Chapter 9

DISCRETE MATHEMATICS IN MANUFACTURING

MARTIN GRÖTSCHEL
Konrad-Zuse-Zentrum
für Informationstechnik and
Technische Universität, Berlin
Germany

1. Introduction. Computer aided design, flexible manufacturing and computer integrated manufacturing have become technological buzzwords of our time. We are fascinated when we see driverless vehicles transport parts through a factory, watch robots executing complicated movements, or observe automated assembly lines producing goods at a speed, and with a quality, unimaginable with manual production methods. However, when our initial fascination is gone, and we examine the details, we quickly realize that enormous improvements are still possible. In fact, improvements can often be made without costly technical changes: by organizing the production flow in a different way, by designing the products better, by scheduling the jobs differently, or by controlling the machines in a more effective manner. Manufacturing, in general, provides rich opportunities for important mathematical contributions to significant real-world problems and, simultaneously, provides a virtually untapped source of mathematical problems, interesting in their own right.

Although the number of mathematically oriented journals, books and papers in the manufacturing field is rapidly increasing, there is still a huge gap between what could be done, and what actually is done. It is my opinion that there are at least two reasons for this phenomenon. In my experience, many of the talented engineers who build and operate complicated manufacturing systems so ingeniously, simply do not have the background in the rather new mathematical techniques that are necessary to handle the issues to be discussed in the sequel; and they sometimes do not believe that mathematics can help. Secondly, only a few mathematicians are willing to go through the laborious and occasionally painful process of understanding, analyzing and modeling complex manufacturing systems and then discussing their findings with the practitioners. Both parties suffer as a result. Companies in particular miss opportunities for more efficient and cost-effective production, and mathematicians opportunities to identify and solve challenging problems, problems that arise in connection with one of the most fascinating technical developments of our time.

It is not my intention here to survey the mathematical problems that arise in this area. Rather, I will concentrate on those aspects that involve the techniques of discrete mathematics. Many of the problems I am aware of are combinatorial optimization problems. Due to the richness of the field of manufacturing, it is impossible to list all the different problem types. Thus, I will concentrate on practical applications and their mathematical models, applications that the members of my research group have worked on in recent years in cooperation with industry. This work was begun at the University of Augsburg and is continuing in Berlin. In most of the cases reported here, the industry partners were, and often still are, branches of Siemens and Siemens Nixdorf.

I have organized this paper following the natural method of design and production in a typical electronics company. The design phase marks the beginning of a product. I will outline issues from this phase in Section 2. In the next phase, components are produced. The related issues of machine control and the like will be discussed in Section 3. In the final phase, the parts are assembled.

The complex management and scheduling problems of highly automated assembly systems will be described in Section 4. I will touch upon the various mathematical problems that arise in these phases only very briefly. A few remarks about the mathematical techniques involved can be found in Section 5. A glimpse of further important issues such as logistics, distribution and safety aspects is also given in Section 5.

An electronics company was a natural choice for my presentation since by far the largest fraction of our projects have been joint efforts with the Siemens corporation. However, other types of companies, such as automotive, chemical and machine construction companies, could just as well have served to demonstrate the use of mathematics in manufacturing applications.

2. The Design Phase. Companies with bad products and effective production will not survive. Good products, products that customers value, are what makes a company successful. But the financial success of a company depends to a large extent on how an idea is realized and on how the resulting product is manufactured. The transformation of an idea into a producible item is called the design phase. Disregarding the (very vital) aspects of style, or look-and-feel of a product, what is important from our perspective is that a product is designed in such a way that it can be easily and cheaply manufactured. This task is hard to quantify and tremendously complex. The usual approach to tackle it is to break the task (often hierarchically) into several subproblems that are more manageable and to hope that the overall solution is "reasonable".

We outline this general aspect here by describing a few tasks that arise in the design of electronic circuits.

2.1. VLSI Design. By looking at the computers on our desks and comparing them with machines 10 or 20 years old, we can observe the incredible improvements that very large scale integration (VLSI) has brought about. Hundreds of thousands or even millions of transistors integrated on a few square centimeters of silicon (a chip) perform an enormous number of operations at breathtaking speed. This large scale integration is one of the most significant technological revolutions of our time. Mathematics is used here in various stages of the chip design phase. A brief outline follows.

Once the full task of a circuit is described, the logic has to be determined that will perform all the desired operations. This logic is then cast in silicon. Today's approach for physically realizing the logic design is to begin with predefined small cells that perform certain simple logic operations, and then to connect these cells with wires (so-called nets) so that the combination of cells and nets realizes the abstract logic model physically. Depending on the chosen technology, the design rules and customer requirements, the cells have a certain shape (usually rectangular) and size, and the wires need a certain width and require a certain distance to other wires or units on the chip. Given cells and nets together with additional technological constraints, the task is to place the cells and route the nets so that either the size of the resulting chip is as small as possible, or the resulting chip fits into a given frame. There are many additional side constraints that have to be met. For instance, certain cells have to be close to each other, certain nets have to have a maximum or minimum length, etc.

Why do we want the chip to be small? Of course, smaller chips can usually be run at higher cycle speeds and are faster, but a major reason is that the yield of chip production decreases nonlinearly with increasing chip size. Modern chip production in futuristic, extremely clean and almost fully automated factories is basically a fixed cost operation, relatively independent of the number of wafers processed. Therefore, it makes a tremendous difference whether 20% or 80% of the chips produced per day are defective or not. Also, the smaller the chip size, the more likely is a higher, top quality output.

Let us now look at a specific technology, the so-called sea-of-cells technology, one that is currently in wide use. Here a rectangular "master chip" is given. Among the feasible master chips, a master is usually chosen that is as small as possible so that one can hope to realize the given circuit on it. This master is subdivided into, say, m base cells. All logic cells are rectangular and have a size equal to some multiple of the base cell size. Suppose n logic cells are given, connected by z nets. The task roughly is to assign the n logic cells to the base cells so that all cells fit, no two logic cells overlap, all nets can be routed, and such that the total net length (the sum of the lengths of all nets) is as small as possible.

It turns out that this problem is enormously complicated. At least at present, it seems impossible

to handle the placement and routing problem simultaneously in a sound mathematical model for realistic problem instances.

Thus, the whole task is subdivided into several hierarchical problems. Depending on instance sizes and algorithmic approaches, the following phases are considered in general chip design: global placement, global routing, local placement, local routing, layer assignment and (possibly) compaction. These phases are processed in an iterative manner and may be repeated until satisfactory results are achieved. An excellent account of this area can be found in [L90].

2.2. Placement in Sea-of-Cells Technology. We will now look at the placement problem (without distinguishing between global and local placement) for chip design in sea-of-cells technology. Figure 1 shows a small master consisting of 26 × 18 base cells and the outer frame of pad cells; the nine dark rectangles are logic cells placed on this master.

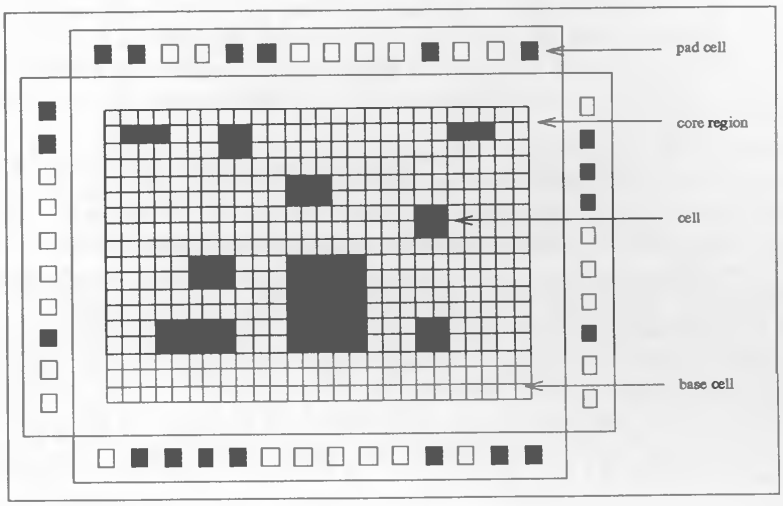

Figure 1: Example of a sea-of-cells master

In the last decade many algorithms have been developed for a solution of this placement problem. They can be categorized as follows.

The most popular and probably still most frequently used approach is based on the min-cut heuristic for bipartitioning graphs. We define a hypergraph $G = (V, E)$, where the node set represents the logic cells. For every net of the given circuit, we define a hyperedge consisting of all logic cells that the net connects. The idea is to recursively partition the node set of the hypergraph into two sets of roughly equal size such that the number of edges in the associated cut is as small as possible; [SK72], [Br77]. The main drawback when working in this scheme is that the global view of the problem gets lost. Moreover, the seemingly easy problem of finding a minimum cut with both sides of about equal size is an $\mathcal{NP}$-hard problem itself.

Another class of algorithms has been developed that model the placement problem mathematically in such a way that a relaxation of it can be solved to optimality. One such relaxation can be interpreted as a (nonlinear) energy model, where the objective function approximates the total wiring length and the side constraints assure that the trivial placement (all cells are assigned to the same base cell) is excluded [KlSJ88]. The solution of such a relaxation is usually not feasible for the placement problem (e.g., cells overlap) and thus the solution of the relaxation has to be modified heuristically in order to obtain a solution of the underlying discrete optimization problem.

We now address the placement problem by introducing boolean variables

$$x_{ik} = \begin{cases} 1, & \text{if cell } i \text{ is assigned to base cell } k, \\ 0, & \text{otherwise.} \end{cases}$$

Let $o(ik, jl)$ and $d(ik, jl)$ denote, respectively, the number of overlapping base cells and the distance if cells i and j are assigned to base cells k and l. Then a corresponding quadratic 0/1-optimization problem can be stated as follows:

$$
\begin{aligned}
& min && \sum_{k=1}^{m}\sum_{l=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{n}(c_{ij}d(ik,jl)+\lambda\cdot o(ik,jl))x_{ik}x_{jl} \\
& s.t. && \sum_{k=1}^{m} x_{ik}=1 \quad \text{for all} \quad i=1\ldots n, \\
& && x_{ik}\in\{0,1\} \quad \text{for all} \quad i=1\ldots n,\ k=1\ldots m,
\end{aligned} \tag{1.1}
$$

where λ is the penalty parameter for the overlaps, and the coefficients c_{ij} denote the number of nets between cells i and j. Problem (1.1) is a (slightly imprecise) model of the placement problem. The equations in (1.1) guarantee that all cells are placed, but the requirement that two cells may not overlap on a chip has been dropped since it leads to a tremendous number of additional inequalities. This requirement is taken into account indirectly by a suitable increase of the objective function value, when cell overlaps occur.

One can show that (1.1) is $\mathcal{NP}$-hard. The complexity of (1.1) and its sheer size for real problems suggest the idea of decomposing large instances into smaller ones. For example, the problem "soc4", mentioned later, has 2776 logic cells and 13440 base cells on the master chip. This leads to a quadratic 0/1-program with 37,309,440 variables. The decomposition is carried out in such a way that the global view does not get lost. For a solution of the decomposed problems, several heuristics have been developed; cf. [JMRW92], [W92] for a detailed outline of the procedure.

In Table 1 the quadratic 0/1-approach sketched above is compared to two state-of-the-art algorithms used in industry, namely the min-cut placement procedure and a method based on the energy model in [KlSJ88]. The three algorithms were applied to four electronic circuits, called soc1, ..., soc4, that consist of 602 to 2776 logic cells. Their performance was measured by the total wiring length estimated by an industrial routing algorithm for the respective placements. The resulting wiring lengths are reported in Table 1.

	min-cut	energy	0/1 QP
soc1	212258	180445	169892
soc2	194732	189683	185592
soc3	766622	652129	553575
soc4	623159	506160	497285

Table 1

The running time of the quadratic 0/1-programming algorithm is about ten times as large as the roughly identical running times of the other two heuristics. Using certain clustering techniques for the quadratic approach in addition, see [W92], the running times can be made comparable with the running times of the other two heuristics without loss of the solution quality. The placement of chip "soc1" obtained with the quadratic 0/1-programming approach is shown in Figure 2.

2.3. Routing. Let us now focus on the next step in the hierarchical design process of electronic circuits, the routing problem. We assume that all cells (logic and I/O-cells) are placed, and a list of nets is given. Each net is viewed as a set of points, where each point corresponds to a terminal (also called pin) of a cell. The routing problem deals with connecting the pins of the nets by wires. In fact, the problem is quite complicated, because certain given design rules must be taken into account and an objective function, such as the total wire length, must be minimized. Since the routing problem is $\mathcal{NP}$-hard and usually of extremely large scale, the problem is often decomposed into special cases that can then be handled more easily. A large variety of such special cases is considered in practice and in the literature.

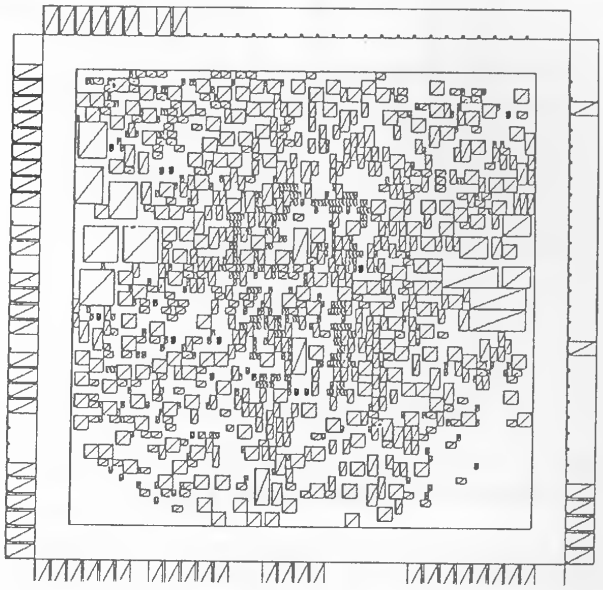

Figure 2: Cell placement in sea-of-cells technology

We want to model the routing problem as a Steiner tree packing problem in a graph. One way of introducing a graph $G = (V, E)$ here is to define nodes for subareas of the whole routing area, and to link nodes that represent adjacent subareas by an edge. In addition, we assign edge capacities and edge weights to each of the edges. The nets are represented in this graph by subsets of the node set. Let a graph $G = (V, E)$ and a node set $T \subseteq V$ be given. We call an edge set $S \subseteq E$ a Steiner tree in G for T, if the graph $(V(S), S)$ (consisting of the edge set S and all nodes $V(S)$ that appear as endnodes of edges in S) contains an $[s,t]$-path for each pair of nodes $s,t \in T$. With this definition, the routing problem can be stated formally as follows.

(1.2) The Routing Problem

Given: A graph $G = (V, E)$ with edge capacities $c_e \in \mathbb{N}$ and edge weights $w_e \in \mathbb{N}$ for all $e \in E$ and a netlist $\mathcal{N} = \{T_1, \ldots, T_N\}$, $T_k \subseteq V, k = 1, \ldots, N$.

Problem: Find edge sets $S_1, \ldots, S_N \subseteq E$ such that

(i) S_k *is a tree in* G *for* T_k, $k = 1, \ldots, N$,

(ii) $|\{k | e \in S_k\}| \leq c_e$ *for all* $e \in E$, *and*

(iii) $\sum_{k=1}^{N} \sum_{e \in S_k} w_e$ *is minimal.*

We call an N-tuple of edge sets $(S_1, \ldots, S_N)$ that satisfies (i) and (ii) of (1.2) a packing of Steiner trees. Problem (1.2) can then be designated as the Steiner tree packing problem. It is not surprising that problem (1.2) is $\mathcal{NP}$-hard. Indeed, it includes several $\mathcal{NP}$-hard problems as special cases; see, for example, [GaJ77], [Ka72], [KrL84], [KPS90].

Figure 3 indicates a 5×3 underlying grid graph (the grid is not drawn), in which each edge has capacity 1, and three nets, each consisting of three pins. The three Steiner trees forming the nets are drawn with solid, dashed and dotted lines, respectively.

Our approach to the Steiner tree packing problem is to consider it from a polyhedral point of view and to use linear programming techniques. We define, for a given instance $(G, \mathcal{N}, c, w)$, a polyhedron P whose vertices correspond uniquely to the packings of Steiner trees for that instance. Thus, the Steiner tree packing problem reduces to the problem of minimizing the objective function $w^T x$ over the polyhedron P.

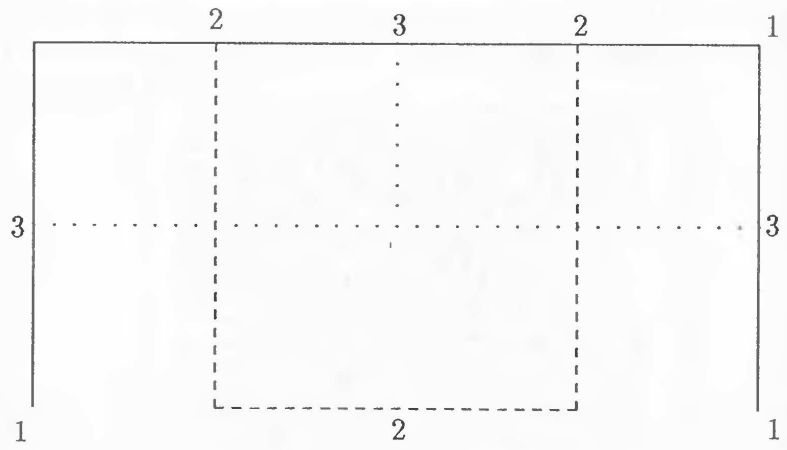

Figure 3: A packing of three Steiner trees

What we need for the application of linear programming techniques is a complete or at least "good" inequality description of the polyhedron P. In [M92] a large number of valid inequalities for P is described. In fact, many of these inequalities also define facets of P if the underlying graph G is complete and the terminal sets are disjoint. The machinery involved in describing these inequalities is rather complicated and thus we omit detailed statements of these results. These inequalities form the basis for the development of cutting plane algorithms. We have implemented a cutting plane algorithm for special instances of the Steiner tree packing problem, when the graph G is a rectangular grid graph and the terminals of the nets lie on the outer face of G. In VLSI design, these instances are known as switchbox routing problems. We have tested our algorithms on many benchmark problems from the literature. For a detailed documentation of these results see [M92].

At its current stage this approach is not yet able to handle instances as large as those that can be treated with the standard routing heuristics used presently in industry. The remarkable feature of this approach, however, is that very good (provable) lower bounds on the total wiring length can be computed, a result that none of the other currently used approaches can provide. In fact, for the (small) problem sizes considered so far, quality guarantees of less than 1% or even provable optimality have been achieved.

2.4. Via Minimization. We continue with our chip design example and assume that the placement and the routing phase have been completed successfully. Usually routing algorithms disregard the requirement that two different nets may not cross. Routings with such defects are referred to as transient. A transient routing of eigth nets is shown in Figure 4. The reason for ignoring net crossings is that the physical routing can be done on two or more layers of a chip. In case two nets cross in a transient routing, a wire segment of one of the nets can be assigned to a different layer so that no physical wire crossing occurs. The connections between the wire segments of a net on different layers are provided by so-called vias. In chip manufacturing the vias are produced by means of a delicate chemical process. Many vias on a chip increase the probability that, at the end of the production process, the chip will have a short or not work correctly. Moreover, vias use considerably more space than wires and thus a large number of vias may lead to an increase of the chip size.

Virtually the same problem occurs in the design of printed circuit boards. Here, the vias are produced by mechanically drilling a hole through the board (lasers are also used). A danger is that boards may crack in the drilling phase, another reason for minimizing the number of vias. Furthermore, the drilling process is quite time-consuming (see Section 3.2), and hence reducing the number of vias leads to a reduction in the overall production time.

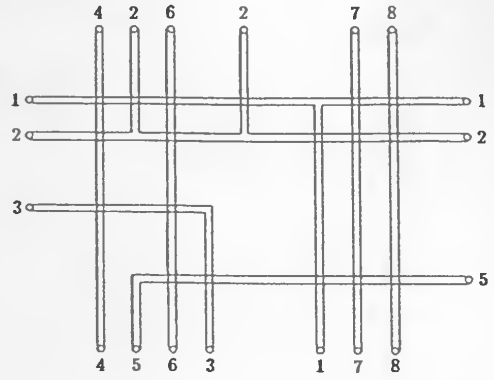

Figure 4: A transient routing

Due to these facts, it is desirable to assign the wires to layers of a chip or a printed circuit board in such a way that no two wires cross and the number of vias is minimum. We call this task the via minimization or layer assignment problem.

For the case of two layers (assuming that the transient routing contains no k-way junctions for $k \geq 4$) [Pi84] and [CKC83] have shown independently that this problem is solvable in polynomial time through reduction to a max-cut problem in a planar graph. However, these reductions do not cover certain side constraints required in practice. In many cases one of the two layers is preferred and pins are preassigned to some layer. [BaGJR88] pointed out that Pinter's reduction can be generalized to the via minimization problem subject to layer preference and pin preassignments. However, the max-cut problem that results from this reduction is $\mathcal{NP}$-hard. The transformation is elegant but technically rather involved and will not be explained here. The reader is invited to find the minimum number of vias for the problem shown in Figure 4 before looking at Figure 5 where an optimum solution needing four vias is displayed. The vias are indicated by the little open squares (where the nets change color, i. e., lay

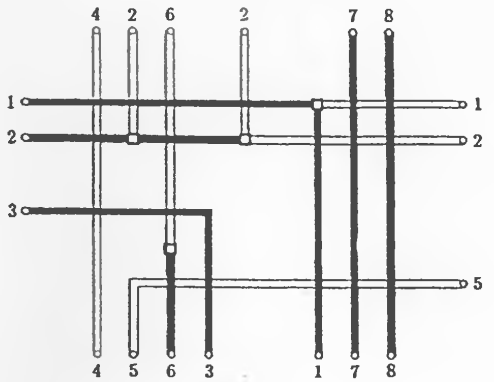

Figure 5: A via-minimal layer assignment

We have developed both simple and elaborate heuristics for the via minimization problem and an exact cutting plane algorithm that is based on polyhedral investigations of the max-cut problem. These algorithms are described in [BaGJR88] and [GJR89].

Tables 2 and 3 report the success of this approach. The symbols $C1, \ldots, C6$ denote six chips from industry. For each chip, the via minimization problem comes in two versions. In one case, no pin was preassigned, and in the other, certain pins were required to be on one of the layers of the chip. Of course, this latter requirement will generally result in more vias. Table 2 reports on the results of the via optimization without pin preassignments, Table 3 shows those with pin preassignments. The row labels have the following meaning:

#	nodes	$\widehat{=}$	number of nodes of the so-called reduced layout graph,
#	edges	$\widehat{=}$	number of edges of the reduced layout graph.

These two numbers indicate the sizes of the assoiated max-cut problems.

#	vias original	$\widehat{=}$	number of vias in the (original) industry design,
#	vias heuristic	$\widehat{=}$	number of vias in our best heuristic solution,
#	vias optimal	$\widehat{=}$	minimum number of vias for this chip found by the cutting plane algorithm,
	improvement	$\widehat{=}$	improvement in percent ((# vias original − # vias optimal) / # vias original).

	C 1	C 2	C 3	C 4	C 5	C 6
# nodes	827	979	1326	1201	1365	924
# edges	1445	1775	2480	2606	2234	1740
# vias original	421	434	683	650	782	630
# vias heuristic	272	347	513	475	610	525
# vias optimal	264	334	500	467	608	516
improvement	37.29%	23.04%	26.79%	28.15%	22.25%	18.10%

Table 2: Via minimization without pin preassignment

	C 1	C 2	C 3	C 4	C 5	C 6
# nodes	828	980	1327	1202	1366	925
# edges	1749	2102	2844	2915	2557	2008
# vias original	421	434	683	650	782	630
# vias heuristic	302	376	563	504	645	585
# vias optimal	302	376	561	482	643	585
improvement	28.27%	13.36%	17.86%	25.85%	17.77%	7.14%

Table 3: Via minimization with pin preassignment

These results were achieved with algorithms designed and implemented by M. Jünger and G. Reinelt. The heuristics approximately solve these via minimization problems within a few seconds, while the cutting plane algorithm (yielding a provable optimum solution) only requires a few minutes on a workstation. The results displayed in these tables show that this optimization approach results in a considerable reduction of the number of vias compared to the industry solutions and thus leads to more favorable chip designs. Moreover, optimum solutions could easily be computed and the heuristics (these are special purpose methods taking the special features of this problem into account) did extremely well.

This discussion of some features of the design phase of a manufacturing process was meant to show that even seemingly minor details such as via minimization lead to very interesting mathematical problems, and that their solution may have a significant impact on other aspects of production. The mathematical theory on topics of this type, whether combinatorial or continuous, is not very well-developed. Only few isolated cases have been analyzed. There is still considerable room for further exciting developments.

3. Control of Machines. Let us now suppose that the design phase has been completed. The product designers have determined how to subdivide the production process into various tasks and how the different parts of a product must be manufactured.

In this section we will consider the manufacturing process on a single machine. The additional problems that arise when taking transport and sequencing into account will be discussed in the next section. In the typical situation, an object enters a machine, and the machine, controlled by a computer program, performs various operations on that object. The questions that arise are: how to do the jobs as fast as possible, and how to schedule the jobs such that idle times, say those induced by retooling or positioning moves, are as short as possible.

We have noticed in our work, dealing with situations like this, that slightly different technical devices, a few details in machine capabilities, and decisions of the production managers may lead to quite different mathematical models. In fact, whole new ranges of problems do arise in this area. We will exemplify these statements by considering two problems in printed circuit board production. We follow the paper [GJR91] in our presentation.

3.1. The Plotting Problem. Complex printed circuit boards are usually produced by a photochemical process. For each layer of the board, the pattern of wires and contacts is produced by covering the board with light sensitive material, exposing this material to light, etching, cleaning, etc. The process is similar to the usual production of photographs. The structures that later should appear on the board are first "drawn" on a mask (a negative) that is placed between the board and the light source, so that certain parts of the board are not exposed to light. These unexposed areas are to form the conductors, pads and contacts of the layer. The question we address here is the generation of the masks. The masks are made of glass and the patterns on the glass are generated optically either using ultraviolet light or laser beams. In our case, a photo plotter is used for the mask production. Figure 6 shows an example of one layer of a printed circuit board. It is one of our test cases.

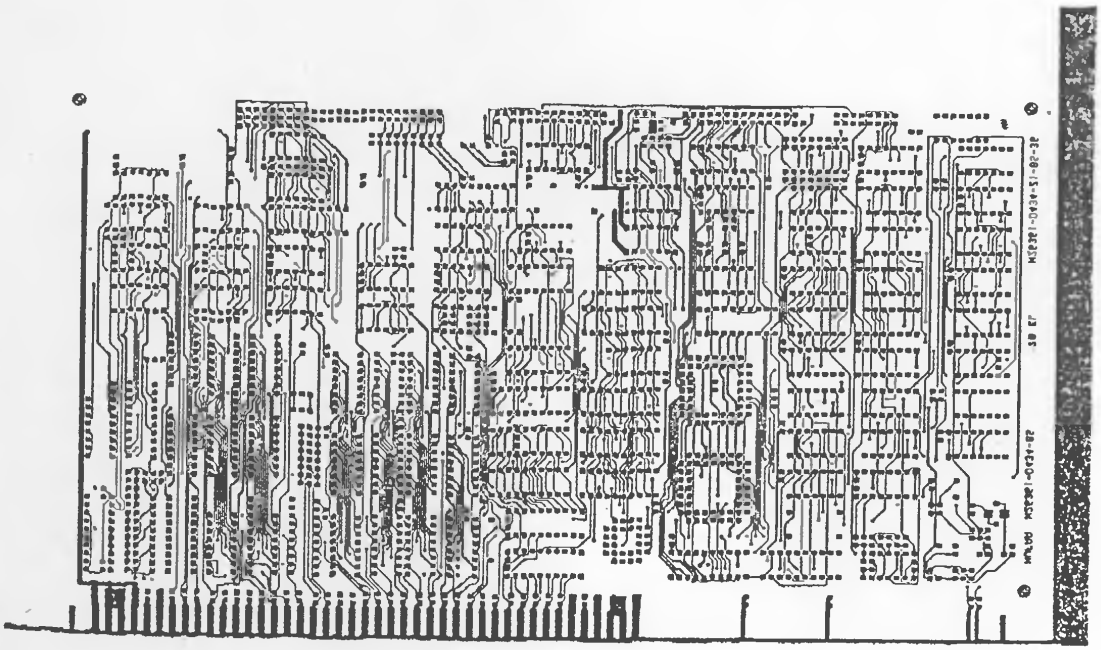

Figure 6: A layer of a printed circuit board

The photo plotter works as follows. It has two modes, a "drawing mode" for plotting lines and a "flashing mode" for plotting points. As one can see from Figure 6, points may be of various sizes

and shapes, and lines may be of various widths. Before plotting, an aperture is chosen that produces the required shape or width.

Points are plotted by moving the light source to certain coordinates on the board, choosing the aperture, and flashing the light. Lines are plotted by moving the head to one end of the line, choosing the aperture, opening the shutter, moving along the line with the open shutter and closing the shutter at the end of the (not necessarily straight) line.

The above is nothing but a basic description of the principle of the plotting process. Plotting machines are offered in various mechanical forms. In some cases, only the head of the light source is moved, in some cases only the compound table, and in others the head moves in one direction and the table in the other. For our purposes, the actual mechanics are irrelevant. We only need to know the time it takes to move from one point to another. For the sake of exposition we will assume a model in which only the head moves.

There is, given a pattern, nothing to be done about the time needed for drawing and flashing. This process requires a certain fixed time depending on the plotter characteristics. What can be optimized is the time needed for positioning head moves, i. e., moves of the head without drawing. We will now describe the mathematical modelling of the plotting process in some detail. Depending on technological side constraints, there are several options of modelling this problem mathematically. We discuss two examples that came up in our application.

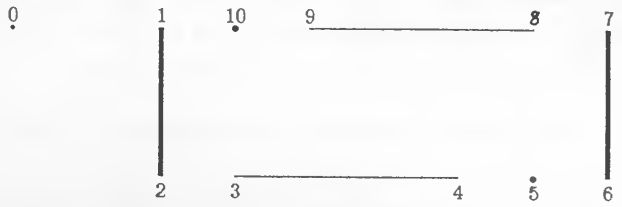

Figure 7: First plotting problem

Consider the line and point plotting problem shown in Figure 7. It seems obvious how to proceed. We begin at the starting position 0, move to point 1, choose the "drawing mode", and switch the aperture to "drawing thick lines". Then, while moving, we draw the line from 1 to 2, move to point 3 and change the aperture to "thin drawing", draw the line from 3 to 4, move to point 5 and change to the "point flashing mode", etc. The trouble here is that during every positioning move the aperture has to be changed. The head is (in many cases) a mechanically delicate device that — after a certain number of aperture changes — has to be readjusted or substituted. This is a costly procedure. Therefore it may be wise to proceed as follows. One first chooses an aperture, plots everything that can be plotted with this aperture, changes the aperture, etc. This approach decomposes the problem into various plotting subproblems and adds a new problem, namely that of an optimal sequence of apertures. For Figure 7, an optimum solution in this case would be as follows. We first choose the thick drawing aperture, go from 0 to 1, draw the line from 1 to 2, move to point 6, draw the line from 6 to 7, change to the thin drawing aperture and move to 8, draw the line from 8 to 9, move to 3, draw the line from 3 to 4, change to the flashing mode and move to 5, flash, move to 10, flash, and return to 0.

Clearly, the second choice produces longer (and sometimes substantially longer) positioning head moves and — provided that aperture changes do not require more time than the moves — longer machine running times.

One has three choices in practice. Either one ignores aperture changes and uses the first option, or one decides to decompose the plotting problem into subproblems (one for each mode and each aperture), or one mixes the two by penalizing those positioning moves where also aperture changes occur. In each case, the person responsible for mask plotting has to decide — based on his knowledge of the technical characteristics of the machine — whether or not aperture changes are considered crucial operations that have to be kept at a minimum and which of the three options should be used.

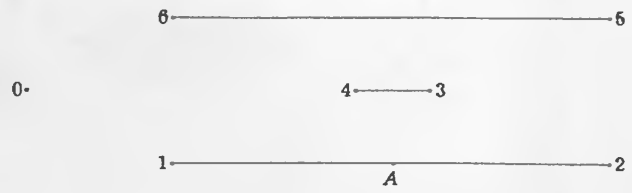

Figure 8: Second plotting problem

There is another option that may produce shorter positioning head moves: preemption. Consider Figure 8, where all three lines are drawn with the same aperture. An optimum solution here is to move from 0 to 1, draw the line from 1 to 2, move to 3, draw the line from 3 to 4, move to 5, draw the line from 5 to 6 and return to 0. We have so far made the assumption, without explicitly stating it, that whenever the plotter has started drawing a line it continues drawing until the other endpoint of the line is reached. If we allow interruptions (technically often called preemptions) we could do better. For instance, in Figure 8 we could draw the line from 1 to 2 only until point A is reached then we move to point 3, draw the line from 3 to 4, return to point A and continue drawing from A to 2, etc. This choice would produce shorter positioning head moves than the "optimum solution" sketched above. So the question is whether preemptions should be allowed or not. There is no general answer. In each individual case one has to make a decision based on the particular conditions.

In our case the decision was to execute as few aperture changes as possible and not to allow preemptions. Therefore, the following combinatorial optimization problems arose.

(3.1) Point Flashing Subproblem.
Given an aperture, select all points that are flashed with this aperture and determine a shortest Hamiltonian path through these points.

(3.2) Line Drawing Subproblem.
Given an aperture, select all lines that have to be plotted with this aperture and determine a sequence of these lines such that the total distance travelled by positioning moves is as short as possible.

(3.3) Aperture Sequencing Subproblem.
Determine a sequence of apertures and, for each aperture, a starting point and a terminal point for the point flashing or line drawing process such that the total time for positioning moves (with and without aperture changes) is as short as possible.

The aperture sequencing problem is by far the most complicated and we do not have any idea of how to solve it. Fortunately, in our practical problems, it was of almost no importance. We treated it by using a simple heuristic.

The point flashing problem is, after an easy transformation, equivalent to a symmetric travelling salesman problem (TSP). The line drawing problem turns out be a so-called rural postman problem. A rich mathematical theory and a substantial algorithmic toolbox exist for the TSP, see [LLRS85], whereas the rural postman problem has basically been neglected in the literature up to now.

To solve the plotting problem in practice we had to meet additional computational side constraints. The organization of the production process required the problem to be solved almost in real time, i. e., for each instance, five minutes of computation time were available on a 3 MIPS machine for the solution of the point flashing, the line drawing and the aperture sequencing problem. Within this time limit good solutions had to be produced. [GJR91] and [R92] describe some of the heuristics that were designed and implemented to satisfy these constraints.

The methodology developed also involved the fast solution of problems from computational geometry, such as computing Voronoi diagrams, Delaunay triangulations or convex hulls. These techniques were utilized to reduce problem sizes in order to satisfy the running time requirements.

Our codes were tested on real-world masks by the engineers at Siemens. The solutions of the industry heuristics were compared with our solutions by measuring the running times on the real plotting machines. The savings turned out to be tremendous. We had to solve line drawing problems with up to 38,621 lines and flashing problems with up to 2,496 flashes. The running time reductions for the positioning moves using our fastest heuristics ranged from 14% in the worst case up to 83% in the best case. Further improvements of an approximately additional 10% could be achieved by more elaborate and time-consuming heuristics. These savings resulted in capacity increases of the plotting machine in the range of 5% to 35%, see [GJR91] for more detail.

3.2. The Drilling Problem. In Section 1.4 we introduced the via minimization problem for chips. Its analogue for printed circuit boards (PCBs) has direct consequences for the printed circuit board production process: the fewer the vias, the shorter the production time.

The practical problem arising in this application is the following. To connect a conductor on one layer with a conductor on another layer or to position (in a later stage of the PCB production) the pins of IC's, holes have to be drilled through the board. The holes may be of different diameters. To drill two holes of different diameters consecutively, the head of the machine must move to a tool box and change the drilling equipment. This is quite time-consuming. Thus, it is clear at the outset that one has to choose some diameter, drill all holes of the same diameter, change the drill, drill the holes of the next diameter, etc.

There is no tool changeover problem here since, in any case, after loading of the boards the machine head is at the initial position (where the tool box is) and after having drilled all holes of one diameter it has to return to the initial position to pick up the new drill. Thus, our drilling problem can be viewed as a sequence of symmetric travelling salesman problems, one for each diameter resp. drill, where the "cities" are the initial position and the set of all holes that can be drilled with one and the same drill. The "distance" between two cities is the time it takes to move the head from one position to the other. The aim here again is to minimize the travel time for the head of the machine. The (quite substantial) time needed to drill a hole cannot be influenced at all. This is a fixed production time. As for the plotting application described before, severe bounds on the running time were required for our heuristic.

The problem sizes that came up in this particular application ranged from about 500 to about 2500 "cities". For the real test cases we obtained from Siemens Nixdorf the following was achieved. All instances were solved in less than 2 minutes on a 3 MIPS computer by our heuristics. The time needed for positioning moves by our solutions was up to 55% lower than the respective time of the solution used in industry. Since the drilling time is quite substantial this does not mean a halfing of the production time. The reduction of the positioning moves resulted in a decrease of the total production time of 5% to 20% on the average. The results were considered quite remarkable by the engineers responsible for the CNC-machines and it was decided to incorporate our codes into the software controlling the machines in order to speed up production.

The dots on Figures 9 and 10 indicate the holes of a typical drilling problem for printed circuit boards. Figure 9 shows the positioning moves of the original industry solution, Figure 10 the positioning moves of our solution. Our tour is 51.83% shorter.

A detailed account of these results is given in [GJR91]. A thorough description of the design and analysis of the heuristics is contained in [R92]. A side effect of this investigation was the compilation of a large number of real-world travelling salesman problems. A library, called TSPLIB, that includes the instances of this study and many other TSP instances was set up. It can be accessed by e-mail. A description of this library and its use can be found in [R91].

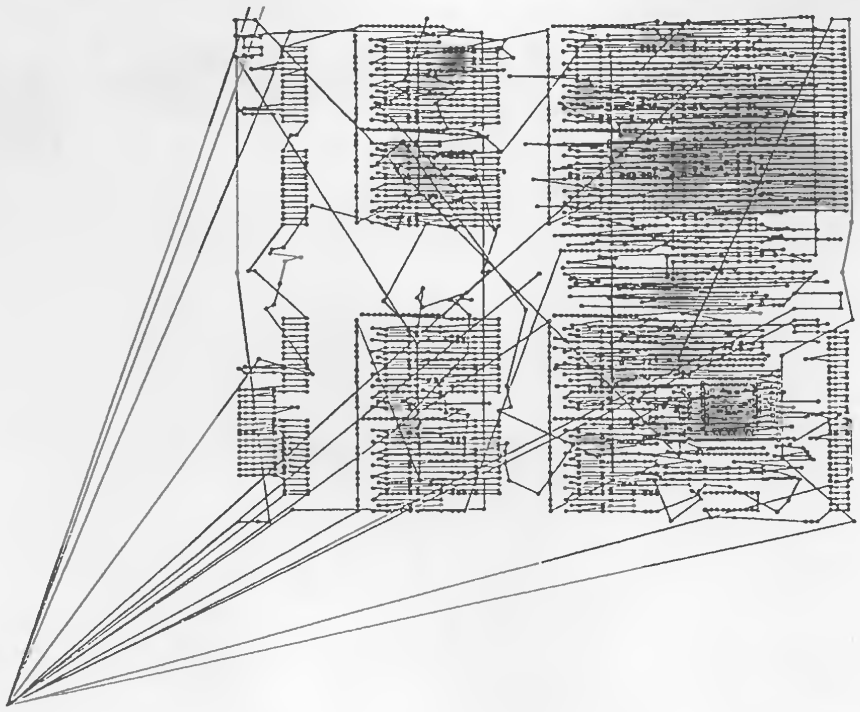

Figure 9: Positioning moves, original solution

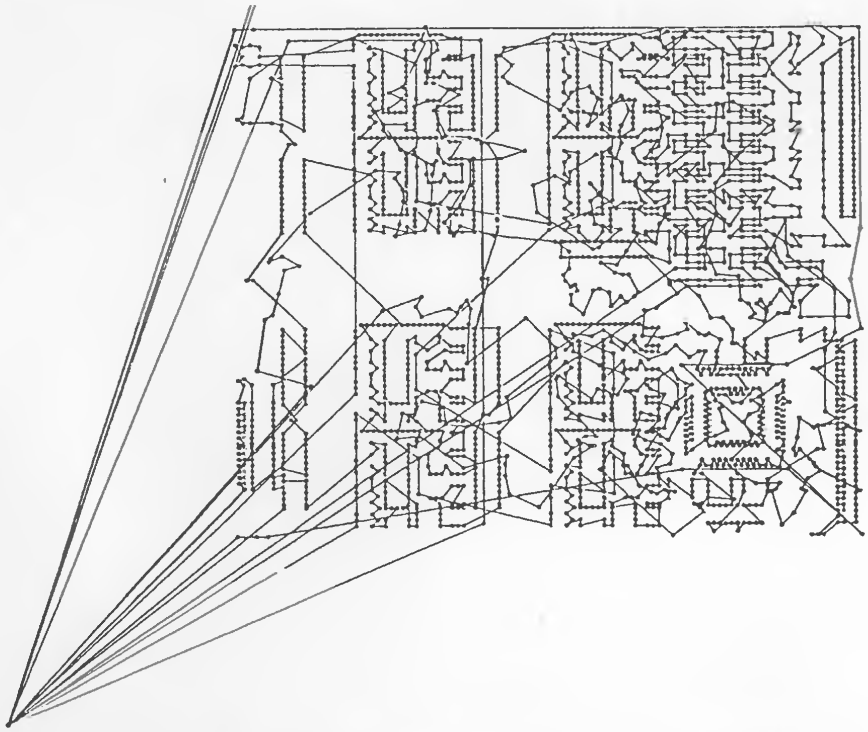

Figure 10: Positioning moves, "optimized" solution

4. Assembling the Parts. Having fine tuned the individual machines that produce the parts of a product and having improved their performance by using optimization methods as described in the previous section, we now focus on the task of putting the parts together to obtain the final product. A main problem here is to design the assembly process and to schedule the jobs to be executed in such a way that the production cost is small and the output per time unit high. These global goals are often replaced by (occasionally) more operational or measurable aims such as: all machines are utilized to the full, the work loads of the factory branches are balanced, stocks are kept at a minimum level under the condition that quick changes in the production volume and pattern are still possible, etc. No doubt, this is a tremendously difficult task.

Mathematics is, at present, rarely used in the design phase of the assembly process, i. e., when the machines are chosen, the factory floor is layed out, the transport systems are selected and dimensioned, etc. However, once these basic decisions are made, simulation systems are often employed that are based on mathematical tools. These simulation systems are used to get a feeling for the flow of goods, the throughput, the possible bottlenecks of the system, etc. Simulation methods have become indispensable and valued tools for the design of large production systems. Nevertheless, there is considerable room for the use of further and more sophisticated mathematical tools in this design process.

Mathematicians are, in general, only asked for their expertise in cases where the system does not work as expected, or when the work load of the system has changed considerably and ways are sought to run the system efficiently without making significant technical changes, or when certain components of the system turn out to be severe bottlenecks and reorganization of work seems feasible for curing this disease.

I will explain these aspects by means of two projects that were jointly carried out with Siemens Nixdorf, Augsburg.

4.1. Flexible Printed Circuit Board Production. The Siemens Nixdorf, Werk für Systeme in Augsburg produces, among other items, all main frame computers of the company. Important parts of computers and other electronic products are printed circuit boards (PCBs). Printed circuit board production is a division that manufactures PCBs for Siemens Nixdorf itself and other outside customers.

The whole PCB production process consists of various stages. We have already met two such phases in the previous section (plotting and drilling). We now focus on the final assembly. The PCB frame is finished and the task is to place various electronic components on the PCBs. There are components of different types and mechanical properties. For instance, some components are glued onto the board, some have "little legs" that have to be positioned into certain holes (drilled in a previous stage) and that are soldered in a later phase of the PCB production, etc.

In general terms, the flexible PCB assembly line (called FALKE at Siemens Nixdorf) has the following properties. At the head of the system, the so-called HEAD-cell, the boards are put onto carriers and fed into the system. There are automatic conveyor belts for the transport of the carriers. The carriers may enter some or all of six "cells", where each cell consists of a series of CNC-machines each carrying out certain feeding or other operations. There are buffers in front of each machine that can hold up to 25 carriers. Except for the initial system feeding and certain manual operations for the insertion of special parts, the entire process including transport, component feeding, etc. is controlled by a computer system and is fully automated. The whole assembly line is an impressive, automatic flexible manufacturing system.

The problem with the FALKE line was the following. Due to changes in the design of PCBs over time the distribution of work among the machines of the system was somewhat unbalanced. Certain machines were constantly working, while others were frequently idle. Although some machines had a higher total workload than others, it was not always the same machines that appeared to be the production bottleneck. So the question was: Is it possible to increase the throughput of the system without making any changes in the general control software or any technical changes? We were not allowed, for this type of investigation, to retool the machines, to change the order of jobs to be executed on individual PCB's, or to modify the control software for the conveyor system, the feeders or buffers. Such changes would simply have incurred costs and dangers too high for the reliability of the assembly line. The only control parameter left to be influenced by optimization was the feeding

of the system.

At the time of analysis of the FALKE system about 1500 to 2000 circuit boards were produced per day consisting of about 20 to 40 different types. In total, about 500 different PCB types could be automatically assembled on the system, without changing the machine settings.

The managers of the FALKE system learn about the planned production volume for a day about one to two days ahead of time. This gives enough lead time to arrange for the shipment of required boards and components. Having no other option, we proposed using this time for also finding a sequence of the carriers such that the completion time of the whole set of PCBs is minimized.

An immediate question arises. Given a sequence of PCBs, how can one compute the time it needs to finish the work? Of course, we assume here that no artificial idling occurs, i.e., that all machines work and that each job is done at the earliest possible time, etc.

In fact, we were not able to come up with an analytic formula for the completion time. The difficulty is mainly due to long distance interdependencies that are hard to model analytically. For example, some carriers do not enter certain cells or machines (an average PCB is processed on about two thirds of the machines), carriers may overtake each other, a carrier that blocks another at some machine may be blocked in return by this other one at a later stage.

So we decided to design, very carefully, a special purpose simulation model of the FALKE line by measuring all transport times, feeding times and obeying all rules and side constraints of the system. Programming and validating this simulation tool took about 9 month of two persons' work. In the end, we had a simulation tool that could simulate an eight hour shift of the FALKE line in about 5 minutes on a PC. Two weeks of real production were recorded and the parameters of our simulation tool were adapted in such a way that it faithfully reproduced the real production in our computer model of the production line.

We then invented a number of heuristics to improve the sequencing of the jobs. A typical iterative improvement heuristic starts with some sequence of jobs and changes this sequence using some myopic optimization rule. Then it computes the completion time of the new sequence. If it is better (or only slightly worse), it takes the new sequence and applies the same rule. If after a number of applications of this rule, no improvements have been made, other sequence changing techniques are applied until no significant progress is visible. The trouble here is that evaluating the new sequence takes some minutes of running time, and thus, not too many sequences can be tested. To achieve acceptable running times for the heuristics, a fast lower bounding procedure for the completion time was invented. This lower bound was used as the objective function value for the heuristic. The real completion times were only occasionally computed by means of the simulation model.

After having tuned these heuristics, we ran them on the production data of the two weeks available to us. We improved upon the completion time, compared to the runs of the real system, in the range of 3.3% to 12.7%. On the average the completion time could be reduced by 6.7%.

This improvement was significantly less than the management had expected. But we could also show that the expectations were much too high. By means of our lower bounding procedure we could prove that there was not as much room for improvement as thought. We could show that, for the days considered, a maximum total speed up of about 20% might be possible. (We actually believe that this is an overestimation.) So, on the average, about 13% of further completion time reduction might be possible at most.

What turned out to be the most significant drawback of our heuristic solutions was the following property. Since we know that the HEAD-cell is manually operated we expected the person in charge to make a few sequencing errors. Thus we randomly perturbed our "optimized" carrier sequences a little and noticed that a few changes could result in the loss of all the gains that we had worked for so hard, i. e., our solutions turned out to be quite unstable. Due to this fact and the not so significant improvements, the company decided not to change the present system.

In some sense, our efforts were in vain. Techniques of combinatorial optimization did not lead here to significant improvements. What we learned is that we do not have a good understanding of such complex and complicatedly interlinked systems like the FALKE line. We do not seem to have the right tools yet to control these systems efficiently. In fact, what is really needed is an on-line control system that makes adjustments of the job schedule whenever interrupts, machine breakdowns, conveyor stops, etc. occur. I think we are still far away from a good mathematical

understanding of such systems. In fact, it may be reasonable to consider designing less complex production systems, production systems that are efficiently controllable. After all, flexibility does not help if it can't be used.

This work has — to a large extent — been carried out by Petra Bauer together with Siemens Nixdorf engineers and is documented in [Bau90]. A paper describing in detail the findings surveyed above will appear in the near future.

4.2. Optimizing a PC Factory. The Werk für Arbeitsplatzsysteme, Augsburg is the plant of Siemens Nixdorf where all PCs (and some other related products) are manufactured. In the fall of 1989, we started a joint project aimed at looking for possibilities for production improvements. The goal, in the long run, was to design and implement a software package that optimizes the material flow through the factory and allows for the production of the required PCs, monitors, keyboards, etc. in shortest possible time with high flexibility. This is obviously a very ambitious goal, and we are aware of the fact that we are unlikely to achieve it. The point, though, is to keep the real goal in mind whenever optimization and simulation tools are designed for particular components and branches of the factory. Clearly, the output and efficiency of the whole factory is what really matters and not the speed of a few machines. Figure 11 shows

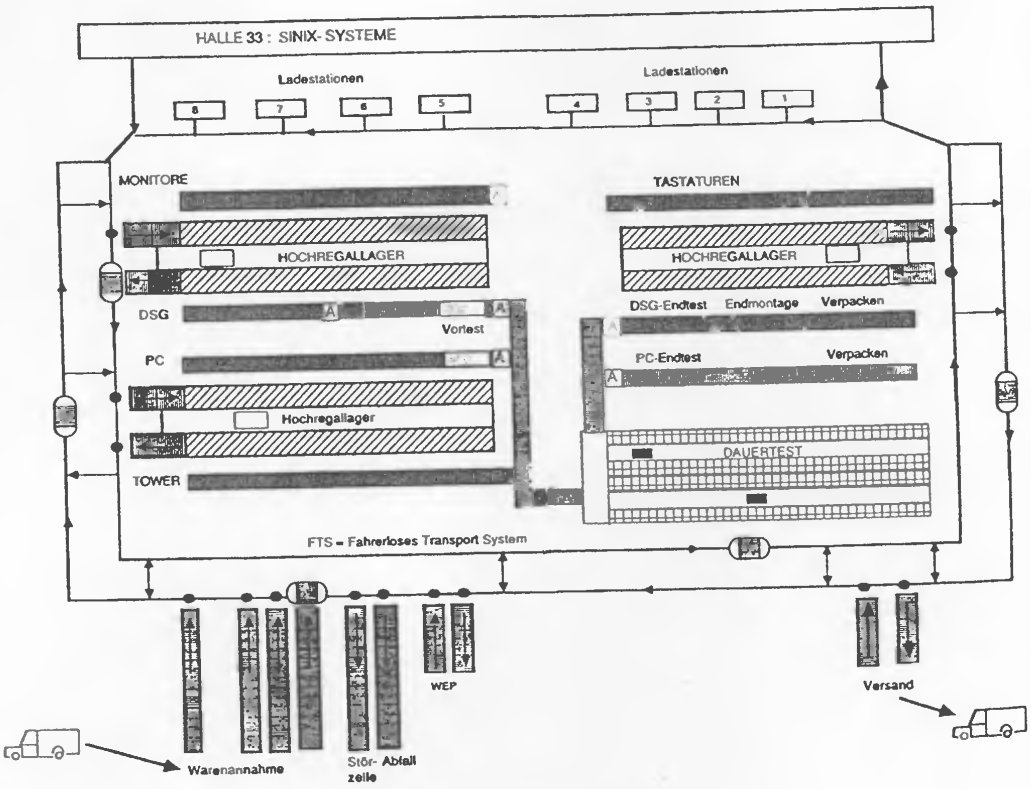

Figure 11: Sketch of the factory floor

a symbolic sketch of one hall of the factory. We give now a rough description of the operations involved in the assembly.

There is a receiving area, where all parts necessary for the production are supplied by trucks. Among these are, for instance, the printed circuit boards mentioned in the previous section. These are manufactured in a factory about one km away. The parts arrive in or are loaded into three types of boxes and given a bar code identifying them. These boxes are moved by a conveyor to several loading points where automatic driverless vehicles pick up the boxes and transport them to their destination. The vehicles are controlled by a software system that decides when which vehicle executes which transportation job.

A typical destination of such a vehicle is the Input/Output buffer of one of the many automatic storage systems. When the driverless vehicle arrives, a mechanical device moves the box on the I/O-buffer and signals to the control of the storage system that the box has to be transported by the stacker crane to its storage location.

The PC and other assembly lines are located on each side of the storage systems such that all parts needed at a certain point on the assembly line are delivered upon request automatically to the desired location by the stacker crane. The control system of the stacker crane decides where to put incoming boxes, when to remove empty boxes and in what sequence all current orders are processed. All this is done on-line.

One of the assembly lines is fully automated, robots do all of the jobs. At the other lines, manual work is involved and the speed of production can be adjusted by varying the number of workers at the lines.

Once a product like a PC is finished, a conveyor moves it to a "hot line" where it is tested by means of software for about twenty four hours. There is a scheduling and a capacity problem here. The control system of the hot line has to determine where to locate a PC, how to perform the tests and whether it pays to relocate the PCs at night in order to be in a position to do all the necessary transportation work fast in case of high traffic volume.

The PCs that have failed the test are moved to a repair area and may reenter the hot line later. The other PCs will run through further manual tests before they are packed and moved to the distribution center. Storing all the products (including manuals, software, etc.), combining these parts to existing orders, and combining orders to reasonable truck loads is another complicated and highly automized process that we do not want to describe here.

In a project that is mainly executed by Norbert Ascheuer and Atef Abdel Hamid with the help of some mathematics students and a support team of Siemens Nixdorf, we have designed and implemented a simulation model that covers, at present, the receiving area, the transportation system (i.e., the driverless vehicles), and the automatic storage systems. This simulation model has been validated by comparing its performance with the data of several weeks of real production. The various models have been validated individually and in joint operation.

The role of the simulation system is twofold. Its use as part of a planning system where decisions about the work load of the day and the distribution of the jobs are made is currently being tested. The simulation system is employed here to check not only whether all parts needed for the planned production volume are available but also to check whether the work force is sufficient or whether bottlenecks in the transportation system and the parts supply will arise.

The main objective of designing the simulation models, however, was to analyse the performance of optimization approaches and compare them with the existing system. To convince the engineers that our analysis is sound and that our system speedup predictions will show up in practice, validation was indispensible. Since outlining the whole work would be too space-consuming, I will concentrate here on the automatic storage system and the stacker crane operation.

The storage systems that are in use at the Siemens Nixdorf Werk für Arbeitsplatzsysteme are huge frames that hold all the material needed at the assembly lines next to them. The initial planning question is how to choose the size of the storage frame to make sure that it will be large enough to hold all the items required in the production. At this point in time, it is also decided which type of transportation and which storage boxes are going to be utilized. In our case, there are three types of boxes of different sizes. Each box type needs differently shaped storage locations.

We have mathematically modelled various optimization questions associated with the design and management of this storage system. A first problem is how to subdivide the frame into storage units such that the expected storage volume of different boxes is sufficient and all boxes are as close as possible to their output buffer, i.e., the problem is to determine locations such that all needed

boxes fit and the expected moves of the stacker crane for box transportation are as short as possible. This leads to general integer programming approaches or set packing models. We are not sure yet whether or not our models achieve what is needed in practice. Further modelling iterations may be necessary.

Once the locations are fixed and once the system is in operation we have to decide, every time a new box arrives, where to locate it among the available empty storage locations such that the stacker crane moves are as short as possible. This requires on-line decisions that should take also into account which further boxes are already in the material flow pipeline and are likely to arrive soon. It turns out that, for two of the three box types, this problem can be formulated as an (ordinary) rectangular assignment problem. Typical sizes of such assignment problems are 20×800 and can be handled with the assignment code of Kleinschmidt, see [AKP89], in a few seconds on a PC. For one type of small boxes the associated model is a special case of the generalized assignment problem. Unfortunately, it is $\mathcal{NP}$-hard. We have developed fast heuristics, e.g., based on matching techniques, and a cutting plane algorithm for its solution. The heuristics provide satisfactory approximate solutions very quickly.

For a detailed description of all these aspects of the storage system and the theoretical and algorithmical investigations of the generalized assignment problem, see [AH92].

Let us now turn to the moves of the stacker crane. By analysing all the technical details, measuring running times of the crane, reaction times of the systems, times for moving boxes, etc. Norbert Ascheuer has implemented a simulation system that models the behaviour of the stacker crane and has compared it with the real moves. Table 4 shows the validation results for all operations of one week. One job consists of the positioning move, the pick up of a box, the move with the box, and the delivery of the box. The time needed for moving and handling one box on day 1, for instance, was 76.27 sec. on the average, see column ØRL of Table 4. This time was obtained by measuring all operations of the day and averaging them out. The simulation model produced an average job length of 76.83 sec. for this day (column ØSIM) resulting in 0.73% deviation (column %DEV). Deviations are bound to occur since always some interruptions and manual interactions occur that are not taken into account by the simulation model; see column ØDEV for average and column max DEV for maximun deviations that occured throughout a day. The time for executing the simulation model for a whole day was about 12 seconds on a PC.

	ØRL	ØSIM	ØDEV	max DEV	% DEV	Time
1	76.27	76.83	2.42	20	0.73	11.47
2	75.45	76.34	1.96	11	1.17	11.60
3	78.81	78.95	2.01	9	0.18	11.17
4	76.19	76.58	2.09	9	0.51	11.08
5	77.11	76.73	2.06	20	0.49	12.30

Table 4: Validation of the simulation model

The results of the validation process, which was more elaborate than indicated here, were considered very satisfactory by our industry partners and it was decided to accept proposals that are based on the use of this simulation model.

At any point in time, the stacker crane control system has to decide which of the jobs to execute next. In the old system this was done by a priority based FIFO rule. We proposed to replace this heuristic by solving the following problem. Whenever a set of jobs is given or modified, schedule the jobs in such a way that the unloaded travel time (or positioning time, i.e., the time the stacker crane moves without carrying a box) is as short as possible. This problem turns out to be a directed Hamiltonian path problem; it is trivially equivalent to an asymmetric TSP. We solve this directed Hamiltonian path problem as follows. At a general point in time, the stacker crane is executing some job or idle and there are some jobs that the stacker crane has to execute in the future. If one job is finished, the crane starts the first job of the sequence. Due to calls from the assembly line or deliveries of the driverless vehicles new jobs are created. Whenever a new job comes up, we run a very fast insertion heuristic to schedule the new job and call the resulting sequence our new solution. Then a more elaborate but still fast heuristic tries to improve the present solution. Finally, a branch-and-bound code is activated that determines an optimum solution of the present problem.

It may happen that a new job is created while we are still computing. In that case we stop the optimization process and turn to the new enlarged problem.

	# TT	uTr-P	uTr-0	I %	max-TT	Ø# TT
1	416	8599	8325	3.18	6	2.31
2	421	8655	8141	5.93	8	1.94
3	405	8238	7956	3.42	6	2.27
4	398	8017	7634	4.77	8	1.93
5	447	9411	8951	4.88	8	2.13

Table 5: Minimizing the unloaded travel time
(normal conditions)

The asymmetric TSPs that come up in this system are very small under normal conditions (see max-TT and Ø#TT in Table 5 for largest and the average instance sizes). Even under heavy load conditions (see max-TT and Ø#TT in Table 7) the sizes of the assymetric TSP instances arising have never exceeded 40 "cities" during the days for which we collected data. Thus we decided to use a branch-and-bound code instead of investing into the extensive effort of implementing a cutting plane algorithm. Currently we are using the code of Fischetti and Toth [FT92] that solves the instances arising here in reasonable time. We used the simulation model to compare our approach with the old system. Table 5 shows the results for the week that was used for the simulation model validation. The running time improvements (see column I%) of 3% to 6% were very disappointing. By analyzing the data it turned out that the week considered was a week of low production volume. On the average, there were only two jobs to schedule at a time (see column Ø#TT) and thus almost nothing to optimize.

Tables 6 and 7 report about two different periods of heavy load. These usually occur after the breakdown of some part of the production system. In such cases jobs to be executed start to pile up. The running time improvements (see columns I%) in these cases ranged from 15% to 40%. In fact, this is exactly what is needed in the system: quick recovery from "catastrophies". The results of our optimization approach were considered so satisfactory that our software for scheduling the moves of the stacker crane has been put into use at the Siemens Nixdorf Werk für Arbeitsplatzsysteme. It has considerably improved the capacity of the storage system.

	# TT	uTr-P	uTr-O	I %
1	18	344	239	30.52
2	10	194	143	26.28
3	18	347	211	39.19
4	26	507	330	34.91
5	20	388	283	27.06

Table 6: Minimizing the unloaded travel time
(heavy load conditions)

	# TT	uTr-P	uTr-O	I %	max-TT	Ø# TT
1	50	917	693	24.42	29	13.32
2	49	974	749	23.10	20	8.32
3	50	1007	783	22.24	26	12.23
4	49	889	662	25.53	31	15.24
5	50	985	839	14.82	25	13.08

Table 7: Minimizing the unloaded travel time
(heavy load conditions)

# TT	:	number of transportation tasks (TT)
uTR-P	:	unloaded travel time with priority rule (in sec.)
uTr-O	:	unloaded travel time with optimization (in sec.)
I %	:	improvement in % (((uTR-P) – (uTr-O)) / (uTr-P))
max-TT	:	maximal number of TT handled at the same time
Ø# TT	:	average number of TT handled at the same time

We are aware of the fact that our approach may not be the best way to handle on-line decisions of the type described above. But the scientific literature on handling complex on-line situations such as this is one is almost nonexisting. We were happy that we could improve a bottleneck situation in the factory considerably, but we know that deeper theoretical investigations are necessary to get a good understanding of such cases. We are currently experimenting with other approaches that lead to new combinatorial optimization problems and are going to compare these on a practical and theoretical basis with the approach outlined above.

More details on the results of our stacker crane optimization will appear soon in joint papers with the Siemens Nixdorf partners. The mathematics involved and a thorough analysis of the practical achievements will be described in [As93].

5. More Applications and Some Mathematics. This paper consists of a list of a few problems arising in the world of manufacturing and my attempts to describe the use of mathematics in their solution. I have tried to indicate that the line of attack taken is a general approach, not confined to the special cases discussed here. But I also have to admit that there is no guarantee of success. In every particular case, substantial joint work of practitioners and theoreticians is necessary to contribute significantly to the solution of the problems coming up. Due to lack of space many important areas in manufacturing where discrete mathematics plays or could play an essential role could not be mentioned at all. I will indicate a few further fruitful topics in the subsequent subsection. I also note that the mathematical theory involved in the solution of the problems outlined above has been purposely avoided here. A glimpse at it will be provided in Section 5.2.

5.1. A Few Further Applications. A vast application area for discrete mathematics is the field of logistics and transportation. Factory internal questions such as where to store parts and how to move them are vital for production speed. Due to the increasing emergence of just-in-time manufacturing processes and the high traffic volume associated with these production techniques, the planning and operating of shipments of parts and components have become a major topic. This involves questions of what to order when and in which quantities. This affects the planning of the fleets of trucks that are necessary to handle the transportation volume and concerns the actual planning of tours to satisfy all requirements and minimize transportation costs. Moreover, shifts of drivers have to be determined such that all work regulations are met and the labor costs are as low as possible. Clearly, the latter question extends to plant management. Finding a cost-effective mix of the work force, assigning the right people to the right job, and planning the work so that the idle time is minimum is of substantial importance. Problems of this type lead to integer, mixed-integer or combinatorial optimization problems of very large scale.

Lot sizing problems have always been a major concern in industry. They play an increasing role since many advanced machines (for instance in the chemical industry) can be operated cheaply at high production volume but have very high set-up costs. Balancing these with the cost for inventory and the desire of keeping the stocks at a level such that all expected demands can be supplied is a challenging task.

I would also like to mention a topic that is typically neglected. There are discussions in the public press about whether or not airplanes or nuclear power plants are safe. Machine safety is rarely a public issue in general. What I have in mind here is not only the security of workers or the ecological systems, but also the safeguarding of machines against unwanted moves or self-destruction. The ordinary machines on factory floors, the automatic assembly and transfer lines are becoming more and more complicated. Tools for engineers are scarcely available to help them design machines that do not only operate correctly in standard situations, but machines that are also safe in case

certain components or control sensors break. Severe accidents or the breakdown of the production process may occur if the control system of a machine is unable to handle a complicated mix of errors.

I have worked on such a topic together with Klaus Truemper (University of Texas at Dallas) jointly with the Grob Corporation (Mindelheim, Germany). Problems as indicated above can often be formulated as problems in algorithmic logic. A main task here is to design algorithms for the satisfyability problem of a logic formula (say, given in conjunctive normal form) that decide very quickly if a formula has a solution or not. It is not enough to have a fast answer in case there is a solution, but one also has to know quickly if there is none. This second requirement makes the design of a different type of algorithm for the $\mathcal{NP}$-complete satisfyability problem necessary. The standard algorithmic approaches do a reasonable job if solutions of a logic formula exist but have unpredictable running times otherwise. Predictability of the solution time here is paramount for the safety of the systems. For example, if a pilot learns after 5 minutes that an operation he performed was dangerous, it may be too late. Klaus Truemper has developed a very effective software system, called Leibniz, to handle such situations. It is based on new results on ternary matroids; see [T90], [T93].

Let me close this journey through applications with an important question that I am often asked by students and that I am unable to answer satisfactorily. How does one model practical situation mathematically? Are there guidelines for this approach? I simply don't know how to teach this methodology other than by giving examples, by describing basic situations that frequently arise, by showing "little tricks" that help, by outlining approaches that are hopeless, and by giving heuristic reasons why some models work and why others don't work. The work described in this paper reflects this fact.

5.2. A Glimpse at the Mathematics. Due to lack of space it is impossible to describe all the mathematical theory that has been developed to solve the problems mentioned in the previous sections. I will list here a few of the techniques used and point to the literature where in-depth treatments can be found.

It is indeed fortunate when one encounters a well-studied and polynomially solvable combinatorial optimization problem in a real-world application. Easy problems, such as shortest path, max-flow or min-cost flow problems do, in fact, frequently arise as subproblems of more general problem types. In such cases, one can look in standard textbooks like [Sch86] or [NW88] or recent survey papers such as [AMO89] or [GoTT90] for flow problems to get information about available theory and existing fast algorithms.

For example, among the cases discussed here, two of the box assignment problems described in Section 4.2 turned out to be ordinary assignment problems. We knew that Peter Kleinschmidt (Passau) had recently implemented a fast assignment code. He subsequently tuned his algorithm for the instance structure that arises in our application. It easily solves all of the instances of interest to us.

In the via minimization case reported in Section 2.4, a certain setting of the parameters (occurring in practice) results in max-cut problems for planar graphs. There exist a beautiful theory and several algorithmic approaches for solving this problem, but no implementation was available. In her master's thesis, Petra Mutzel [Mu90] closed this gap and came up with a practically and theoretically fast code for the solution of such cases. This involved the implementation of a planarity testing and the associated embedding algorithm, of a graph dualization algorithm and a certain transformation procedure, and of a shortest path and a matching algorithm.

$\mathcal{NP}$-hard problems need special treatment. What to do depends on the sizes of the problem instances that arise in the practical situation under investigation, the computers available and on the running time bounds given.

For instance, we noticed that in the stacker crane optimization problem of Section 4.2 asymmetric travelling salesman problems arise that have small to moderate size. Branch and bound algorithms based on assignment relaxations, such as the one described in, are able to solve such instances reasonably well. Standard techniques that can be found in any textbook on combinatorial optimization suffice. There is no need to use more involved machinery in such cases.

The design of heuristics for large-scale problems is of particular importance. In some cases known basic approaches work well only for small problems. Further investigation of the problem

structure, of suitable data structures and their running time analysis is required to extend these heuristics to large-scale problems. For example, in TSP applications like the plotting and drilling problems discussed in Chapter 3, where points are given by their coordinates and the distances are defined by a metric, the use of techniques from computational geometry (e.g., Voronoi diagrams and convex hulls) helped to speed up existing algorithms tremendously and to achieve good quality solutions in very short time. Such heuristic techniques were thoroughly investigated in [R92]. Even the design of relatively simple heuristics may lead to mathematical problems of general interest.

Substantial mathematical work is necessary if one wants to design algorithms that produce provably optimal solutions or provide very good quality guarantees. At present, the most successful general approach in this case is the use of polyhedral combinatorics as the backbone of LP-based cutting plane or branch-and-cut algorithms. This technique associates with every instance of a combinatorial optimization problem a polyhedron such that the optimum solutions of the instance of the combinatorial optimization problem are precisely the optimum vertex solutions of the corresponding programs over the associated polyhedron. The difficulty is that the construction of the polyhedron is nothing but a theoretical device. A complete and nonredundant description of the associated polyhedron by means of linear inequalities and equations has to be found by special case investigations. Moreover, turning these theoretical investigations into efficient algorithmic tools requires further work and insight.

Many combinatorial optimization problems are currently investigated by following this approach. The survey papers [GP85], [PaG85] describe this approach for the travelling salesman problem. A more general survey of the area of polyhedral combinatorics is [Pu89]. I would like to demonstrate this technique using the max-cut problem as an example.

If $G = (V, E)$ is a graph and $W \subseteq V$ a node set, then the edge set $\delta(W) := \{ij \in E | i \in W, j \in V \backslash W\}$ is called a cut. (If $\emptyset \neq W \neq V$, the removal of this edge set will disconnect the nodes in W from the nodes in $V \backslash W$.) An instance of the max-cut problem is given by a graph $G = (V, E)$ with weights $c_e \in \mathbb{R}$ for all edges $e \in E$. The task is to find a cut $\delta(W)$ of G of maximum weigth $c(\delta(W)) := \sum_{e \in \delta(W)} c_e$. The max-cut problem comes up, for instance, in via minimization, see Section 2.4, and is used in statistical mechanics to calculate ground states of spinglasses, see [BaGJR88].

To associate a polyhedron with an instance of the max-cut problem we proceed as follows. Given $G = (V, E)$ we consider the vector space $\mathbb{R}^E$, where each component of a vector $x = (x_e)_{e \in E} \in \mathbb{R}^E$ is indexed by an edge of E. If $F \subseteq E$, we define the incidence vector $\chi^F \in \mathbb{R}^E$ of F by setting $\chi^F_e = 1$ if $e \in F$ and $\chi^F_e = 0$ if $e \notin F$. The cut polytope $\mathrm{CUT}(G)$ associated with G is nothing but the convex hull of all incidence vectors of cuts of G, i.e.,

$$\mathrm{CUT}(G) = \mathrm{conv}\{\chi^{\delta(W)} \in \mathbb{R}^E \mid W \subseteq V\}. \tag{5.1}$$

It follows from this construction that the cuts of G are in one-to-one correspondence with the vertices of the cut polytope. This implies that the optimum vertex solutions of the linear program

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.} & x \in \mathrm{CUT}(G) \end{array} \tag{5.2}$$

are precisely the optimum solutions of the max-cut problem defined by $G = (V, E)$ and the edge weights c_e. Hence, we could solve the max-cut problem by linear programming techniques if we knew a description of $\mathrm{CUT}(G)$ by means of linear inequalities. To find inequalities that are valid for $\mathrm{CUT}(G)$ and define facets of $\mathrm{CUT}(G)$ is an interesting research topic. The paper [DL91] of over 80 pages length surveys its state-of-the-art. The cut polytope has received so much attention because it arises in so many different ways. For instance, it is related to Eulerian subgraphs by graph duality; embedding problems of semi-metric spaces in functional analysis lead to investigations of cut cones [AsD82]; and cycles in binary matroids form a far reaching generalization [BaG86], [GT89].

A first step of investigation is usually to formulate (5.2) as an integer linear program. One of the first theorems taught in graph theory states that a cut and a circuit meet in an even number of

edges. Thus, if $C \subseteq E$ is a circuit and $F \subseteq C$ of odd cardinality, then every incidence vector of a cut must satisfy the so-called "odd circuit inequality"

$$\sum_{e \in F} x_e - \sum_{e \in C \setminus F} x_e \leq |F| - 1 \,.$$

Every incidence vector obviously satisfies the "trivial constraints" $0 \leq x_e \leq 1$ for all $e \in E$ and thus it follows that all inequalities,

$$\text{(5.3)} \qquad \begin{array}{lll} \text{(i)} & 0 \leq x_e \leq 1 & \text{for all } e \in E, \\ \text{(ii)} & \displaystyle\sum_{e \in F} x_e - \sum_{e \in C \setminus F} x_e \leq |F| - 1 & \text{for all circuits } C \subseteq E, \text{ and all } F \subseteq C, |F| \text{ odd} \end{array}$$

are valid for $\mathrm{CUT}(G)$. Let us set

$$\mathrm{P}(G) := \{x \in I\!R^E \mid x \text{ satisfies (5.3) (i) and (ii)}\}.$$

Then, clearly, $\mathrm{CUT}(G) \subseteq P(G)$. In fact, one can easily prove the following.

(5.4) PROPOSITION. *For every graph $G = (V, E)$*

$$\mathrm{CUT}(G) = \mathit{conv}\{x \in P(G) \mid x \quad \mathit{integral}\}.$$

This implies that

$$\text{(5.5)} \qquad \begin{array}{lll} \max & c^T x & \\ \text{(i)} & 0 \leq x_e \leq 1 & \text{for all } e \in E, \\ \text{(ii)} & \displaystyle\sum_{e \in F} x_e - \sum_{e \in C \setminus F} x_e \leq |F| - 1 & \text{for all circuits } C \subseteq E, \text{ and for all } F \subseteq C, |F| \text{ odd} \\ \text{(iii)} & x_e \in \{0, 1\} & \text{for all } e \in E \end{array}$$

is an integer programming formulation of (5.2).

An immediate question arises. Do we really need the integrality conditions in (5.5), i. e., does $\mathrm{CUT}(G) = \mathrm{P}(G)$ hold? A result due to Barahona and Majhoub [BaM86] answers this question.

(5.6) THEOREM. *Let $G = (V, E)$ be a graph. Then $\mathrm{P}(G) = \mathrm{CUT}(G)$ if and only if G is not contractible to the complete graph K_5.*

Theorem (5.6) together with the Wagner-Kuratowski theorem for planar graphs yields that $\mathrm{P}(G) = \mathrm{CUT}(G)$ holds for planar graphs. A considerable generalization of (5.6) is the polyhedral characterization of those binary matroids that have the sums of circuits property; see [Sey81] and [GT89].

Let us return from this theoretical line of investigation, which is of interest in its own right, to more algorithmic issues. If we can solve the linear program that arises from (5.5) by dropping the integrality constraints, i. e., the so-called LP-relaxation of (5.5), then we will obtain an upper bound on the maximum weight of a cut. A quick count shows that the number of odd circuit inequalities grows exponentially with the graph size. Hence, there is no way to input these inequalities into a computer in polynomial time. Do we have to give up?

An obvious idea now is to check whether we really need all odd circuit inequalites and to determine which are redundant. This amounts to characterizing those inequalities of the system (5.3) (i), (ii) that define facets of the cut polytope. The following is shown in [BaM86].

(5.7) THEOREM. *Let $G = (V, E)$ be a graph.*

(a) The dimension of $\mathrm{CUT}(G)$ is equal to $|E|$.

(b) For every edge $e \in E$, the following statements are equivalent:

(b_1) $x_e \geq 0$ defines a facet of $\mathrm{CUT}(G)$,

(b_2) $x_e \leq 1$ defines a facet of CUT(G),
(b_3) e does not belong to a triangle.
(c) Let $C \subseteq E$ be a circuit and $F \subseteq C$, $|F|$ odd, then the odd circuit inequality

$$\sum_{e \in F} x_e - \sum_{e \in C \setminus F} x_e \leq |C| - 1$$

defines a facet of CUT(G) if and only if C has no chord.

This result reduces the number of necessary inequalities considerably. For instance, for the complete graph K_n, no trivial constraint is facet-defining and only triangles are chordless circuits. Thus, only $O(n^3)$ inequalities remain, i.e., in this case the LP- relaxation has polynomial size. But in general, there are still exponentially many facet-defining odd circuit constraints.

Nevertheless, we do not have to give up. Help comes from powerful results that are based on the ellipsoid method. To formulate these we have to introduce further concepts.

Let P be a polyhedron. By definition, there exist a matrix A and a vector b such that P$= \{x | Ax \leq b\}$, and, equivalently, there are finite sets S, T of vectors such that P$= conv(V) + cone(T)$, where $cone(T)$ denotes the set of all points that are nonnegative linear combinations of elements of T. We call P rational if all entries of A and b (or equivalently, all entries of the vectors in S and T) are rational. The encoding length of an inequality $a^T x \leq \alpha$, a and α rational, is the number of binary digits needed to encode a and α, where a rational number is encoded by encoding its numerator and denominator. We say that a rational polyhedron P has facet-complexity at most φ ($\varphi \in I\!N$) if there is an inequality system $Ax \leq b$ such that $P = \{x | Ax \leq b\}$ and each inequality of the system has encoding length at most φ.

Note that polyhedra with very many facets may have small facet-complexity. Consider, for example, the inequality system (5.3). The encoding length of the inequalities in (i) is 4, while the maximum encoding length of the inequalities in (ii) is $2|E| + \lceil log|E| \rceil + 1$. Hence, the facet-complexity of P(G) is at most $3|E|$, say. This implies that the facet-complexity of the polytopes P(G) is polynomial in the encoding length of G. One can similarly prove that the facet-complexity of cut polytopes is polynomial in the encoding length of the associated graphs.

Let $\mathcal{P}$ be a class of rational polyhedra. We say that the optimization problem for $\mathcal{P}$ can be solved in polynomial time if, for any polyhedron $P \in \mathcal{P}$ and any rational vector c, the linear program

$$\begin{array}{ll} \max & c^T x \\ \text{s.t.} & x \in P \end{array}$$

can be solved in time polynomial in the encoding length of c and the facet-complexity of P.

The separation problem for a rational polyhedron $P \subseteq I\!R^n$ is the following. Given a rational vector $y \in I\!R^n$, decide whether $y \in P$ and if not, determine a vector $c \in I\!R^n$ such that $c^T y > max\{c^T x | x \in P\}$. Observe that c yields a hyperplane separating y from P.

Let $\mathcal{P}$ be a class of rational polyhedra. We say that the separation problem for $\mathcal{P}$ can be solved in polynomial time if, for any polyhedron $P \in \mathcal{P}$ and any vector y, the separation problem for P and y can be solved in time polynomial in the encoding length of y and the facet-complexity of P.

The following result, using the ellipsiod method, was proved in [GLS81], see also [GLS88] for an in-depth treatment of this subject.

(5.8) THEOREM. *Let $\mathcal{P}$ be a class of rational polyhedra. Then the following two statements are equivalent:*
(a) The optimization problem for $\mathcal{P}$ can be solved in polynomial time.
(b) The separation problem for $\mathcal{P}$ can be solved in polynomial time.

What help does (5.8) provide in the study of cut polytopes? Note that the classes $\mathcal{P}_1 :=$ $\{$CUT(G)$|G$ a graph$\}$, $\mathcal{P}_2 := \{P(G) | G$ a graph$\}$ are classes of rational polyhedra where the polyhedra in $\mathcal{P}_1$ and $\mathcal{P}_2$ have a facet-complexity that is polynomial in the encoding length of the associated graph. We really would like to prove that the optimization problem for $\mathcal{P}_1$ can be solved in polynomial time and that means in time polynomial in $|E| + |V|$. Due to the $\mathcal{NP}$-hardness of the max-cut problem this seems to be impossible. But we can optimize over $\mathcal{P}_2$ using the equivalence of (a) and (b) in (5.8). Namely, it was shown in [BaM86] that the separation problem for $P(G)$, G a graph, can be solved in polynomial time. The hard part, of course, is, given a vector y, to check whether y satisfies all odd circuit constraints and if not to find one of these inequalities that is violated by y.

The result follows by a tricky transformation of this problem to a sequence of shortest path problems as follows.

Let $G = (V, E)$ be a graph and $y \in \mathbb{Q}^E$. We first check whether $0 \leq y_e \leq 1$ for all $e \in E$ by substitution. If not, a separating hyperplane is at hand. Otherwise we construct a new graph $H = (V' \cup V'', E' \cup E'' \cup E''')$ consisting of two disjoint copies $G' = (V', E')$ and $G'' = (V'', E'')$ of G and an additional edge set E''' that contains, for each $uv \in E$, the two edges $u'v'', u''v'$. The edges $u'v' \in E'$ and $u''v'' \in E''$ get the weight y_{uv}, while the edges $u'v'', u''v' \in E'''$ get the weigth $1 - y_{uv}$. For each node $u \in V$, we calculate a shortest (with respect to the weights just defined) path in H from $u' \in V'$ to $u'' \in V''$. Such a path contains an odd number of edges of E''' and corresponds to a closed walk in G containing u. Clearly, if the shortest of these $[u', u'']$-paths has length at least 1 then y satisfies all odd circuit constraints, otherwise there exists a circuit C and a set $F \subseteq C, |F|$ odd, such that y violates the corresponding inequality.

Summing all these observations up, we obtain that for any graph G, the LP-relaxation of (5.5) can be solved in polynomial time and thus the max-cut problem for graphs not contractible to K_5 (and hence for planar graphs) can be solved in polynomial time. One has to admit, though, that these algorithmic results are not practical. If one replaces the ellipsoid method (that is needed to derive the polynomial time bounds) by the simplex method and the use of cutting plane techniques one obtains a practically useful algorithm for solving the LP-relaxation of (5.5).

This is the the central idea for a relatively efficient branch-and-cut algorithm for the max-cut problem that is based on the LP-relaxation of (5.5), that uses further classes of facet-defining cutting planes for CUT(G) heuristically and that employs additional heuristic techniques. This algorithm solves max-cut problems on graphs with several thousand nodes to optimality, see [BaGJR88].

Techniques like the one described above form the backbone of many recent successful attempts to solve hard large scale combinatorial optimization problems to optimality, see for example [GH91], [PaR91]. There are further approaches based, for instance, on Lagrangian relaxations and the use of nondifferentiable convex minimization or based on geometry of numbers and basis reduction for lattices, see [LS90], [CRSS92]. It is impossible to cover these approaches here.

5.3. Final Remarks. The purpose of this paper was to show, by means of a few examples, how and where interesting mathematical problems of a discrete nature arise in the field of manufacturing and to indicate the important role mathematics could play if all parties involved in manufacturing were aware of the contributions mathematics is able to make. The focus was on describing a few real-world applications and to indicate the difficulties that arise in mathematically modelling complex situations. There are cases where we have a solid understanding of the practical problems, a rich associated mathematical theory and sophisticated algorithmic tools. In other cases suitable mathematical models are still missing, sound theories have not yet emerged and substantial, probably quite difficult mathematical work still lies before us.

REFERENCES

[AH92] A. Abdel Hamid, *Optimization aspects of automatic storage systems*, Dissertation, Technische Universität Berlin, 1992.

[AKP89] H. Achatz, P. Kleinschmidt, K. Paparrizos, *A dual algorithm for the assignment problem*, Working Paper, Universität Passau, 1989.

[AMO89] R.K. Ahuja, T.L. Magnanti, J. B. Orlin, *Network flows*, in: G. L. Nemhauser, A. H. R. Rinnooy Kan, M. J. Todd (eds.), *Optimization*, Handbook in Operations Research and Management Science, North-Holland, Amsterdam, 1989, pp. 211-369.

[As93] N. Ascheuer, *On-line-optimization of flexible manufacturing systems*, Dissertation, Technische Universität Berlin, 1993, to appear.

[AsD82] P. Assouad. M. Deza, *Metric subspaces of L^1*, Publications Mathématiques d'Orsay, vol. 3, 1982.

[BaG86] F. Barahona, M. Grötschel, *On the cycle polytope of a binary matroid*, Journal of Combinatorial Theory B, 40 (1986), pp. 40-62.

[BaGJR88] F. Barahona, M. Grötschel, M. Jünger, G. Reinelt, *An application of combinatorial optimization to statistical physics and circuit layout design*, Operations Research, 36 (1988), pp. 493-513.

[BaM86] F. Barahona, A.R. Mahjoub, *On the cut polytope*, Mathematical Programming, 36 (1986), pp. 157-173.

[Bau90] P. Bauer, *Optimale Steuerung des FALKE-Automaten*, Projektdokumentation, Internal Report, Augsburg, 1990.

[Br77] M.A. Breuer, *Min Cut Placement*, J. Design Aut. &. Fault Tol. Comp., 1 (1977), pp. 343-362.

[CKC83] R.-W. Chen, Y. Katjitani, S.-P. Chan, *A graph-theoretic via minimization algorithm for two-layer printed circuit bourds* , IEEE Trans. Circuits and Syst., 30 (1983), pp. 284-299.

[CRSS92] W. Cook, T. Rutherford, H. Scarf, D. Shallcross, *Integer Programming using Lovász-Scarf basis reduction*, in preparation.

[DL91] M. Deza, M. Laurent, *A survey of the known facets of the cut cone*, Institut für Ökonomie und Operations Research, Universität Bonn, Report No. 91722-OR, 1991.

[FT92] M. Fischetti, P. Toth, *An additive bounding procedure for the asymmetric TSP*, Mathematical Programming, 53 (1992), pp. 173-197.

[GaJ77] M.R. Garey, D.S. Johnson, *The rectilinear Steiner Tree problem is $\mathcal{NP}$-complete*, SIAM J. Appl. Math., 32 (1977), pp. 826-834.

[GoTT90] A. V. Goldberg, É. Tardos, R. E. Tarjan, *Network flow algorithms*, in: B. Korte, L. Lovász, H. J. Prömel, A. Schrijver (eds.), *Paths, Flows, and VLSI-Layout*, Springer, Berlin, 1990, pp. 101-164.

[GH91] M. Grötschel, O. Holland, *Solution of large-scale symmetric travelling salesman problems*, Mathematical Progamming, 51 (1991), pp. 141-202.

[GJR89] M. Grötschel, M. Jünger, G. Reinelt, *Via minimization with pin preassignments and layer preference*, Zeitschrift für Angewandte Mathematik und Mechanik, 69 (1989), pp. 393-399.

[GJR91] M. Grötschel, M. Jünger, G. Reinelt, *Optimal control of plotting and drilling machines: a case study*, Zeitschrift für Operations Research, 35 (1991), pp. 61-84.

[GLS81] M. Grötschel, L. Lovász, A. Schrijver, *The ellipsiod method and its consequences in combinatorial optimization*, Combinatoria, 1 (1981), pp. 169-197.

[GLS88] M. Grötschel, L. Lovász, A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer, Berlin, 1988.

[GP85] M. Grötschel, M. W. Padberg, *Polyhedral theory* in: E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (eds.), *The Traveling Salesman Problem*, Wiley, Chichester, 1985, pp. 251-305.

[GT89] M. Grötschel, K. Truemper, *Decomposition and optimization over cycles in binary matroids*, Journal of Combinatorial Theory B, 46 (1989), pp. 306-337.

[JMRW92] M. Jünger, A. Martin, G. Reinelt, R. Weismantel, *Quadratic 0/1 optimization and a decomposition approach to the placement of electronic circuits*, Mathematical Programming, 1992, to appear.

[Ka72] R. M. Karp, *Reducibility among combinatorial problems*, R. E. Miller, J. W. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 1972, pp. 85-103.

[KlSJ88] J. M. Kleinhans, G. Sigl, F. M. Johannes, *Gordian: A new global optimization/ rectangle dissection method for cell placement*, IEEE Int. Conference on CAD ICCAD-88, 1988, pp. 506-509.

[KPS90] B. Korte, H. J. Prömel, A. Steger, *Steiner trees in VLSI-Layout*, in: B. Korte, L. Lovász, H. J. Prömel, A. Schrijver (eds.), *Paths, Flows, and VLSI-Layout*, Springer, Berlin, 1990, pp. 185-214.

[KrL84] M. R. Kramer, J. van Leeuwen, *The complexitiy of wire-routing and finding minimum area layout for arbitrary VLSI circuits*, in: F.P. Preparata (ed.), *Advances in Computing Research*, 1984, pp. 129-146.

[LLRS85] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys (eds.), *The Traveling Salesman Problem* Wiley, Chichester, 1985, pp. 251-305.

[L90] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley, Chichester, 1990.

[LS90] L. Lovász, H. Scarf, *The generalized basis reduction algorithm*, Cowles Foundation, Discussion Paper No. 946, Yale University, June 1990.

[M92] A. Martin, *Packen von Steinerbäumen: Polyedrische Studien und Anwendungen*, Dissertation, Technische Universität Berlin, 1992.

[Mu90] Petra Mutzel, *Implementierung und Analyse eines Max-Cut-Algorithmus,* Diplomarbeit, Universität Augsburg, 1990.

[NW88] G. Nemhauser, L, A. Wolsey, *Integer and Combinatorial Optimization*, Wiley, Chichester, 1988.

[PaG85] M. Padberg, M. Grötschel, *Polyhedral computations*, in: E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, (eds.), *The Traveling Salesman Problem*, Wiley, Chichester, 1985, pp. 307-360.

[PaR91] M. Padberg, G. Rinaldi, *A branch-and-cut-algorithm for the resolution of large-scale symmetric traveling salesman problems*, SIAM Review, 33 (1991) pp. 60-100.

[Pi84] R.Y. Pinter, *Optimal layer assignment for interconnect*, J. VLSI Comput. Syst., 1 (1984), pp. 123-137.

[Pu89] W. R. Pulleyblank, *Polyhedral combinatorics*, in: G. L. Nemhauser, A. H. R. Rinnooy Kan, M. J. Todd (eds.), *Optimization*, Handbook in Operations Research and Management Science, North-Holland, Amsterdam, 1989, pp. 371-446.

[R91] G. Reinelt, *TSPLIB - A traveling salesman problem library*, ORSA Journal on Computing, 3 (1991), pp. 43-49.

[R92] G. Reinelt, *Contributions to Practical Traveling Salesman Problem Solving*, Springer, Heidelberg, 1992, pp. 43-49.

[Sch86] A. Schrijver, *Theory of Linear and Integer Programming*, Wiley, Chichester, 1986.

[Sey81] P.D. Seymour, *Matroids and multicommodity flows*, European Journal of Combinatorics, 2 (1981), pp. 257-290.

[SK72] D.G. Schweikert, B. W. Kernighan *A proper model for the partitioning of electrical circuits*, Proceedings Design Automation Conference, 1972, pp. 56-62.

[T90] K. Truemper, *Leibniz System: User's Manual*, Leibniz Company, Plano, Texas, 1990.
[T93] K. Truemper, *Logic Decomposition*, 1993, in preparation.
[W92] R. Weismantel, *Plazieren von Zellen: Analyse und Lösung eines quadratischen* 0/1*-Optimierungsproblems*, Dissertation, Technische Universität Berlin, 1992.