THE COMPLEXITY OF DESIGN AUTOMATION PROBLEMS

Sartaj Sahni*+
University of Minnesota

Atul Bhatt++
Sperry Corp.

Raghunath Raghavan*+++
Mentor Graphics

ABSTRACT
This paper reviews several problems that arise in the area of design automation. Most of these problems are shown to be NP-hard. Further, it is unlikely that any of these problems can be solved by fast approximation algorithms that guarantee solutions that are always within some fixed relative error of the optimal solution value. This points out the importance of heuristics and other tools to obtain algorithms that perform well on the problem instances of interest.

KEYWORDS AND PHRASES
design automation; complexity; NP-hard; approximation algorithm.

---

*The work of these authors was supported in part by the National Science Foundation under grants MCS 78-15455 and MCS 80-005856.

+ Address: Department of Computer Science, University of Minnesota, Minneapolis, MN 55455
++ Address: Sperry Corporation, P.O. Box 43942, St.Paul, MN 55l64.
+++ Address: Mentor Graphics, Beaverton, OR 97005.

1. INTRODUCTION

Over the past twenty years, the complexity of the computer design process has increased tremendously. Traditionally, much of the design has been done manually, with computers used mainly for design entry and verification, and for a few menial design chores. It is felt that such labor-intensive design methods cannot survive very much longer. There are two main reasons for this.

The first reason is the rapid evolution of semiconductor technology. Increases in the levels of integration possible have opened the path for more complex and varied designs. LSI technology has already taxed traditional design methods to the limit. With the advent of VLSI, such methods will prove inadequate. As a case in point, the design of the Z8000 microprocessor, which qualifies as a VLSI device, took 13,000 man-hours and many years to complete. In fact, it has been noted [NOYC77] that industry-wide, the design time (in man-hours per month) has been increasing exponentially with increasing levels of integration. Clearly, design methods will have to go from labor-intensive to computer-intensive.

Secondly, labor-intensive methods do not adequately accomodate the increasingly more stringent requirements for an acceptable design. Even within a given technology, improvements are constantly sought in performance, cost, flexibility, reliability, maintainability, etc. This increases the number of iterations in the design cycle, and thus requires smaller design times for each design step.

Industry-wide, the need for sophisticated design automation (DA) tools is widely recognized. To date, most of the effort in DA has concentrated on the following stages of the design process: physical implementation of the logic design, and testing. DA for the early stages of the design process, involving system specification, system architecture and system design, is virtually nonexistent. Though DA does not pervade the entire design process at this time, there are a number of tools that aid in, rather than automate, certain design steps. Such computer-aided design tools can dramatically cut design times by boosting designer productivity. We shall restrict our attention to problems encountered in developing tools that automate, rather than aid in, certain design steps.

In the light of the need for more advanced and sophisticated DA tools, it becomes necessary to re-examine the problems tackled in developing such tools. They must be thoroughly analyzed and their complexity understood. (The term "complexity" will be defined more precisely, using concepts from mathematics and computer science, later in this chapter.) A better understanding of the inherent difficulty of a problem can help shape and guide the search for better solutions to that problem. In this paper, several problems commonly encountered in DA are investigated and their complexities analyzed. Emphasis is on problems involving the physical implementation and testing stages of the design process.

Section 2 contains a brief description, in general terms, of the DA problems considered. The concepts of complexity and nondeterminism are introduced and elaborated upon in Section 3. This section also includes other background material.

The problems described in Section 2 are analyzed in Section 4. Each problem is mathematically formulated and described in terms of its complexity. Most problems under discussion are shown to be NP-hard. In addition, a brief account in Section 5 describes ways of attacking

these problems via heuristics and what are called "usually good" algorithms.

The book edited by Breuer [BREU72a] and a survey paper by him [BREU72b] provide a good account on DA problems, techniques for solutions, and their applications to digital system design. Though these efforts are about ten years old, the problems as formulated therein are still very representative of the kinds of problems encountered in designing large, fast systems using MSI/LSI technology and a hierarchy of physical packaging. The book by Mead and Conway [MEAD80] describes a design methodology that appears to be appropriate for VLSI. This design methodology gives rise to a number of design problems, some of which are similar to problems encountered earlier, and some that have no counterpart in MSI/LSI-based design styles. W. M. van Cleemput [CLEE76] has compiled a detailed bibliography on DA related disciplines. The computational aspects of VLSI design are studied in the book [ULLM84]. David Johnson's ongoing column "The NP-Completeness Column" in the Journal of Algorithms, Academic Press, is a good source for current research on NP-completeness. In fact, the Dec. 1982 column is devoted to routing problems, [JOHN82].

2. SOME DESIGN AUTOMATION PROBLEMS

There are numerous steps in the process of designing complex digital hardware. It is generally recognized that the following classes of design activities occur:

(1) System design.
This is a very high-level architectural design of the system. It also defines the circuit and packaging technologies to be utilized in realizing the system. (While it might appear that this is a bit too early to define the circuit and packaging technologies, such is not the case. It is necessary if one is to obtain cost/performance and physical sizing estimates for the system. These estimates help confirm that the system will be well-suited, cost- and performance-wise, to its intended application.) Clearly, system design defines the nature and the scope of the design activities to follow.

(2) Logical design
This is the process by which block diagrams (produced following the system design) are converted to logic diagrams, which are basically interconnected sets of logic gates. The building blocks of the logic diagrams (e.g., AND, OR and NOT gates) are not necessarily representative of the actual circuitry to be used in implementing the logic. (For example, programmable logic arrays, or PLA's, may be used to implement chunks of combinational logic.) Rather, these building blocks are primitives that are 'understood' by the simulation tools used to verify the functional correctness of the logic design.

(3) Physical design.
This is the process by which the logical design is partitioned and mapped into the physical packaging hierarchy. The design of a package, or module, at any level of the physical packaging hierarchy includes the following activities: (i) further partitioning of the logic 'chunk' being realized between the sub-modules (which are the modules at the next level of the hierarchy) contained within the

given module; (ii) placement of these sub-modules; and (iii) interconnection routing.

The design process is considered to be essentially complete following physical design. However, another important pre-manufacturing design step is prototype verification and checkout, wherein a full-scale prototype is fabricated as per the design rules, and thoroughly checked. The engineers may make some small changes and fixups to the design at this point ("engineering changes").

These design steps exist both in 'conventional' hardware design (i.e., using MSI/LSI parts) and in VLSI design. In conventional design, the design steps mentioned above occur more or less sequentially, whereas in some VLSI design methodologies, there is much overlap, with system, logical and physical design decisions occurring, in varying degrees, in parallel.

The design step that has proved to be the most amenable to automation is physical design. This step, which contains the most tedious and time-consuming detail, was also the one that received the most attention from researchers early on. Our discussion on DA problems will concentrate on the class of physical design problems, and to a lesser extent, on testing problems.

In discussing physical design automation problems, we shall further classify them into various sub-classes of problems. Although these sub-classes are intimately related (in that they are all really parts of a single problem), it is preferable to treat them separately because of the inherent computational complexity of the total problem. Actually, each of these problems represents a general class of problems whose precise definition is strongly influenced by factors such as the level (in the wiring hierarchy of IC chip, circuit card, backplane, etc.) of design, the particular technology being employed, the electrical constraints of the circuitry, and the tools available for attacking the problems. The specific problem, in turn influences the size of the problem, the selection of parameters for constraints and optimization, and the methodology for designing the solution techniques.

2.1 Some Classes of Design Automation Problems

2.1.1 Implementation Problems

For lack of a better term, we shall classify as "implementation problems" all those problems encountered in the process of mapping the logic design onto a set of physical modules. These implementation problems include the following types of problems.

(a) *Synthesis*

This problem deals with the translation of one logical representation of a digital system into another, with the constraint that the two representations be functionally equivalent.

This problem arises because the building blocks of the logic design are determined by the functional primitives understood by the logic simulation system, and not by the functionalities of the circuits most conveniently implemented in the given semiconductor technology. (So, though the choice of the underlying technology influences the logic designer, insofar as he takes advantage of its strengths and compensates for its weaknesses, the primitives in which his design eventually gets

expressed are not altered.) Consequently, there is a need to re-write the design in terms of the circuit families supported by the given technology.

Synthesis is a major bottleneck in designing computer hardware. Manual synthesis, apart from being slow, is quite error-prone. Designers often find themselves spending half their time correcting synthesis errors. Unfortunately, automated synthesis is far from viable at this point in time, and much work needs to be done in this area.

(b) *Partitioning*

The partitioning problem is encountered at various levels of the system packaging hierarchy. In very general terms, the problem may be described as follows. Given a description of the design to be implemented within a given physical package, the problem is to subdivide the logic among the sub-assemblies (i.e., packages at the next level of the hierarchy) contained within the given package, in a way that optimizes certain predetermined norms.

Quantities of interest in the partitioning process are:

(1) The number of partition elements (i.e., distinct sub-assemblies) [KODR69].

(2) The size of each partition element. This is an indication of the amount of space needed to physically implement the chunk of logic within that partition element.

(3) The number of external connections required by each sub-assembly.

(4) The total number of electrical connections between sub-assemblies [HABA68][LAWL62].

(5) The (estimated) system delay [LAWL69]. This points to the fact that proper partitioning is an extremely key element in optimizing the system performance. In fact, in many design methodologies, partitioning, at least at the early (and critical) stages of the design process, is still done manually by extremely skilled designers, in order to extract the maximum performance from the logic design.

(6) The reliability and testability of the resulting system implementation.

(c) *Construction of a Standard Library of Modules*

The library is a set of fully-designed modules that can be utilized in creating larger designs. The problem in creating libraries is deciding the functionalities of the various modules that are to be placed in the library. The process involves balancing the richness of functionality provided against the need to keep within reasonable bounds the number of distinct modules (which is related to the total cost of creating the library). Notz et al. [NOTZ67] propose measures which aid in the periodic update of a standard library of modules.

The problem of library construction is intimately related to the partitioning problem. The library should be constructed with a good idea of what the partitioning will be like in the various logic designs that use that library (though parts of a library may be based on earlier successful sub-designs). On the other side of the coin, partitioning is often done based on a good understanding of the library's contents.

In SSI terms, the library is the 7400 parts catalog. In LSI terms, the

parts in the library are far more complex functionally. Library construction is usually quite expensive. The logical and physical designs of each part in the library have to be totally optimized to extract the maximum performance while requiring the least space and power, given a specific semiconductor technology.

d) *Selection of Modules from a Standard Library*

Given a partition of a circuit along with a standard library of modules, the selection problem deals with finding a set of modules with either minimal total cost or minimal module count to implement the logic in the partition.

2.1.2 Placement Problems

In the most general terms the placement problem may be viewed as a combinatorial problem of optimally assigning interrelated entities to fixed locations (cells) contained in a given area. The precise definition of the interrelated entities and the location is strongly dependent on the particular level of the backplane being considered and the particular technology being employed. For instance, we can talk of the placement of logic circuits within a chip, of chips on a chip carrier, of chip carriers on a board, or of boards on a backplane. As stated earlier, the particular level influences the size of the problem, the choice of norms to be optimized, the constraints, and even the solution techniques to be considered.

The optimization criterion in placement problems is generally some norm defined on the interconnections and in practice a number of goals are to be satisfied. The main goal is to enhance the wireability of the resulting assembly, while ensuring that wiring rules are not violated. Some of the norms used are listed below:

(1) Minimizing the expected wiring congestions [CLAR69].

(2) Avoidance of wire crossovers [KODR62].

(3) Minimizing the total number of wire bends (in rectilinear technologies) [POME65].

(4) Elimination of inductive cross-talk by minimum shielding techniques.

(5) Elimination/suppression of signal echoes.

(6) Control of heat dissipation levels.

It can be seen that satisfying all the above-mentioned goals is a virtually impossible task. In most practical applications the norm minimized is the total weighted wire length.

In the context of VLSI, the placement problem is concerned almost exclusively with enhancing wireability. A difference is that the cell shapes and locations are not fixed, and the relative (or topological) placement of the cells is the important thing. Before absolute placement on silicon occurs, the cell shapes and the amount of space to be allowed for wiring have to be determined. This gives placement a different flavor from the MSI/LSI context. Furthermore, the term 'placement' does not have a standard usage in 'VLSI'. For instance, it has been used to describe the problem of determining the relative ordering of the terminals emanating from a cell.

2.1.3 Wiring Problems

These are also referred to as interconnection or routing problems, and they involve the process of formally defining the precise conductor paths necessary to properly interconnect the elements of the system. The constraints imposed on an acceptable solution generally involve one or more of the following:

(1) Number of layers (planes in which paths may exist).
(2) Number and location of via holes (feedthrough pins) or paths between layers.
(3) Number of crossovers.
(4) Noise levels.
(5) Amount of shielding required for cross-talk suppression.
(6) Signal reflection elimination.
(7) Line width (density).
(8) Path directions, e.g., horizontal and/or vertical only.
(9) Interconnection path length.
(10) Total area or volume for interconnection.

Due to the intimate relationship that exists between the placement and the wiring phases, it can be seen that many of the norms considered for optimization are common to both phases.

Approaches to wiring differ based on the nature of the wiring surface. There are two broad approaches: one-at-a-time wiring, and a two-stage coarse-fine approach.

One-at-a-time wiring is most appropriate in situations where the wiring surface is wide open, as is typically the case with PC cards and back-panels. In practice, one-at-a-time wiring is generally viewed as comprising the following subproblems:

(1) Wire list determination.
(2) Layering.
(3) Ordering.
(4) Wire layout.

Wire list determination involves making a list of the set of wires to be laid out. Given a set of points to be made electrically common, there are a number of alternative interconnecting wire sets possible.

Layering assigns the wires to different layers. The layering problem involves minimizing the number of layers, such that there exists an assignment of each wire to one of the layers that results in no wire intersections anywhere. The

ordering problem decides when each wire assigned to a layer is to be laid out. Since optimal wire layout appears to be a computationally intractable problem, all wire layout algorithms currently in use are heuristic in nature. Therefore, this sequence or ordering not only affects the total interconnection distance, but also may lead to the success or failure of the entire wiring process. Last but not least, the

wire layout problem, which seems to have attracted more interest than the others, deals with how each wire is to be routed, i.e., specifying the precise interconnection path between two elements.

A criticism of one-at-a-time wiring has been that, when a wire is laid out, it is done without any prescience. Thus, when a wire is routed, it might end up blocking a lot of wires that have yet to be considered for wire routing. To alleviate this problem, a two-stage approach, based on [HASH71], is often used.. Here, the wiring surface is usually divided into rectangular areas called channels. In the first stage, often referred to as *global routing*, an algorithm is used to determine the sequence of channels to be used in routing each connection. In the second stage, usually called *channel routing*, all connections within each channel are completed, with the various channels being considered in order.

This two-stage approach, which is generically referred to as the *channel routing approach*, is very appropriate for wire routing inside IC's, where the wiring surface is naturally divided into channels. There are, however, situations where the channel routing approach is not appropriate. In particular, it is not very appropriate for wide open wiring surfaces, where all channel definition becomes artificial. With wire terminals located all over the wiring surface, it becomes difficult to define non-trivial channels, while ensuring that all terminals are on the sides of (and not within) channels. Even when channel definition is possible, applying the channel routing approach to relatively uncluttered wiring surfaces results in many instances of the notorious channel intersection (or switchbox) problem. The effects of the coupling of constraints between channels at channel intersections are usually undesirable, and can destroy the effectiveness of the channel routing approach when the number of channel intersections is large in relation to the problem size.

For very large systems, where the number of interconnections may be in the tens of thousands, an interesting approach to the general wiring problem, given a wide open wiring surface, is the *single row routing approach*. It was initially developed by So [SO74] as a method to estimate very roughly the inherent routability of the given problem. However, as it produces very regular layouts (which facilitates automated fabrication), it has been adopted as being a viable approach to the wiring problem.

Single row routing consists of a systematic decomposition of the general multilayer wiring problem into a number of independent single layer, single row routing problems. There are four phases in this decomposition ([TING78]) :

(1) Via assignment
In this phase, vias are assigned to the different nets such that the interconnection becomes feasible. Note that after the via assignment is complete, wires to be routed are either horizontal or vertical. Design objectives in this phase include:
    (a) minimizing the number of vias used.
    (b) minimizing the number of columns of vias that result.

(2) Linear placement of via columns
In this phase, an optimal permutation of via columns is sought that minimizes the maximum number of horizontal tracks required on the board.

(3) Layering
The objective in this phase is to evenly distribute the edges on a number of layers. The edges are partitioned between the various layers in such a way that all edges on a layer are either horizontal or vertical.

(4) Single Row Routing
In this final phase, one is presented with a number of single row connection patterns on each layer. The problem here is to find a physical layout for these patterns, subject to the usual wiring constraints.

If one of the objectives during the via assignment phase is minimizing the projected wiring density, the single row wiring approach in fact becomes an effective application of a two-stage channel routing-like approach to situations in which the wiring surface is wide open.

The apparent complexity of the general wiring problem has sparked investigations into topologically restricted classes of wiring problems. One such class of problems involves the wiring of connections between the terminals of a single rectangular component, with wiring allowed only outside the periphery of the component. A norm to be minimized is the area required for wiring.

Another restricted wiring problem is *river routing*. The basic problem is as follows. Two ordered sets of terminals (a_1, a_2, .. , a_n) and (b_1, b_2, .. , b_n) are to be connected by wires across a rectangular channel, with wire i connecting terminal a_i to terminal b_i, $1 \le i \le n$. The objective is to make the connections without any wires crossing, while attempting to minimize the separation between the two sets of terminals (i.e., the channel width).

River routing has found applications in many VLSI design methodologies. When a top-down design style is followed, it is possible to ensure that by and large, the terminals are so ordered on the perimeter of each block that, in the channel between any adjacent pair of blocks, the terminals to be connected are in the correct relative order on the opposite sides of the channel (i.e., a river routing situation exists).

The requirement concerning the proper ordering of the terminals of each block is admittedly quite difficult to always meet. However, it is a less severe requirement to meet than the one imposed in design systems like Bristle Blocks [JOHA79]. In the latter system, wiring is conspicuously avoided by forcing the designer to design modules in a plug-together fashion; the blocks must all fit together snugly, and all desired connections between blocks are made to occur by actually having the associated terminals touch each other. In such a design environment, all channel widths are zero, and thus there can be no wiring.

2.1.4 Testing Problems

Over the years, fault diagnosis has grown to be one of the more active, albeit less mature, areas of design automation development. Fault diagnosis comprises:

1) Fault detection, and
2) Fault location and identification.

The unit to be diagnosed can range from an individual IC chip, to a board-level assembly comprising several chip carriers, to an entire system containing many boards. For proper diagnosis, the unit's behavior

and hardware organization must be thoroughly understood. Also essential is a detailed analysis of the faults for which the unit is being diagnosed. This in turn involves concepts like fault modeling, fault equivalence, fault collapsing, fault propagation, coverage analysis and fault enumeration.

Fault diagnosis is normally effected by the process of *testing*. That is, the unit's behavior is monitored in the presence of certain predetermined stimuli known as tests. Testing is a general term, and its goal is to discover the presence of different types of faults. However, over the years, it has come to mean the testing for physical faults, particularly those introduced during the manufacturing phase. Testing for non-physical faults, e.g., design faults, has come to be known as *design verification*, and it is just emerging as an area of active interest. In keeping with the industry trend and to avoid confusion, we shall use the terms 'testing' and 'design verification' in the sense described above.

2.1.4.1 Testing

The central problem in testing is *test generation*. Majority of the effort in testing has been directed towards designing automatic test generation methods. Test generation is often followed by test verification, which deals with evaluating the effectiveness of a test set in diagnosing faults. Both test generation and test verification are extremely complex and time consuming tasks. As a result, their development has been rather slow. Most of the techniques developed are of a heuristic nature. The development process itself has consistently lagged behind the rapidly changing IC technologies. Hence, at any given time, the testing methods have always been inadequate for handling the designs that use the technologies existing in the same time frame. Most of the automatic test generation methods existing today were developed in the 1970's. They mainly addressed fault detection, and were based on one of the following: path sensitizing, D-algorithm [ROTH66], or Boolean difference [YAN71]. They basically handled logic networks implemented with SSI/MSI level gates. Also, most of the techniques considered only combinational networks, and almost all of them assumed the simplified single stuck-at fault model. As regards test verification, to date, formal proof has been almost impossible in practice. Most of the verification is done by fault simulation and fault injection.

The advent of LSI and VLSI, while improving cost and performance, has further complicated the testing problem. The different architectures and processing complexities of the new building blocks (e.g., the microprocessor) have rendered most of the existing test methods quite incapable. As a result, it has become necessary to re-investigate some of the aspects of the testing problem. Take for instance the single stuck-at fault model. For years, the industry has clung to this assumption. While being adequate for prior technologies, it does not adequately cover other fault mechanisms, like bridging shorts or pattern sensitivities. Furthermore, for testing microprocessors, PLA's, RAM's, ROM's and complex gate arrays, testing at a level higher than the gate level appears to make more sense. This involves testing at the functional and algorithmic or behavioral levels. Also, more work needs to be done in the area of fault location and identification. The method presented in [ABRA80] attempts to achieve both fault detection and location without requiring explicit fault enumeration.

Finally, there is the prudent approach of designing for testability in

order to simplify the testing problem. Design for testability first attracted attention with the coming of LSI. Today, with VLSI, its need has become all the more critical. One of the main problems in this area is deriving a quantitative measure of testability. One way is to analyze a unit for its controllability and observability [GOLD79], which quantities represent the difficulty of controlling and observing the logical values of internal nodes from the inputs and outputs respectively. Most existing testability measures , however, have been found to be either crude or difficult to determine. The next problem is deriving techniques for testability design. A comprehensive survey of these techniques is given in [GRAS80] and [WILL82]. Most of them are of an ad hoc nature, presented either as general guidelines, or hard and fast rules. A summary of these techniques appears in Figure 2.1.

Figure 2.1

2.1.4.2 Design Verification

The central issue here is proving the correctness of a design. Does a design do what it is supposed to do? In other words, we are dealing with the testing for design faults. The purpose of design verification is quite clear. Design faults must be eliminated, as far as possible, before the hardware is constructed, and before prototype tests begin. The increasing cost of implementing engineering changes given LSI/VLSI hardware has enhanced the need for design verification.

Compared to physical faults, design faults are more subtle and serious and can be extremely difficult to handle. Hence, the techniques developed for physical faults cannot be effectively extended to design faults. To date, very little effort has been devoted to formalizing design verification. Designs are still mostly checked by ad hoc means, like fault simulation and prototype checkout.

Like other disciplines, design verification too has not been spared the impact of LSI/VLSI. Some of these influences are listed below.

Ratification or matching the design specification was accomplished in the pre-LSI era by gate-level simulation. This may no longer be sufficient. The simulations need to be more detailed, and they need to be done at higher levels, like the functional and behavioral levels. Also, techniques are required for determining the following:

(1) Stopping rules for simulation.
(2) The extent of design faults removed.
(3) A quantitative measure for the correctness or quality of the final design.

Validation. In the pre-LSI days, this was restricted to the testing of hardware on the test floor. Moreover, the testing process was not formalized or systematic, and hence lacked thoroughness and rigor. Today, validation mainly involves testing the equivalence of two design descriptions. The descriptions may be at different levels. Thus, before being compared, they need to be translated to a common level. For example, one can construct symbolic execution tree models of the design descriptions to be compared.

Timing Analysis. In the past, it was sufficient to analyze only single "critical" paths. The technology rules of LSI/VLSI are so complex that the identification of these critical paths has become extremely difficult. Statistical timing analysis methods need to be investigated, in order to cope with the tremendous densities and wide range of tolerances imposed by LSI/VLSI.

Finally, research has also started in developing design techniques to alleviate the need for, and/or facilitate, the design verification process.

3. COMPLEXITY AND NONDETERMINISM

3.1 Complexity

By the *complexity of an algorithm*, we mean the amount of computer time and memory space needed to run this algorithm. These two quantities will be referred to respectively as the time and space complexities of the algorithm. To illustrate, consider procedure MADD (Algorithm 3.1): it is an algorithm to add two mxn matrices together.

```
line procedure MADD(A,B,C,m,n)
    {compute C=A+B}
 1   declare A(m,n),B(m,n),C(m,n)
 2   for i ← 1 to m do
 3     for j ← 1 to n do
 4       C(i,j) ← A(i,j)+B(i,j)
 5     endfor
 6   endfor
 7  end MADD
```

Algorithm 3.1 Matrix addition.

The time needed to run this algorithm on a computer comprises two components: the time to compile the algorithm and the time to execute it. The first of these two components depends on the compiler and the computer being used. This time is, however, independent of the actual values of n and m. The execution time, in addition to being dependent on the compiler and the computer used, depends on the values of m and n. It takes more time to add larger matrices.

Since the actual time requirements of an algorithm are very machine-dependent, the theoretical analysis of the time complexity of an algorithm is restricted to determining the number of steps needed by the algorithm. This step count is obtained as a function of certain parameters that characterize the input and output. Some examples of often-used parameters are: number of inputs; number of outputs; magnitude of inputs and outputs; etc.

In the case of our matrix addition example, the number of rows m and the number of columns n are reasonable parameters to use. If instruction 4 of procedure MADD is assigned a step count of 1 per execution, then its total contribution to the step count of the algorithm is mn, as this instruction is executed mn times. [SAHN80, Chapter 6] discusses step count analysis in greater detail. Since the notion of a step is somewhat inexact, one often does not strive to obtain an exact step count for an algorithm. Rather, asymptotic bounds on the step count are obtained. Asymptotic analysis uses the notation O, Ω, Θ and o. These are defined below.

Definition: [Asymptotic Notation] f(n) = O(g(n)) (read as "f of n is big oh of g of n") iff there exist positive constants c and n_0 such that f(n)≤cg(n) for all n, n≥n_0. f(n) = Ω(g(n)) (read as "f of n is omega of g of n") iff there exist positive constants c and n_0 such that f(n)≥cg(n) for all n, n≥n_0. f(n) is Θ(g(n)) (read as "f of n is theta of g of n") iff there exist positive constants c_1, c_2, and n_0 such that c_1g(n)≤f(n)≤c_2g(n) for all n, n≥n_0. f(n) = o(g(n)) (read as "f of n is little o of g of n") iff $\lim_{n\to\infty} f(n)/g(n) = 1$. □

The definitions of O, Ω, Θ, and o are easily extended to include functions of more than one variable. For example, f(n,m) = O(g(n,m)) iff there exist positive constants c, n_0, and m_0 such that f(n,m)≤cg(n,m) for all n≥n_0 and all m≥m_0.

Example 3.1: 3n+2=O(n) as 3n+2≤4n for all n, n≥2. 3n+2=Ω(n) and 3n+2=Θ(n). $6*2^n+n^2=O(2^n)$. 3n=O(n^2). 3n=o(3n) and 3n=O(n^3). □

As illustrated by the previous example, the statement f(n)=O(g(n)) only states that g(n) is an upper bound on the value of f(n) for all n, n≥n_0.

It doesn't say anything about how good this bound is. Notice that if n=O(n), n=O(n^2), n=O($n^{2.5}$), etc. In order to be informative, g(n) should be as small a function of n as one can come up with such that f(n)=O(g(n)). So, while we shall often say that 3n+3=O(n), we shall almost never say that 3n+3=O(n^2).

As in the case of the "big oh" notation, there are several functions g(n) for which f(n)=Ω(g(n)). g(n) is only a lower bound on f(n). The theta notation is more precise than both the "big oh" and omega notations. The following theorem obtains a very useful result about the order of f(n) when f(n) is a polynomial in n.

Theorem 3.1: Let f(n)=$a_m n^m + a_{m-1} n^{m-1} + \cdots + a_0$, $a_m \neq 0$.
(a) f(n) = O($n^{m)}$
(b) f(n) = Ω(n^m)
(c) f(n) = Θ(n^m)
(d) f(n) = o($a_m n^m$) □

Asymptotic analysis may also be used for space complexity.

While asymptotic analysis does not tell us how many seconds an algorithm will run for or how many words of memory it will require, it does characterize the growth rate of the complexity (see Figure 3.1). So, if procedure MADD takes 2 milliseconds (ms) on a problem with m=100 and n=20, then we expect it to take about 16ms when mn=16000 (the complexity of MADD is Θ(mn)). For sufficiently large values of n, a Θ(n^2) algorithm will be faster than a Θ(n^3) algorithm.

We have seen that the time complexity of an algorithm is generally some function of the instance characteristics. This function is very useful in determining how the time requirements vary as the instance characteristics change. The complexity function may also be used to compare two algorithms A and B that perform the same task. Assume that algorithm A has complexity Θ(n) and algorithm B is of complexity Θ(n^2). We can assert that algorithm A is faster than algorithm B for "sufficiently large" n. To see the validity of this assertion, observe that the actual computing time of A is bounded from above by n for some constant c and for all n, n≥n_1 while that of B is bounded from below by dn^2 for some constant d and all n, n≥n_2. Since cn≤dn^2 for n≥c/d, algorithm A is faster than algorithm B whenever n≥max{n_1, n_2, c/d}. One should always be cautiously aware of the presence of the phrase "sufficiently large" in the assertion of the preceding discussion. When deciding which of the two algorithms to use, we must know whether the n we are dealing with is in fact "sufficiently large". If algorithm A actually runs in 10^6n milliseconds while algorithm B runs in n^2 milliseconds and if we always have n≤10^6, then algorithm B is the one to use. To get a feel for how the various functions grow with n, you are advised to study Figure 3.1 very closely. As is evident from the figure, the function 2^n grows very rapidly with n. In fact, if an algorithm needs 2^n steps for execution, then when n=40 the number of steps needed is approximately 1.1*10^{12}. On a computer performing one billion steps per second, this would require about 18.3 minutes. If n=50, the same algorithm would run for about 13 days on this computer. When n=60, about 36.56 years will be required to execute the algorithm and when n=100, about 4*10^{13} years will be needed. So, we may conclude that the utility of algorithms with exponential complexity is limited to small n (typically n≤40).

Figure 3.1

Algorithms that have a complexity that is a polynomial of high degree are also of limited utility. For example, if an algorithm needs n^{10} steps, then using our one billion step per second computer we will need 10 seconds when n=10; 3,171 years when n=100; and $3.17*10^{13}$ years when n=1000. If the algorithms complexity had been n^3 steps instead, then we would need one second when n=1000, 16.67 minutes when n=10,000; and 11.57 days when n=100,000.

Table 3.1 gives the time needed by a one billion instruction per

second computer to execute a program of complexity f(n) instructions. One should note that currently only the fastest computers can execute about one billion instructions per second. From a practical standpoint, it is evident that for reasonably large n (say n>100) only algorithms of small complexity (such as n, nlogn, n^2, n^3, etc.) are feasible. Further, this is the case even if one could build a computer capable of executing 10^{12} instructions per second. In this case, the computing times of Table 3.1 would decrease by a factor of 1000. Now, when n = 100 it would take 3.17 years to execute n^{10} instructions, and $4*10^{10}$ years to execute 2^n instructions.

Table 3.1

Another point to note is that the complexity of an algorithm cannot always be characterized by the size or number of inputs and outputs. The time taken is often very data dependent. As an example, consider Algorithm 3.2. This is a very primitive backtracking algorithm to determine if there is a subset of {W(1),W(2),...W(n)} with sum equal to M. This problem is called the *sum of subsets* problem. Procedure SS is initially invoked by the statement:

$$X \leftarrow SS(1,M)$$

```
procedure SS(i,P)
  {determine if W(i:n) has a}
  {subset that sums to P}
  global W(1:n),n
  if i>n then return(false)
  case
    :W(i)=P: return(true)
    :W(i)<P: if SS(i+1,P-W(i))
             then return(true)
             else return(SS(i+1,P))
            endif
```

```
   :else: return (SS(i+1,P))
  endcase
end SS
```

Algorithm 3.2

Procedure SS returns the value *true* iff a subset of W(1:n) sums to M. Observe that if $\sum_{i=1}^{n} W(i)$=M then SS runs in Θ(n) time. If no subset of W(1:n) sums to M, then SS takes $\Theta(2^n)$ time to terminate. For other cases, the time needed is anywhere between Θ(n) and $\Theta(2^n)$. So, the complexity of SS is Ω(n) and $O(2^n)$. In cases like SS where the complexity is quite data dependent, one talks of the best case, worst case and average (or expected) complexity. A precise definition of these terms can be found in [SAHN81]. Here, we shall rely on our intuitive understanding of these terms.

Most of the complexity results obtained to date have been concerned with the worst case complexity of algorithms. The value of such analyses may be debated. Procedure SS has a worst case time complexity of $\Theta(2^n)$. This just tells us that there exist inputs on which this much time will be spent. However, it might well be that for the inputs of "interest", the algorithm is far more efficient, perhaps even of complexity $O(n^2)$.

As another example, consider the much publicized Khachian algorithm for the linear programming problem. This algorithm has a worst case complexity that is a polynomial function of the number of variables, equations, and size (i.e., number of bits) of the coefficients. The Simplex method is known to have a worst case complexity that is exponential in the number of equations. So, as far as the worst case complexity is concerned, Khachian's algorithm is superior to the Simplex method. Despite this, Khachian's algorithm is impractical while the Simplex method has been successfully used for years to solve reasonably large instances of the linear programming problem. The Simplex method works well on those instances that are of "interest" to people.

It has recently been shown [DANT80] that under reasonable assumptions, the expected (or average) complexity of the Simplex method is in fact $O(m^3)$ where m is the number of equations (the expected number of pivot steps in 3.5m, and each such step takes $O(m^2)$ time).

Thus, the notion of average complexity seems to better capture the complexity one might observe when actually using the algorithm. Average complexity analysis is however far more difficult than worst case analysis and has been carried out successfully for only a limited number of algorithms. As a result of this, this paper will be concerned mainly with the worst case complexity of design automation problems. One should keep in mind that while many of the design automation problems will shown to be "probably" intractable in terms of worst case complexity, these results do not rule out the possibility of very efficient expected behavior algorithms.

3.2 Nondeterminism

The commonly used notion of an algorithm has the property that the result of every step is uniquely defined. Algorithms with this property are called deterministic algorithms. From a theoretical framework, we can remove this restriction on the outcome of every operation. We can allow algorithms to contain operations whose outcome is not uniquely defined but is limited to a specific set of possibilities. The machine executing such operations is allowed to choose any one of these outcomes. This leads to the concept of a *nondeterministic algorithm*. To specify such algorithms we introduce three new functions:

(a) **choice**(S) ... arbitrarily choose one of the elements of set S;
(b) **failure** ... signals an unsuccessful completion;
(c) **success** ... signals a successful completion.

Thus the assignment statement X ← **choice**(1:n) could result in X being assigned any one of the integers in the range [1,n]. There is no rule specifying how this choice is to be made. The **failure** and **success** signals are used to define a computation of the algorithm. One way to view this computation is to say that whenever there is a set of choices that leads to a successful computation, then one such set of choices is made and the algorithm terminates successfully. A nondeterministic algorithm terminates unsuccessfully iff there exists no set of choices leading to a success signal. A machine capable of executing a nondeterministic algorithm in this way is called a *nondeterministic machine*.

Example 3.2: Consider the problem of searching for an element x in a given set of elements A(1) to A(n), n≥1. We are required to determine an index j such that A(j) = x or j = 0 if x is not in A. A nondeterministic algorithm for this is:

```
j ← choice(1:n)
if A(j) = x then print (j); success endif
print ("0"); failure.
```

From the way a nondeterministic computation is defined, it follows that the number "0" can be output iff there is no j such that A(j) = x. The computing times for **choice**, **success**, and **failure** are taken to be O(1). Thus the above algorithm is of nondeterministic complexity O(1). Note that since A is not ordered, every deterministic search algorithm is of complexity at least O(n). □

Since many choice sequences lead to a successful termination of a nondeterministic algorithm, the output of such an algorithm working on a given data set may not be uniquely defined. To overcome this difficulty, one normally considers only decision problems, i.e., problems with answer 0 or 1 (or true or false). A successful termination always yields the output 1 while unsuccessful terminations always yield the output 0.

In measuring the complexity of a nondeterministic algorithm, the cost assignable to the **choice**(S) function is O(log k) where k is the size of S. So, strictly speaking, the complexity of the search algorithm of Example 3.2 is O(log n). The time required by a nondeterministic algorithm performing on any given input depends upon whether or not there exists a sequence of choices that leads to a successful completion. If such a sequence exists, then the time required is the minimum number of steps leading to such a completion. If no choice sequence leads to a successful completion, then the algorithm takes O(1) time to make a failure

termination.

Nondeterminism appears to be a powerful tool. Algorithm 3.3 is a nondeterministic algorithm for the sum of subsets problem. Its complexity is O(n). The best deterministic algorithm for this problem has complexity $O(2^{n/2})$ (see HORO74).

```
procedure NSS(W,n,M)
 declare X(1:n),W(1:n),n,M
 for i ← 1 to n do
   X(i) ← choice({0,1})
 end
 if Σ_{i=1}^{n} W(i)X(i) = M then success
                 else failure
 endif
end NSS
```

Algorithm 3.3 Nondeterministic sum of subsets algorithm.

3.3 NP-hard and NP-complete Problems

The *size* of a problem instance is the number of digits needed to represent that instance. An instance of the sum of subsets problem is given by (W(1),W(2),...,W(n),M). If each of these numbers is nonnegative and integer, then the instance size is $\lceil \sum_{i=1}^{n} \log_2 W(i) \rceil + \lceil \log_2 M \rceil$ if binary digits are used. An algorithm is of *polynomial* time complexity iff its computing time is O(p(m)) for every input of size m and some fixed polynomial p().

Let P be the set of all decision problems that can be solved in deterministic polynomial time. Let NP be the set of decision problems solvable in polynomial time by nondeterministic algorithms. Clearly, P NP. It is not known whether P = NP or P ≠ NP. The P = NP problem is important because it is related to the complexity of many interesting problems (including certain design automation problems). There exist many problems that cannot be solved in polynomial time unless P = NP. Since, intuitively, one expects that P ≠ NP, these problems are in "all probability" not solvable in polynomial time. The first problem that was shown to be related to the P = NP problem, in this way, was the problem of determining whether or not a propositional formula is satisfiable. This problem is referred to as the *Satisfiability problem*.

Theorem 3.2: Satisfiability is in P iff P = NP.

Proof: See [HORO78] or [GARE79]. □

Let A and B be two problems. Problem A is *polynomially reducible* to problem B (abbreviated A reduces to B, and written as A α B) iff the existence of a deterministic polynomial time algorithm for B implies the existence of a deterministic polynomial time algorithm for A. Thus, if A α B and B is polynomially solvable, then so also is A. A problem A is *NP-hard* iff Satisfiability α A. An NP-hard problem A is *NP-complete* iff A ε NP.

Observe that the relation α is transitive (i.e., if A α B and B α C

then A α C). Consequently, if A α B and Satisfiability α A then B is NP-hard. So, to show that any problem B is NP-hard, we need merely show that A α B where A is any known NP-hard problem. Some of the known NP-hard problems are:

NP1: Euclidean Steiner Tree [GARE77]
Input: A set X = $\{(x_i,y_i)|1\leq i\leq n\}$ of points.
Output: A finite set Y = $\{(a_i,b_i)|1\leq i\leq m\}$ of points such that the minimum spanning tree for XY is of minimum total length over all choices for Y. The distance between two points (t,u) and (v,w) is $[(t-v)^2+(u-w)^2]$.

NP2: Manhattan Steiner Tree [GARE77]
Input: Same as in NP1.
Output: Same as in NP1, except that the distance between two points is taken to be |t-v|+|u-w|.

NP3: Euclidean Traveling Salesman [GARE76b]
Input: Same as in NP1.
Output: A minimum length tour going through each point in X. The Euclidan distance measure is used.

NP4: Euclidean Path Traveling Salesman [PAPA77]
(also called Euclidean Hamiltonian Path)
Input: Same is in NP1.
Output: A minimum length path that visits all points in X exactly once. The Euclidean distance measure is used.

NP5: Manhattan Traveling Salesman [GARE76b]
Input: Same as in NP3.
Output: Same as in NP3, except that the Manhattan distance measure is used.

NP6: Manhattan Path Traveling Salesman [PAPA77]
(also called Manhattan Hamiltonian Path)
Input: Same as in NP1.
Output: Same as in NP4, except that the Manhattan distance measure is used.

NP7: Chromatic Number I [EHRL76]
Input: A graph G which is the intersection graph for straight line segments in the plane.
Output: The minimum number of colors needed to color G.

NP8: Chromatic Number II [EHRL76]
Input: Same as in NP5.
Output: 'Yes' if G is 3-colorable and 'No' otherwise.

NP9: Partition [KARP72]
Input: A multiset A = $\{a_i|1\leq i\leq n\}$ of natural numbers.
Output: "Yes" if there is a subset B {1,2,...,n} such that $\sum_{i\,fP(moB} a_i = \sum_{i\,fP(nmB} a_i$. *"No" otherwise.*

NP10: 3-Partition [GARE75]

Input: A multiset A = {a_i|1≤i≤3m} of natural numbers, and a bound B, such that

(i) $\sum_{a_i \in A} a_i = mB$

(ii) B/4 < a_i < B/2 for 1≤i≤3m
Output: "Yes" if A can be partitioned into m disjoint sets $A_1, A_2, \cdots A_m$ such that, for 1≤i≤m, $\sum_{a_j \in A_i} a_j = B$*; "No" otherwise.*

NP11: Knapsack(maximization) [KARP72]
Input: Multisets P = {p_i|1≤i≤n} and W = {w_i|1≤i≤n} of natural numbers and another natural number M.
Output: x_i ε {0,1} such that $\sum_i p_i x_i$ *is maximized and* $\sum_i w_i x_i \le M$.

NP12: Knapsack(minimization)
Input: Same as in NP11, except replace set P by K = {k_i|1≤i≤n}.
Output: x_i ε {0,1} such that $\sum_i k_i x_i$ *is minimized and* $\sum_i w_i x_i \ge M$.

NP13: Integer Knapsack [LUEK75]
Input: Multiset W = {w_i|1≤i≤n} of nonnegative integers and two additional nonnegative integers M and K.
Output: "Yes" if there exist nonnegative integers x_i, 1≤i≤n such that $\sum w_i x_i \le M$ *and* $\sum w_i x_i \ge K$. *"No" otherwise.*

NP14: Quadratic Assignment Problem [SAHN76]
Input: $c_{i,j}$ *, 1≤i≤n, 1≤j≤n.*
$d_{k,q}$ *, 1≤k≤m, 1≤q≤m.*
Output: $x_{i,k} \in \{0,1\}$ *, 1≤i≤n, 1≤k≤m, such that*

(a) $\sum_{k=1}^{m} x_{i,k} \le 1$ *, 1≤i≤n*

(b) $\sum_{i=1}^{n} x_{i,k} = 1$ *, 1≤k≤m*

and $\sum_{\substack{i,j=1 \\ i \ne j}}^{n} \{ \sum_{\substack{k,q=1 \\ k \ne q}}^{m} c_{i,j} d_{k,q} x_{i,k} x_{j,q} \}$ *is minimized.*

A listing of over 200 known NP-hard problems can be found in [GARE79]. The importance of showing that a problem A is NP-hard lies in the P = NP problem. Since we don't expect that P = NP, we don't expect NP-hard problems to be solvable by algorithms with a worst case complexity that is polynomial in the size of the problem instance. From Figure 3.1 and Table 3.1, it is apparent that if a problem cannot be solved in polynomial time, then it is intractable, for all practical purposes. If A is NP-complete and if it does turn out that P = NP, then A will be polynomially solvable.

4. COMPLEXITY OF DESIGN AUTOMATION PROBLEMS

In Section 4.1 we illustrate how one goes about showing that a problem is NP-hard or NP-complete. We consider three examples from the design automation area. Over thirty design automation problems are described in Section 4.2. With each problem, a discussion of its complexity is included.

4.1 Showing Problems NP-hard and NP-complete

4.1.1 Circuit Realization

In this problem, we are given a set of r modules. Associated with module i is a cost c_i, $1 \le i \le r$. Module i contains m_{ij} gates of type j, $1 \le j \le n$. We are required to realize a circuit C with gate requirements $(b_1,b_2,...,b_n)$, i.e., circuit C consists of b_j gates of type j. $(x_1,...,x_r)$ realizes circuit C iff

$$\sum_{i=1}^{r} m_{ij}x_i \ge b_j, \quad 1 \le j \le n$$

and x_i is a natural number, $1 \le i \le r$.

The cost of the realization $(x_1,...,x_r)$ is $\sum_{i=1}^{r} c_i x_i$. We are interested in obtaining a minimum cost realization of C.

Theorem 4.1: The circuit realization problem is NP-hard.

Proof: From Section 3.2, we see that it is sufficient to show that Q α circuit realization, where Q is any known NP-hard problem. We shall use Q = NP13 = integer knapsack (Section 3.2).

Let $(w_1,w_2,...,w_p)$, M, and K be any instance of the integer knapsack problem. Construct the following circuit realization instance:

n = 1 ; b_1 = K ; r = p ; $m_{i,1} = w_i$, $1 \le i \le p$; $c_i = w_i$, $1 \le i \le p$

Clearly, the least cost realization of the above circuit instance has a cost at most M iff the corresponding integer knapsack instance has answer "yes". So if the circuit realization problem is polynomially solvable, then so also is NP13. But NP13 is NP-hard. So, circuit realization is also NP-hard. □

In order to show that an NP-hard problem Q is NP-complete, we need to show that it is in NP. Only decision problems (i.e., problems for which the output is "yes" or "no") can be NP-complete. So, the circuit realization problem cannot be NP-complete. However, we may formulate a decision version of the circuit realization problem : Is there a realization with cost no more than S? The proof provided in Theorem 4.1 is valid for this version of the problem too. Also, there is a nondeterministic polynomial time algorithm for this decision problem (Algorithm 4.1). So, the decision version of the circuit realization problem is NP-complete.

```
procedure CKT(b,S,m,r,n,c)
  {Bs there a circuit realization with cost ≤ S?}
 declare r, n, c(r), m(r,n), b(n), S, x(r)
 q ← max_j {b(j)}
 for i ← 1 to r do {obtain x_i s nondeterministically}
   x(i) ← choice(0:q)
 endfor
 for i←1 to n do {check feasibility}
   if Σ_{j=1}^{r} m(j,i)∗x(i) < b(i) then failure endif
 endfor
 if Σ_{i=1}^{r} c(i)x(i) > S then failure
                    else success
end CKT
```

Algorithm 4.1

4.1.2 Euclidean Layering Problem

A wire to be laid out may be defined by the two end points (x,y) and (u,v) of the wire. (x,y) and (u,v) are the coordinates of the two end points. In a Euclidean layout, the wire runs along a straight line from (x,y) to (u,v). Figure 4.1(a) shows some wires laid out in a Euclidean manner. Let W = { $[(u_i,v_i),(x_i,y_i)]$ | 1≤i≤n } be a set of n wires. In the Euclidean layering problem, we are required to partition W into a minimum number of disjoint sets $W_1, W_2, ..., W_k$ such that no two wires in any set W_i cross. Figure 4.1(b) gives a partitioning of the wires of Figure 4.1(a) that satisfies this requirement. The wires in W_1 and W_2 can now be routed in separate layers.

Figure 4.1

Theorem 4.2: The Euclidean layering problem is NP-hard.

Proof: We shall show that the known NP-hard problem NP7 (Chromatic Number I) reduces to the Euclidean layering problem. Let G=(V,E) be any intersection graph for straight line segments in the plane. Let W be the corresponding set of straight line segments. Note that |W|=|V| as V has one vertex for each line segment in W. Also, (i,j) is an edge of G iff the line segments corresponding to vertices i and j intersect in Euclidean space. From any partitioning W_1,W_2,... of W such that no two line segments of any partition intersect, we can obtain a coloring of G. Vertex i is assigned the color j iff the line segment corresponding to vertex i is in the partition W_j. No adjacent vertices in G will be assigned the same color as the line segments corresponding to adjacent vertices intersect and so must be in different partitions. Furthermore, if G can be colored with k colors, then W can be partitioned into W_1,...,W_k.

Hence, G can be colored with k colors iff W can be partitioned into k disjoint sets, no set containing two intersecting segments. So, if we could solve the Euclidean layering problem in polynomial time, then we could solve the chromatic number problem NP7 in polynomial time by first obtaining W as above and then using the polynomial time algorithm to minimally partition W. From the partition, a coloring of G can be obtained. Since NP7 is NP-hard, it follows that the Euclidean layering problem is also NP-hard. □

The above equivalence between NP7 and the Euclidean layering problem was pointed out by Akers [BREU72].

A decision version of the Euclidean layering problem would take the form: Can W be partitioned into ≤ k partitions such that no partition contains two wires that intersect? The proof of Theorem 4.2 shows that this decision problem is NP-hard. Procedure ELP (Algorithm 4.2) is a nondeterministic polynomial time algorithm for this problem. Hence, the decision version of the Euclidean layering problem is NP-complete.

```
procedure ELP(W,n,k)
  {n = |W|}
  wire set W, integer n,k;
  L(i) ← 0, 1≤i≤k
  for i←1 to n do {assign wires to layers}
    j ← choice(1:k)
    L(j) ← L(j) ∪ {[(u_i,v_i),(x_i,y_i)]}
  endfor
  for i←1 to k do {check for intersections}
    if two wires in L(i) intersect then failure
    endif
  endfor
  success
end ELP
```

Algorithm 4.2

4.1.3 Rectilinear Layering Problem

This problem is similar to the Euclidean layering problem of Section 4.1.2. Once again, we are given a set W = {[(u_i,v_i),(x_i,y_i)]1≤i≤n} of wire end points. In addition, we are given a pxq grid with horizontal and vertical lines at unit intervals. We may assume that each wire end point is a grid point. Each pair of wire end points is to be joined by a wire that is routed along grid lines alone. No two wires are permitted to cross or share the same grid line segment. We wish to find a partition $W_1,W_2,...,W_k$ of the wires such that k is minimum and the end point pairs in each partition can be wired as described above. The end point pairs in each partition can be connected in a rectilinear manner in a single layer. The complete wiring will use k layers.

Theorem 4.3: The rectilinear layering problem is NP-hard.

Proof: We shall show that if the rectilinear layering problem can be solved in polynomial time, then the known NP-hard problem NP10 (3-Partition) can also be solved in polynomial time. Hence, the rectilinear layering problem is NP-hard.

Let A = $\{a_1,a_2,...,a_{3m}\}$; B; $\sum a_i$ = mB; B/4 < a_i < B/2 be any instance of the 3-Partition problem. For each a_i, we construct a size subassembly and enforcer subassembly ensemble as shown in Figure 4.2(a). Figure 4.2(b) shows how the ensembles for the n a_is are put together to obtain the complete wiring problem. The grid has dimensions (B+1)x(i_1+i_2+1) where

$$i_1 = \sum a_i + m = mB + m$$

and

$$i_2 = \sum[a_i + 2(m-1)] + m = mB + 6m^2 - 6m + m$$

Note that all wire end points are along the bottom edge of the grid.

Figure 4.2

The valve assembly shown in Figure 4.2(b) is similar to the enforcer subassembly except that it contains m wires instead of m-1. As is evident, no two wires of the valve assembly can be routed in the same layer. Hence, at least m layers are needed to wire the valve.

An examination of the ensemble for each a_i reveals that:

(i) no 2 wires in the enforcer subassembly can be routed on the same layer, obviously.
(ii) a wire from the size subassembly cannot be routed on the same layer with a wire from the enforcer subassembly.
(iii) all wires in a size subassembly can be routed on the same layer.

Therefore, at least m layers are required to route each ensemble. Hence, the rectilinear layering problem defined by Figure 4.2(b) needs at least m layers.

If the 3-Partition instance has a 3-Partition $A_1, A_2,...,A_m$, then only m layers are needed by Figure 4.2(b). In layer i, we wire the size subassemblies for the three a_js in A_i as well as one wire of the valve and one wire from each of the 3m-3 enforcer subassemblies corresponding to the 3m-3 a_js not in A_i.

On the other hand, if Figure 4.2(b) can be wired in m layers, then there is a 3-Partition of the a_is. Since no layer may contain more than 1 wire from the valve, each layer contains exactly B wires from the size ensembles. If a wire from the size ensemble for a_i is in layer j, then all a_i wires from this ensemble must be in this layer. To see this, observe that the remaining m-1 layers must each contain exactly 1 wire from a_i's enforcer subassembly and so can contain no wires from the size subassembly. Hence, each layer must contain exactly three size ensembles. The 3-Partition is therefore A_i={ j |the size subassembly for j is in layer i}.

So, Figure 4.2(b) can be wired in m layers iff the 3-Partition instance has answer "yes". Hence, the rectilinear layering problem is NP-hard. □

As in the case of the problems considered in Sections 4.1.1 and 4.1.2, we may define a decision version of the rectilinear layering problem and show that this version is NP-complete.

4.2 Mathematical Formulation and Complexity of Design Automation Problems

4.2.1 Implementation Problems

IP1: Function Realization
Input: A Boolean function B and a set of component types $C_1,C_2,...,C_k$. C_i realizes the Boolean function F_i.
Output: A circuit made up of component types $C_1,C_2,...,C_k$ realizing B and using the minimum total number of components.
Complexity: NP-hard. The proof can be found in [IBAR75a], where IP1 is called P6.

IP2: Circuit Correctness

Input: A Boolean function B and a circuit C.
Output: "yes" if C realizes B and "no" otherwise.
Complexity: NP-hard. The proof can be found in [IBAR75a], where it is called P5. It is shown that tautology reduces to P5 (IP2).

IP3: Circuit Realization
Input: Circuit requirements $(b_1,b_2,...,b_n)$ with the interpretation that b_i gates of type i are needed to realize the circuit; modules 1,2,...,r with composition m_{ij}, where module i has m_{ij} gates of type j; module costs c_i, where c_i is the cost of one unit of module i.
Output: Nonnegative integers $x_1,x_2,...,x_n$ such that

$\sum_i m_{ij}x_i \geq b_j$, i≤j≤n and

$\sum_i c_i x_i$ is minimized.

Complexity: NP-hard. See Section 4.1.1.

IP4: Construction of a Minimum Cost Standard Library of Replaceable Modules
Input: A set $\{C_1,C_2,...,C_n\}$ of logic circuits such that circuit C_i contains y_{ij} circuits of type j, 1≤j≤r; and a limit, p, on the number of circuits that can be put into a module.
Output: A set M=$\{m_1,m_2,...,m_k\}$ of module types, with module m_i containing a_{ij} circuits of type j, such that:

(i) $\sum_j a_{ij} \leq p, \quad 1 \leq i \leq k$;

(ii) $\sum x_{ij}\, a_{jq} \geq y_{iq}$, $1 \leq i \leq n$ and $1 \leq q \leq r$;

x_{ij} = smallest number of modules m_j needed in implementing C_i.

(iii) $\sum_i \sum_j x_{ij}$ is minimum over all choices of M.

Complexity: NP-hard. Partition (NP9) reduces to IP4 as follows. Let A=$\{a_1,a_2,...,a_n\}$ be an arbitrary instance of NP9. Construct the following instance of IP4. The set $\{C_1,C_2,...,C_n,C_{n+1}\}$ has the composition

$y_{ii} = a_i$, $1 \leq i \leq n$;

$y_{ij} = 0$, $1 \leq i,j \leq n$ and $i \neq j$;

$y_{n+1,i} = a_i$, $1 \leq i \leq n$;

$p = (\sum_i a_i)/2$.

Clearly, there exists a set M such that $\sum_i \sum_j x_{ij} = n+2$ iff the corresponding partition problem has answer "yes".

IP5: Construction of a Standard Library of a Minimum Number of

Replaceable Module Types
Input: Same as in IP4. In addition, a cost bound C is specified.
Output: A minimum cardinality set M = $\{m_1,m_2,...,m_k\}$ with a_{ij}, 1≤i≤k, 1≤j≤r as in IP4 for which there exist natural numbers x_{ij} such that

$\sum_j x_{ij} a_{jm} \geq y_{im}$, 1≤m≤r, 1≤i≤n, and

$\sum x_{ij} \leq C$.

Complexity: NP-hard. Partition (NP9) reduces to IP5, as follows. Given an arbitrary instance of partition, the equivalent instance of IP5 is constructed exactly as described for IP4. In addition, let C = n+2. Clearly, k=2 can be achieved iff the corresponding partition problem has answer "yes".

IP6: Minimum Cardinality Partition
Input: A set V={1,2,...,n} of circuit nodes; a symmetric weighting function w(i,j) such that w(i,j) is the number of connections between nodes i and j; a size function s(i) such that s(i) is the space needed by node i; and constants E and S which are, respectively, bounds on the number of external connections and the space per module.
Output: Partition P=$\{P_1,P_2,...,P_k\}$ of V of minimum cardinality such that :

(a) $\sum_{i \in P(mod P_j} s(i) \leq S,\ 1 \leq j \leq k;$

(b) $\sum_{i \in P(mod P_j \text{ and } q \in P(nm P_j} w(i,q) \leq E,\ 1 \leq j \leq k.$

Complexity: NP-hard. Partition (NP9) reduces to this problem, as follows. Let A = $\{a_1,a_2,...,a_n\}$ be an arbitrary instance of partition.
Equivalent instance of IP6:

$s(i) = a_i,\ 1 \leq i \leq n;$

$S = (\sum_i a_i)/2;$

$w(i,j) = 0,\ 1 \leq i,j \leq n;$

$E = 0.$

There is a minimum partition of size 2 iff the partition instance has a subset that sums to S.

IP7: Minimum External Connections Partition I
Input: V, w, s, and S as in IP6.
Output: A partition P of V such that:

(a) $\sum_{i \in P(mod P_j} s(i) \leq S,\ 1 \leq j \leq k;$

(b) $\sum_j \{ \sum_{i \in P(mod P_j \text{ and } q \in P(nm P_j} w(i,q) \}$ *is minimized.*

Observe that the summation of (b) actually gives us twice the total number of inter-partition connections.
Complexity: NP-hard. This problem is identical to the graph partitioning problem (ND14) in the list of NP-complete problems in [GARE79].

IP8: Minimum External Connections Partition II
Input: V, w, s, and S as in IP6. A constant r.
Output: A partition P=$\{P_1,...,P_k\}$ of V such that:

(a) $k \le r$;

(b) $\sum_{i \in P(mo P_j} s(i) \le S, 1 \le j \le k;$

(c) $\max_j \{ \sum_{i \in P(mo P_j \text{ and } q \in P(nm P_j} w(i,q)\}$ *is minimized.*
Complexity: NP-hard. Partition (NP9) can be reduced to IP8 as described above for IP6.

IP9: Minimum Space Partition
Input: V, w, s, and E as in IP6. In addition, a constant r.
Output: A partition P = $\{P_1,P_2,...,P_k\}$ of V such that:

(a) $k \le r$;

(b) $\sum_{i \in P(mo P_j \text{ and } q \in P(nm P_j} w(i,q) \le E, 1 \le j \le k;$

(c) $\max_j \{ \sum_{i \in P(mo P_j} S(i)\}$ *is minimized.*
Complexity: NP-hard. Partition (NP9) can be reduced to IP9 in a manner very similar to that described for IP6.

IP10: Module Selection Problem
Input: A partition element A' (as in the output of IP6) containing y_i circuits of type i, $1 \le i \le r$; a set M = $\{m_j | 1 \le j \le n\}$ of module types, with z_j copies of each module type m_j. Each m_j has a cost h_j and contains a_{ij} circuits of type i, $1 \le i \le r$, $1 \le j \le n$.
Output: An assignment of non-negative integers $x_1,x_2,...,x_n$, $0 \le x_j \le z_j$ to minimize the total cost $\sum_{j=1}^{n} x_j h_j$ and subject to the constraint that all circuits in A' are implemented, i.e., $\sum_{j=1}^{n} a_{ij} x_j \ge y_i$, $1 \le i \le r$.
Complexity: NP-hard. IP10 contains the 0/1 Knapsack problem (NP12) as a special case. Given an arbitrary instance w, M, K of the 0/1 Knapsack problem, the equivalent instance of IP10 has

r = 1; Y_1 = M; and $z_j=1, a_{ij}=w_j, h_j=k_j$; i = 1, $1 \le j \le n$.

4.2.2 Placement Problems

PP1: Module Placement Problem
Input: m; p; s; N=$\{N_1,N_2,...,N_s\}$, $N_i\{1,...,m\}$; D(p x p) = $[d_{ij}]$; and W(1:s) = $[w_i]$. m is the number of modules. p is the number of available

positions (or slots, or locations); s is the number of signals; N_i, 1≤i≤s are signal nets; d_{ij} is the distance between positions i and j; and w_i is the weight of net N_i, 1≤i≤s.
Output: X(m x p) = $[x_{ij}]$ such that $x_{ij} \in \{0,1\}$ and

(a) $\sum_{j=1}^{p} x_{ij} = 1$;

(b) $\sum_{i=1}^{m} x_{ij} \leq 1$;

(c) $\sum_{i=1}^{s} w_i f(i,X)$ is minimized.

x_{ij} is 1 iff module i is to be assigned to position j. Constraints (a) and (b), respectively, ensure that each module is assigned to a slot and that no slot is assigned more than one module. f(i,X) measures the cost of net N_i under this assignment. This cost could, for example, be the cost of a minimum spanning tree; the length of the shortest Hamiltonian path connecting all modules in the net; the cost of a minimum Steiner tree; etc. In general, the cost is a function of the d_{ij}s.
Complexity: NP-hard. The quadratic assignment problem (NP14) is readily seen to be a special case of the placement problem PP1. To see this, just observe that every instance of NP14 can be transformed into an equivalent instance of PP1 in which $|N_i|$=2 for every net and f(i,X) is simply the distance between the positions of the two modules in N_i. So, PP1 is NP-hard.

PP2: One-Dimensional Placement Problem
Input: A set of components B = $\{b_1,b_2,...,b_n\}$; a list L = $\{N_1,N_2,...N_m\}$ of nets on B such that:

$N_i \leq$ B, 1≤i≤m;

$\bigcup N_i$ = B; N_iN_j = , i≠j.

Output: An ordering σ of B such that the ordering

$B_\sigma = \{b_{\sigma(1)},b_{\sigma(2)},....,b_{\sigma(n)}\}$

minimizes max {number of wires crossing the interval between $b_{\sigma(i)}$ and $b_{\sigma(i+1)} \mid 1 \leq i \leq n-1\}$.

Complexity: NP-hard. The problem is considered in [GOTO77].

4.2.3 Wiring Problems

WP1: Net Wiring With Manhattan Distance
Input: A set P of pin locations, P = $\{(x_i,y_i)|1≤i≤n\}$; set F of feedthrough locations, F = $\{(a_i,b_i)|1≤i≤m\}$; and a set E = $\{E_i|1(<=i(<=r\}$ of equivalence classes. Each equivalence class defines a set of pins that are to be made electrically common.
Output: Wire sets $W_i = \{[(t_j^i,u_j^i),(v_j^i, w_j^i)]|1≤j≤q_i\}$ such that all pins in E_i are made electrically common. Each (t_j^i, u_j^i) and (v_j^i, w_j^i) is either a pin

location in E_i or is a feedthrough pin. No feedthrough pin may appear as a wire end point in more than one W_i. The wire set, $\cup W_i$ is such that $\sum_{i,j}(|t_j^i - v_j^i| + |u_j^i - w_j^i|)$ is minimum.
Complexity: NP-hard. The Manhattan Steiner Tree problem (NP2) is a special case of WP1. To see this, let

$$E = \{P\} \text{ and } F = \{(a_i,b_i)(a_i,b_i) \notin P\} = \bar{P}$$

Hence WP1 is NP-hard.

WP2: Net Wiring With Euclidean Distance
Input: P, F, and E as in WP1.
Output: Wire sets as in WP1 but $\sum_{i,j}[(t_j^i - v_j^i)^2 + (u_j^i - w_j^i)^2]$ is minimized.
Complexity: NP-hard.
The Euclidean Steiner tree problem (NP1) is a special case of WP2, in exactly the same way that NP2 was a special case of WP1. Hence, it is NP-hard.

WP3: Euclidean Spanning Tree
Input: A set P of pin locations, $P=\{(x_i,y_i) \mid 1 \le i \le n\}$;
Output: A spanning tree for P of minimum total length. The distance between two points (a,b) and (c,d) is the Euclidean metric $[(a-c)^2+(b-d)^2]$.
Complexity: Polynomial. An O(n log n) algorithm to find the minimum spanning tree is presented in [SHAM75].

WP4: Manhattan Spanning Tree
Input: P, as in WP3.
Output: A spanning tree for P of minimum total length. The distance between two points (a,b) and (c,d) is the Manhattan metric $|a-c|+|b-d|$.
Complexity: Polynomial. An O(n log n) algorithm that finds the minimum spanning tree is presented in [HWAN79].

WP5: Degree Constrained Wiring with Manhattan Distance
Input: P, E, and F as in WP1 and a constant d.
Output: W_is as in WP1 but with the added restriction that at most d wires may be incident on any pin or feedthrough location.
Complexity: NP-hard. WP5 contains the Manhattan Hamiltonian path problem (NP6) as a special case. To see this, let E = {P}, d = 2 and F = . Hence WP5 is NP-hard.

WP6: Degree Constrained Wiring With Euclidean Distance
Input: P, F, and E as in WP1 and a constant d.
Output: W_is as in WP2 but with the added restriction that at most d wires may be incident on any pin.
Complexity: NP-hard. WP6 contains the Euclidean Hamiltonian path problem (NP4) as a special case. The argument is analogous to that presented for WP5.

WP7: Length Constrained Wiring (Manhattan Distance)
Input: P, F, and E as in WP1, and a constant L.

Output: W_is as in WP1 with the added requirement that

$|t_j^i - v_j^i| + |u_j^i - w_j^i| \leq L$.

Complexity: NP-hard.
WP7 contains WP1 as a special case, when L = ∞. Since WP1 is NP-hard, so is WP7.

WP8: Length Constrained Wiring (Euclidean Distance)
Input: P, F, and E as in WP1, and a constant L.
Output: W_is as in WP2 with the added requirement that

$[(t_j^i - v_j^i)^2 + (u_j^i - w_j^i)^2] \leq L$.

Complexity: NP-hard. WP8 contains WP2, which is itself NP-hard, as a special case, when L = ∞.

WP9: Euclidean Layering Problem I
Input: A set W of wires, W = {[(u_i,v_i),(x_i,y_i)]1≤i≤n}. (u_i,v_i) and (x_i,y_i) are the coordinates of the end points of wire i.
Output: A partitioning $W_1,W_2,...,W_k$ of W such that W_iW_j = , i≠j; ∪ W_i = W and no two wires in any W_i intersect. The end points of wires are connected by straight wires (i.e., in Euclidean manner). k is to be minimum.
Complexity: NP-hard. See Section 4.1.2.

WP10: Euclidean Layering Problem II
Input: W as in WP9 and a constant r.
Output: A partitioning of W into sets $W_1,W_2,...,W_r$, and X. No two wires in any W_i intersect when end points are connected by a straight wire. |X| is minimum.
Complexity: NP-hard when r=3. The corresponding intersection graph is 3-colorable iff x=. Since deciding 3-colorability of intersection graphs is NP-hard (NP8), WP10 is also NP-hard.

WP11: Manhattan Layering Problem I
Input: Same as in WP9.
Output: Same as in WP9, except that the end points of each wire are connected in a Manhattan manner (i.e., a straight run along the x-axis and a straight run along the y-axis).
Complexity: Status unknown.

WP12: Manhattan Layering Problem II
Input: Same as in WP10.
Output: Same as in WP10, except that wire end points are connected in Manhattan manner.
Complexity: Status unknown.

WP13: Rectilinear Layering Problem
Input: A pxq grid; a wire set W={[$(u_i,v_i)$$(x_i,y_i)$]1 ≤i≤ n}. (u_i,v_i) and (x_i,y_i) are grid points which are the end points of wire i. All the wires are constrained to be routed along the grid lines only.
Output: A partition of W into $W_1,W_2,...,W_r$ such that

(i) $W_i W_j$ = ,i≠j ; $\cup W_i$=W;

(ii) all wires ∈ W_i can be routed along the grid lines without intersections.

(iii) r is a minimum.

Complexity: NP-hard. See Section 4.1.3.

WP14: Grid Routing
Input: Set W of wires as in WP9 and a rectangular mxn grid. The end points of wires correspond to grid points.
Output: A routing for each wire such that no two wires intersect and all wire segments are on grid segments.
Complexity: NP-hard ([KRAM82] and [RICH84]).

WP15: Single Bend Grid Routing
Input: Same as in WP14.
Output: Maximum number of wires that can be routed on the grid using at most one bend per wire.
Complexity: NP-hard [RAGH81].

WP16: Minimum Layer Single Bend Grid Routing
Input: Same as in WP14.
Output: Minimum number of layers needed to route all the wires in W using at most one bend per wire.
Complexity: NP-hard [RAGH81].

WP17: Single Row Layering Problem
Input: A set of vertices V = {1,2,...,n} evenly spaced along a line; a list of nets L = {N_1,N_2,...,N_m} such that N_i V, 1≤i≤m; $\cup N_i$ = V; $N_i N_j$ = i≠j; integers c_u and c_l: the respective upper and lower street capacities.
Output: A decomposition of L into L_1,L_2,..., L_r such that

(i) $L_i L_j$ = , i≠j; $\cup L_i$ = L

(ii) all nets ∈ L_i have single layer single row realizations that require no more than c_u and c_l tracks in the upper and lower streets respectively.

(iii) r is minimum.

Complexity: NP-hard. By setting c_u=0 and c_l=B+1,3-Partition (NP10) can be reduced to WP15 in a manner very similar to that described for WP13.

WP18: Single Row Routing with Non-Uniform Conductor Widths
Input: V and L as in WP17; in addition a natural number valued function t, where t_i is the width of the conductor used to route net N_i.
Output: A layout for the nets that minimizes
max {total width required in the upper street, total width required in the lower street}
Complexity: NP-hard.
Partition (NP9) reduces to this problem, as shown. Given an arbitrary instance of partition A = {a_1,a_2 ... a_n} the equivalent instance of WP16 is:
V = {1,2, ... 2n};
N_i = {i,2n+1-i} , 1≤i≤n;
t_i = a_i, 1≤i≤n.

Clearly, there exists a realization with upper street width = lower street width = $(\sum a_i)/2$ iff the corresponding partition instance has answer "yes".

WP19: Single Row Routing Problem
Input: V and L as in WP17.
Output: A layout for L that minimizes
max {number of tracks needed in upper street, number of tracks needed in lower street}
Complexity: NP-hard. See [ARNO82].

WP20: Minimum Width Single Row Routing
Input: V and L, as in WP17.
Output: A layout for L that minimizes (number of tracks needed in upper street + number of tracks needed in lower street).
Complexity: Status unknown.

WP21: Single Row Routing With Fewest Bends I
Input: V and L, as in WP17.
Output: A layout for L that minimizes the total number of bends in the wiring paths.
Complexity: NP-hard [RAGH84].

WP22: Single Row Routing With Fewest Bends II
Input: V and L, as in WP17.
Output: A layout for L that minimizes the maximum number of bends in any one wire.
Complexity: NP-hard [RAGH84].

WP23: Single Row Routing With Fewest Interstreet Crossings I
Input: V and L, as in WP17.
Output: A layout for L that minimizes the total number of conductor crossings between the upper and lower streets.
Complexity: NP-hard [RAGH84].

WP24: Single Row Routing With Fewest Interstreet Crossings II
Input: V and L, as in WP17.
Output: A layout for L that minimizes the maximum number of conductors between an adjacent pair of nodes.
Complexity: NP-hard [RAGH84].

WP25: One Component Routing
Input: A rectangular component of length l and height h having n pins along its periphery and a set of two point nets defined on these pins.
Output: A two layer wiring of the nets such that all vertical runs are on one layer and all horizontal runs are on the other. Wires can run only around the component. The area of the smallest rectangle that circumscribes the component and all routing paths is minimized.
Complexity: $O(n^3)$ [LaPa80a]. When an arbitrary number of rectangular components are present and each net may consist of several pins, the routing problem is NP-hard [SZYM82b].

WP26: River Routing
Input: Two ordered sets $X = (x_1, x_2, \cdots, x_n)$ and $Y = (y_1, y_2, \cdots, y_n)$ of

pins separated by a wiring channel. Each set is divided into blocks of consecutive pins. While the relative ordering of blocks is fixed, their relative positioning is not.
Output: A one layer wiring pattern connecting x_i to y_i, 1≤i≤n. The channel dimensions necessary to accomplish this wiring are given by the vertical distance (separation) needed between the two rows of pins and the horizontal length (spread) of the channel. The channel area is the product of spread and separation. The output wiring should optimize these channel dimensions.
Complexity: If the wiring channel is assumed to be a single layer grid (hence wires must be rectilinear) a placement of the blocks that minimizes the channel separation can be found in O(n logn) time; for a given separation a placement with minimal spread can be determined in O(n) time; and a placement minimizing channel area may be obtained in $O(n^2)$ time [LEIS81]. When the position of each pin is not fixed and wires are not constrained to run along grid lines (but must still consist of horizontal and vertical runs with some minimum separation) the channel separation can be minimized in $O(n^2)$ time [DOLE81]. When the wires can take on any shape and the pin positions are fixed, minimum length wiring can ba done in $O(n^2)$ time [TOMP80]. If two layers are allowed with one devoted to horizontal runs and the other to vertical runs, then the minimum separation can be found in O(n) time [SIEG81] provided upto two vertical wires are permitted to overlap. The offset that minimizes the separation for both the single layer and restricted two layer case can be found in O(nlogn) time [SIEG81]. Furthermore, the offset that leads to the minimum area circumscribing rectangle may be found in $O(n^3)$ time for both cases.

WP27: Channel Routing
Input: A set of nets. Each net is a pair of pins on opposite sides of a rectangular channel.
Output: A two layer wiring of the nets such that no wire has more than one horizontal segment. Horizontal segments are to be layed out in one layer and vertical segments in the other. The number of horizontal tracks used is minimized.
Complexity: NP-hard [LaPa80b]. The problem remains NP-hard if doglegs are allowed and nets are permitted to contain any number of pins from both sides of the channel [SZYM82a]. Several good heuristics for two layer channel routing exist ([DEUT76], [YOSH82], [MARE82], [RIVE81], [FIDD82]). All of these allow doglegs and those of [MARE82] and [RIVE81] permit horizontal and vertical segments to share layers. Lower bounds on the number of tracks needed are developed in [BROW81]. Routing in the T-shaped and X-shaped junctions that result from the intersection of rectangular channels is considered in [PINT81].

4.2.4 Fault Detection Problems

Let C be an n-input 1-output combinational circuit. Let Z be the set of all possible single stuck at 0 (s-a-0) and stuck at 1 (s-a-1) faults. The tuple ($i_1, i_2, \cdots, i_n$, j, F(0), F(1)) is a fault detection test for C iff each of the following is satisfied:

(a) $i_k \varepsilon \{0,1\}$, 1≤k≤n; jε{0,1}

(b) F(0) Z; F(1) Z; F(0) F(1) = ; F(0) F(1) ≠ .

(c) (i) There is a s-a-0 fault at one of the locations in F(0) iff C with inputs $i_1,i_2,...,i_n$ has output j.

(ii) There is a s-a-1 fault at one of the locations in F(1) iff C with inputs $i_1,i_2,...,i_n$ has output j.

A *test set*, T, is a set of fault detection tests. T is a test set for L Z iff (i) the union of all the F(0)s for the tests in T is L and (ii) the union of all the F(1)s for the tests in T is L. If L = Z then T is a test set for C. Circuit C is irredundant iff it has a test set.

FDP1: Irredundancy
Input: A combinational circuit C.
Output: "yes" iff the circuit is irredundant (i.e. all s-a-0 and s-a-1 faults can be detected by I/0 experiments); "no" otherwise.
Complexity: NP-hard. See problem P1 in [IBAR75a].

FDP2: Line Fault Detection
Input: Same as for FDP1.
Output: "yes" iff a fault in a particular input line can be detected by I/0 experiments; "no" otherwise.
Complexity: NP-hard. See problem P2 in [IBAR75a].

FDP3: All Faults Detection
Input: Same as for FDP1.
Output: "yes" iff all single input faults can be detected by I/0 experiments; "no" otherwise.
Complexity: NP-hard. See problem P3 in [IBAR75a].

FDP4: Output Fault Detection
Input: Same as for FDP1.
Output: "yes" iff faults in the output line can be detected by I/0 experiments; "no" otherwise.
Complexity: NP-hard. See problem P4 in [IBAR75a].

FDP5: Minimal Test Set
Input: Same as for FDP1.
Output: If C is irredundant, a minimal test set for C.
Complexity: NP-hard. See [IBAR75a].

5. HEURISTICS AND USUALLY GOOD ALGORITHMS

Having discovered that many of the interesting problems that arise in design automation are computationally difficult (in the sense that they are probably not solvable by a polynomial time algorithm), we are left with the issue of alternate paths one might take in solving these problems. The three most commonly tried paths are:

(a) Obtain a heuristic algorithm that is both computationally feasible and that obtains "reasonably" good solutions. Algorithms with this latter property are called *approximation algorithms*. We are interested in good, fast (i.e., low order polynomial; say O(n), O(n log n), O(n^2), etc.) approximation algorithms.

(b) Arrive at an algorithm that always finds optimal solutions. The complexity of this algorithm is such that it is computationally feasible for "most" of the instances people want to solve. Such an algorithm will be called a *usually good algorithm*. The Simplex algorithm for linear programming is a good example of a usually good algorithm. Its worst case complexity is exponential. However, it is able to solve most of the instances given it in a "reasonable" amount of time (much less than the worst case time).

(c) Obtain a computationally feasible algorithm that "almost" always finds optimal solutions. An algorithm with this property is called a *probabilistically good* algorithm.

5.1 Approximation Algorithms

When evaluating an approximation algorithm, one considers two measures: algorithm complexity, and the quality of the answer (i.e., how close it is to being optimal). As in the case of complexity, the second measure may refer to the worst case or the average case.

There exist several categories of approximation algorithms. Let A be an algorithm that generates a feasible solution to every instance I of a problem P. Let $F^*(I)$ be the value of an optimal solution, and let F'(I) be the value of the solution generated by A.

Definition: A is an *absolute approximation* algorithm for P iff $|F^*(I)-F'(I)| \le k$ for all I, with k a constant. A is an *f(n)-approximate* algorithm for P iff $|F^*(I)-F'(I)|/F^*(I) \le f(n)$ for all I. n is the size of I and we assume that $|F^*(I)|>0$. An f(n)-approximate algorithm with $f(n) \le \varepsilon$ for all n and some constant ε is an ε-*approximate* algorithm.

Definition: Let A(ε) be a family of algorithms that obtain a feasible solution for every instance I of P. Let n be the size of I. A(ε) is an *approximation scheme* for P iff for every ε>0 and every instance I, $|F^*(I)-F'(I)|/F^*(I) \le \varepsilon$. An approximation scheme whose time complexity is polynomial in n is a *polynomial time approximation* scheme. A *fully polynomial time approximation* scheme is an approximation scheme whose time complexity is polynomial in n and 1/ε. For most of the heuristic algorithms in use in the design automation area, little or no effort has been devoted to determining how good or bad (relative to the optimal solution values) these are. In what follows, we briefly review some results that concern design automation. The reader is referred to [HORO78, Chap 12] for a more complete discussion of heuristics for NP-hard problems. For most NP-hard problems, it is the case that the problem of finding absolute approximations is also NP-hard. As an example, consider problem IP3 (circuit realization). Let

$$\min \sum_i c_i x_i$$

(1) subject to

$$\sum_i m_{ij} x_i \ge b_j,\ 1 \le j \le n$$

and

$$x_i \ge 0 \text{ and integer}$$

be an instance of IP3. Consider the instance:

$$\min \sum_i d_i x_i$$

(2) subject to

$$\sum_i m_{ij} x_i \ge b_j,\ 1 \le j \le n$$

$$x_i \ge 0 \text{ and integer}$$

where $d_i=(k+1)c_i$. Since the values of feasible solutions to (2) are at least k+1 apart, every absolute approximation algorithm for IP3 must produce optimal solutions for (2). These solutions are, in turn, optimal for (1). Hence, finding absolute approximate solutions for any fixed k is no easier than finding optimal solutions. Horowitz and Sahni [HORO78, Chap 12] provide examples of NP-hard problems for which there do exist polynomial time absolute approximation algorithms. It has long been conjectured ([GILB68]) that, under the Euclidean metric,

$$\frac{\textit{length of minimum spanning tree}}{\textit{length of optimum Steiner tree}} = \frac{F'}{F^*} = 2/3$$

Hence,

$$\frac{|F'-F^*|}{F^*} \leq \frac{2-\sqrt{3}}{\sqrt{3}} \leq 0.155$$

Hence, the O(n log n) minimum spanning tree algorithm in [SHAM75] can be used as a 0.155-approximate algorithm for the Euclidean Steiner tree problem.

For the rectilinear Steiner tree problem, it is known ([HWAN79],

[LEE76]) that

$$\frac{\textit{length of minimum spanning tree}}{\textit{length of optimum Steiner tree}} = \frac{F'}{F^*} \leq 3/2$$

Hence,

$$\frac{|F'-F^*|}{F^*} \leq 1/2$$

The O(n log n) spanning tree algorithm in [HWAN79a] can be used as a 0.5-approximate algorithm for the Steiner tree problem.

Since both the Euclidean and rectilinear Steiner tree problems are strongly NP-hard, they can be solved by a fully polynomial time approximation scheme iff P=NP. (See [HORO78, Chapter 12] for a definition of strong NP-hardness and its implications).

[SHAM75] suggests an O(n log n) approximation algorithm that finds a traveling salesman tour that is not longer than twice the length of an optimal tour, using the Euclidean minimum spanning tree. This is a 1-approximate algorithm, and it is possible to do better. [CHRI76] contains a 0.5-approximate algorithm for this problem. Sahni and Gonzalez [SAHN76] have shown that there exists a polynomial time ε-approximation algorithm for the quadratic assignment problem iff P=NP.

5.2 Usually Good Algorithms

Classifying an algorithm as "usually good" is a difficult process. From the practical standpoint, this can be done only after extensive experimentation with the algorithm. The Simplex method is regarded as good only because it has proven to be so over years of usage on a variety of instances. An analytical approach to obtain such a classification comes from probabilistic analysis. [KARP75 and 76] has carried out such an analysis for several NP-hard problems. Such analysis is not limited to algorithms that guarantee optimal solutions. [KARP77] analyzes an approximation algorithm for the Euclidean traveling salesman problem. The net result is a fast algorithm that is expected to produce near optimal salesman tours. Dantzig [DANT80] analyzes the expected behavior of the Simplex method.

Kirkpatrick et al. ([KIRK83 and VECC83]) have proposed the use of simulated annealing to obtain good solutions to combinatorially difficult design automation problems. Experimental results presented in these papers as well as in [NAHA85], [GOLD84], and [ROME84] indicate that simulated annealing does not perform as well as other heuristics when the problem being studied has a well defined mathematical model. However, for problems with multiple constraints that are hard to model, simulated annealing can be used to obtain solutions that are superior to those obtainable by other methods. Even in the case of easily modeled problems, simulated annealing may be used to improve the solutions obtained by other methods.

6. CONCLUSIONS

Under the worst case complexity measure, most design automation problems are intractable. This conclusion remains true even if we are interested only in obtaining solutions with values guaranteed to be "close" to the value of optimal solutions. The most promising approaches to certifying the value of algorithms for these intractable problems appear to be: probabilistic analysis and experimention. Another avenue of research that may prove fruitful is the design of highly parallel algorithms (and associated hardware) for some of the computationally more difficult problems.

7. REFERENCES

[ABRA80] Abramovici, M. and M. A. Breuer, "Fault diagnosis base on effect-cause analysis: An introduction", Proceedings 17th Design Automation Conference, 1980, pp. 69-76.

[ARNO82] Arnold, P.B., "Complexity results for circuit layout on double-sided printed circuit boards", Bachelor's thesis, Harvard University, 1982.

[BREU66] Breuer, M. A., "The application of Integer Programming in Design Automation", Proc. SHARE Design Automation Workshop, 1966.

[BREU72a] Breuer, M.A.(ed.), Design Automation of Digital Systems, Vol.1, Theory and Techniques, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[BREU72b] Breuer, M.A., "Recent Developments in the Automated Design and Analysis of Digital Systems", Proceedings of the IEEE, Vol. 60, No. 1, January 1972, pp. 12-27.

[BROW81] Brown, D. and R. Rivest, "New lower bounds for channel width", in VLSI Systems and Computations, ed. Kung et al., Computer Science Press, pp. 178-185, 1981.

[CHRI76] Christofedes, N., "Worst-case Analysis of a New Heuristic for a Traveling Salesman Problem", Mgmt. Science Research Report, Carnegie Mellon University, 1976.

[CLAR69] Clark, R.L., "A Technique for Improving Wirability in Automated Circuit Card Placement, Rand Corp. Report R-4049, August 1969.

[CLEE76] vanCleemput, W.M., "Computer Aided Design of Digital Systems", 3 volumes, Digital Systems Lab., Stanford University, Computer Science Press, Potomac, MD, 1976.

[DANT80] Dantzig, G., "Khachian's algorithm: a comment"' SIAM News, vol 13 no. 5, Oct. 1980.

[DEJK77] Dejka, W. J., "Measure of testability in device and system design", Proceedings 20th Midwest Symposium on Circuits and Systems, Aug. 1977, pp. 39-52.

[DEUT76] Deutsch, D., "A dogleg channel router", Proceedings 13th Design Automation Conference, pp. 425-433, 1976.

[DOLE81] Dolev, D., et al., "Optimal wiring between rectangles", 13th Annual Symposium On Theory Of Computing, pp. 312-317, 1981.

[EHRL76] Ehrlich, G.S., S.Even and R.E.Tarjan, "Intersection graphs of Curves in the Plane", Jr. Combin. Theo., Ser. B, 21, 1976, pp. 8-20.

[EICH77] Eichelberger, E. B. and T. W. Williams, "A logic design structure for LSI testability", Proceedings 14th Design Automation Conference, 1977, pp. 462-468.

[FIDD82] Fidducia, C. and R. Rivest, "A greedy channel router", Proceedings 19th Design Automation Conference, 1982, pp. 418-424.

[GARE75] Garey, M.R. and D.S. Johnson, "Complexity Results for

Multiprocessor Scheduling under Resource Constraints", SIAM J. Comput., 4, pp.397-411.

[GARE76a] of Near-Optimal Graph Coloring", JACM, 23, 1976, pp.43-49.

[GARE76b] Geometric Problems", Proc. 8th Annual ACM Symposium on Theory of Computing, ACM, NY., 1976, pp.10-22.

[GARE77] Garey, M.R., R.L.Graham and D.S.Johnson, "The Complexity of Computing Steiner Minimal Trees", SIAM Jr.Appl.Math., 32, 1977, pp.835-859.

[GARE79] Garey,A Guide to the Theory of NP-completeness", W.H.Freeman and Co., San Francisco, CA, 1979.

[GILB68] Gilbert, E.N. and H.O. Pollak, "Steiner Minimal Trees", SIAM J Appl. Math., January 1968, pp.1-29.

[GOLD79] Goldstein, L. H., "Controllability/observability analysis of digital circuits", IEEE Trans. on Circuits and Systems, vol. CAS-26, no. 9, Sept. 1979, pp. 685-693.

[GOLD84] B. Golden and C. Skiscim, Using simulated annealing to solve routing and location problems, University of Maryland, College of Business Administration, Technical Report, Jan. 1984.

[GOTO77] Goto S., I. Cederbaum and B. S. Ting, "Suboptimum Solution of the Backboard Ordering with Channel Capacity Constraint", IEEE Transactions on Circuits and Systems, Vol. CAS-24, November 1977, pp.645-652.

[GRAS80] Grason, J. and A. W. Nagle, "Digital test generation and design for testability", Proceedings 17th Design Automation Conference, 1980, pp. 175-189.

[HABA68] Habayeb, A.R., "System Decomposition, Partitioning, and Integration for Microelectronics", IEEE Trans. on System Science and Cybernetics, Vol.SSC-4, No.2, July 1968, pp.164-172.

[HASH71] Hashimoto, A. and J. Stevens, "Wire routing by optimizing channel assignment within large apertures", Proceedings 8th Design Automation Conference, 1971, pp. 155-169.

[HORO74] Horowitz, E. and S.Sahni, "Computing Partitions with Applications to the Knapsack Problem",JACM, 21, 1974, pp.277-292.

[HORO78] Horowitz, E. and S.Sahni, Fundamentals of Computer Algorithms, Computer Science Press, Potomac, MD, 1978.

[HWAN76] Hwang, F.K., "On Steiner Minimal Trees wioth Rectilinear Distance", SIAM J Appl. Math, January 1976, pp.104-114.

[HWAN79a] Rectilinear Minimal Spanning Trees", JACM, 26, 1979, pp.177-182.

[HWAN79b] Rectilinear Steiner Trees", IEEE Transactions on Circuits and Systems, Vol. CAS-26, January 1979, pp.75-77.

[IBAR75a] Ibarra, O.H. and S.Sahni, "Polynomially Complete Fault Detection Problems", IEEE Trans. on Computers, Vol.C-24, March 1975, pp.242-249.

[IBAR75b] Ibarra, O.H. and C. E. Kim, "Fast Approximation

Algorithms for the Knapsack and Sum of Subset Problems", JACM, 22, 1975, pp.463-468.

[IBAR77] Ibaraki, T., T.Kameda and S.Toida, "Generation of Minimal Test Sets for System Diagnosis", University of Waterloo, 1977.

[JOHA79] Johannsen, D., "Bristle blocks: A silicon compiler"' Proceedings 16th Design Automation Conference, 1979.

[JOHN82] Johnson, D., "The NP-Completeness Column: An ongoing guide", Jr of Algorithms, Dec 1982, Vol 3, No 4, pp. 381-395.

[KARP72] Karp, R., "On the Reducibility of Combinatorial Problems", in R.E.Miller and J.W.Thatcher(eds.), Complexity of Computer Computations, Plenum Press, NY, 1972, pp.85-103.

[KARP75] Karp, R., "The Fast Approximate Solution of Hard Combinatorial Problems", Proc. 6th Southeastern Conf. on Combinatorics, Graph Theory, and Computing, Winnipeg, 1975.

[KARP76] Karp, R., "The Probabilistic Analysis of Some Combinatorial Search Algorithms", University of California, Berkeley, Memo No.ERL-M581, April 1976.

[KARP77] Karp, R., "Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane", Math. of Oper. Res., 2(3), 1977, pp.209-224.

[KIRK83] S. Kirkpatrick, C. Gelatt, Jr., and M. Vecchi, Optimization by simulated annealing, *Science*, Vol 220, No 4598, May 1983, pp. 671-680.

[KODR62] Kodres, U.R., "Formulation and Solution of Circuit Card Design Problems Through Use of Graph Methods", in G.A.Walker(ed.), Advances in Electronic Circuit Packaging, Vol.2, Plenum Press, New York, NY, 1962, pp.121-142.

[KODR69] Kodres, U.R., "Logic Circuit Layout", The Digest Record of the 1969 Joint Conference of Mathematical and Computer Aids to Design, October 1969.

[KRAM82] Kramer, M.R., and J. van Leeuwen, "Wire-routing is NP-Complete" Technical Report, Computer Science Dept., University of Utrecht, The Netherlands, 1982.

[LAWL62] Lawler, E.L., "Electrical Assemblies with a Minimum Number of Interconnections", IEEE Trans. on Electronic Computers (Correspondence), Vol.EC-11, February 1962, pp.86-88.

[LAWL69] Lawler, E.L., K.N.Levitt and J.Turner, "Module Clustering to Minimize Delay in Digital Networks", IEEE Trans. on Computers, Vol.C-18, January 1969, pp.47-57.

[LAWL77] Lawler, E.L., "Fast Approximation Algorithms for Knapsack Problems",Proc. 18th Ann. IEEE Symp. on Foundations of Computer Science, 1977, pp.206-213.

[LEE76] Lee, J.H., N.K. Bose and F.K. Hwang, "Use of Steiner's Problem in Suboptimal Routing in Rectilinear Metric", IEEE Transactions on Circuits and Systems, Vol. CAS-23, July 1976, pp.470-476.

[LEIS81] Leiserson, C. and R. Pinter, "Optimal placement for river

routing", in VLSI Systems and Computations, Kung et al. editors, Computer Science Press, pp.126-142, 1981.

[LaPa80a] La Paugh, A., "A polynomial time algorithm for routing around a rectangle", 21st Annual IEEE Symposium on Foundations of Computer Science, pp. 282-293, 1980.

[LaPa80b] La Paugh, A., "Algorithms for integrated circuit layout: An analytic approach", MIT-LCS-TR-248, Doctoral dissertation, MIT, 1980.

[LUEK75] Lueker, G.S., "Two NP-complete Problems in Nonnegative Integer Programming", Report No.178, Computer Science Lab., Princeton University, Princeton, NJ, 1975.

[MARE82] Marek-Sadowska, M. and E. Kuh, "A new approach to channel routing", Proceedings 1982 ISCAS Symposium, IEEE, pp. 764-767, 1982.

[MEAD80] Mead, C. and L. Conway, "Introduction to VLSI systems", Addison-Wesley, 1980.

[NAHA85] Nahar, S., S. Sahni and E. Shragowitz, "Experiments with simulated annealing", 1985 Design Automation Conference.

[NOTZ67] Notz, W.A., E.Schischa, J.L.Smith and M.G.Smith, "Large Scale Integration; Benefitting the Systems Designer", Electronics, February 20, 1967, pp.130-141.

[NOYC77] Noyce, R. N., "Microelectronics", Scientific American, Sept. 1977, pp. 62-69.

[PAPA77] Papadimitriou, C.H.",The Euclidean Traveling Salesman Problem is NP-Complete", Theoretical Computer Science 4, 1977, pp.237-244.

[PINT81] Pinter, R., "Optimal routing in rectilinear channels", in VLSI Systems and Computations, ed. Kung et al., pp.153-159, 1981.

[POME65] Pomentale, T., "An Algorithm for Minimizing Backboard Wiring Functions", Comm. ACM, Vol.8, No.11, November 1965, pp.699-703.

[RAGH81] Raghavan, R., J. Cohoon, and S. Sahni, "Manhattan and rectilinear routing", Technical Report 81-5, University of Minnesota, 1981.

[RAGH84] Raghavan, R., and S. Sahni, "The complexity of single row routing", IEEE Transactions on Circuit and Systems, vol. CAS-31, No 5, May 1984, pp. 462-472.

[RICH84] Richards, D., "Complexity of single-layer routing", IEEE Transactions on Computers, March 1984, pp. 286-288.

[RIVE81] Rivest, R., A. Baratz, and G. Miller, "Provably good channel routing algorithms", in VLSI systems and Computations, ed. Kung et al., pp. 153-159, 1981.

[ROME84] F. Romeo, A. Vincentelli, and C. Sechen, Research on simulated annealing at Berkeley, Proceedings ICCD, Oct. 1984, pp 652-657.

[ROTH66] Roth, J. P., "Diagnosis of automatic failures: A calculus and a method", IBM Jr of Syst. and Dev., no. 10, 1966, pp. 278-291.

[SAHN76] Sahni, S. and T.Gonzalez, "P-Complete Approximation Problems", JACM, 23, 1976, pp.555-565.

[SAHN81] Sahni, S., "Concepts in discrete mathematics", Camelot Publishing Co., Fridley, Minnesota, 1981.

[SHAM75] Shamos, M.I. and D.Hoey, "Closest Point Problems", 16th Annual IEEE Symposium on Found. of Comp. Sc., 1975, pp.151-163.

[SIEG81] Siegel, A. and D. Dolev, "The separation for general single layer wiring barriers", in VLSI Systems and Computations, Kung et al. editors, Computer Science Press, pp. 143-152, 1982.

[SO74] So, H.C., "Some Theoretical Results on the Routing of Multilayer Printed Wiring Boards", IEEE Symposium on Circuits and Systems, 1974, pp.296-303.

[SZYM82a] Szymanski, T., "Dogleg channel routing is NP-complete", unpublished manuscript, Bell Labs, 1982.

[SZYM82b] Szymanski, T. and M. Yannanakis, Unpublished manuscript, 1982.

[TOMP80] Tompa, M., "An optimal solution to a wiring routing problem", 12th Annual ACM Symposium On Theory Of Computing, pp. 161-176, 1980.

[TING78] Ting, B.S. and E.S. Kuh, "An Approach to the Routing of Multilayer Printed Circuit Boards", IEEE Symposium on Circuits and Systems, 1978, pp.902-911.

[ULLM84] Ullman, J., Computational Aspects of VLSI, Computer Science Press, Maryland, 1984.

[VECC83] M. Vecchi and S. Kirkpatrick, Global wiring by simulated annealing, *IEEE Trans. On Computer Aided Design*, Vol CAD-2, No 4, Oct. 1983, pp 215-222.

[WILL82] Williams, T. W. and K. P. Parker, "Design for testability: A survey", IEEE Trans. on Computers, vol. C-31, no. 1, 1982, pp. 2-15.

[WOJT81] Wojtkowiak, H., "Deterministic systems design from functional specifications", Proceedings 18th Design Automation Conference, 1981, pp. 98-104.

[YAN71] Yan, S. S. and Y. S. Tang, "An efficient algorithm for generating complete test sets for combinational logic circuits", IEEE Trans. on Computers, 1971.

[YAO75] Yao, A., "An O(|E|log log|V|) Algorithm for Minimum Spanning Trees", Information Processing Letters, 4(1), 1975, pp.21-23.

[YOSH82] Yoshimura, T. and E. Kuh, "Efficient algorithms for channel routing", IEEE Transactions on DA, pp. 1-15, 1982.