INTERFACES BETWEEN OPERATIONS RESEARCH AND COMPUTER SCIENCE

J.K. Lenstra

Centre for Mathematics and Computer Science (CWI), Amsterdam

The subject of this lecture is the relation between two disciplines, both of which pertain to the use of mathematics in industry. Operations research emerged out of optimization questions in warfare logistics in the 1940's and quickly established itself as one of the cornerstones of industrial mathematics. Computer science and engineering is a younger discipline, which pervades all activities relating to the processing and manipulation of quantitative information at an ever increasing rate.

I cannot possibly give a complete overview of all interfaces between operations research and computer science. The time is too short and my background is too limited. So you should expect a biased view of some interfaces.

Let me give a short preview. I will make only a few remarks on the impact of operations research on computer science and concentrate on the influence in the other direction. Another bias is the emphasis on the combinatorial side of operations research rather than on the nonlinear or stochastic ones. I will go through issues of computation, complexity and analysis of algorithms, and then mention three new tools: randomization, parallelism (both quite briefly), and interaction. A substantial part of my lecture will be devoted to interactive computing or, more precisely, to the integration of interaction and algorithmics in what is nowadays called "decision support systems". I will finish with some remarks on the subject of expert systems.

Operations research vs. computer science

The basic claim which I would like to make is that operations research and computer science cannot exist without each other. More specifically, the evolution of either area to a discipline in itself has only been possible due to

the use of concepts, results and techniques developed in the other area.

As I said before, I will concentrate on the impact of computer science on operations research. That is not to say that the influence in the other direction is less important. It is hard to resist the temptation to spend the entire hour on this subject. Many applications of operations research occur in the design, analysis and control of computing devices. This has been true since the very beginning of automated computing. The three examples I will give are the more topical applications of the 1980's.

First, the *layout of integrated circuits*. Locating several given components on a chip and routing the required connections between them in such a way that the total area is minimized, is a highly complex problem. It is, in fact, a combinatorial optimization problem, and many ideas from location, routing and scheduling theory could contribute to its solution. It is my impression that this interface is at a very early stage of exploration.

On a larger scale, the *performance of computer systems* is a subject of increasing interest. From an operations research point of view, it calls for the analysis of the behavior of communication networks and thus belongs to the application area of queueing theory. This interface is in rapid development.

In the third place, the *design and control of distributed systems* leads to many algorithmic problems. Consider, for example, a network of processors that cooperate to perform a certain task. Each processor has its own memory. Which data must be stored where so as to achieve a reasonable balance between storage and access cost? This is just an example of a question in distributed computing to which operations research could contribute.

It appears that computer science needs operations research. In that sense, it has already a substantial impact by providing a wealth of challenging decision problems.

The influence that I will emphasize, however, is more fundamental. Without the achievements of computer science, the practical application of operations research methods as well as our theoretical understanding of their behavior would have been very limited.

Computations, efficiency, and complexity

Operations research needs computer science. This was already evident at the origin of our discipline. The whole idea of solving large scale decision

problems by iterative techniques was only a feasible idea in view of the existence, or the prospect of developing, automated equipment for performing massive amounts of computation. This applies to the work on linear programming and network optimization that started in the late 1940's. It is also true for most of the subsequent methodological developments, such as cutting plane methods, dynamic programming, branch and bound, and Markov programming.

So far, the influence of computer science was basically restricted to providing the machinery needed. During the 1960's, the impact deepened and acquired a component that was more algorithmic in nature. Operations researchers became interested in the efficiency of computer algorithms. In order to reduce the time and space requirements of their implementations, they had to use sophisticated data structures. With this, operations research started to rely on the art of computer programming.

The advent of computational complexity theory in the beginning of the 1970's can, in retrospect, be seen as a natural culmination of this development. At the time, it was a surprising and exciting event. Complexity theory studies the inherent limitations to the efficiency of algorithms. It provides a simple tool to distinguish between the tractable problems and the intractable, or probably intractable, ones. This distinction is now common practice in combinatorial operations research. For the easy problems, a fast optimization algorithm is available. For the hard problems, one has to choose: if an optimal solution is required, then one must settle for some tedious form of enumeration; if speed is desired, then one must be satisfied with an approximate solution. Linear programming is an example of an easy problem; integer programming is probably hard.

In what follows, I shall first take a closer look at the concepts of complexity and then return to the study of algorithms.

A closer look at complexity

You all know what a graph is: a collection of nodes and a collection of edges, each of which links two nodes together. The graph in Figure 1 is connected, because you can get from each node to any other. The graph in Figure 2 is disconnected. The graph in Figure 3 is Hamiltonian, since there is a cycle which visits each node exactly once; such a Hamiltonian cycle is indicated by wiggly lines. The graph in Figure 4 is not Hamiltonian; you might want to prove this.

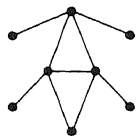


Figure 1 A connected graph.

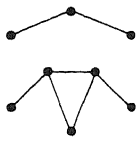


Figure 2 A disconnected graph.

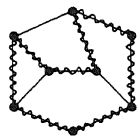


Figure 3 A Hamiltonian graph.

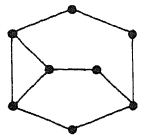
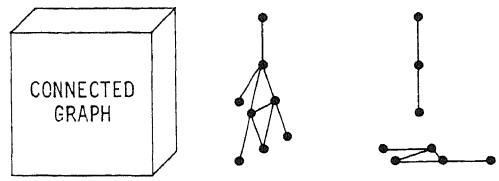
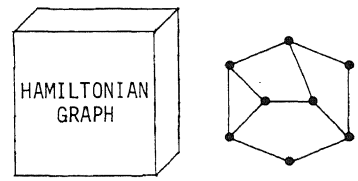


Figure 4 A non-Hamiltonian graph.



(a) ? (b) Right. (c) Wrong.

Figure 5 Buying a connected graph.



(a) ? (b) ??

Figure 6 Buying a Hamiltonian graph.

Suppose that you are attending a conference and you want to buy a present for the people at home. You go to a graph store and ask for a connected graph. The shopkeeper puts a box on the counter (Figure 5(a)). You want to check this, open the box, and take out the graph. When it sticks together, it is connected (Figure 5(b)); when it falls apart, it is not (Figure 5(c)). The point I want to make here is that you can easily test a graph for connectivity by yourself; it takes an amount of time proportional to the number of edges.

Now you want to buy something special: a Hamiltonian graph. Again, there is a box (Figure 6(a)). You open it - but now you may find yourself in trouble (Figure 6(b)): there is no fast method available which can test any given graph for Hamiltonicity. Trial and error may work, but it does not have to. However, the shopkeeper can easily convince you, namely by pointing out a Hamiltonian cycle as in Figure 3; this takes an amount of time proportional to the number of nodes.

This is exactly the difference between the problem classes P and NP. Both classes contain only decision problems, which require a yes/no answer; I will return to optimization problems shortly. P contains all those problems for which one can easily come up with the correct answer. NP contains all those problems for which one can easily be convinced of the correctness of the yes answer by checking a given structure: a Hamiltonian cycle in the example, a "certificate" in terms of complexity theory.

These definitions only make sense if the notion of "easiness" is formalized. A computation is easy if its running time is bounded by a *polynomial* function of the size of the problem under consideration. For a graph on n nodes, checking all nodes or all edges takes time polynomial in n, but generating all permutations of the node set in the hope of finding a Hamiltonian cycle is superexponential.

What are the virtues of an algorithm when it runs in polynomial time? First of all, its robustness. An algorithm that is polynomial on one machine is polynomial on any other reasonable type of machine, including theoretical models and commercial computers (but excluding parallel machines). Secondly, its asymptotic behavior. Any polynomial function in n is ultimately, when n is large enough, smaller than any superpolynomial function. In the third place, its practical efficiency. Polynomial algorithms tend to work well in practice. Some polynomial algorithms are pretty bad, but it seems to be the case that once a problem has been shown to belong to P, a truly efficient method is found sooner or later. Finally, polynomiality allows us to come to grips with computational complexity in a theoretical sense. It serves to explain why some problems appear to be harder than others. More generally, it has proven to be a fundamental concept in the broad area of computational mathematics.

Any problem in P also belongs to NP, so P is a subclass of NP. I have indicated that the connectivity problem is a member of P and that Hamilton-

icity is in NP. If it could be shown that Hamiltonicity is outside P, then the problem would have no solution in polynomial time and one would justifiably call it "hard". Such a proof seems to be beyond the reach of present-day mathematics. However, we can do slightly less. It can be shown that the Hamiltonicity problem is a generalization of any other problem in NP. Hamiltonicity is *NP-complete*, i.e., it is representative of the entire class NP. If Hamiltonicity would belong to P, then all other problems in NP would be easy as well and P would be equal to NP. No one believes this to be true, for the simple reason that NP seems to be so much richer than P. It follows that the Hamiltonicity problem is unlikely to be easy and therefore "probably hard". See Figure 7.

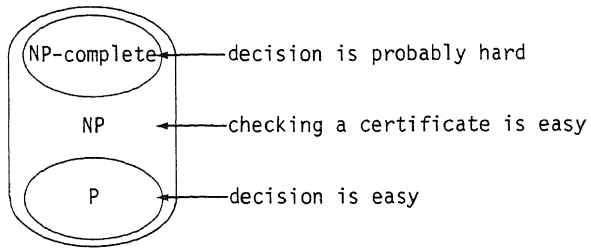


Figure 7 A likely map of NP.

Next to Hamiltonicity, many other combinatorial decision problems have been shown to be NP-complete. I have not told you how results of this type are obtained. That is of secondary importance here; suffice it to say that it is conceptually a simple affair, although it can be technically very intricate.

What is the use of all this for operations research? Complexity theory deals with yes/no problems, in operations research we have optimization problems. If a problem has a polynomial optimization algorithm, then it is said to be easy (or well solved, or tractable). If a problem is at least as hard as some NP-complete problem, then it is said to be NP-hard. This should not be the last word on the problem, but the first. It tells you that you cannot expect to find a guaranteed optimum in worst case polynomial time. You have to give in on either speed or solution quality.

Analysis of algorithms

Complexity theory has given a new impetus to the analysis of algorithms. In studying the behavior of an algorithm, you have to distinguish between its

efficiency and its effectivity. Efficiency measures the resources you need: time, space, or (in case of parallel computing) the number of processors. Effectivity measures the extent to which you reach your goal. As the goal is usually an optimum solution, effectivity stands for solution quality or, more precisely, for the absolute or relative difference between the solution value obtained and the optimum value. It is the principal performance criterion for approximation algorithms.

The traditional way to analyze the behavior of an algorithm is the *empirical* one. You actually run one or more algorithms on one or more computers using one or more test problems and tabulate the results. There are some difficulties: Are the implementations of the algorithms of the same quality? How do the computers compare? Do the test problems form a fair collection? And, finally, how do you statistically validate the experimental results? In spite of this, empirical analysis is widely applied and generally very useful. However, it is not the only resort. Two new approaches have emerged, of a more theoretical nature: worst case analysis and probabilistic analysis.

The purpose of a *worst case* analysis is to provide guarantees on the performance of an algorithm. Complexity theory is basically concerned with this type of analysis. Performance guarantees are solid but may be pessimistic, since the isolated difficult problem instance has to be accounted for. The simplex method for linear programming, for example, requires exponential time on a class of artificial instances, but the method performs quite satisfactorily in practice. It is reassuring to know that a list schedule of jobs on identical parallel machines is never longer than twice the optimal schedule, but it is usually much closer.

Worst case analysis can give a misleading picture of the typical case. Thus the ultimate explanation of why algorithms behave as they do must be of a *probabilistic* nature.

A probabilistic analysis requires first of all the specification of a probability distribution over the set of all problem instances. Several random graph models have been well studied, but for many other combinatorial structures the choice of a reasonable probability model is far less obvious. Moreover, the technical difficulties encountered in a probabilistic analysis are formidable. The main reasons for this are the special structure of problem instances and solutions, as well as the interdependence between the various steps of an algorithm. What happens at a node of a search tree, for example,

depends highly on what has happened at its predecessors, and no real way has been found around the resulting mathematical obstacles.

Nevertheless, progress has been made on various fronts. One of these is probabilistic efficiency analysis, an approach that is now standard for the basic algorithms in computer science. In operations research, a substantial amount of work has been done to explain the success of the simplex method. A great challenge here is to give rigorous proofs of the polynomial expected running time of various enumerative methods, in order to confirm informal analyses or empirical evidence. Secondly, there is the area of probabilistic effectivity analysis. The empirical behavior of approximation algorithms suggests that the worst case is seldom met in practice, but theoretical verification remains very difficult. Most research of this type is actually based on probabilistic value analysis, the third and perhaps most surprising area. Many hard optimization problems, notably those with a geometric structure such as routing and location problems in the plane, allow a simple probabilistic description of their optimum solution value in terms of the problem parameters. The shining example here is the planar traveling salesman problem: the length of a shortest tour through n cities, uniformly distributed over a circle of area 1, is almost surely equal to $\beta\sqrt{n}$, where β is a constant that can be estimated numerically.

Three new tools

Computer science has enabled us to perform large scale computations. It has taught us to achieve efficiency and also to accept the limits to efficiency. It has indicated how to attempt a formal analysis of algorithmic behavior.

In the last few years, it has given us three new tools: *randomization*, *parallelism*, and *interaction*. Randomization and parallelism are the most important new algorithmic concepts in theoretical computer science. Parallelism and interaction have become relevant topics due to the availability of new computer architectures. And interaction and randomization represent new modes of approximation: a randomized algorithm tosses a coin at certain points in order to decide how to proceed; an interactive method is not even completely algorithmic but relies on man-machine interaction. I will make a few brief remarks on two of these subjects and spend the rest of my time on the third one: interaction.

In a randomized algorithm, the stochasticity is inside the algorithm and

not in the problem instances, as in the case of probabilistic analysis. A randomized algorithm makes guesses along the way and hence mistakes, but with a bounded probability. One of the earliest examples is Rabin's primality test. It runs in polynomial time and gives moral, although no absolute, certainty about the primality of the input number. The most topical example in operations research is the principle of simulated annealing. This is a technique for iterative improvement, based on neighborhood search, which accepts deteriorations with a small and decreasing probability in the hope of avoiding bad local optima and getting settled in the global optimum. The investigation of the randomization principle in operations research is still at a very early stage. Much can be expected along these lines in the near future.

Parallel algorithms are designed to be executed on a collection of processors that operate in parallel and communicate with each other. For certain formal models of parallel computation, algorithms have been designed for the basic problems in computer science as well as for many optimization problems. The entire theoretical approach to the complexity of problems and the efficiency and effectivity of algorithms is being extended to this area. At the more practical side, many types of parallel and pseudoparallel architectures are now becoming available, from mainframes for vectorized computations to networks of micros for distributed computations. This leads to a broad range of research questions. For computer scientists: How do the results for formal models of parallel computing translate to realistic models? For operations researchers: How do we solve problem X on architecture Y? Is it a good idea anyhow? And for everyone: Are there one or two models that will be accepted as standards for the future?

CAR: Computer Aided Routing

The vehicle I will use to discuss interactive computing is CAR, a system for computer aided routing that we have been developing in Amsterdam over the past two years.

Let me start by describing the practical decision situation in which the first release of CAR will be installed. It concerns the operational distribution planning for the hanging garment division of Van Gend & Loos, the largest Dutch road transportation firm. They have one central depot, a number of vehicles, and each day a collection of customers. Each vehicle has its own capacity. Each customer has a demand, a time window (i.e., a time interval in

which he must be visited), and a priority (indicating if the visit may be postponed until tomorrow or not). The time windows appear to be increasingly restrictive, especially for garment shops located in urban pedestrian areas. In addition to this, some customers may have a supply rather than a demand, and the combination of supply and demand may lead to precedence constraints between customers. The purpose of our involvement is to help in developing a tool for improved planning. Improved in terms of lower costs, better service to the customers, and a more even work load for the drivers.

An important requirement, which we imposed ourselves, is the functional flexibility of the system. It should enable the planner to perform the traditional planning more efficiently on the one hand, and it should be able to construct a complete plan by itself on the other hand. In different words, it should assist as an automatic scratch pad and advise as an automatic pilot.

Another requirement is of major concern to the information engineers of Van Gend & Loos. The system should be implemented on a small configuration at the loading site. But it should also be integrated into the administrative organization of the firm. In the current situation, the planning is done by hand and the registration is done at a later stage on the firm's mainframe. In the new situation, the information of each order will be fed into a microcomputer at a much earlier stage. Both machines need to be linked, and Van Gend & Loos has to seriously reconsider the structure and control of its information flows.

The mathematical model motivated by this practical decision situation is a fairly typical vehicle routing problem. We have to find a tour for each vehicle, starting and finishing at the depot and collectively visiting all customers, such that three conditions are satisfied. First, the total load allocated to any vehicle should not exceed its capacity at any point in time; this is the *clustering* aspect. Secondly, the departure time at any customer should fall inside his time window (early arrivals are allowed but lead to waiting time); this is the *scheduling* aspect. Thirdly, the total travel time should be minimized; this is the *routing* aspect.

The overall problem is very hard and you cannot hope to solve it to optimality for realistic problem sizes. In fact, each of the three aspects I mentioned represents a problem which is NP-hard, independently of the other two aspects. We have chosen to follow an approximative two-phase approach suggested by Fisher and Jaikumar: in the first phase, we cluster the customers into

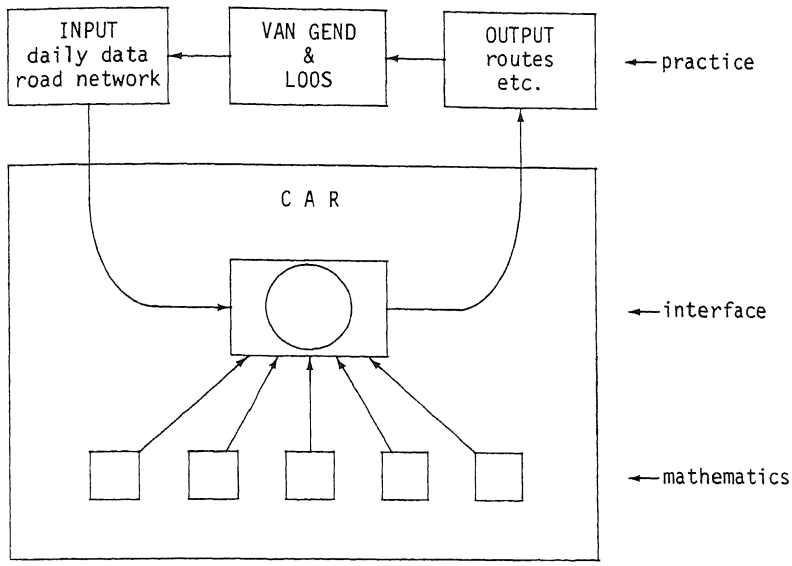


Figure 8 CAR.

vehicles by solving a generalized assignment problem; in the second phase, we route each vehicle through its customers by solving a traveling salesman problem. This does not yet take care of the scheduling aspect. Time windows are a relatively neglected complication in vehicle routing theory. We have designed routing algorithms that take account of time windows as efficiently as one could hope for, but their incorporation in the clustering phase remains a challenge.

After describing practice and sketching the mathematics inspired by it, I now want to discuss the implementation of the entire system. See Figure 8 and note that there are three levels. The top level stands for *practice*, consisting of the input to CAR, its output, and the rest of the outer world. The bottom level represents *mathematics*, a collection of algorithmic modules for solving several types of well-defined subproblems: clustering, routing, routing subject to time windows, and so on. The modular setup facilitates an extension of the system to a broader range of practical situations. Extensions that we would like to attack in the near future relate to multiple depots and heterogeneous commodities. The middle level is the core of the system: the *interface* between practice and mathematics where the man-machine interaction takes place. Its

implementation heavily relies on advanced information technology for the graphical display of data and solutions in a variety of ways. Some technical data: CAR is written in the C programming language and uses a C implementation of the Graphical Kernel System; implementations are available on an IBM PC/AT with an IBM Professional Graphics Display and on an IBM 5160 (PC/RT) with an IBM 5085 Graphics Display.

I should emphasize again that the solution approach followed by CAR is not purely algorithmic but that the man-machine interaction is an essential feature. Interaction has a threefold advantage in that it adds to effectivity, efficiency and acceptability. First, the cooperation between man and machine leads to better solutions. The machine cannot be beaten in solving well-defined detailed problems. The human planner is superior in judging fuzzy situations, in recognizing global patterns, and in observing *ad hoc* constraints which do not form part of the underlying models. Secondly, these better solutions are obtained faster, because interaction allows for flexibility in manipulating data and in selecting solutions. Finally, an interactive system is more readily accepted. The human planner is not replaced by a black box but gets a versatile tool.

Decision support systems

At the CWI we are involved in the development of other interactive planning systems in which the same design philosophy is applied. One of these concerns the operational production planning for assembly lines in the Dutch clothing industry. Rather than going into any detail here, I would like to discuss the combination of interaction and algorithmics in more general terms.

In the few formal descriptions of decision support systems which I have seen, whether they were logical, technical or functional, there was always an implicit three-level structure, which is made explicit in Figure 9. The core of the system is the man-machine interaction, the dialogue between user and computer. On the practical level, the input consists of all kinds of data and also "scenarios", i.e., solutions and strategies proposed to the system. The output is, of course, decision support. And the communication with other systems is a very important subject but it falls outside the decision support system. On the mathematical level, there is a collection of quantitative models and methods.

The system must be able to perform a broad range of functions. This range should include the assisting role of an automatic scratch pad and the advisory

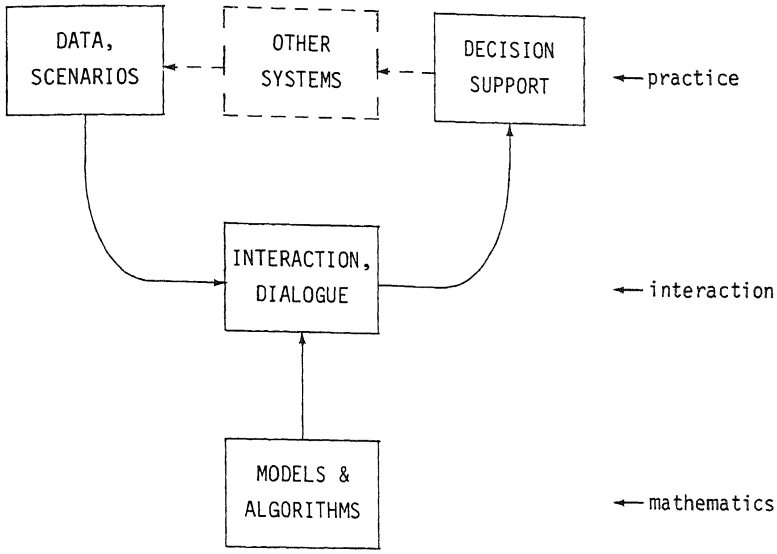


Figure 9 Decision support system.

role of an automatic pilot, which I already discussed in the context of CAR. In the first role, the interaction is essential and the computations are usually restricted to "what-if analyses", i.e., evaluations of given scenarios. In the second role, the algorithmics is essential; it should be sufficiently powerful to propose solutions of a reasonable quality.

I do not want to attempt to give anything like a definition of what a decision support system is. I do want to claim, however, that the ability to play both roles is a necessary functional condition for a system in order to qualify as such. The roles are at the extremes of the spectrum, and there is much inbetween. The aim to integrate the functions of assistant and advisor implies the need to combine interaction and algorithmics. This makes interactive decision support one of the great challenges on the interface between operations research and computer science.

Let me briefly compare traditional operations research with this concept of a decision support system. On the practical level, decision making, sometimes in terms of commands, is replaced by decision support, in terms of suggestions. On the level of information technology, the classical black box becomes transparent. On the mathematical level, the algorithms are no longer

at the core of the system. They are less visible to the user and may seem to be of secondary importance. However, for the investigator who is primarily motivated by the mathematics of operations research, a decision support system is like the wooden horse of Troy, enabling him to disguise his models and methods in an attractive fashion. Information technology provides facilities for manipulating information, but these facilities only pertain to the form. The practical situation in question has to give substance to the information, and some sort of mathematical abstraction is needed to make the manipulation meaningful.

Expert systems

I would like to spend the last few minutes on expert systems. Please do not ask me what an expert system is. As the term suggests, it should encapsulate human expertise, in terms of knowledge and inference power, for a certain application area. But it remains a very vague notion, at least in operations research. The expertise of today may be common knowledge tomorrow. Sometimes a straightforward collection of heuristic rules is called an expert system because it outperforms experts. That is fine, but it deprives the notion of its contents.

I will sketch one type of system which, in my opinion, deserves the name. It receives as input the description of a problem type or situation (rather than the data of a specific problem instance). It gives as output a mathematical model and a suggestion of a suitable algorithmic approach (rather than a specific numerical solution). Input and output are concepts of a higher order than we are used to in traditional operations research or interactive decision support. And the transformation can only be made on the basis of a formal representation of operations research expertise.

Back in 1975, we built a system of this type. It is called MSPCLASS and handles a class of 4,536 deterministic machine scheduling problems. As any expert system, it has a knowledge base and an inference engine. The knowledge base consists of a subclass of known easy problems, a subclass of known NP-hard problems, and a partial order on the entire class. The partial order is denoted by an arrow: $\rightarrow$; $X \rightarrow Y$ means that problem Y is at least as hard as problem X. The inference engine applies four rules:

- if $X \rightarrow Y$ and Y is easy, then X is easy;
- if $X \rightarrow Y$ and X is NP-hard, then Y is NP-hard;

- if X is easy and there is no easy Y with X→Y, then X is maximally easy;
- if Y is NP-hard and there is no NP-hard X with X→Y, then Y is minimally NP-hard.

MSPCLASS registrates our current knowledge by partitioning the entire class into three subclasses of easy, NP-hard and open problems. It also determines the borderlines between the subclasses by identifying the maximal and minimal problems in each subclass. This feature turned out to be helpful in suggesting future research.

In a technical sense, there is nothing sophisticated about MSPCLASS. All we need are elementary operations on the product of seven directed graphs. In a functional sense, MSPCLASS represents the bottom of the type of system I am discussing. The algorithmic suggestion implied by the complexity classification is certainly a very global one. However, in a conceptual sense, MSPCLASS fully qualifies, although we did not realize this at the time.

One of the things we intend to do in the near future is to build a system of this type for a broader input class of more practical relevance, the routing and scheduling of vehicles and crews, and also with a broader range of output statements than the bare distinction between easy and NP-hard. The system might consist of two phases: one phase in which a question-and-answer game transforms a practical decision situation into a rough model, and a second phase in which true expertise is applied to strip the rough model to a tractable model. This might lead to a third phase in which a decision support system is selected and applied.

It is in the development of such systems where the needs from practice and the possibilities offered by theory, from complexity theory to mathematical programming, are meeting.

Acknowledgement

The work which I reported on decision support and expert systems is carried out by the combinatorial optimization group at the Centre for Mathematics and Computer Science (CWI) in Amsterdam. The views which I expressed on these subjects are based on discussions with members of the group. I am pleased to acknowledge the contributions by J.M. Anthonisse, G.A.P. Kindervater, B.J. Lageweg, and M.W.P. Savelsbergh.

Bibliographical notes

A volume of annotated bibliographies [O'hEigeartaigh *et al.*, 1985] provides classified reviews of the recent literature in four interface areas, namely complexity [Papadimitriou, 1985], probabilistic analysis [Karp *et al.*, 1985], randomization [Maffioli *et al.*, 1985], and parallelism [Kindervater & Lenstra, 1985]. A tutorial introduction to the latter subject is given by Kindervater & Lenstra [1986]. For the Fisher-Jaikumar algorithm and much more on vehicle routing, see the annotated bibliography and the survey by Christofides [1985a, 1985b]. MSPCLASS is described by Lageweg *et al.* [1982].

References

N. Christofides (1985a). Vehicle routing. [O'hEigeartaigh *et al.*, 1985], 148-163.

N. Christofides (1985b). Vehicle routing. [Lawler *et al.*, 1985], 431-448.

R.M. Karp, J.K. Lenstra, C.J.H. McDiarmid, A.H.G. Rinnooy Kan (1985). Probabilistic analysis. [O'hEigeartaigh *et al.*, 1985], 52-88.

G.A.P. Kindervater, J.K. Lenstra (1985). Parallel algorithms. [O'hEigeartaigh *et al.*, 1985], 106-128.

G.A.P. Kindervater, J.K. Lenstra (1986). An introduction to parallelism in combinatorial optimization. *Discrete Appl. Math. 14*, 135-156.

B.J. Lageweg, J.K. Lenstra, E.L. Lawler, A.H.G. Rinnooy Kan (1982). Computer-aided complexity classification of combinatorial problems. *Comm. ACM 25*, 817-822.

E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (eds.) (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester.

F. Maffioli, M.G. Speranza, C. Vercellis (1985). Randomized algorithms. [O'hEigeartaigh *et al.*, 1985], 89-105.

M. O'hEigeartaigh, J.K. Lenstra, A.H.G. Rinnooy Kan (eds.) (1985). *Combinatorial Optimization: Annotated Bibliographies*, Wiley, Chichester.

C.H. Papadimitriou (1985). Computational complexity. [O'hEigeartaigh *et al.*, 1985], 39-51.