2) probe. In all 21 graphs, the cost obtained from method B was better than the median cost in the baseline set; however, in only 12 of the 21 graphs did it beat the *best* of the 20 trials with random starts. Furthermore, the cost of this placement was less than the best of set C (the 20 *probe* starts) for only one graph, and the difference in that case was 0.3%. We conclude that while this method for starting pairwise interchange is a good one, it offers no particular advantage when compared to other probes associated with low-cost eigenvectors.

This conclusion can lead to improvements of existing placement methods. For instance, consider the methods we reviewed in section 1.2.1.3 for constructive placement by constraint relaxation. Each of these procedures defines a problem with a unique relaxed solution, and works from there to a legal placement. Our work is the first demonstration of the possible advantages of exploiting whole *subspaces* of good relaxed placements when searching for good legal placements.

7.3.3. Real circuits

It is important to test our methods on real circuits. Very few circuit specifications have been published for use in testing placement algorithms. [Stevens72] published the net lists for five printed circuit boards from the ILLIAC IV. These circuits have from 67 to 151 components, in each case including 15 I-O pads. The components are placed on a 15-column grid, with the pads fixed in place along the bottom.

We use the weighting function described in section 5.1 to represent the nets. We have not yet implemented two-dimensional versions of iterated probes that incorporate fixed components. Instead, we use iterated probes to modify the x and y vectors in alternating stages, by applying the "class" constraints discussed in section 6.2. We increase the number of eigenvectors (of the reduced B matrix described in section 5.3) from stage to stage. (See appendix 4 for the parameters used.)

For each circuit we compare the lowest cost(x, y) from 20 runs of iterated probes (using mixed iteration) with that from 20 runs of exhaustive pairwise interchange. On average, the cost of our best placement was 3.7% greater. To discover the source of difficulty for probe methods in placing these circuits, we experimented briefly with variants on the legal positions. We tried one-dimensional placement with I-O pads fixed at the edge, and with pads free to move to any position. Our observations indicate that the difficulty is due more to the constraints associated with fixed components than the extension to real circuits and two dimensions, *per se*.

Chapter 8

8. Conclusions

In this chapter we review the main contributions of this work, and consider some directions for future research. We conclude by mentioning some problems in areas other than circuit placement for which our techniques may be useful.

8.1. Review

We began with the following circuit placement problem: given n legal positions, n components, and an n-by-n matrix of connections between components, assign the components to legal positions so that the sum of squared connection distances is minimized. We transformed this to an equivalent problem in which every feasible placement is represented by a point in $n-1$ dimensions, and the object is to find the point furthest from the origin. This was accomplished by expressing each placement as a weighted sum of certain eigenvectors, which contribute independently to placement cost.

We showed that it is possible to find the feasible point with maximum projection on any given direction in the transformed problem space. We call this operation a "probe". Individual probes can be used to produce good points, and iterated probes can be used to produce *sequences* of points at increasing distance from the origin. We found that a particularly effective way to apply iterated probes is to use successive stages that project the furthest-point problem into spaces with increasing numbers of dimensions. This allows early stages to focus on a few of the dimensions that have the greatest potential value.

We showed that methods to upper-bound the distance of the furthest point from the origin in k dimensions, for $k<n$, can be used to obtain lower bounds on placement cost that are better than those given by previous techniques. The upper bounds required can be proven with arbitrarily high probability using $O(\sqrt{k}(c_\varphi)^{k-1})$ random probes, where the desired tightness of the bound determines the constant c_φ. We also showed that the furthest point in a k-dimensional projection can be precisely determined by an adaptive algorithm that requires $O(n^{2(k-1)})$ probes in the worst case.

We generalized our approach to handle fixed components. Finally, we implemented several placement algorithms based on probes and tested them against an exhaustive pairwise interchange heuristic on a class of randomly generated test cases. For one-dimensional placement problems, our techniques were faster and produced slightly better solutions. For two-dimensional problems, the results depended on the size of the problem. In small cases (up to about 64 components), an exact implementation of iterated probes produced the best results, but it was substantially slower than pairwise interchange. In large cases (256 or more components), an approximate iterated probes heuristic was faster than pairwise interchange and gave equally good results. We obtained the best results for cases of intermediate size by using probes to generate the initial placements and pairwise interchange to improve them.

8.2. Directions for future work

Our investigation of circuit placement methods using multiple eigenvectors has opened several areas for future research. These include analytical questions, experimental tests, and possible extensions of the methods.

8.2.1. Analytical questions

Having introduced a new class of algorithms and lower-bound techniques, we face a fundamental question: For what kinds of problems can we expect these methods to perform well? It is not obvious how to pose this question mathematically. We could begin by defining a quality measure for problem instances. For an arbitrary integer k, let $Q(k)$ represent the maximum, over all legal placements, of the fraction of $\sum_r \alpha_r^2$ (squared eigenvector coefficients) that can be attained by the first k eigenvectors. Large values of $Q(k)$ signify favorable cases for our methods.

On the basis of experimental observations, we might conjecture that for uniform interval placement of random d-regular graphs, the expected value of $Q(1)$ converges to 1 as n goes to infinity. For other classes of connection matrices and legal positions, we could try to prove lower bounds on the expectation of $Q(k)$. Another challenge is to characterize *unfavorable* problem classes, *i.e.*, cases in which $Q(k)$ is very small.

The answers to such problems hinge on the probabilities that certain regions of the eigenvector domain contain feasible solution points. More specifically, consider all one-dimensional placement problems on n components with legal positions satisfying $\sum_{i=1}^{n} l^2[i]=1$: the feasible placements x are permutations of the l's. Let the orthonormal eigenvectors of the matrix B defined in (2.0) be u_r, and let each x be transformed into a point with n coordinates $\alpha_r(x)=x^T u_r$. For any such problem, all $n!$ feasible points lie on the unit sphere. We are interested in the likelihood that one or more of these points lie outside various cylinders of the form $\sum_{r=1}^{k} \alpha_r^2 = f$.

The distribution of the $n!$ points is determined by the closeness of the eigenvectors u_r to permutations of the legal positions; its analysis thus

requires an understanding of how the sets of components of the u_r's are distributed. To date very little research has been done to characterize the distributions of eigenvector components that arise from natural classes of matrices.

If we assume that the $n!$ points are distributed uniformly on the unit sphere, then it is easy to prove that $Q(1)$ approaches 1 as n increases. We follow the argument used to analyze random probe sets in chapter 4. With high probability, a set of $n!$ random points on the unit sphere in R^n contains at least one point near any given direction d, where *near* can be defined (for any $\delta > 0$) as within an angle of $\frac{e}{n}(1+\delta)$ radians. Of course, there is no reason to believe that the points from circuit placement problems are randomly distributed.

Other questions are relevant to the analysis of algorithms we have discussed. Recall from our description of iterated probes that if a probe aimed at point p discovers that p has the maximum projection in that direction, we call p "stable". How many points can be stable? For general furthest-point problems, it is possible for every point to be stable (consider the vertices of a regular polytope). It would be interesting to estimate the number of stable points in problems derived from circuit placement. A related problem is to estimate the maximum possible number of steps in an iterated-probes sequence.

In section 4.3 we computed the numbers of vertices ($|V|$) and facets ($|F|$) of convex hulls of placement polytopes in R^k: both are of order $O(n^{2(k-1)})$. We showed that $|V|+|F|$ probes suffice to map out the entire convex hull, but also that it is sometimes possible to find the furthest vertex with far fewer probes. Another interesting, albeit difficult problem is to analyze the expected number of probes needed by our heuristic to find the furthest point.

8.2.2. Experimental tests

Most of the algorithms we discussed have been implemented. A notable exception is procedure *ProbeForHull* of section 4.3. The primary application we discussed for this procedure was to prove lower bounds on placement cost; but because it finds points that are far from the origin, it can also be used as a placement algorithm. The implementation of this procedure itself involves some interesting issues in algorithm design, *e.g.*, how to update the convex hulls efficiently when new points are found. Once the algorithm is implemented, we can compare its performance with techniques such as random probes for lower bounds or iterated probes for placements.

The results of tests of our techniques with random graphs have been encouraging. Preliminary results with the ILLIAC IV boards (section 7.3.3), which include several fixed components, were less successful. We have not yet tested two-dimensional iterated probes on these examples. In any case, more work is needed to assess the merits of our techniques for practical problems with fixed components.

8.2.3. Possible extensions

The experiments in chapter 7 demonstrate that the quality of two-dimensional placements generated with probes depends largely on the accuracy of the algorithm used for Euclidean^2 blue-green matching. Thus an important challenge is to develop improved algorithms for this problem. The ideal algorithm would yield the optimal solution in a running time guaranteed to be significantly shorter than the $O(n^3)$ associated with the Hungarian algorithm. Even if this goal proves elusive, several alternatives are available.

The approach in [Karp80] finds the optimal solution to arbitrary linear assignment problems, and has an *expected* running time of $O(n^2 \log n)$ when the entries of the cost matrix are independent. The running time analysis does not apply to our problems, but it is possible that this algorithm will provide substantial speedup for them as well.

In iterated probes, each blue-green matching determines a new probe, and each probe aims to maximize a new objective function. There may be more effective ways to conduct iterations than solving for the optimal value at every step. For example, the following procedure *improves* the objective function whenever this is possible, without doing all the work needed to optimize it.

We construct a directed graph in which each node represents a legal position. The edge from i to j gets a cost corresponding to the effect on the objective function of reassigning the component at position i to position j. By itself any such move is infeasible because j is already occupied, but any directed *cycle* of such moves leads to a new feasible assignment. The net change in the objective function from such a cycle equals the sum of the costs of the edges used. The objective function can therefore be improved if and only if the graph has a cycle of negative total cost. Detecting a negative-cost cycle to improve an assignment is generally much easier than computing the optimal assignment. However, it can take $O(n^3)$ time in the *worst* case ([Lawler76]).

Many techniques besides iterated probes can be developed to find points far from the origin in our transformed problem. For instance, suppose that a sequence of iterated probes has converged to a stable point p. To locate a point further from the origin, one strategy is to determine the local structure of the convex hull of the feasible points. It should be possible to find

the facets that contain p by using adaptive probes (as in *ProbeForHull*) in this neighborhood. If any of these facets contains a point of greater magnitude than p, we could in turn map out *its* incident facets.

We can also consider modifying the original problem formulation to address additional practical considerations in our framework. Appendix 5, which suggests a way to make our cost function more sensitive to potential routing congestion, is a first attempt.

Finally, it would be useful to develop techniques analogous to the probe for more general placement problems. For instance, it should be possible to handle placement problems with surplus legal positions, and one-dimensional placement of components with different widths. The difficulty in both these cases is that $\sum x^2[i]$ is no longer constant for all feasible placements. In our derivation, equation (2.5) relies on this property.

8.2.4. Applications to partitioning

Although we developed probe techniques for circuit placement, they can also be used in other applications. In particular, the case of our problem known as graph partitioning (section 2.1.4) can arise in many situations, *e.g.*, allocating computer programs to pages of memory, data files to storage devices, or employees to offices, as well as circuit components to layout regions.

In each situation, the number of connections between blocks of the partition is a natural measure of cost. To represent partitions we use vectors whose elements are $n/2$ copies of each of two numbers, so that each block corresponds to the components with the same number. Using combinations of eigenvectors as probes for good *partitions* allows us to apply our methods in a variety of problem domains.

Appendix 1

Proof that UIP is NP-hard

We transform any instance of "graph partition," an NP-complete decision problem mentioned in section 2.1.4, into an instance of "uniform-interval placement" (UIP). We then show that the optimal solution to the UIP instance provides the correct answer to the original graph partition problem. Since we can perform the problem transformation in polynomial time, this constitutes a proof that UIP is NP-hard.

Graph partition

INSTANCE: a graph (V,E) on nodes 1 to $n=|V|$ (n even); an integer k.

QUESTION: Is there a partition of V into disjoint sets S and T of $n/2$ nodes, such that no more than k edges in E run between S and T?

Transformation to UIP

We produce an instance of UIP with $n+2H+1$ circuit components (which we refer to from now on as nodes), where $H=\frac{n^3}{2}$. Nodes 1 through n correspond to the nodes in V; $n+1$ through $n+2H+1$ are new. The symmetric matrix C is determined by the entries c_{ij} with $i \le j$. We set $c_{ij}=0$ for all $i \le j$ <u>except</u>

$$\text{for } 1 \le i < j \le n \text{ and } <i,j> \in E,\ c_{ij}=1 \qquad (|E| \text{ "original" edges;})$$

$$\text{for } n+1 \le i \le n+2H,\ c_{i,i+1}=n^{12} \quad (n^3 \text{ "chain" edges;}) \text{ and}$$

$$\text{for } 1 \le i \le n,\ c_{i,n+H+1}=n^5 \quad (n \text{ "star" edges.})$$

Define legal positions at $(-H-\frac{n}{2},0)$, $(-H-\frac{n}{2}+1,0)$, $\cdots$, $(H+\frac{n}{2},0)$. We will prove that the answer to the original graph partition problem is "yes" if and only if the optimal solution to this instance of UIP has cost(UIP) $< n^{15} + D \cdot n^5 + (k+1)n^6$, where

$$D=(nH-\frac{n^2}{4})\cdot(H+n+1)-n^2\frac{H}{4}+(\frac{n^3}{3}+\frac{n^2}{2}+\frac{n}{6}).$$

Proof:

"Yes" for graph partition → optimal cost(UIP) $< n^{15} + D\cdot n^5 + (k+1)n^6$.

Place nodes $n+1$ through $n+2H+1$ in order at positions $(-H,0)$ through $(H,0)$; assign the $\frac{n}{2}$ nodes of S arbitrarily to the positions with $x<-H$ and the $\frac{n}{2}$ nodes of T to the positions with $x>H$.

The n^3 chain edges all have length 1; each costs n^{12}, for a total of n^{15}. A dry calculation shows that the n star edges have a total cost of Dn^5. We thus need only show that the placement cost associated with the original edges is less than $(k+1)n^6$. By assumption S and T may be chosen so at most k edges run between them; each of these has squared length at most $(n^3+n)^2$. Fewer than $\frac{n^2}{4}$ edges are *internal* to S and T, and each has squared length at most $(\frac{n}{2})^2$. We thus need

$$k(n^6+2n^4+n^2)+\frac{n^2}{4}(\frac{n^2}{4})<(k+1)n^6 \text{ or simply}$$

$$k\cdot 2n^4+k\cdot n^2+\frac{n^4}{16}<n^6,$$

and this is true because we always have $k\le\frac{n^2}{4}$.

Optimal cost(UIP) $< n^{15} + D\cdot n^5 + (k+1)n^6$ → "yes" for graph partition.

We prove that a placement for this instance of UIP can achieve the specified cost only if it puts the "original" nodes at the extreme positions as in the above configuration, and the corresponding S:T partition separates fewer than $k+1$ edges. First observe that nodes $n+1$ through $n+2H+1$ must be placed in sequence (in either the above order or its reverse) because only then do all the chain edges achieve their minimum possible cost,

namely n^{12}. In any other arrangement, the cost of some chain edge would be at least $4 \cdot n^{12}$, and the total cost of the chain edges would be at least $n^{15} + 3 \cdot n^{12}$. Given that $D < n^7$, cost(UIP) would be greater than we assumed.

Since the new nodes must be placed in sequence with no gaps, nodes 1 through n must occupy two blocks of consecutive positions: a at the left and $n-a$ at the right. The total cost of the star edges equals $n^5 \cdot [D + (2H+n+1)(a-\frac{n}{2})^2]$. This cost is minimized when $a = \frac{n}{2}$, and for any other choice of a it would be at least n^8 larger. The blocks must therefore be balanced ($a = \frac{n}{2}$) if cost(UIP) is as assumed.

When nodes $n+1$ through $n+2H+1$ occupy positions $(-H,0)$ through $(H,0)$ in order, our assumption about cost(UIP) implies that the cost of the original edges is less than $(k+1)n^6$. Since the cost of each original edge that runs between the left and right sides is greater than n^6, fewer than $k+1$ original edges can do so. Thus the UIP placement provides a "yes" answer to the original graph partition problem, as claimed.

Appendix 2

"Minimum projection magnitude" is NP-hard

In section 2.4, we claimed that to minimize $|x^T(Vd)|$ for fixed Vd is NP-hard. We transform any instance of "set partition," an NP-complete decision problem [Garey79, page 223], into an instance of minimum projection magnitude.

Set Partition

INSTANCE: Finite set A with $|A|$ even, and a positive integer $s(a)$ for each $a \in A$.

QUESTION: Is there a subset A' of A such that $|A'| = |A|/2$ and

$$\sum_{a \in A'} s(a) = \sum_{a \in A - A'} s(a)?$$

Transformation to Minimum Projection Magnitude

We produce a vector l of $|A|$ legal positions, half of which equal -1 and half of which equal 1. (Feasible vectors are those that satisfy $x \in \{\Pi l\}$ for permutations Π.) Let Vd be a vector with the numbers $s(a)$ as components. The minimum-projection-magnitude problem is to find a feasible x that minimizes $|x^T(Vd)|$.

It is easy to verify that the answer to set partition is "yes" if and only if the solution to the minimum-projection-magnitude problem is 0.

Appendix 3

Proof of (4.11): $\tan^2\varphi(S) < \frac{.256+\ln(k)}{4}$

The error is determined by a representative "worst-covered vector", x. By symmetry, we can restrict our search for x to the positive orthant. Again by symmetry, it suffices to find the worst vector in any of the $k!$ cones that correspond to the different orderings of the components: for definiteness, suppose $x_1 \geq x_2 \geq \cdots \geq x_k \geq 0$. This cone is bounded by the probes $p_1=(1,0,\ldots,0)$, $p_2=\frac{1}{\sqrt{2}}(1,1,0,\ldots,0)$, $p_3=\frac{1}{\sqrt{3}}(1,1,1,0,\ldots,0),\ldots,$ $p_k=\frac{1}{\sqrt{k}}(1,1,\ldots,1)$.

If a nonzero vector x makes the same angle with each of k distinct probes and x lies in the cone that these probes determine, then in the sense of (4.9), x is the vector in the cone that is worst-covered by those probes. In the example at hand, we solve for x by equating the quantities $\frac{1}{\sqrt{j}}\sum_{i=1}^{j} x_i$ for $j = 1$ to k (the inner products of x with the probes), and then verify that the solution lies in the cone. For convenience fix $x_1=1$, so that $\cos^2(x,p_1)=1/\|x\|^2$, and $\tan^2\varphi=\|x\|^2-1$. Thus we have

$$x_1=1$$
$$x_1+x_2=\sqrt{2}$$
$$x_1+x_2+x_3=\sqrt{3}$$
$$x_1+x_2+\cdots+x_k=\sqrt{k}.$$

The trivial solution is $x_j=\sqrt{j}-\sqrt{j-1}$, so x indeed lies in the cone. We conclude that $\tan^2\varphi=\sum_{j=2}^{k}(\sqrt{j}-\sqrt{j-1})^2$.

We now show that this sum is less than $\frac{.256+ln(k)}{4}$, which will complete the proof of (4.11). For $j=2$ and $j=3$, the bound holds by inspection. Let $j \geq 4$. We use the following lemma:

LEMMA: For all $j \geq 2$, $\left[\sqrt{j} - \sqrt{j-1}\right]^2 < \frac{1}{4(j-.6)}$.

From this lemma, we have (for $j \geq 4$)

$$\sum_{j=2}^{k}\left[\sqrt{j} - \sqrt{j-1}\right]^2 < \tfrac{1}{4}\left[\frac{1}{1.4} + \frac{1}{2.4} + \sum_{j=4}^{k}\frac{1}{j-.6}\right]$$

$$< \tfrac{1}{4}\left[\frac{1}{1.4} + \frac{1}{2.4} - ln(2.4) + ln(k-.6)\right]$$

$$< \frac{.256 + ln(k-.6)}{4} < \frac{.256 + ln(k)}{4}.$$

It remains only to prove the lemma. First observe that for $j \geq 2$,

$$\frac{1}{4\left[1 - \frac{.3}{j}\right]} < .3$$

(Equality holds when $j = 1.8$). The rest is algebra. First divide by $(j-.3)j$:

$$\frac{1}{4(j-.3)^2} < \frac{.3}{(j-.3)j}$$

Add $1 - \frac{1}{j-.3}$:

$$1 - \frac{1}{j-.3} + \frac{1}{4(j-.3)^2} < 1 + \frac{-j+.3}{(j-.3)j}$$

Take square roots:

$$1 - \frac{1}{2(j-.3)} < (1 - \tfrac{1}{j})^{\frac{1}{2}} \quad or$$

$$1 - (1 - \tfrac{1}{j})^{\frac{1}{2}} < \frac{1}{2(j-.3)}$$

Finally, squaring both sides and multiplying by j:

$$\left[\sqrt{j} - \sqrt{j-1}\right]^2 < \frac{j}{4(j-.3)^2} < \frac{1}{4(j-.6)},$$

which concludes the proof of the lemma.

Appendix 4

Details of experiments

Iterated probes for UIP (section 7.2.1): The number of dimensions used in the first stage ($k[1]$) equals $\lceil .4\sqrt{n} \rceil$. The number of dimensions (s) used to select the random starting direction for each trial is chosen uniformly at random from integers in the range $[k[1]-\sqrt{k[1]}, k[1]+\sqrt{k[1]}]$. The value of H for the initial probe equals λ_{s+1}. Each stage after the first uses twice as many active dimensions as the previous one, until the number exceeds $n/2$. The maximum number of iterated probe steps allowed in a stage with k eigenvectors equals the nearest integer to $2n/k$. The parameter ϵ equals 0: *i.e.*, iteration continues no matter how much power in the solution is concentrated in the current dimensions.

Choice of legal grid points (section 7.3): Let m be the number of components to be placed. (Recall that m is typically slightly less than n, because only the nodes in the largest connected component are placed.) The points lie in r evenly-spaced rows, where r equals $\lfloor \sqrt{m} \rfloor$. We initialize a first column with points in every row. Proceeding to the right, we initialize additional columns, spaced at uniform intervals. When m is not a multiple of r, the rightmost column has points in the lower $m \bmod r$ rows, and its top rows are left empty.

The legal positions are normalized so that the sums of the x positions and of the y positions equal 0, and the sums of squares equal 1.

The cost matrix for *assct* (section 7.3): The algorithm *assct* for linear assignment requires an integer cost matrix; we must therefore sacrifice some precision in representing the squared distances between blue and green points in this matrix. We do so by multiplying the coordinates of all

the points by an appropriate scale factor and rounding each result to the nearest integer before computing squared distances.

The sums of squares of legal positions in x and in y equal 1; the corresponding sums can be slightly larger for the positions determined by the probe. We use a scale factor of 10,000: this provides sufficient precision for the values of n being tested, and is small enough to avoid overflow with 32-bit integers.

Mixed iteration for placement of ILLIAC IV boards (section 7.3.3): We use the parameters described above for UIP, with two exceptions. First, the number of dimensions used in the first stage ($k[1]$) equals $\lceil\sqrt{n}\rceil$.

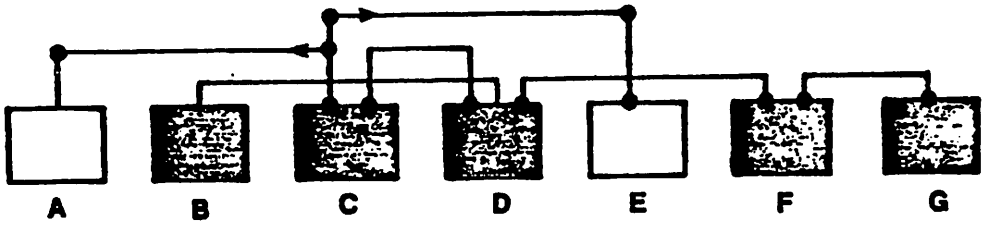
Second, the choice of H for the initial probe is more complicated. We usually set it to the average of the $n-1-s$ eigenvalues associated with eigenvectors not used for that probe. But because the eigenvectors are selected in decreasing order of $|u_r{}^T g| - A\lambda_r$ (equation (5.6)), it is possible for this average to be less than some "active" eigenvalues. In this case H is set to the least eigenvalue greater than all the active ones.

Finally, there is the question of what value of A to use when ranking the eigenvectors according to (5.6). On the basis of empirical tests, we chose $A = 0.13$ for these circuits.

Appendix 5

A way to make our cost function sensitive to congestion

Congestion at the center of the placement tends to cause the most severe routing difficulties. As stated, our cost function is insensitive to this problem. Suppose that only two wiring tracks are available: then we can illustrate "insensitivity" with a small example.



The connection between positions C and E is a serious problem: it makes the placement infeasible, since it requires a third track. The connection between C and A is no problem. But since C-A and C-E have equal length, they are currently assigned the same cost.

To sensitize our cost function to central congestion, we would like to make wires that cross the center more costly per unit length. An expedient that achieves this goal is to move the legal positions in the problem specification away from the middle. (*E.g.*, move positions A,B,C to the left and E,F,G to the right.) In the evaluation of any placement, connections that traverse the center (*e.g.*, C-E) then cost more than comparable connections that do not (*e.g.*, C-A).

References

Ajtai84.
M. Ajtai, J. Komlós and J. Tusnády, On optimal matchings, *Combinatorica 4*, 4 (1984), 259-264.

Avis83.
D. Avis, A survey of heuristics for the weighted matching problem, *Networks 13* (1983), 475-493.

Barnes82a.
E. R. Barnes, An algorithm for partitioning the nodes of a graph, *SIAM Journal on Algebraic and Discrete Methods 3* (1982), 541-550.

Barnes82b.
E. R. Barnes and A. J. Hoffman, Partitioning, spectra and linear programming, IBM Research Report 9511 (#42058), Aug. 1982.

Barnes84.
E. R. Barnes, A. Vannelli and J. Q. Walker, A new procedure for partitioning the nodes of a graph, IBM Research Report 10561 (#47264), June 1984.

Blanks85a.
J. P. Blanks, Use of a quadratic objective function for the placement problem in VLSI design, Doctoral Dissertation, University of Texas at Austin, April 1985.

Blanks85b.
J. P. Blanks, Near-optimal placement using a quadratic objective function, in *Proceedings of the 22nd Design Automation Conference*, June 1985, 609-615.

Boppana87.
R. Boppana, Personal communication, 1987.

Charney68.
H. L. Charney and D. L. Plato, Efficient partitioning of components, in *Proceedings of the 5th Annual Design Automation Workshop*, July 1968, 16-1 to 16-21.

Cheng84.
C. Cheng and E. S. Kuh, Module placement based on resistive network design, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems CAD-3*, 3 (July 1984), 218-225.

Cover65.
T. M. Cover, Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition, *IEEE Transactions on Electronic Computers EC-14* (1965), 326-334.

Cover67.
T. M. Cover, The number of linearly inducible orderings of points in d-space, *SIAM Journal of Applied Mathematics 15*, 2 (March 1967).

Dobkin.
D. Dobkin, H. Edelsbrunner and C. K. Yap, Probing convex polytopes, in *Proceedings of the 18th ACM Symposium on Theory of Computing*, 424-432.

Dunlop85.
A. E. Dunlop and B. W. Kernighan, A procedure for the placement of standard-cell VLSI circuits, *IEEE Transactions on Computer-Aided Design CAD-4*, 1 (Jan. 1985), 92-98.

Füredi81.
Z. Füredi and J. Komlós, The eigenvalues of random symmetric matrices, *Combinatorica 1*, 3 (1981), 233-241.

Fiduccia82.
C. M. Fiduccia and R. M. Mattheyses, A linear time heuristic for improving network partitions, in *Proceedings of the 19th Design Automation Conference*, June 1982, 175-181.

Frankle86.
J. Frankle and R. M. Karp, Circuit placements and cost bounds by eigenvector decomposition, in *Proceedings of the IEEE International Conference on Computer-Aided Design*, Nov. 1986, 414-417.

Garey76.
M. R. Garey, D. S. Johnson and L. Stockmeyer, Some simplified NP-complete graph problems, *Theoretical Computer Science 1* (1976), 237-267.

Garey79.
M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company, San Francisco, 1979.

Goodman86.
J. E. Goodman and R. Pollack, Upper bounds for configurations and polytopes in R^d, *Discrete & Computational Geometry 1* (1986), 219-227.

Goto81.
S. Goto, An efficient algorithm for the two-dimensional placement problem in electrical circuit layout, *IEEE Transactions on circuits and systems CAS-28*, 1 (Jan. 1981).

Hall70.
K. M. Hall, An r-dimensional quadratic placement algorithm, *Management Science 17*, 3 (Nov. 1970), 219-229.

Hanan72.
M. Hanan and J. M. Kurtzberg, A review of the placement and quadratic assignment problems, *SIAM Review 14*, 2 (April 1972), 324-342.

Hartoog86.
M. R. Hartoog, Analysis of placement procedures for VLSI standard cell layout, in *Proceedings of the 23rd Design Automation Conference*, June 1986, 314-319.

IMSL87.
IMSL, *The IMSL User's Manual*, IMSL, Inc., Houston, Texas, 1987. For information about IMSL routines, phone (800)-222-4675.

Just86.
K. M. Just, J. M. Kleinhans and F. M. Johannes, On the relative placement and the transportation problem for standard-cell layout, in *Proceedings of the 23rd Design Automation Conference*, June 1986, 308-313.

Karp75.
R. M. Karp, On the computational complexity of combinatorial problems, *Networks 5* (1975), 45-68.

Karp80.
R. M. Karp, An algorithm to solve the m x n assignment problem in expected time O (mn log n), *Networks 10* (1980), 143-152.

Kernighan70.
B. W. Kernighan and S. Lin, An efficient procedure for partitioning graphs, *Bell System Technical Journal*, Feb. 1970, 291-307.

Kirkpatrick83.
S. Kirkpatrick, C. D. G. Jr. and M. P. Vecchi, Optimization by simulated annealing, *Science 220*, 4598 (May 13, 1983).

Lawler76.
E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New York, 1976.

Metropolis53.
N. Metropolis, A. W. Rosenbluth, A. H. Teller and E. Teller, Equation of state calculations by fast computing machines, *Journal of Chemical Physics 21*, 6 (June 1953), 1087-1093.

Muller59.
M. E. Muller, A note on a methods for generating points uniformly on n-dimensional spheres, *Communications of the Association for Computing Machinery 2*, 4 (1959), 19-20.

Munkres57.
J. Munkres, Algorithms for the assignment and transportation problems, *Journal of the Society for Industrial and Applied Mathematics 5*, 1 (March 1957).

Nahar85.
S. Nahar, S. Sahni and E. Shragowitz, Experiments with simulated annealing, in *Proceedings of the 22nd Design Automation Conference*, June 1985, 748-752.

Nahar86.
S. Nahar, S. Sahni and E. Shragowitz, Simulated annealing and combinatorial optimization, in *Proceedings of the 23rd Design Automation Conference*, June 1986, 293-299.

Otten82.
R. H. J. M. Otten, Automatic floorplan design, in *Proceedings of the 19th Design Automation Conference*, June 1982, 261-267.

Papadimitriou82.
C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

Parlett80.
B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1980.

Preas86.
B. T. Preas and P. G. Karger, Automatic placement: a review of current techniques, in *Proceedings of the 23rd Design Automation Conference*, June 1986, 622-629.

Quinn, Jr.79.
N. R. Quinn, Jr. and M. A. Breuer, A force directed component placement procedure for printed circuit boards, *IEEE Transactions on Circuits and Systems CAS-26*, 6 (June 1979), 377-388.

Rankin55.
R. A. Rankin, The closest packing of spherical caps in n dimensions, *Glasgow Math. Association Proceedings 2* (1955), 139-144.

Rogers63.
C. A. Rogers, Covering a sphere with spheres, *Mathematika 10* (1963), 157-164.

Sahni80.
S. Sahni and A. Bhatt, The complexity of design automation problems, in *Proceedings of the 17th Design Automation Conference*, 1980, 402-411.

Schweikert72.
D. G. Schweikert and B. W. Kernighan, A proper model for the partitioning of electrical circuits, in *Proceedings of the 9th Design Automation Workshop*, June 1972, 56-62.

Sechen86.
C. Sechen and A. Sangiovanni-Vincentelli, Timberwolf 3.2: a new standard-cell placement and global routing package, in *Proceedings of the 23rd Design Automation Conference*, June 1986, 423-439.

Seidel86.
R. Seidel, Constructing higher-dimensional convex hulls at logarithmic cost per face, in *Proceedings of the 18th ACM Symposium on Theory of Computing*, May 1986, 404-413.

Sidel'nikov73.
V. M. Sidel'nikov, On the densest packing of balls on the surface of an n-dimensional Euclidean sphere and the number of binary code vectors with a given code distance, *Soviet Math Dokl. 14*, 6 (1973), 1851-1855.

Sloane81.
N. J. A. Sloane, Tables of sphere packings and spherical codes, *IEEE Transactions on Information Theory IT-27*, 3 (May 1981), 327-338.

Soukup81.
J. Soukup, Circuit layout, *Proceedings of the IEEE 69*, 10 (Oct. 1981), 1281-1304.

Stevens72.
J. E. Stevens, Fast heuristic techniques for placing and wiring printed circuit boards, Doctoral Dissertation, Computer Science Department, University of Illinois, 1972.

Ullman84.
J. D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Inc., Rockville, Maryland, 1984.