Mathematical models for planning support

Leo G. Kroon, Rob A. Zuidwijk

ERIM REPORT SERIES *RESEARCH IN MANAGEMENT*	
ERIM Report Series reference number	ERS-2003-032-LIS
Publication	April 2003
Number of pages	30
Email address corresponding author	l.kroon@fbk.eur.nl
Address	Erasmus Research Institute of Management (ERIM) Rotterdam School of Management / Faculteit Bedrijfskunde Rotterdam School of Economics / Faculteit Economische Wetenschappen Erasmus Universiteit Rotterdam P.O. Box 1738 3000 DR Rotterdam, The Netherlands Phone: +31 10 408 1182 Fax: +31 10 408 9640 Email: info@erim.eur.nl Internet: www.erim.eur.nl

Bibliographic data and classifications of all the ERIM reports are also available on the ERIM website:
www.erim.eur.nl

ERASMUS RESEARCH INSTITUTE OF MANAGEMENT

REPORT SERIES

RESEARCH IN MANAGEMENT

BIBLIOGRAPHIC DATA AND CLASSIFICATIONS		
Abstract	In this paper we describe how computer systems can provide planners with active planning support, when these planners are carrying out their daily planning activities. This means that computer systems actively participate in the planning process by automatically generating plans or partial plans. Active planning support by computer systems requires the application of mathematical models and solution techniques. In this paper we describe the modeling process in general terms, as well as several modeling and solution techniques. We also present some background information on computational complexity theory, since most practical planning problems are hard to solve. We also describe how several objective functions can be handled, since it is rare that solutions can be evaluated by just one single objective. Furthermore, we give an introduction into the use of mathematical modeling systems, which are useful tools in a modeling context, especially during the development phases of a mathematical model. We finish the paper with a real life example related to the planning process of the rolling stock circulation of a railway operator.	
Library of Congress Classification (LCC)	5001-6182	Business
	5201-5982	Business Science
	HD 30.213	Decision support systems
Journal of Economic Literature (JEL)	M	Business Administration and Business Economics
	M 11 R 4	Production Management Transportation Systems
	C 59	Modelling, other
European Business Schools Library Group (EBSLG)	85 A	Business General
	260 K 240 B	Logistics Information Systems Management
	240 C	Decision support systems
Gemeenschappelijke Onderwerpsontsluiting (GOO)		
Classification GOO	85.00	Bedrijfskunde, Organisatiekunde: algemeen
	85.34 85.20	Logistiek management Bestuurlijke informatie, informatieverzorging
	85.03	Methoden en technieken, operations reseach
Keywords GOO	Bedrijfskunde / Bedrijfseconomie	
	Bedrijfsprocessen, logistiek, management informatiesystemen	
	Planning, beslissingsondersteunende informatiesystemen, modellen, optimalisatie	
Free keywords	Planning, planning support, mathematical models, optimization, modeling process	

Mathematical models for planning support

Leo G. Kroon[1,2]

Rob A. Zuidwijk[1]

[1] Erasmus University Rotterdam
Rotterdam School of Management
P.O. Box 1738, NL-3000 DR, Rotterdam
The Netherlands

[2] NS Reizigers, Department of Logistics
P.O. Box 2025, NL-3500 HA Utrecht
The Netherlands

Abstract: In this paper we describe how computer systems can provide planners with active planning support, when these planners are carrying out their daily planning activities. This means that computer systems actively participate in the planning process by automatically generating plans or partial plans. Active planning support by computer systems requires the application of mathematical models and solution techniques. In this paper we describe the modeling process in general terms, as well as several modeling and solution techniques. We also present some background information on computational complexity theory, since most practical planning problems are hard to solve. We also describe how several objective functions can be handled, since it is rare that solutions can be evaluated by just one single objective. Furthermore, we give an introduction into the use of mathematical modeling systems, which are useful tools in a modeling context, especially during the development phases of a mathematical model. We finish the paper with a real life example related to the planning process of the rolling stock circulation of a railway operator.

1. Introduction

In this paper we describe how computer systems can provide planners with active planning support, when these planners are carrying out their daily planning activities. This means that computer systems actively participate in the planning process by automatically generating plans or partial plans. Later on, these plans can be completed and fine-tuned manually by the planners. Automatically generating plans by a computer system requires the application of mathematical *models* and mathematical *solution techniques*, as will be described in this paper. Here a mathematical model is an abstract representation of the practical planning problem in mathematical terms. Solving the model leads to a solution of the planning problem, which is an abstract representation of the required plan. The currently available hard- and software for solving mathematical models allows for the generation of appropriate solutions for practical planning problems, both in a pro-active and in a re-active way.

The advantages of the automatic generation of plans are obvious: planning support based on mathematical models may lead to *better* plans, but it may also lead to a *reduction* of the *throughput*

time of the planning process. The value of better plans comes almost by definition: a better plan may lead to cost reductions, improved service performance, or other advantages. Furthermore, a shorter throughput time is important because of the flexibility of the planned system: the shorter the throughput time of the planning process, the quicker the planned system can accommodate to changes in the environment. It may also reduce the need for re-active planning. Another advantage of planning support based on mathematical models is the fact that it allows for the generation of several plans at the same time based on different scenarios, instead of the generation of just one single plan.

Mathematical models and solution techniques may be used in a variety of application areas to support both long term and short term planning processes. For example, there are applications in the financial world, in transportation planning, in production planning, in marketing, in manpower planning, etc. Here a planning process is interpreted as a decision making process aiming at the selection of appropriate actions that are to be carried out in practice such that the planned system will achieve certain objectives (Ackoff, 1962). The planned system may be large, such as a company that wants to minimize its costs or to maximize its service performance. It may also be small, such as a truck driver who wants to reach his destination along the shortest or the fastest path as possible.

The remainder of this paper is structured as follows. In Section 2, we start with an example of a simple planning problem, and we develop a mathematical model that can be used to solve this problem. This example is used to explain certain concepts, and serves as a reference in the remainder of this paper. In Section 3, we generalize the description of Section 2 by presenting the planning process based on mathematical models in more general terms. Section 4 lists a couple of modeling techniques as well as a couple of solution techniques. This section also presents some background information on the theory of computational complexity, which is highly relevant in a planning context. In Section 5, we describe several methods that can be used to deal with multiple objectives. Section 6 describes how mathematical models can be used to support both *pro-active* and *re-active* planning processes. Section 7 discusses modeling systems, which are quite useful tools in a modeling context, especially during the development phase of a mathematical model. In Section 8, we describe some issues that are relevant when applying mathematical models in practice. Section 9 presents a real life example related to the planning process of the rolling stock circulation of a railway operator. The paper is finished in Section 10 with some conclusions and final remarks.

2. A simple Planning Problem

We start with the description of a simple planning problem and we discuss several modeling issues and solution methods based on this example. The example is used to explain certain concepts and as a reference in the remainder of the paper. The example problem deals with a number of “jobs” that need to be processed on a number of “machines”. Each of the jobs requires a certain amount of processing time on either of the machines. The problem that needs to be solved is: “use the machine capacity in an efficient way while processing the jobs”. More specific descriptions of “efficient use” are: “process all jobs in as little time as possible, given the number of available machines” (the required amount of time is called the *Makespan*), or “use as little machines as possible, given a certain amount of time in which all jobs need to be processed”. Later on, we shall discuss decision making on the assignment of jobs to machines where these two objectives are combined.

We shall first present a tiny example of an instance of such a problem. In this example, there are two machines and there are six jobs to be processed on either of the machines. The processing time of job j (here j is an integer value running through the values 1,…,6) is denoted in an abstract way as p_j. Table 1 provides the processing times for the six jobs (in hours):

Job number j	1	2	3	4	5	6
Processing time p_j	8	7	4	3	3	3

Table 1. Processing times of the jobs.

One remark should be made about these processing times. Obviously, it is assumed here that the processing times are all known in advance. In practice, such data is typically obtained through experience in the past. One can use average processing times for each job type. More advanced models than the one presented here may take into account variability of processing times by introducing stochastic processing times and by using stochastic optimization techniques. However, this strongly complicates the situation and falls outside the scope of this introductory example. Observe further that we implicitly assume that the processing times of the jobs are independent of the machines. In other words, the machines can be considered as identical for this set of jobs. After the model formulation of this simple planning problem, we shall discuss other variants of this problem, which is usually referred to as a "job scheduling problem" (Baker, 1975).

Although the problem formulation may seem quite concrete, it is rather abstract in fact. This is due to the fact that the notions "machine", "job" and "processing time" refer to the structure of the problem more than to actual objects in reality. Indeed, this model can be applied to situations where "machines" are teams of people working on a project, a clerk working on a document, a straddle carrier moving a maritime container on a container terminal, and so on. Of course, in each of these cases, the problem formulation may require further specification in one direction or the other.

This paper focuses on mathematical models, so let's *make* a mathematical model that can be used to tackle the job scheduling problem! The aim of the model should be to generate good decisions about the allocation of the jobs to the machines. Therefore, we need "decision variables" that are used to represent the decisions to be taken. Furthermore, in order to specify what is meant by "good" decisions, we need an "objective function" that expresses what we are aiming at. Moreover, there are certain "constraints" that we want our decisions to respect, such as the fact that each job is to be carried out on one of the machines and not on both. We are only interested in decisions that respect all the specified constraints, i.e. the decisions that are "feasible".

In order to "model" this job scheduling problem, we introduce decision variables that are to describe how the jobs are allocated to the machines. If we want to express that job j is processed on machine m, we state that the value of the decision variable $A_{j,m}$ equals one. Otherwise, the value of this decision variable equals zero. Formally, $A_{j,m}$ is a binary decision variable (binary means that it only assumes the values 0 or 1) satisfying:

$$A_{j,m} = 1 \text{ if and only if job } j \text{ is processed on machine } m \qquad (1)$$

Note that the indicated decision variables are indeed *variable*, in the sense that values for these variables are not known in advance (i.e. before the model has been solved). Another decision variable that we need is the *Makespan* denoting the makespan corresponding to a certain assignment of jobs to machines. The latter is equal to the total time required to process all jobs. In this example, we state as objective that the *Makespan* is to be minimized. In the remainder of this paper, we shall identify decision variables by using capitals as opposed to input parameters such as the processing times p_j.

In order to get a feasible plan, the decision variables $A_{j,m}$ need to satisfy several constraints. First, each job is to be processed on exactly one machine. This is expressed by the following constraints:

$$A_{j,1} + A_{j,2} = 1 \qquad \text{for } j = 1,\ldots,6 \qquad (2)$$

Since $A_{j,m}$ only assumes the values 0 or 1, these constraints state that either $A_{j,1} = 1$ or $A_{j,2} = 1$. This is equivalent to stating that job j is to be processed either on machine 1 or on machine 2.

Furthermore, the *Makespan* is not less than the total processing times on each of the machines. This can be expressed by the following constraints:

$$\sum_j p_j A_{j,m} \le Makespan \qquad \text{for } m = 1, 2 \qquad (3)$$

For m=1, this constraint states that the total processing time on machine 1 equals the sum of the processing times of the jobs that have been assigned to machine 1. Indeed, the processing time p_j of job j is added to the left-hand side of this constraint only if the decision variable $A_{j,1}$ has the value 1, and otherwise it is not added. For m=2, a similar explanation holds. Now the constraints state further that the *Makespan* is not less than the total processing times on both machines.

Figure 1 recapitulates the complete mathematical model, whose solution provides an assignment of jobs to machines resulting in a minimum *Makespan*, in a more general form. Here the number of jobs is denoted by J and the number of machines is denoted by M.

Minimize *Makespan*

by changing the binary variables

$A_{j,m}$ $\quad j = 1,..,J,\ m = 1,..,M$

under the constraints

$\sum_m A_{j,m} = 1 \quad$ for $j = 1,..,J$

$\sum_j p_j A_{j,m} \le Makespan \quad$ for $m = 1,..,M$

$A_{j,m} \in \{0,1\} \quad$ for $j = 1,..,J,\ m = 1,..,M$

Figure 1. A mathematical model for solving the simple job scheduling problem.

The model can be solved by using an appropriate mathematical solution technique. Here “solving” the model refers to the process of computing appropriate values for the decision variables, such that all constraints are satisfied and such that the objective function has an optimal value. The latter means that none of the feasible assignments of jobs to machines has a strictly lower *Makespan*.

The model described here is an Integer Programming model. For solving such a model, one may use a so-called Branch & Bound solution technique as is provided by general purpose solvers such as CPLEX or LINDO. An optimal solution for the instance described in Table 1 is shown in Figure 2. Since this instance is very small, the computation time for finding an optimal solution is negligible.

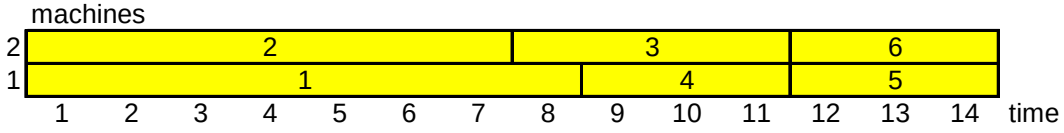

Figure 2. Optimal solution for the simple job scheduling problem.

```
Makespan = 14

A(1,1) = 1          A(2,1) = 0          A(3,1) = 0
A(1,2) = 0          A(2,2) = 1          A(3,2) = 1

A(4,1) = 1          A(5,1) = 1          A(6,1) = 0
A(4,2) = 0          A(5,2) = 0          A(6,2) = 1
```

Table 2. Optimal solution in terms of the decision variables.

The tabular representation of the solution shown in Figure 2 is listed in Table 2. Obviously, the solution is not unique. For example, the jobs 4 or 5 can be exchanged with job 6. Furthermore, since the machines are identical, they can be interchanged as well. In other words, replacing $A_{j,m}$ by 1- $A_{j,m}$ for all jobs and both machines gives another feasible schedule with the same makespan.

In the case that the number of machines is variable (as opposed to the example described so far), the makespan of the optimal solution clearly depends on the number of machines involved. Calculation shows a relation between the *Makespan* of the optimal solution and the *Number of Machines* in Figure 3. Indeed, the ideal *Makespan* would follow the relation

$$Makespan \times Number\ of\ Machines = 28,$$

which would mean a full utilization of the available machines during the processing of the jobs. Figure 3 shows the relation between the minimum *Makespan* and the available number of machines. The lower "curve" in this figure indicates full machine utilization, assuming that jobs can be cut into subjobs of any size. The latter is also known as *pre-emptive* processing of the jobs.

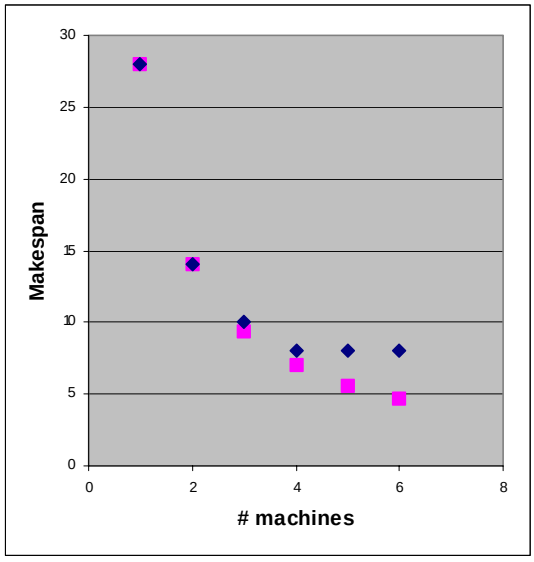

Figure 3. Trade-off between *Makespan* and *Number of Machines*.

There are several variants of the scheduling problem and hence to the mathematical model that are worth mentioning. First of all, one may consider the situation that the machines are not identical, and

that the jobs need to be processed first on machine 1 and then on machine 2. In such cases, a job often refers to a (physical) object that requires consecutive processing on the two machines. For example, machine 1 adds a circuit board to a computer assembly, while machine 2 adds the cover and seals the product. Both processes related to each job (in this case a computer assembly) take a certain amount of time on each of the machines and in a fixed order (first the process on machine 1, and then the process on machine 2). The preceding model did not take into account such precedence relations. This variant of the planning problem is also known as the Two Machine Flow Shop Problem (see Johnson, 1954).

Other job scheduling problems are described in Baker (1975). For example, a more complex variant of the Two Machine Flow Shop Problem is the Two Machine Job Shop Problem. In this variant there are again jobs that need to be processed on two machines, but in this case the order in which the jobs are to be processed on the machines is not the same for all jobs: each job may have its own routing along the machines. Further obvious variants are the general Flow Shop Problem and the general Job Shop Problem with more than two machines.

Another issue that may be relevant in practice has to do with the dynamic character of practical scheduling problems. This dynamic character may be modeled to some extent by introducing so-called *release* dates and *due* dates for the jobs: the processing of a job on a machine cannot start before the job's release date and is to be completed before the job's due date.

Although, at first sight, these variants of the scheduling problem may seem to be rather similar, there are strong differences between them. Especially from the point of view of mathematical models, solution techniques and computational complexity, there are strong differences, as is explained later.

An important representative of a *routing* problem with many important applications in practice is the so-called Traveling Salesman Problem (Lawler et al., 1985). Here the problem is to find the shortest route along a number of locations that returns in the end to the same place where the route started initially. The Traveling Salesman Problem has many important applications.

3. The mathematical modeling process

As we have seen in the preceding discussion, mathematical models and solution techniques can be used to support decision making processes in practice. First, one needs to model certain parts of reality. Here one can choose between a large variety of model types, and it requires some skills (besides mathematical ones) to choose the appropriate type of model, next to choosing, for example, the appropriate level of detail of the model. Pidd (1999) describes the modeling process as "Just Modeling Through". This puts emphasis on the development of the model as a process (the modeling process) instead of on the end product (the model). Mathematical models can be seen as entities on their own, which can be used to solve planning problems. However, this approach underexposes important aspects of problem solving by means of models. Indeed, let's consider the modeling process in somewhat more detail. This process has been represented schematically in Figure 4.

Generating a proper *Problem Formulation* is difficult and not without ambiguities. For example, a planner may consider the legacy approach as satisfying or even as irreplaceable, while at a management level one agrees on extended automatic support of the planning process. The scope of the problem may also depend on the viewpoint of the problem owner. Indeed, if one chooses the scope of the problem too small, then a solution to the posed problem will not solve the actual problem, and if one chooses the scope of the problem too large, then one will probably be forced to oversimplify. The problem owner, who states the problem and sponsors the development of the model, and the planners, who are supposed to use the model in practice, need not be the same persons nor share the same viewpoints, which may complicate their communication with the model builder. The model builder should provide a *Model Formulation* that rephrases the problem in terms of a mathematical model.

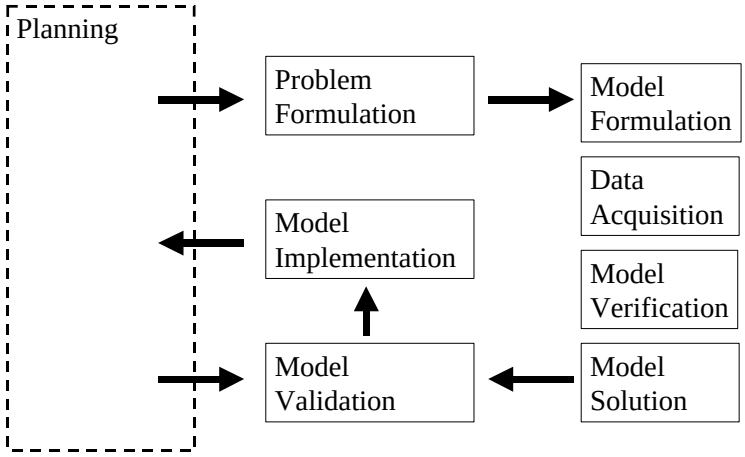

Figure 4. The mathematical modeling cycle.

One of the advantages of the modeling process is the fact that alternatives, constraints and objectives have to be described explicitly during this process. In practice, the discussions that should lead to this explicit description of alternatives, constraints and objectives are sometimes already as valuable as the final solutions generated by the model, since they generate a lot of insight into the planning process itself and into the hierarchy of preferences that is used by the planners.

In practice, it may be necessary to adapt the original problem formulation to the modeling possibilities. Indeed, one may need to restrict oneself to a tractable sub-problem, or reformulate it in such a way that both the required input and the required output are quantities that are measurable and that make sense. For example, it may be necessary to state the “quality of resources” in more specific terms (e.g. productivity or flexibility) or to state that a “good” schedule is a schedule that respects the due dates and costs less than a certain amount of money.

The formulation of the model itself starts with the choice for a specific model type. Some examples of these are presented in Section 4. Important decisions in formulating the model further are the model size (e.g. the level of detail), the scope of the model (what should be included and what not?), the decision variables (as opposed to fixed parameters), the decision objective(s), and the constraints.

Mathematical models need to be fed with a lot of data, require *Data Acquisition* (see Figure 4). In the modeling process, this certainly is an issue. It does not make sense to make a large model that requires a lot of data, if such data are not available. Furthermore, the phrase “Garbage In, Garbage Out” expresses the need for high quality input data. On the other hand, one should not focus on the available data too much, as it may blur the requirements to solve the problem at hand (“Don’t fall in love with data”, as Pidd says).

Figure 5 represents the general situation, where data (input parameters) are fed into the model, and where the model provides decisions and an objective value as outcome. The decisions are represented by the values of the decision variables of the model, which are determined by applying an appropriate solver to the model. This process is described in Section 4.

As mathematical models may be rather complicated, they need to be tested: does the model actually compute what the model builder expects it to compute? If the model provides “optimal solutions” that do not satisfy the posed constraints, something is obviously wrong. It can be concluded that thorough *Model Verification* (see Figure 4), i.e. verifying that the model and the applied solution techniques are correct, is required.

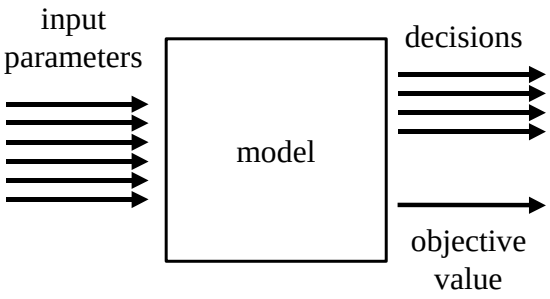

Figure 5: Input (data) and output (decisions, objective value) of a model.

Nevertheless, even if the model works all right, the outcomes, the *Model Solution*, need not be valid. Non-realistic parameter values or other fatal deviations from reality may result in model outcomes that are not usable in practice. Since a model always describes only certain parts of reality and in an abstract way, this so-called *Model Validation* (see Figure 4) can be a serious hurdle. In this stage, feedback from the planners is indispensable, since the solutions generated by the model should be acceptable to the planners. Usually a first evaluation of the obtained solutions by the planners leads to new alternatives or to new constraints that are to be taken into account. However, after several iterations, this process may lead to a model and solutions that are indeed acceptable.

After one has obtained an initial solution that seems to be acceptable in practice, a next useful step is to carry out an extensive *sensitivity analysis*. This is especially the case if parts of the input data are uncertain or only approximately known, or if several scenarios with different input values need to be discerned. To that end, one may study the response of the model outcomes to (small) changes in the input parameters (see also Figure 5). Somewhat related to sensitivity analysis is the notion of a *robust* solution to an optimization problem. An optimal solution is called *robust* when it is optimal or nearly optimal for a wide range of values for the input parameters. One may also require that the objective value of the solution is not too sensitive to changes within this input parameter range.

If one agrees that the model provides valid output, then one can start with *Model Implementation* (see Figure 4). This final step usually comes down to establishing a tool based on the model that can actually used by the planners. For that reason, the model has to be implemented into a user-friendly decision support system, since only experts in Operations Research or Mathematics know how to work with a mathematical model in its basic form. However, a model becomes much more effective in practice if it can be used by the planners themselves, who are the real domain experts. A decision support system should allow the planners to specify the scenarios to be studied in a user-friendly way based on the available data, and it should provide appropriate solutions for these scenarios within an acceptable amount of time, preferably in cooperation with the planners.

Obviously, there are many more requirements for a decision support system, but in this paper we will not elaborate on these. We only want to note here that the specifications of such a decision support system may lead to a loop back to the formulation of the underlying mathematical models. Therefore, the step-by-step description in Figure 4 should be enriched with feedback loops that result in a process referred to as “prototyping”. Finally, it is not uncommon that, once a decision support system is operational in practice, it leads to new questions that the users of the system may want to answer with it. The implementation of such functionalities into the system may require the adaptation or the extension of the underlying mathematical models. It can be concluded that the modeling cycle depicted in Figure 4 actually never ends in practice.

4. Mathematical models, algorithms, and complexity

As was indicated already in Section 3, mathematical models exist in a large variety. Relevant classes that can be distinguished are the class of *optimization* models and the class of *simulation* models. Optimization models are used to determine a set of decisions satisfying certain restrictions that are "optimal" in a certain sense. Therefore, such models are also called "how-to" models. For example, how to assign jobs to machines, or how to minimize the operational costs of the production process?

On the other hand, a simulation model provides an abstract description of a real-life system, and allows one to study the behavior of this system by simulating the system under varying conditions. Therefore, simulation models are also called "what-if" models. For example, what happens with the customer waiting time if we double the number of counters? Simulation models may be used to evaluate a system along several dimensions. Improvements of the system can be studied by varying the operating conditions of the system in the simulation model.

Other classes of models that can be distinguished are the classes of *deterministic* and *stochastic* models. Within a deterministic model, all input data are assumed to be deterministic and given, whereas in a stochastic model some of the input data has a stochastic character.

Within an *operational* planning context, it seems that *deterministic optimization models* are more relevant than other types of models. First, in a planning context, one wants to know how certain processes are to be carried out. Hence for providing active support in the planning process, optimization models seem to be more appropriate than simulation models. Second, if the planning process has an operational nature with a relatively short planning horizon, then it is usually defendable to assume that the input data is deterministic. The latter holds in particular because stochastic optimization models are usually much more complex than deterministic ones. Nevertheless, it is obvious that also in operational planning processes, stochasticity may play an important role. However, this may also be handled by applying an adequate sensitivity analysis afterwards.

Usually, models that are used for supporting *tactical* or *strategic* planning process are of a different nature than models for supporting operational planning process. The former have to take into account a longer time horizon, which may lead to a higher level of uncertainty. In this case, an extensive analysis of several scenarios may be even more important than in an operational planning context. Furthermore, in such models usually a lower level of detail is taken into account.

4.1 Deterministic optimization models

Within the class of deterministic optimization models for supporting operational planning processes, at least the following modeling techniques can be distinguished:

- Linear Programming
- (Mixed) Integer Programming
- Network Modeling
- Non-Linear Programming
- Goal Programming
- Dynamic Programming
- Constraint Programming

In this paper, we do not go into the details of the mentioned modeling techniques, since we consider the details of these techniques to lie outside the scope of this paper. Such details can be found in any Operations Research textbook (e.g. Hillier and Lieberman, 1967, or Wagner, 1975). Anyway, the common elements of all techniques are: (*i*) a representation of a set of potential decisions in terms of decision variables, (*ii*) a set of restrictions to be satisfied, represented by a set of mathematical constraints, and (*iii*) one or more objective functions that describe the objective(s) to be pursued in the

planning process. The model for solving the simple job scheduling problem described in Section 2 is an Integer Programming model. In practice, the most widely used modeling technique is Linear Programming, probably due to the fact that it is a relatively simple technique for which many powerful solvers and other tools are commercially available.

The techniques for solving a mathematical model are usually related to the applied modeling technique. For example, for solving a Linear Programming model, one may apply the Simplex Method (Dantzig, 1948). For solving Mixed Integer Programming models, one may apply Branch & Bound or Branch & Cut methods (Gomory, 1958). Columns generation methods are also popular nowadays (Barnhart et al., 1998). And for solving Network Models, one may apply techniques for finding Shortest Paths, Minimum Spanning Trees, Matchings, or Single Commodity Flows, such as Maximum Flows or Minimum Cost Flows (Gondran and Minoux, 1984).

Solutions for a certain problem instance of a deterministic optimization problem may also be generated by applying general purpose approximation techniques such as Simulated Annealing (Aarts and Van Laarhoven, 1987) or Tabu Search (Glover and Laguna, 1997), or by applying heuristics. Again, the details of the mentioned techniques fall outside the scope of this paper, but the following section presents some relevant considerations.

The choice for using a certain modeling technique may be guided by several aspects. First, in a certain application, one technique may be more appropriate than another. For example, if a certain problem is known to be *NP-hard*, then Linear Programming and Network techniques will not be sufficiently powerful for solving it, as will be explained in the next section. Similarly, if the involved problem has certain structural non-linear elements that are so dominant that linearization will lead to unacceptable results, then one will have to use non-linear optimization techniques. Hence, knowledge of the fundamental characteristics of a certain problem may be a useful guide in selecting an appropriate modeling technique. The available software may also be useful here. Software for Linear Programming and Mixed Integer Programming is usually more easily accessible than software for other techniques. Therefore, they are more often used than other techniques. Besides these aspects, each modeler usually has a preference of his own for certain techniques based on personal experience.

4.2 Computational complexity

In practice, it turns out that some problems are much harder to solve than others. Whereas Linear Programming or Network models with hundreds of thousands of decision variables or constraints can typically be solved easily to optimality using the currently available hard- and software, Mixed Integer or Non-Linear models with several hundreds of variables may be too complex already to be solved. On the other hand, also with Mixed Integer or Non-Linear models one may be lucky and be able to solve quite large instances within a reasonable amount of time. Anyway, the progress in computational power that was achieved over the last decades, both by improvements in the available hardware and by algorithmic improvements, is tremendous, as is witnessed by Bixby (2002).

The foregoing has to do with the intrinsic computational complexity of the underlying problems. The study of this subject started around 1970 with the seminal work by Cook (1971). The theory of computational complexity usually relates the computational effort for solving a certain problem instance to the size of the instance. For example, in a job scheduling problem, a larger number of jobs J usually leads to a longer computing time. Then the way this computing time depends on the number of jobs is an indication of the complexity of the problem: a problem for which the computing time is bounded by J^3 is preferable over a problem for which the computing time is bounded by 2^J.

Details on the theory of computational complexity can be found in Garey and Johnson (1979). This theory is basically related to the complexity of decision problems: problems that can be answered

either by “yes” or by “no”. This is not a loss of generality, since it is easy to transform an *optimization* problem into a series of *decision* problems. The theory of computational complexity distinguishes on one hand the class of decision problems whose instances can be solved in a polynomial amount of time. The latter means that there exists an exponent *m* such that, if the number *N* is a measure for the size of an instance of the problem, then the time required for solving the instance is bounded by N^m. This class is called the class of polynomially solvable problems, or the class *P*. Problems in *P* are usually considered as well solved, since for most problems in *P* even large instances can be solved in a reasonable amount of time, especially if the exponent *m* is low. Examples of problems in *P* are problems that can be described by Linear Programming models and by several Network models, such as Shortest Path problems, Matching problems, and Single Commodity Flow problems.

A second class of problems that is distinguished in the theory of computational complexity is called the class of Non-deterministic Polynomially solvable problems, or the class *NP*. This class contains the problems with the property that a proposed solution for an instance of the problem can be *verified* to be feasible or not in an amount of time that is polynomial in the size of the instance. Obviously, problems in *P* also belong to *NP*, since *generating* a solution is more difficult than *checking* the feasibility of a proposed solution. However, the class *NP* also contains a large set of problems for which, as of today (2002), no algorithm has been found that can generate feasible solutions in a polynomial amount of time. An example of such a problem is the earlier mentioned Traveling Salesman Problem. It was recognized by Cook (1971) that many of these *hard* problems in *NP* are equivalent in computational complexity in the following sense: there exists a subset of hard problems in *NP* with the property that the existence of a polynomial time algorithm for solving one of these problems would imply the existence of a polynomial time algorithm for all others in this subset. Therefore this subset of these hard *decision* problems in *NP* was called *NP*-complete. The subset of hard *optimization* problems was called the set of *NP*-hard problems.

The class of *NP*-hard problems currently contains problems of many different kinds for which polynomial time algorithms are lacking despite enormous research efforts over many years. Therefore many researchers consider it as highly unlikely that a polynomial algorithm for one of these *NP*-hard problems will ever be found. However, a positive result of these research efforts is that it has given a better insight into the boundary between the problems in *P* and the *NP*-hard problems. For example: the well-known Two-Machine Flow Shop problem can be solved easily in polynomial time by applying Johnson’s rule (Johnson, 1954), but the Three Machine Flow Shop Problem is already *NP*-hard (Baker, 1975). This research has also revealed several special variants of the *NP*-hard hard Traveling Salesman Problem that belong to the set *P* (Lawler et al., 1985).

However, most practical optimization problems are known to be *NP*-hard, due to the many complicating issues that are to be taken into account. Therefore, searching for an algorithm that provides optimal solutions for all instances of these problems is probably not successful. Nevertheless, it may happen that large instances of these problems can be solved to optimality, since, if a problem is known to be *NP*-hard, then this is only a statement about the worst-case instances of the problem. It may happen that practical instances of an *NP*-hard problem can be solved much more easily than the sometimes pathological worst-case instances that are responsible for the problem’s *NP*-hardness.

Figure 6 represents the problem classes *P*, *NP* and *NP*-complete in a schematic way (based on the assumption that the classes *P* and *NP*-complete are disjoint). Note that the given classification of problems is just a rough one. Currently, also more subtle classifications exist. For example, within the class *NP*, one may distinguish between the problems that are *NP*-complete in the *weak* sense and those that are *NP*-complete in the *strong* sense. Furthermore, many problems do not belong to the class *NP*. For example, in the simple job scheduling problem described in Section 2, the problem of finding *all*

schedules that meet a certain *Makespan* does not belong to the class *NP*, simply because the number of such schedules is not polynomial in the number of jobs, and thereby not in the size of the input.

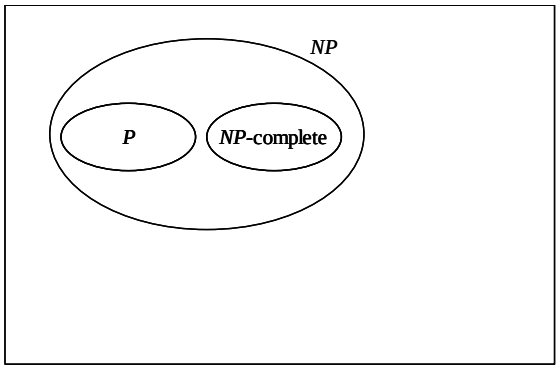

Figure 6. The problem classes *P*, *NP*, and *NP*-complete.

4.3 Algorithms and heuristics

For solving a mathematical optimization model one may use a *general purpose* solver, which basically can be applied for all problems within a certain class, or by a *special purpose* solver, which has been developed especially for solving instances of a single specific problem.

The commercially available solvers CPLEX and LINDO are examples of general purpose solvers, which can be used for solving problems that have been modeled as Linear or Mixed Integer programs. An advantage of such a general purpose solver is the fact that it provides a lot of flexibility: as soon as one has an appropriate description of a problem in terms of a mathematical model and one also has the required data for an instance, one can use the solver to find solutions for the instance. However, a disadvantage of a general purpose solver is the fact that it may not be able to solve instances of problems of a certain practical size, due to its general purpose character. This may be a consequence of the fact that the problem at hand belongs to the class of *NP*-hard problems.

If this is indeed the case, then one will be forced to develop a special purpose solver, which may better take advantage of the special structure of the involved problem. Unfortunately, developing a special purpose solver usually takes a lot of creativity and time, and, as a consequence, also a lot of money. An additional problem is that a special purpose solver may be less flexible than a general purpose solver: as soon as the characteristics of the problem change over time, which happens quite often in practice, then the special purpose solver may not be able to handle the modified problem adequately, whereas the general purpose solver is still appropriate after modification of the model.

An alternative to a solver that determines solutions for which it is guaranteed that they are optimal, given the involved objective function, is a *heuristic*. Heuristics are solution methods that are usually based on approximation techniques, rules of thumb, or on a decomposition of the problem into less complex subproblems. The solutions provided by a heuristic may not be optimal, given the involved objective function, but they may be “acceptable” or “good”. The latter means that the solutions are at least as good as the solutions that have been obtained manually. Furthermore, an appropriate heuristic determines such acceptable or good solutions in a fraction of the time that is required by an optimal solver. However, it should be noted that a really simple heuristic that is developed for solving a very complex problem may not lead to acceptable solutions in practice. In fact, planners usually have so much planning experience that they easily outperform a heuristic if the latter is too simple.

A heuristic can be classified as a *generating* heuristic or as an *improvement* heuristic. A generating heuristic generates an initial solution for a certain problem instance, and an improvement heuristic

tries to improve an initial solution. For example, a simple generating heuristic for the earlier mentioned Traveling Salesman Problem is the well-known Nearest Neighbor rule (Lawler et al., 1985). A class of simple heuristics, parameterized by the parameter δ, for solving instances of the simple job scheduling problem described in Section 2 could be the following:

1. Sort the jobs in order of decreasing processing time
2. Compute the average processing time per machine *T*: $T = \frac{1}{2}\left(\sum_j p_j\right)$
3. Set $T_1 = T_2 = 0$
4. For *j* := 1 to *J* do
   If $T_1 + p_j \le T + \delta$
   Then assign job *j* to machine 1, and set $T_1 := T_1 + p_j$
   Else assign job *j* to machine 2, and set $T_2 := T_2 + p_j$

In other words, "long" jobs are assigned to machine 1 as long as the total processing time on machine 1 does not exceed the value $T + \delta$. In this heuristic, the processing time of the sorting step 1 is proportional to *J*log(*J*). The processing time of the assignment step 4 is obviously linear in *J*. Since the running time of such a heuristic is obviously extremely low, one may try this heuristic for several values of the parameter δ and retain the best result. If this very simple generating heuristic is applied to the instance of the simple job scheduling problem described in Section 2 with the parameter $\delta = 0$, then it produces the schedule with *Makespan* 16 which is represented in Figure 7.

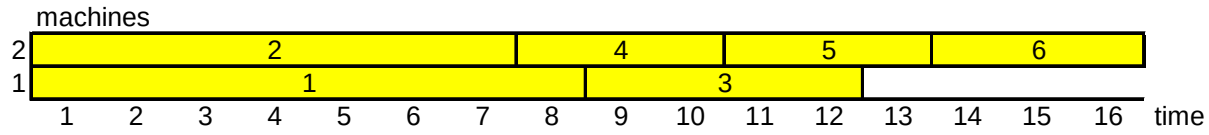

Figure 7. The schedule produced by the generating heuristic with $\delta = 0$.

If this heuristic is applied to this instance with the parameter $\delta = 1$, then the result is a schedule with *Makespan* 15. A simple improvement heuristic inspired by the schedule in Figure 7 is the rule: "Assign the last job to the machine with the lowest workload." This rule would also result in a schedule with *Makespan* 15. Improvement heuristics usually try to exchange certain parts of an obtained solution, as in the well-known 2-Opt and 3-Opt heuristics for the Traveling Salesman Problem. A more sophisticated improvement heuristic in this area is the Variable-Opt method developed by Lin and Kernighan (Lawler et al., 1985). An improvement heuristic in the same vein for improving a solution of the job scheduling problem described in Section 2 could be the following:

1. For each job j_1 assigned to machine 1 do
   For each job j_2 assigned to machine 2 do
   If switching the assignments of jobs j_1 and j_2 reduces the *Makespan*
   Then switch the assignments of jobs j_1 and j_2
2. Repeat step 1 until there are no further improvements

If this improvement heuristic is applied to the schedule that was produced by the generating heuristic described above with the parameter $\delta = 0$ (see Figure 7), then it indeed provides an improvement: the

assignments of jobs 2 and 3 are interchanged. However, the resulting schedule, represented in Figure 8, is still not optimal, since its *Makespan* equals 15.

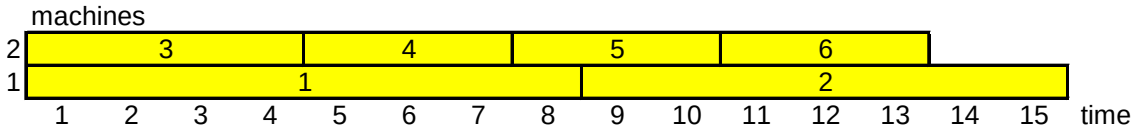

Figure 8. The schedule produced by the improvement heuristic.

In this improvement heuristic, the processing time for step 1 is obviously proportional to J^2. The number of times that step 1 can be executed is bounded by J. Therefore, the total processing time of this improvement heuristic is certainly proportional to J^3.

When dealing with approximation methods or with heuristic methods, it may be useful to develop lower bounds or upper bounds. Such bounds give an indication of the gap between an obtained solution and the optimal solution. For example, if one has a solution with objective function value V, and one can prove by applying lower bounding techniques that the minimum value of the objective function cannot be less than $0.95V$, then one knows that the minimum value of the objective function does not differ from the obtained value by more than 5%.

If the problem at hand is a minimization problem, then a *lower* bound can be generated by a relaxation of some of the constraints: the optimal solution of the relaxed problem gives a lower bound to the optimal solution of the original problem. This is useful if the relaxed problem can be solved more easily than the original problem. For example, if the problem at hand is an Integer Program, then a straightforward relaxation is its Linear Programming relaxation. This means that, instead of requiring the decision variables to be integer valued, they are just required to be real valued. This simple relaxation often leads to useful lower bounds. A lower bound for the *Makespan* of an instance of the simple job scheduling problem described in Section 2 is obviously $T = \frac{1}{2}\left(\sum_j p_j\right)$.

If the problem at hand is a minimization problem, then each feasible solution provides an *upper* bound to the minimum value of the objective function. Indeed, the solution is feasible, but it may be non-optimal. Therefore, the minimum value of the objective function will be less than or equal to the objective function value of the obtained feasible solution. In other words: the latter is an upper bound.

There exist many situations in which it is very difficult to develop an appropriate heuristic. This may be the case if finding a feasible solution for a problem is already difficult, let alone finding an acceptable or a good solution. In particular, in situations with very limited resources, a straightforward heuristic may not be capable of generating solutions effectively. In such situations, an approach that may work for finding an initial solution may be to use a general purpose solver and let it run until an acceptable solution has been found. This may work, since such solvers sometimes find a good solution quickly and then spend a lot of time on proving that this solution is optimal or nearly optimal.

An important aspect of automatically generating solutions for planning problems is the following: one sometimes states that solutions will be accepted by the planners only if the applied solution technique is very similar to the planning techniques used by the planners themselves. Only then it would be possible to understand the automatic solution process, and only then the obtained solution could be accepted in practice. However, in our opinion, the final result is much more important than the applied solution technique. The *automatic dishwasher* in the kitchen provides a relevant *metaphor* for this. Dishwashers provide the same result as the manual dishwashing process, if not a better. But

this result is obtained along a completely different way. Dishwashers that clean dishes exactly as humans do (pick up a cup, put it into a soap basin, use a brush to clean it, etc.) simply do not exist.

5. Evaluation of a solution

In Section 2, we introduced a simple job scheduling problem, where the *Makespan* was to be minimized for a given set of jobs and a given number of machines. However, in practical planning situations it is very rare that only one single objective function is used to evaluate the obtained solutions: in most practical situations, at least two objective functions are to be considered. Usually these objective functions are conflicting, which means that optimizing the first objective function leads to a completely different solution than optimizing the second objective function.

For example, one may want to minimize the production costs and at the same time to maximize the quality of the services to the customers. Another important example of multiple objectives, mainly related to the application of mathematical models, is to use the objective of minimizing the infeasibility of a solution next to the "real" objective of the planning problem at hand. Indeed, in practice there may be so many constraints to be taken into account that a solution satisfying all constraints simply does not exists: the problem is *over-constrained*. In such a case one may look for a solution that violates the constraints as few as possible.

In order to deal with optimization problems with multiple objectives, an extensive Multi-Criteria optimization theory has evolved over the last decades (e.g. Keeney and Raiffa, 1976). Examples of well-known Multi-Criteria optimization techniques are Goal Programming and Analytical Hierarchy Processing. Again, we do not go into the details, but we provide some general remarks.

If multiple objective functions are to be considered, then an important issue is the fact that one cannot really speak about *the* optimal solution of an instance of a planning problem, which can be determined in an *objective* way. The obtained solution depends on the way the multiple objectives are combined or weighted with each other. The user of the Multi-Criteria optimization technique is required to provide external information about his or her preferences. The foregoing also implies that optimality in practical terms may be quite different from optimality in mathematical terms.

For example, Figure 3 illustrates for the simple job scheduling problem described in Section 2 how the minimum *Makespan* and the *Number of Machines* are related there. Say that we were to minimize the total operational costs, where the *Makespan* incurs costs per hour w_1 and where resource costs per machine are equal to w_2. For each schedule involving a given *Makespan* and a given *Number of Machines*, the following total costs occur:

$$Total\ Costs = w_1 \times Makespan + w_2 \times Number\ of\ Machines$$

Not every combination of *Makespan* and *Number of Machines* is feasible: only the points in the plane above the curve in Figure 3 indicate feasible combinations. Observe that we have reduced the two objective functions into one single objective function by expressing both objectives in terms of costs. This need not always be possible, in particular if the objectives are very different.

More generally, consider the situation shown in Figure 9. This figure corresponds to an instance of a planning problem to be solved, thereby taking into account two objective functions F_1 and F_2. Both objective functions are to be minimized. The shaded area in Figure 9 represents the set of feasible combinations of values of these objective functions for the instance of the planning problem at hand. This set of feasible combinations of objective function values is also called the *function space*. Note the similarity of this figure with Figure 3. Obviously, in Figure 9 the objective functions are conflicting, since the minimum value for F_1 (denoted by m_1) cannot be reached at the same time as the

minimum value for F_2 (denoted by m_2). The point P in Figure 9 corresponding to the minimum value for F_1 *and* the minimum value for F_2 is called the *ideal* point or the *Utopic* point.

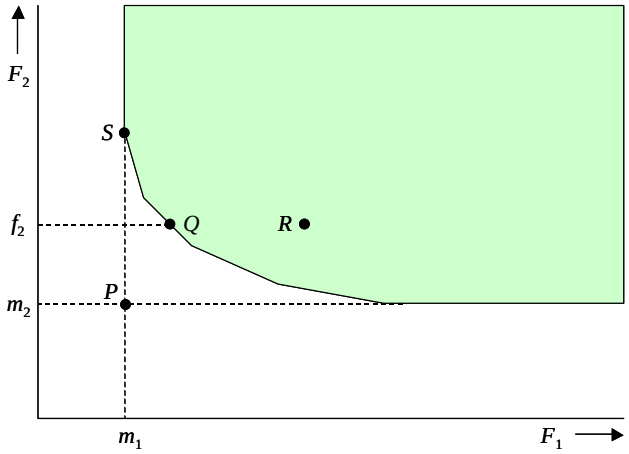

Figure 9. Function space corresponding to two conflicting objective functions.

Point Q is an example of a *Pareto-optimal* or *efficient* point in the function space: both objective functions can be improved with respect to their values in Q, but *not* at the same time. In other words: for all points Q' in the function space with $F_1(Q') < F_1(Q)$ it holds that $F_2(Q') \geq F_2(Q)$ and for all points Q' with $F_2(Q') < F_2(Q)$ it holds that $F_1(Q') \geq F_1(Q)$. Obviously, point R in Figure 9 is not Pareto-optimal, since both objective functions F_1 and F_2 can be improved there.

It is well-known, that optimizing the combined objective function $F = w_1 \times F_1 + w_2 \times F_2$ (where w_1 and w_2 are non-negative weights) over all feasible solutions of an instance of a planning problem leads to a Pareto-optimal point in the function space. Indeed, the obtained point in the function space is the point where a so-called *iso-cost* line with the equation $F = w_1 \times F_1 + w_2 \times F_2$ just touches the function space. This geometric interpretation implies that a higher weight for one of the objectives will lead to a lower value for this objective. Studying the set of Pareto-optimal points of a planning problem may provide insight into the relative importance of the different objective functions. It may also help in determining a trade-off between these objectives, since it shows how much of some of the objectives is to be sacrificed in order to get a certain improvement in the others.

Instead of combining several objective functions into one weighted objective function, one may also treat the objective functions in a so-called *lexicographic* way. That is, one first looks for a solution by minimizing the first objective function. Then, given the minimum value for the first objective function, one minimizes the second objective function. This is repeated until all objective functions have been taken into account. If this lexicographic method is applied to the instance represented in Figure 9 (first F_1 and then F_2), then this leads to the Pareto-optimal point S in the figure.

A somewhat more subtle method than the lexicographic one uses some of the objective functions as a constraint, and then optimizes over the others. For example, if in the example shown in Figure 9 one minimizes F_1 under the additional constraint that $F_2 \leq f_2$, then one obtains the Pareto-optimal point Q in the function space.

6. Pro-active and re-active planning

In this section we explain the differences between pro-active planning and re-active planning, and we show that both planning processes can be supported by applying mathematical models.

So far, we assumed that the planning process starts completely from scratch. That is, before starting the planning process, no plans or partial plans are available yet. Also the models that we described so far aimed at generating plans completely from scratch, not taking into account earlier generated plans. In this case, we talk about a *pro-active* planning process. In this case, the planning process may be triggered by the fact that the plans are to be updated with a certain fixed frequency. For example, a public transport company usually updates his basic logistic plans at least once per year.

However, there is necessarily a certain time interval between the start of the pro-active planning process and the execution of the plan in practice. In complex planning processes, it is even usual that there is a certain *planned* time interval between the *end* of the pro-active planning process and the execution of the plan. The latter is meant as a buffer in order to be able to cope with uncertainties in the duration of the pro-active planning process. The initial plan that is generated in the pro-active planning process is based on the information that is available at the moment that this plan is generated. However, as time goes by, it may happen that, during the generation of the initial plan or after the initial plan has been generated, new information becomes available and has to be taken into account in the plan as well. An example of this is a rush order that has to be planned and carried out as soon as possible. Another example may be that, on second thought, the processing time of a certain job should be longer than it was estimated earlier. A final example concerns the resource availability: after the pro-active plan has been generated, it may turn out that one of the machines is temporarily not available due to maintenance. Of course, if the machine maintenance department had communicated the maintenance plan of the machines earlier, then this would not have happened.

Note that the execution of a plan in practice usually needs to be monitored continuously. This may lead to updates of the plan in real time. Indeed, due to external disruptions during the execution of the plan, the plan may become infeasible and therefore it has to be updated. For example, the operational traffic control center of a public transport company has to deal with external disruptions quickly and adequately in order to have as few as possible delays in the operational transport processes.

In all these cases, the initial plan has to be updated in order to be able to take into account the additional information, thereby changing the initial plan as few as possible. Furthermore, during the execution of the plan, one usually wants to be able to return to the initial plan as soon as possible, since the plan is considered as an important guideline for managing the operational processes. Requirements for the latter are that the initial plan is sufficiently *robust* and *stable*, so that it can be picked up easily after certain external disruptions have taken place.

Generating an updated plan based on an initial plan and on additional information that was not available during the pro-active planning process is called *re-active* planning. Usually, in a re-active planning process, other objectives are pursued than in a pro-active planning process. This may be due to the fact that a re-active planning process is carried out closer to or during the execution of the plan in practice. The latter implies that a re-active planning process is carried out under a certain time pressure. As a consequence, the main objective in a re-active planning process is to find a *feasible* plan that takes into account the additional information, and that *differs* from the initial plan *as few as possible*. Here the underlying idea is that in the pro-active planning process more time has been available to optimize the quality of the initial plan. Therefore, if the modified plan differs as few as possible from the initial plan, it cannot be too far from optimality. Although in the re-active planning process one usually focuses on restoring the feasibility of the plan by modifying the initial plan as few as possible, also other objectives may be pursued then. This is in particular the case if a simple update of the initial plan turns out to be insufficient and a more rigorous modification of the plan is required.

6.1 An example of re-active planning

In the following, we illustrate how a mathematical model can be applied usefully also in a re-active planning process in order to take into account some new information. This illustration is based again on the simple job scheduling problem described in Section 2. Again, we consider the instance of this job scheduling problem shown in Table 1. We also assume that in the pro-active planning process the initial plan shown in Figure 2 was generated. This initial plan is now considered as part of the input of the re-active planning process and it is represented by the parameters $a_{j,m}$ for j=1,…,J and m=1,…,M. That is, $a_{j,m}$ =1 if and only if in the original plan job j is to be processed by machine m.

Next suppose that, after this initial plan has been generated, one realizes that the processing time of job 6 as been estimated too optimistically. A better estimate seems to be 4 hours instead of 3 hours. Furthermore, also a rush order with job number 7 and a processing time of 3 hours is to be taken into account. One way to incorporate this additional information into the model is represented in Figure 10. This model is basically the same as the model described in Section 2. The only difference is found in the objective function. Due to this modified objective function, the model focuses on staying as close as possible to the original plan. Indeed, if $a_{j,m}$ =1, then a value 1 for the decision variable $A_{j,m}$ is preferred over a value 0, due to the fact that the *Objective* is to be minimized. Similarly, if $a_{j,m}$ =0, then a value of 0 for the decision variable $A_{j,m}$ is preferred over a value 1.

Note that all assignments have the same weight, but, obviously, the assignments might have been given different weights as well. Note further that the *Makespan* has also been included in the *Objective* with a certain weight w, which describes the relative importance of the *Makespan*. If the *Makespan* would not have been included in the *Objective*, then the optimal solution to the model would obviously be the solution $A_{j,m} = a_{j,m}$ for j=1,…,J and m=1, 2. On the other hand, if the weight w is chosen too large, then the objective of minimizing the differences between the modified and the original plans will be dominated by the objective of minimizing the *Makespan*., which may lead to a modification of many of the initial assignments. Therefore, selecting the weight w is of paramount importance in such cases. Note that also the other Multi-Criteria optimization techniques that were described in Section 5 may be applied in this case, where two objectives play a role.

Minimize *Objective*
by changing the binary variables
$A_{j,m}$ $\quad j = 1,..,J,\ m = 1,..,M$

under the constraints

$$Objective = \sum_{j,m:a_{j,m}=1} (1 - A_{j,m}) + \sum_{j,m:a_{j,m}=0} A_{j,m} + w \times Makespan$$

$$\sum_{m} A_{j,m} = 1 \qquad \text{for } j = 1,..,J$$

$$\sum_{j} p_j A_{j,m} \le Makespan \qquad \text{for } m = 1,..,M$$

$$A_{j,m} \in \{0,1\} \qquad \text{for } j = 1,..,J,\ m = 1,..,M$$

Figure 10. Mathematical model for re-active planning support.

An optimal solution to the modified model is shown in Figure 11. This solution is obtained with the value 3 for the weight w, so that each unit of increase of the *Makespan* is considered as costly as 3 changes of a job assignment. Note that for the original jobs only the assignments of the jobs 1 and 2

have changed. The other assignments are the same as in the initial plan (see Figure 2). Furthermore, job 7 has been assigned to machine 1.

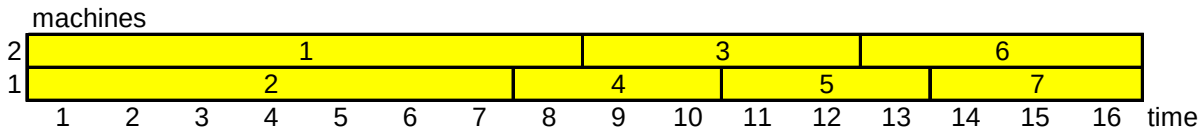

Figure 11. Optimal solution taking into account the additional information.

From the foregoing, it also follows that, if the weight *w* of the *Makespan* in the objective function has a value less than 2, then the assignment of the original jobs in the optimal solution will be the same as in Figure 2 and job 7 will be assigned to machine 1, which results in a *Makespan* 17.

One may want to fix certain parts of the initial plan when generating a modified plan, in particular if one is satisfied with these parts of the initial plan and if these parts of the initial plan do not have a direct relation with the additional information responsible for the need to generate the modified plan. If the modified plan is to be generated by a mathematical model, then such a fixation of certain parts of the initial plan can be accomplished by adding additional constraints to the model. For example, in the model described in Figure 10, one may select a certain subset *S* of jobs, and then add a corresponding set of constraints of the following form:

$$A_{j,m} = a_{j,m} \qquad \text{for all } j \text{ in } S \text{ and for } m = 1,2 \qquad (4)$$

6.2 Conclusion

The foregoing illustrates the fact that it is important that the length of the time interval between the start of the pro-active planning process and the execution of the plan in practice is as short as possible. The shorter the length of this time interval, the less time there will be for outside requests to modify the initial plan. Therefore, there will be less reasons for a re-active planning process then. However, the length of this time interval cannot be less than the throughput time of the pro-active planning process, which obviously cannot be reduced to zero. Therefore, there will always be a reason for a re-active planning process before the operational execution of the plan. Based on these observations, it can be concluded that it is highly important to reduce the throughput times of both the pro-active and the re-active planning process as much as possible. Mathematical models may be useful here.

Note that re-active planning in real time during the operational execution of the plan usually cannot be avoided as well, since external disruptions during the execution of the plan just will happen. In real time re-active planning, one should take into account the stochastic nature of the system explicitly. For example, if, in case of disruptions in a public transport system, the traffic control center suggests certain travelers to take an alternative route, then they should first have made a well-founded estimate of the probability that this alternative route will indeed bring these travelers to their destinations. Thus, forecasting the future states of the planned system is particularly important during the operations. In order to be able to do so, the continuous availability of up-to-date information on the current state of the planned system is of paramount importance.

7. Modeling systems

Modeling systems are extremely useful tools, especially during the development phase of a mathematical model. These systems enable one to translate a conceptual mathematical model into comparable terms that can be understood by a solver. These systems may also provide links to

spreadsheet or database systems for getting the data into the model. They usually provide links to one or more standard solvers such as CPLEX or LINDO as well. Some modeling systems have their own presentation system or they provide links to external presentation systems for presenting the data or the results of the model in a graphical way. Currently, several modeling systems are commercially available. Well-known examples of such systems are OPL Studio, GAMS, AMPL, and AIMMS.

As an example, Figure 12 shows the representation of the model for the simple job scheduling problem described in Section 2 in the modeling system OPL Studio. Note the similarity between the conceptual mathematical model and its representation in OPL Studio. Hence, once one has developed the conceptual mathematical model, its implementation in OPL Studio is usually a straightforward process. Furthermore, it should be noted that, apart from some syntactical details, the modeling languages of the other mentioned modeling systems are very similar to the one of OPL Studio.

```
var float Makespan       in 0..100;
var int   A[Job,Machine] in 0..1;

minimize Makespan,
subject to
{  // each job is to be processed exactly once
   forall (j in Job)
          sum(m in Machine) A(j,m) = 1;

   // the makespan is not less than the load of each machine
   forall (m in Machine)
          sum(j in Job) p(j)*A(j,m) ≤ Makespan;
};
```

Figure 12. Representation of the simple job scheduling problem in OPL Studio.

An advantage of the use of a modeling system is the fact that such systems provide a lot of flexibility in the development phase of a model: within such a modeling system, adapting a model is relatively easy. Thus these systems provide the flexibility to experiment with several formulations of a model, to add or delete decision variables or constraints, etc. This is especially useful in discussions with the problem owner or the planners, when there is still a lot of uncertainty about the specifications and the structure of the model. The flexibility provided by a modeling system may enable the model builder to modify the model within a few minutes and to present the results to the problem owner or the planners immediately thereafter. Furthermore, modeling systems usually also provide a lot of support in the sensitivity analysis of the results obtained by the solver.

Since the currently available modeling systems usually provide links to several standard solvers only, it may happen that these solvers are not powerful enough for solving real life instances of a certain problem. In that case, it will be required to develop a special purpose solver or a heuristic, as was described in Section 4 of this paper. But also in the latter case, the added value of the modeling system is mainly lying in the support provided during the development phase of the model. This is usually one of the hardest parts of the solution process based on mathematical models.

8. Application of mathematical models in practice

As was mentioned earlier, practical planning problems usually belong to the class of *NP-hard* problems, since many complicating practical issues are to be taken into account. The foregoing implies

that the computation times for solving instances of these problems usually increase quickly with the sizes of the instances. On the other hand, in practice one may be inclined to take the instances to be solved as large as possible, since decomposition of an instance into smaller subinstances usually leads to suboptimality. In cases where large instances lead to unacceptable running times, one will have to deal with this adequately. One of the following options may be chosen then:

- One may apply a heuristic or an approximation algorithm in order to compute an approximate solution for the problem instances. Drawbacks of this approach are that the development of such methods may require a lot of time and money, that the resulting method may be not quite robust with respect to changes in the problem, and that the planners may not be satisfied with the results if the approximation method is not powerful or detailed enough.
- One may decompose an instance into subinstances that can be solved to optimality in an acceptable running time. By choosing the subinstances in a clever way, thereby also taking into account decompositions that are relevant in practice, and by considering the inter-dependencies between the subinstances and the results of previous subinstances in the analysis of the next ones, the implied suboptimality may be kept within acceptable limits.

Although mathematical models and solution techniques are intended to actively *support* the planners in their daily planning activities, they will never be able to *replace* these planners. First, planners are usually much more *creative* than mathematical models. If planners really cannot find a solution that satisfies all practical constraints, then they may still find a "feasible solution" by relaxing some of the constraints in a creative way. However, "solutions" that do not fit within the constraints of the model are considered as infeasible by the model. They are not generated by the corresponding solution technique therefore. If, nevertheless, a "feasible solution" needs to be generated by the model, then one or more of the constraints of the model will have to be relaxed explicitly. Note that, in such a case, the model may provide support in finding a "feasible solution" that violates the constraints in some *minimal* way, thereby also taking into account the *hardness* of the constraints. However, incorporating too many of such exceptions into the model may complicate the solution of the model substantially.

Second, planners will be required in the planning process to specify the relative importance of the different conflicting objectives. This may lead to a large number of scenarios, each of which possibly leading to a different solution. These scenarios may be run on different computer systems in parallel, so that the throughput time of this process may be the same as the throughput time for running just a single scenario. Thereafter the expertise of the planners is required for selecting a preferred solution from the generated ones. The latter is in contrast with the situation where the planning process is carried out completely manually. In that case, one is usually already satisfied as soon as one has found exactly one acceptable solution that satisfies all or almost all practical restrictions.

Finally, mathematical solution techniques usually have more *computational power* than human planners, and this computational power is increasing with each new hardware generation and with each further improvement of the applied solution technique. Moreover, these techniques are not bored by doing the same computations over and over again, and their error rate is usually much lower than the human error rate. Therefore, a planning support system that facilitates a certain synergy between the planners' creativity and flexibility and the computational power and endurance of the currently available hardware and optimization techniques may lead to optimal results in practice.

9. A practical application

In this section, we describe a practical application of mathematical models for supporting the solution of real-life planning problems. We focus on the planning of the rolling stock circulation of an operator of passenger trains. It should be noted that mathematical models may provide useful support in other

railway planning processes as well, e.g. long term planning processes related to demand forecasting, capacity planning of the infrastructure, the rolling stock and the train crews, and relatively short term planning processes related to the structure of the timetable, crew scheduling, maintenance routing, and shunting. Many of theses issues are relevant in public transport planning in general as well.

For an operator of passenger trains, the rolling stock circulation is of paramount importance, since it influences both the service to the passengers, the operational costs for the railway operator, and the logistic robustness of the system. The rolling stock circulation gives rise to huge planning problems that have to be solved as fast and as effectively as possible. These planning problems are complex, because usually many complicated rules are to be taken into account, and multiple objectives are to be pursued. Examples of important objectives are (*i*) to maximize the effective capacity, (*ii*) to minimize the number of train unit kilometers, and (*iii*) to minimize the implied number of shunting movements.

The first objective is especially important during the rush hours, since most railway traffic takes place in these periods. Therefore it is important to schedule the rolling stock in such a way that as many passengers as possible can be transported according to the usual service standards. The second objective is obvious, since train unit kilometers are major cost drivers in a railway system. The last objective stems from the desire to have a robust railway system: many additional shunting movements in and around railway stations may disturb the regular railway traffic. Note that the three mentioned objectives are conflicting, so a well-founded trade-off between them has to be made.

Mathematical models may be quite helpful in carrying out such a trade-off between such conflicting objectives. Therefore, quite some research has been carried out in this area recently (e.g. Ben-Khedher et al, 1998; Cordeau et al, 2001; Lingaya et al, 2002; Schrijver, 1993).

9.1 Background information

In the Netherlands, most trains are operated with train units. An example of a train unit of the so-called type *Mat'64* with two carriages is shown in Figure 13. Train units of type *Mat'64* also exist with four carriages. Train units are indivisible units that can move individually in both directions without a locomotive. Train units of the same train type can be combined with each other into longer trains.

Figure 13. A train unit of type *Mat'64* with 2 carriages.

The way the train units can be put together into longer trains is shown in Figure 14. This figure shows a time-space diagram for part of the trains of the 8800 stop train line between Leiden (Ledn) and Utrecht (Ut). Trains of the 8800 line run twice per hour. The numbers at the top of the figure indicate the time axis. The dashed diagonal lines indicate the timetabled trains and the adjacent numbers are the train numbers. Each line represents one train unit of type Mat'64 with two carriages.

Train 8820, for instance, is run with three train units. Upon arrival in Leiden, two train units return to Utrecht on train 8833, and the third train unit remains in Leiden. This train unit is stored on the local shunting yard, and is used only in the afternoon rush hours on train 8865, as is indicated in the figure. Something similar happens with one train unit of train 8822 upon arrival in Leiden as well as with one of the train units of the trains 8829 and 8831 upon arrival in Utrecht.

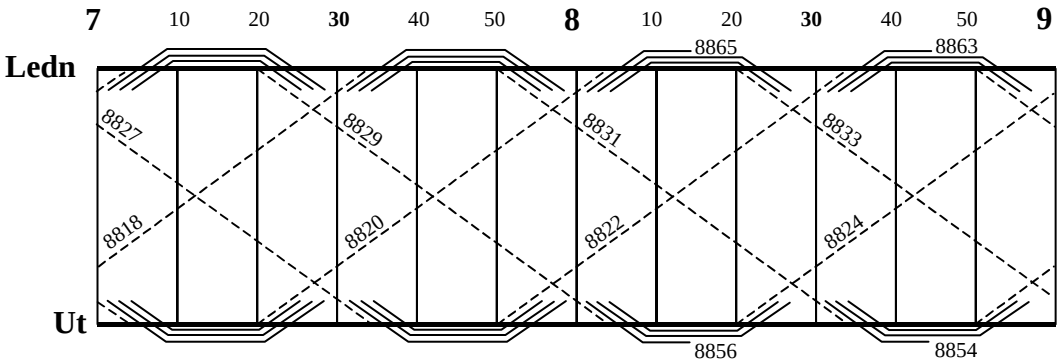

Figure 14. Part of the rolling stock circulation of the 8800 stop trains.

The rolling stock circulation problem does not only have to be solved for each single day, but also in such a way that the consecutive days fit after another: the number of train units ending in the late evening in a certain station should match with the number of train units that are required there the next morning. Nevertheless, in the planning process, usually a decomposition per day is applied. Initially, the ending night balances for day d may be used then as boundary conditions for the rolling stock circulation on day d+1. The final night balances are fixed in a second planning step.

Another complicating factor is the fact that each train unit needs maintenance check-ups regularly, which also have to be planned. However, the latter very short term planning close before the actual operations, because otherwise the probability that the maintenance schedules will have to be re-planned continuously is high.

9.2 Model description

In this section we briefly describe an optimization model that can be used to solve the rolling stock circulation problem for a single type of rolling stock on a single day. In this model, the timetable is assumed to be represented by a set of trips, where each trip t has an origin o_t, a destination d_t, a start time s_t and an end time e_t. The parameters $d_{t,c}$ represent the expected numbers of passengers on trip t in class c, k_c represents the number of seats in class c per train unit, and v denotes the total number of available train units. Apart from these parameters, the model also needs as input several weights w_t^N, w_t^C, w_t^U, and $w_{t,c}^S$ for making a trade-off between the different objectives.

The main decision variables in the model are the variables N_t describing the number of train units to be allocated to trip t. Furthermore, the decision variables $S_{t,c}$ describe the number of shortages of seats on trip t in class c, and the decision variables C_t and U_t describe the number of train units that are coupled/uncoupled to/from a train just before this train starts to carry out trip t. Finally, V_s denotes the number of train units that are available in the early morning in station s before the first train arrives or leaves there. Then the model reads as follows:

$$\text{Min} \sum_{t \in T} (w_t^N N_t + w_t^C C_t + w_t^U U_t + \sum_{c \in C} w_{t,c}^S S_{t,c}) \tag{5}$$

Subject to

$$S_{t,c} = \max\{0, d_{t,c} - k_c N_t\} \qquad \text{for all trips } t \text{ and classes } c \tag{6}$$

$$\sum_{t': o_{t'} = s \wedge s_{t'} \le s_t} N_{t'} \le V_s + \sum_{t': d_{t'} = s \wedge e_{t'} < s_t} N_{t'} \qquad \text{for all stations } s \text{ and all trips } t \text{ with } o_t = s \tag{7}$$

$$N_t = N_{t'} + C_t - U_t \qquad \text{for all pairs of consecutive trips } t' \text{ and } t \qquad (8)$$

$$\sum_{t:o_t=s} N_t = \sum_{t:d_t=s} N_t \qquad \text{for all stations } s \qquad (9)$$

$$\sum_{s \in S} V_s = v \qquad (10)$$

In the above model, the objective function (5) expresses the fact that one wants to minimize a weighted combination of carriage kilometers, shunting movements, and shortages of seats. Note that such a combination of multiple objectives may also be handled differently, as was explained earlier in Section 5 of this paper. Constraints (6) link the shortages of seats per trip and per class to the expected numbers of passengers per trip and per class, and to the allocated length of the train. Note that these constraints are not really linear. However, they can linearized easily. Constraints (7) are the balancing constraints per station: for each station and for each departing train from that station, the number of departing train units before and including the train's departure should not exceed the initial stock of train units at that station plus the number train units that arrived there earlier. Constraints (8) link the number of shunting movements to the numbers of train units in consecutive trips. Constraints (9) require that in each station the number of train units by the end of the day equals the number of train units by the start of the day, so that the circulation can be repeated the next day. Note that this is a simplification, since the rolling stock circulations on consecutive days need not be identical.

According to constraints (10), all train units are stored in one of the stations during the night. Finally, constraints (11) declare all decision variables as integer valued. Figure 15 shows the representation of the above model in terms of the modeling system OPL Studio.

```
minimize   Objective
   subject to
   {  // minimize carriage kilometers, shunting movements, and shortages
         Objective = sum (t in Trip)  ( wn[t]*N[t] +  wc[t]*C[t] + wu[t]*U[t] +
                                        sum (c in Classes) ws{t,c]*S[t,c] );

      // link between passenger demand, train units and shortages
         forall (t in Trip, c in Class)  S[t,c] >= d[t,c] – k[c]*N[t];

      // balance constraint after each departure from a station
         forall (s in Station, t in Trip: o[t]=s)
                sum (t1 in Trip: (o[t1]=s)&(s[t1]<=s[t])) N[t1] <=
                sum (t1 in Trip: (d[t1]=s)&(e[t1]< s[t])) N[t1] + V[s];

      // link between shunting movements and incoming and outgoing train units
         forall (t, t1 in Trip: t = Next[t1])  N[t] = N[t1] + C[t] – U[t];

      // final stock per station equals the initial stock there
         forall (s in Station)
                sum(t in Trip: o[t]=s) N[t] = sum(t in Trip: d[t]=s) N[t];

      // all train units are stored somewhere during the night
         sum (s in Station)  V[s] = v;
   };
```

Figure 15. Representation of a rolling stock circulation model in OPL Studio.

9.3 Model extensions

Basically, the model as described above is a relatively simple extension of a Single Commodity Flow problem on a network that is very similar to the graph in Figure 3. As was noted earlier, Single Commodity Flow problems can be solved efficiently, since they belong to the class *P*. As a consequence, large instances of this problem can be solved to optimality by standard optimization software such as CPLEX or by applying special purpose Single Commodity Flow techniques.

Straightforward extensions of the above model may deal with e.g. (*i*) certain minimum or maximum lengths of trains on certain trips, (*ii*) the fact that shunting movements require time, or (*iii*) prescribed night balances per station. However, dealing with train units of different subtypes requires a far more complex model. In that case, a train with composition *Mat'64-2* / *Mat'64-2* / *Mat'64-4* is different from a train with composition *Mat'64-2* / *Mat'64-4* / *Mat'64-2*: although they have the same capacities for transporting passengers, they have different transition possibilities. Therefore, one has to take into account the positions of the train units within the trains then. As a result, the mathematical model does not only become a Multi Commodity Flow problem, it even becomes something, which may be called an *Ordered* Multi Commodity Flow problem. Such problems are usually hard to solve to optimality. Another highly complicating issue for modeling the rolling stock circulation is the fact that trains are sometimes split or combined in certain locations or the fact that certain trains are operated with locomotive hauled carriages. Nevertheless, a lot of progress in modeling such complicating issues has been achieved recently.

9.4 Practical experiences

The Logistics department of NS Reizigers is responsible for the planning of the timetable, the rolling stock circulation and the crew schedules of NS Reizigers. Within this department, the models described in this section as well as several extensions of them have been implemented as prototypes within the well-known modeling system OPL Studio, and they have been solved by the standard solver CPLEX. These models have proved to be very useful within the planning process of the rolling stock circulation of NS Reizigers. In almost all cases, the results of the models are at least as good as the results obtained manually by the planners, sometimes even in all three objectives. Moreover, the results are usually obtained in just a fraction of the manually required amount of time. This comparison of the model results with the manually obtained results can be considered as a validation of the models, as was described in Section 3.

The availability of the models allows one to study the set of Pareto-optimal solutions of an instance of the rolling stock circulation problem, as was described in Section 5. The latter is useful in determining a trade-off between the different objective functions. For example, one may find out that the shortages of seats can be further reduced, but only at the expense of a very high number of additional train unit kilometers. In that case one will probably be satisfied with the available solution. Also the fact that the models enable one to quickly evaluate the consequences of different scenarios, e.g. related to changing passenger demands, changing availabilities of rolling stock or different rolling stock types, is appreciated both by the planners and by the management of the Logistics department.

Nevertheless, the application of these models usually does not lead to rolling stock schedules that can be applied in practice immediately. The latter is due to the fact that, in practice, the complete rolling stock circulation planning problem is really a huge and complex problem: it has to be solved for all rolling stock types, for all lines, and for a whole week. Indeed the initial rolling stock circulation is usually cyclic with a cycle length of one week. However, the models described so far deal with a limited number of rolling stock types, on a subset of the lines, and for a single day only. However, as was mentioned earlier already, the planners in practice usually also start to solve the

rolling stock circulation problem on a line-by-line basis and for a single day, in order to reduce the complexity of the problem. Hence, it can be concluded that the mathematical models provide the planners with useful initial solutions, but also that these initial solutions have to be fine-tuned and fit together manually in a second step. The latter may involve, for example, the fixing of the night balances. Note that, in principle, the fixing of the night balances could also be supported by mathematical models, but such models have not been developed yet. Furthermore, since a rolling stock circulation that is completely line based may lead to a suboptimal utilization of the rolling stock, several interconnections between the rolling stock circulations of the single lines are made manually. In other words, the inefficiencies that are due to the decomposition of the rolling stock circulation problem on a line by line basis are reduced afterwards as much as possible.

The available models have been implemented as prototype systems and they have not yet been embedded in user-friendly decision support systems. In particular, advanced interfaces between the data bases that could deliver the input data for the models and that could restore the solutions of the models into the data bases are still lacking. However, since the prototype implementations of the models have proved the models' viability in practice, the development of such interfaces, as well as user-friendly decision support systems around them, seems to be a logical next step.

10. Final remarks

In this paper we described how mathematical models and solution techniques can provide planners with *active* support in their daily planning activities. Mathematical models and solution techniques can be used to generate solutions or partial solutions for complex planning problems. These solutions may either be used directly in practice, or the planners may further refine them manually, if they consider this as necessary. In this final section, we summarize some of the raised issues.

In this paper, we pointed out that most practical planning problems belong to the class of *NP*-hard problems. Therefore, the probability that solution methods exist that solve all instances of such a planning problem efficiently is considered as small. For solving practical instances of such a planning problem, one will therefore have to be satisfied with *approximate* solutions. Nevertheless, the time required to generate an acceptable solution is usually shorter than if the solution had to be generated manually. The latter may lead to a reduction of the throughput time of the planning process. This will allow the organization to react faster to changing external circumstances, which may have a positive effect on the flexibility of the organization.

A consequence of the foregoing is that developing a mathematical model requires one to balance the required *level of detail* of the model carefully with the implied *complexity*. However, in practice there is often a tendency to incorporate as many details as possible into a model. The latter may have a detrimental effect on the model's complexity. Creating mathematical models and corresponding solution techniques that are really useful in practice is therefore more an art than a science.

A further important point is that optimality in *practical* terms need not be the same as optimality in *mathematical* terms. In practice, usually several conflicting objectives play a role. Therefore it is impossible to talk about *the* optimal solution then, since the quality of a solution depends on the relative importance that is given to the different objectives. But mathematical solution techniques usually allow one to compute a trade-off between the different objectives: how much of a certain objective is to be sacrificed in order to get a certain improvement in another one?

From the foregoing, we may conclude that, if designed and developed appropriately and applied sensibly, then mathematical models and solution techniques may have a positive influence, both on the obtained solutions and plans, the planning process itself, and the work contents of the planners.

References

Aarts, E.H.L., and P. J. van Laarhoven, Simulated annealing: theory and applications. Kluwer Academic Publishers, 1987.

Ackoff, R., The scientific method. Krieger Publishing Company, 1984.

Baker, K.R., Introduction to sequencing and scheduling. John Wiley, 1975.

Barnhart, C., E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance, Branch-and-price: column generation for solving huge integer programs. *Operations Research*, 46: 316-329, 1998.

Ben-Khedher, N., J. Kintanar, C. Queille, and W. Stripling, Schedule optimization at SNCF: From conception to day of departure", *Interfaces*, 28: 6-23, 1998.

Bixby. R.E., Solving real-world Linear Programs: a decade and more of progress. *Operations Research*, 50: 3-15, 2002.

Cook, S.A., The complexity of theorem-proving procedures. Proceedings of the 3rd annual ACM symposium on the theory of computing. Association for Computing Machinery, New York, 1971.

Cordeau, J.F., F. Soumis, and J. Desrosiers., Simultaneous assignment of locomotives and cars to passenger trains, *Operations Research*, 49: 531-548, 2001.

Dantzig, G.B., Programming in a linear structure. *U.S. Air Force Comptroller*, USAF, Washington D.C, 1948.

Garey, M.R. and D.S. Johnson., Computers and intractability: A guide to the theory of *NP*-Completeness. Freeman, 1979.

Glover, F., and M. Laguna, Tabu Search. Kluwer Academic Publishers, 1997.

Gomory, R.E., Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64: 275-278, 1958.

Gondran, M., and M. Minoux, Graphs and algorithms. John Wiley, New York, 1984.

Hillier, F.S., and G.J. Lieberman, Introduction to Operations Research. Holden-Day, Inc., 1967.

Johnson, S.M., Optimal two- and three stage production schedules with setup times included. *Naval Research Logistics Quarterly*, Vol. 1, 1954.

Keeney, R.K., and H. Raiffa, Decisions with multiple objectives: preferences and value trade-offs. John Wiley, 1976.

Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (eds.), The traveling salesman problem. John Wiley, 1985.

Lingaya, N., J.F. Cordeau, G. Desaulniers, J. Desrosiers, and F. Soumis, Operational car assignment at VIA Rail Canada, *Transportation Research B*, 36: 755-778, 2002.

Pidd, M., Just modeling through: a rough guide to modeling, *Interfaces*, 29: 118-132, 1999.

Schrijver, A., Minimum circulation of railway stock. CWI Quarterly, 6: 205-217, 1993.

Wagner, H.M., Principles of Operations Research. Prentice Hall, 1970.

Publications in the Report Series Research* in Management

ERIM Research Program: "Business Processes, Logistics and Information Systems"

2003

Project Selection Directed By Intellectual Capital Scorecards
Hennie Daniels and Bram de Jonge
ERS-2003-001-LIS
http://hdl.handle.net/1765/265

Combining expert knowledge and databases for risk management
Hennie Daniels and Han van Dissel
ERS-2003-002-LIS
http://hdl.handle.net/1765/266

Recursive Approximation of the High Dimensional max Function
Ş. İl. Birbil, S.-C. Fang, J.B.G. Frenk and S. Zhang
ERS-2003-003-LIS
http://hdl.handle.net/1765/267

Auctioning Bulk Mobile Messages
S.Meij, L-F.Pau, E.van Heck
ERS-2003-006-LIS
http://hdl.handle.net/1765/274

Induction of Ordinal Decision Trees: An MCDA Approach
Jan C. Bioch, Viara Popova
ERS-2003-008-LIS
http://hdl.handle.net/1765/271

A New Dantzig-Wolfe Reformulation And Branch-And-Price Algorithm For The Capacitated Lot Sizing Problem With Set Up Times
Zeger Degraeve, Raf Jans
ERS-2003-010-LIS
http://hdl.handle.net/1765/275

Reverse Logistics – a review of case studies
Marisa P. de Brito, Rommert Dekker, Simme D.P. Flapper
ERS-2003-012-LIS
http://hdl.handle.net/1765/277

Product Return Handling: decision-making and quantitative support
Marisa P. de Brito, M. (René) B. M. de Koster
ERS-2003-013-LIS
http://hdl.handle.net/1765/278

---

* A complete overview of the ERIM Report Series Research in Management:
http://www.erim.eur.nl

ERIM Research Programs:
LIS Business Processes, Logistics and Information Systems
ORG Organizing for Performance
MKT Marketing
F&A Finance and Accounting
STR Strategy and Entrepreneurship

Managing Product Returns: The Role of Forecasting
Beril Toktay, Erwin A. van der Laan, Marisa P. de Brito
ERS-2003-023-LIS
http://hdl.handle.net/1765/316

Improved Lower Bounds For The Capacitated Lot Sizing Problem With Set Up Times
Zeger Degraeve, Raf Jans
ERS-2003-026-LIS
http://hdl.handle.net/1765/326

In Chains? Automotive Suppliers and Their Product Development Activities
Fredrik von Corswant, Finn Wynstra, Martin Wetzels
ERS-2003-027-LIS

Mathematical models for planning support
Leo G. Kroon, Rob A. Zuidwijk
ERS-2003-032-LIS

How and why communications industry suppliers get "squeezed out" now, and the next phase
L-F Pau
ERS-2003-033-LIS
http://hdl.handle.net/1765/317

Financial Markets Analysis by Probabilistic Fuzzy Modelling
Jan van den Berg, Uzay Kaymak, Willem-Max van den Bergh
ERS-2003-036-LIS
http://hdl.handle.net/1765/323

WLAN Hot Spot services for the automotive and oil industries :a business analysis or : "Refuel the car with petrol and information , both ways at the gas station "
L-F Pau, M.H.P.Oremus
ERS-2003-039-LIS

A Lotting Method for Electronic Reverse Auctions
U. Kaymak, J.P. Verkade and H.A.B. te Braake
ERS-2003-042-LIS

Supply Chain Optimisation in Animal Husbandry
J.M. Bloemhof, C.M. Smeets, J.A.E.E. van Nunen
ERS-2003-043-LIS