Working Paper

Design and Implementation of Model-based Decision Support Systems

Marek Makowski

WP-94-86
December 1994

IIASA International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 807 □ Fax: +43 2236 71313 □ E-Mail: info@iiasa.ac.at

Design and Implementation of Model-based Decision Support Systems

Marek Makowski

WP-94-86
December 1994

International Institute for Applied Systems Analysis □ A-2361 Laxenburg □ Austria
Telephone: +43 2236 807 □ Fax: +43 2236 71313 □ E-Mail: info@iiasa.ac.at

Foreword

At IIASA there is a long history of developing methodology for decision support. There is also a lot of experience of applying such methodology to practical problems. The present paper summarizes the experience with multi-objective optimization methods at IIASA and explains different features of applying such methods. It is emphasized that decision support systems should be developed for particular decision environments with the help of modular tools. Several modular tools are described.

Abstract

Decision making often requires the analysis of large amount of data and complex relations. Computerized tools designed and implemented for such purposes are called Decision Support Systems (DSS). A DSS, which is typically a problem specific tool, usually helps in the evaluation of consequences of given decisions and may advise what decision would be the best for achieving a given set of goals. In such cases, an analysis of a mathematical model can support rational decision making.

The paper provides an overview of the methodology of the design and deals with practical aspects related to implementations of model-based decision support systems. In particular, different approaches to the analysis of a model using simulation and optimization are summarized. Various optimization techniques are discussed in this context, including multi-criteria optimization used for a model analysis. The paper summarizes also problems of hardware selection and of software development. Modular software tools applicable to DSSs, including a tool for data interchange, are characterized. Selected issues of implementations of modular solvers and of applications of artificial neural nets to decision support are also presented.

Key Words: decision support, multi-criteria programming, aspiration-led decision support, linear programming, regularization, applications of multi-criteria programming, artificial neural nets.

Contents

1 Introduction **1**

2 Decision Support Systems (DSS) **2**
2.1 General remarks 2
2.2 DSS – a support for a Decision Making Process 4
2.3 Model-based DSS 5
2.3.1 General concepts 5
2.3.2 Model specification 7
2.3.3 Simulation and optimization for decision support 9
2.4 Optimization in DSS 10
2.4.1 Extensions of the traditional formulation of LP problems 12
2.4.2 Multi-objective model analysis 14
2.5 Applications of Artificial Neural Nets in decision support 17

3 Design and implementation of a DSS **19**
3.1 Hardware considerations 19
3.2 Software considerations 19
3.3 Modular software tools 23
3.4 Linear Programming Data Interchange Tool: LP-DIT 25
3.5 Modular solvers of mathematical programming problems 25

4 Conclusions **27**

References **28**

Design and Implementation of Model-based Decision Support Systems

Marek Makowski

1 Introduction

Decision Support System(s) (DSS) are computerized tools used to aid in decision-making. Such tools have wide coverage in theoretical research and are applied for supporting decision making in diverse situations in various fields, including management, engineering and medicine. The variety of situations includes making strategic decisions at a corporate level, strategic and operational planning of means aimed at improving environmental situations, operational water management and solving engineering design problems. DSS that serve such diversified purposes have obviously very different features. Even more confusion is created because of DSS that are designed and applied for similar purposes in different scientific communities are often called differently. Sometimes different terms like Management Information Systems (MIS), Strategic Information Systems (SIS), Expert Systems, Intelligent Decision Support Systems or DSS are used for similar methodology and/or type of application. On the other hand, very often the DSS name is used for very different methods and/or tools.

This paper focuses on selected topics of design and implementation of a DSS. The aim of this paper is not to present this particular application, but rather to use it as an illustration for the discussion of problems that are of a broader interest. The selection of topics for the discussion is based on lessons learned from several applications of different types. An attempt has been made to discuss issues that are common to a relatively numerous class of problems for which DSS have been or can be developed. The remaining part of this paper is organized as follows.

A general characteristic of DSS is summarized in Section 2.1, and a role of a DSS within a decision making process is discussed in Section 2.2. Model based DSS are discussed in Section 2.3. This includes issues of model specification (Section 2.3.2) and an outline of two basic techniques for a model analysis, namely simulation and optimization (Section 2.3.3). In Section 2.4 the role of optimization in decision support is discussed. Although linear programming (LP) problems are considered classical, there are still some useful extensions of the traditional formulation of LP problems. Such extensions are presented in Section 2.4.1. An outline of multi-objective optimization, which is nowadays a basic methodology used for model examination in model-based DSS, is given in Section 2.4.2. Section 2.5 summarizes applications of Artificial Neural Nets (ANN) to decision support.

The second part of the paper is devoted to the discussion of problems related to design and implementation of a DSS. Hardware and software considerations related to implementation of a DSS are discussed in Sections 3.1 and 3.2, respectively. Selected problems related to development and implementation of modular software tools that can

be used for building a DSS are presented in Section 3.3. One of the discussed implementation issues is related to data handling. It has been observed that the only de facto standard for LP problems (MPS format) is not efficient and does not provide complete information for an LP problem. Therefore a modular tool LP-DIT that provides efficient data handling is also briefly characterized in Section 3.4. Next, some experiences with development and applications of modular solvers are presented in Section 3.5. Finally, Section 4 contains conclusions.

2 Decision Support Systems (DSS)

2.1 General remarks

The books related to various issues connected to the design and implementation of DSS number in dozens whereas a list of such issues would even be longer. Therefore we have to restrain ourselves to a small subset of basic problems.

The term *Decision Support System* (DSS) is widely used both in research and in many applications, but there is no agreement about its meaning. We will not contribute to this terminological discussion. Instead of discussing the large set of problems related to the possible definitions of a DSS, we refer the reader to such discussions presented e.g. in [And89, Dav88, Eme87, FlJ91, Jan92, Hop88, LeW89b, Mei76, Nag90, Sil91, Thi88, Thi93]) and we will quote here only the definition proposed by Emery ([Eme87]):

> A DSS provides computer-based assistance to a human decision maker. This offers the possibility of combining the best capabilities of both humans and computers. A human has an astonishing ability to recognize relevant patterns among many factors involved in a decision, recall from memory relevant information on the basis of obscure and incomplete associations, and exercise subtle judgments. A computer, for its part, is obviously much faster and more accurate than a human in handling massive quantities of data. The goal of a DSS is to supplement the decision powers of the human with the data manipulation capabilities of the computer.

Handling massive quantities of data should be understood not only in a traditional sense (as data processing done by a Data Base Management System) but it can also analyze a large amount of logical relations and/or solve mathematical programming problems. Therefore, Operational Research (OR) is one of the key research fields for DSS. However, we would like to stress one key problem expressed by many authors (cf e.g. the summary given by Davis in [Dav88]). Namely that the misunderstanding and misuse of the DSS concept may eventually result in its demise as a distinctive management tool. This concern is well expressed by the already famous statement published already in 1979 by Ackoff (cf [Ack79]):

> More and more people are coming to realize that optimization of all the quantities of life does not optimize the quality of life and that is a limiting objective. In addition, there is a widespread belief that much of the accelerating rate of change is getting us nowhere. [...] Those of us who are engaged in helping others make decisions have the opportunity and the obligation to bring consideration of quality of life – style and progress – into their deliberations. OR [Operational Research] has virtually ignored both the opportunity and the obligation.

This statement is far from being generally accepted in the OR community (cf e.g. a constructive discussion of the role of OR by Chapman [Cha88, Cha92] and by Radermacher [Rad94]). Also many applications (examples of different types of applications are discussed and further references are given e.g. in [EKO90, HoW93, KaZ89, Kee92, KOCZ93, KMW92, KLW91, LeW89a, Mak91b, Nak94, SpW93, Ste86, Ste92b, Tur93, WeW93])

demonstrate wide possibilities of the application of OR for DSS. However the statement by Ackoff should not be forgotten since it serves two purposes: as a perfect warning against misusing OR methods and as a stimulation for using optimization methods in ways that are better suited for solving real-life problems.

In order to avoid possible misunderstandings it is necessary to present the basic characteristics and features of the class of DSS we will be dealing with. Let us start with a brief discussion of the environment in which a DSS may be used. The key person in this consideration is an individual who uses a DSS in real-life situations. By convention such a person is called *a Decision Maker* (DM). By this term we mean both a person who makes real decisions (depending on an application it may be a manager or an engineer or an operator) or an expert who may be his/her advisor. Decisions are made within a *Decision Making Process* (DMP), which, in situations that justify usage of a DSS, is a rather complex sequence of tasks. We assume that a final decision is to be made by the DM and a DSS does not serve as a replacement or control of a DM. In other words, a DSS is not aimed at the automatic selection of decisions.

The following characteristic of a DSS implicitly defines a class of DSS we will be dealing with:

- A DSS is a supportive tool for the management and the processing of large amounts of information and logical relations that helps a DM to extend his habitual domain (cf [Yu90]) and thus help him/her to reach a better decision. In other words, a DSS can be considered as a tool that, under full control of a DM, performs the cumbersome task of data processing and provides relevant information that enables a DM to concentrate on this part of the DMP that can not be formalized.
- A DSS is a problem dedicated system designed for a specific DMP and its environment. The functioning of a DSS should be consistent with the actual environment of a DMP. A DSS is often tuned for a specific DM.
- A DSS is not a *black box type* tool. The structure and functioning of a DSS (including explicit and implicit consequences of assumptions adopted for its design) must be such that a DM understands and accepts them. The user interface of a DSS is designed in such a way that a DM may obtain, from the DSS, information and answers for questions that he considers to be important for a DMP.
- A DSS is not intended to solve a decision problem. Therefore it should not support reaching a single or unique decision nor should it restrict a possible range of decisions.
- A DSS should support a user during a DMP by providing two main functions. First, it allows for examination of consequences of any (feasible) decision. Second, it helps in finding decisions that are the best for attaining goals specified by a user.

A DSS is useful in complex situations for which specification of attainable goals and of rational decisions is impossible. Let us briefly outline a use of a DSS for the example of Regional Water Quality Management (RWQM) problem[1]. A DM may want to consider different measures to improve water quality, the related costs and various standards of water quality. The multicriterion's version of the DSS for the RWQM has less than 100 binary variables but the number of all feasible combinations of the corresponding technologies is far from being manageable. Therefore the DSS for RWQM helps a DM in finding a set of technologies whose implementation would result in achieving conflicting goals expressed by selected aspiration values for economic and environmental criteria. The specified goals are typically not attainable, therefore the DSS finds a solution closest to the

[1]The RWQM is aimed at the design and implementation of a DSS for supporting a selection of a set of alternatives for waste water treatment plants in order to improve the water quality of a river basin, or of a larger region consisting of a number of river basins (cf [MSW95] for details).

specified goals. This in turn usually motivates a DM to change aspiration values for the goals and to repeat the calculations. From time to time a DM may want to check the effect of applying a given (by him/her) selection of technology. Of course, the corresponding solution will not be optimal in the sense of mathematical programming, but nevertheless it is an important function of any DSS to provide also simulation capabilities because a user may not only want to check the effects but sometimes may also want to apply such a solution (e.g. due to its other properties that are not included in the specified goals).

Hopple [Hop88] suggests that *"The human-machine symbiosis is a hallmark of a genuine decision support system"*. The above listed brief characteristics are the necessary main conditions for such a symbiosis. By no means they are sufficient conditions. There is no general specification of sufficient conditions for the implementation of a DSS since this obviously depends on a particular environment of a DMP. The key element of this environment is the habitual domain of a DM (cf [Yu90]). Recognition and understanding by a DSS developer of the DM's habitual domain is essential for design and implementation of a DSS.

2.2 DSS – a support for a Decision Making Process

Making a rational decision often requires access to and the processing of a large amount of data and logical relations which (due to the nature of the problem) cannot or should not be replaced by intuition. In many situations it is not a small task to examine even the possible range of feasible alternatives. Using computers for processing usually results in implementing a DSS. Many different functions that can or should be provided by a DSS can be divided into two sets:

- Data processing in the traditional sense: These functions provide selective retrieval and presentation of information previously stored in a data base. Such functions are typically supported by a Data Base Management System (DBMS) and are useful for many every-day managerial activities such as producing various reports (e.g., periodical and exception), answering ad hoc queries, presenting information in different forms, etc.
- Model processing: Quite often, it is desired to predict consequences of some actions (for example implementing a decision or making a choice) or events (actions that are not controlled by a DM). In such cases, a mathematical model for a real situation is built and such a model is used for analysis of predicted consequences.

A DSS that supports only functions from the first set is called a data-oriented DSS. Model processing functions require a model-based DSS that typically contains also most of the functions of the data-oriented DSS.

We will consider a model based DSS[2] which uses an underlying mathematical model. Such a mathematical model is built for a part of the Decision Making Process (DMP) for which it is possible to implement an abstract (mathematical) model that is good enough to represent the available (often quite complex) knowledge and experience of a user in order to support his/her intuition. A user is a Decision Maker (DM) or an analyst but in this paper we will use the terms *user* and *DM* interchangeably. Therefore such support for decision making is conceptually distinct from the more traditional data-oriented perspectives of decision support. We don't claim that the approach discussed in this paper is better than this more traditional approach. We simply point out that quite often the DMP requires not only data processing in the traditional sense but it also requires the analysis of a large number of logical or analytical relations and processing –

[2]Further on in this paper, DSS means model based DSS, unless explicitly stated otherwise.

in the sense of solving[3] an underlying mathematical model – a large amount of data. In such situations a properly designed and implemented model-based DSS not only performs cumbersome data processing, but it also provides relevant information that enables a DM to concentrate on those parts of the DMP that cannot be formalized.

A key issue for a model-based DSS is its relation to the actual DMP. Especially in managerial situations, a DM is typically confronted with problems that are dependent of each other and a DSS covers only a subset of problems that are considered within the DMP. Part of the DMP is often not representable in the form of a mathematical model. Dealing with a mathematical model is often considered to be *hard system thinking* as opposed to the *soft system thinking.* Relations of these two types of approaches are discussed e.g. by Flood [FlJ91] and Wierzbicki [Wie92c]. The *Shinayakana* approach proposed by Sawaragi (cf [SaN91]) also combines hard and soft system approaches. Another important issue for a successful implementation of a DSS, is the paradigm of *rational choice* (cf Section 2.3.1).

To end this brief overview of the common issues let us just point out two important problems. First, **decision making is not a point event even in situations when it is realistic to assume that the problem perception does not change during the DMP.** Therefore the possibility of using a DSS in a learning and adaptive mode is a critical feature. Secondly, in many complex decision situations, the availability of information may not always be consistent with the authority of using this information for implementing appropriate decisions. The analysis of the accident of the Challenger space shuttle (cf [vG91]) is a good example of related problems.

Relations between the DSS and the whole DMP should be clearly identified and be understood both by the team that implements it (cf [Mak91b]) and by the DM. Therefore mutual understanding – between the DM and the analysts and researchers who design and implement a DSS – is one of necessary conditions for each implementation. The DM must understand consequences of assumptions adopted for a particular DSS, and analysts and researchers must understand the DMP well enough to identify those consequences. However, this condition is not always fulfilled. Quite often a general DSS tool is applied without careful analysis of the consequences. We hope that the arguments collected in this section clearly show that such an approach is unlikely to be successful.

2.3 Model-based DSS

2.3.1 General concepts

For the discussion of the model based, aspiration–led DSS concept, let us assume the following decision making situation:

- A well-defined part of a DMP (for which a DSS is to be implemented) can be represented in the form of a mathematical programming model. Decisions have quantitative characters and therefore can be represented by a set of the model variables, hereafter referred to as decisions[4] $x \in E_x$, where E_x denotes a space of decisions.
- The model defines a set of feasible decisions $X_0 \subseteq E_x$. Therefore x is feasible, iff $x \in X_0$. The set X_0 is usually defined implicitly by a specification of a set of constraints that correspond to logical and physical relations between the variables. The feasibility of decisions given by a DM should be assessed. Decisions computed by a DSS should be

[3]The word *solving* is used here for two approches to the analysis of mathematical models, namely simulation and optimization. See Section 2.3.3 for more details.

[4]For the sake of brevity we call decision variables simply decisions.

feasible, if a feasible solution exists.

- The model can be used for predicting the consequences of decisions proposed by a DM or computed by DSS. The prediction of the consequences can usually be represented by a mapping $y = f(x) \in E_y$, where E_y is a space of consequences (outcomes) of the decisions.
- The consequences of different decisions x can be evaluated by values of criteria $q \in E_q$, where E_q is a space of criteria (sometimes referred to as outcomes, goals, objectives, performance indices, attributes, etc.). Usually E_q is a subspace of E_y, that is, the DM might select some criteria q_i between various outcomes y_j. Sometimes also some of the decision variables x are used as criteria. A partial preordering in E_q is usually implied by the decision problem and has obvious interpretations, such as the minimization of costs competing with the minimization of pollution. However, a complete preordering in E_q cannot usually be given within the context of a mathematical programming model.

The decision problem boils down to a selection of *the best*, among all feasible decisions, decision $\hat{x}$. The key problem here is to understand what *the best* means for a DM who actually makes a decision. The problem of a *rational choice* of a decision has been extensively discussed in a number of publications. A discussion of different approaches to this problem is given e.g. by Keeney [KeR76, Kee92], Rapoport [Rap89], Sawaragi [SaN91], Stewart [Ste92b], Wierzbicki [Wie92c, Wie92a], Yu [Yu90].

Before discussing the multi-objective based approaches, we briefly comment on the most strongly established rationality framework that is based on the concept of maximization of *the multiattribute utility* (MAU) (cf e.g. [Fis64, KeR76, Yu85]). The MAU concept, often also referred to as *multiattribute value function*, assumes that it is possible to construct a function that maps elements of the criteria set E_q into R^1 in such a way that a larger number corresponds to a stronger preference. There are many fundamental and technical difficulties related to the identification of the value function that adequately reflects the preferences of a DM (cf e.g. [Fis79, Rap89]). Moreover, it has been observed by many researchers (cf e.g. [Mac85, TvK85]) that a DM learns about the decision problem during an interaction with a DSS and quite often changes his/her preferences or specifies them inconsistently during this learning process. But an even more important reservation for applications of the MAU concept to decision support was given by Simon [Sim57], who pointed out, against all traditional economic concepts, that people look for satisficing solutions instead of one that maximizes the expected utility. Also Galbraith stressed in [Gal67], that satisficing behavior corresponds to the culture of big industrial organizations.

Simon formulated in [Sim58] another rationality framework, called *bounded rationality* or *satisficing decision making.* This framework has been extended further by many researchers (cf e.g. a summary given by Lewandowski and Wierzbicki in [LeW89b]). One of the main directions in that field was set by Wierzbicki [Wie80], who formulated the *principle of reference point optimization* in multiobjective optimization and decision support. That principle has been extended by Wierzbicki (cf [Wie82, Wie84, Wie86]) to *principles of quasisatisficing decision making* and has been extensively used both in research and in applications (cf [LeW89a]). Parallelly, Nakayama also developed a similar method called the satisficing trade-off method (cf [SNT85]). Similar approaches and their extensions have also been elaborated and applied by many other researchers (cf e.g. [GrW94, KLW91, KoW90, KMW92, LeW89a, LAP94, Mak94b, MiS92, Nak94, Sak93, SeS88, Ste86, WeW93]).

The first natural approaches to the model analysis have been based on simulation and on the application of classical single-criterion optimization. However, it has become

obvious that the specification of a single-objective function, which adequately reflects preferences of a model user, is perhaps the major unresolved difficulty in solving many practical problems as a relevant single-criterion optimization problem. Multiobjective optimization approaches make this problem less difficult, particularly if they allow for an interactive redefinition of the user preferences.

In a typical decision situation it is reasonable to evaluate results of a decision by more than one criterion. Since the classical LP formulation allows for the formulation of only one criterion (goal function), the main objective is selected as the performance index whereas other objectives are converted into constraints whose values are treated as parameters. Although parametric optimization is a sound idea for OR-oriented users, in practical applications it is very difficult to be used. For such problems it is natural to formulate and deal with interactive multiobjective optimization.

From both methodological and practical points of view, it is rational to discuss and to implement a DSS in two stages:

- First, to build a core model that implicitly defines a set of feasible solutions X_0. The core model should include all logical and physical relations between variables but should not contain any constraints corresponding to a preferential structure of a user. Specification of a core model is discussed in Section 2.3.2.
- Second, to provide a methodology and tools for analysis of the model. This can be done in different ways, e.g. by simulation (cf Sec. 2.3.3), by some extensions of single-criterion optimization (cf Sec. 2.4.1) or by one of the implementations of multiobjective programming (cf Sec. 2.4.2).

The first stage is of a more technical nature, that requires modeling skills and knowledge of the problem. The resulting model defines a set X_0 that contains all feasible solutions. Clearly most of the $x \in X_0$ are not acceptable. However, acceptability of a solution should be assessed only by a DM. Including acceptability conditions or preferential structure into a definition of X_0 quite often results in implicit rejection of a large number of feasible solutions. Such a narrowing of X_0 is misleading a user, because in such a case he/she cannot evaluate all feasible solutions. Therefore, a clear distinction between the feasibility and the acceptability of a solution should be maintained in any DSS.

2.3.2 Model specification

When a model-based DSS is desired (cf Sec. 2.2), it can be achieved only for a problem that is understood sufficiently well to build a mathematical programming model[5], which can adequately represent a decision situation. To represent a decision situation means that the model can be used for predicting and evaluating consequences of decisions. Such a model – that implicitly defines a set X_0 (cf Sec. 2.3.1) – is typically composed of (cf e.g. [WiM92]):

- decision variables that represent actual decisions (alternatives, choices, options etc.),
- potential objectives (goals, performance indices) which can be used for evaluation of consequences,
- various intermediate and parametric variables (balance and/or state variables, resources, external[6] decisions),

[5]Mathematical programming model is a mathematical abstraction and should not be confused with programming computers (although building and using the model inevitably requires programming of a computer).

[6]That means not directly controlled by a DM.

- constraining relations (inequalities, equations, etc.) between variables that indirectly determine the set of admissible (feasible, accepted) decisions. Some of the constraints may reflect the logic of handling events represented by variables.
- outcome relations that define goals as functions of decision variables.

Building a mathematical programming model is usually a task that requires good understanding of the problem and good knowledge of model building methodology. The process of specifying the requirements to be met by the modeling process or establishing the specifications that the modeling process must fulfill is called metamodeling and one can also examine a metamodel (through the modeling process – cf [vG91]). There exist many books and articles related to practical problems of modeling (cf e.g. [HuJ90, Wie92a, Wil90]. Specification of a model that will be used in a model-based DSS should meet additional requirements. Such requirements are discussed in more details in [Mak94b].

It is usually not possible to specify uniquely a model that can yield a unique solution reflecting the preferences of a DM. For example, very often it is practically impossible (even for a good analyst or an experienced DM) to specify e.g. values for a group of constraints that would cause a feasible solution to be also acceptable. In order to illustrate this point let us consider the example of the Regional Water Quality Management (RWQM) problem. A DM may want to consider different waste water treatment technologies and the related costs, as well as standards for water quality. However he/she knows that specification of constraints for a group of (either ambient or effluent) water standards may lead to solutions that are too expensive. On the other hand, assuming constraints for costs (with water quality standards being goals) could result in an unacceptable water quality. Values of constraints are in such cases formally parameters in a corresponding optimization problem. But those values are in fact decisions that reflect the preference structure of a user. Setting constraints' value too tight would result in restricting the analysis of the problem to a (possibly small) part of feasible solutions (often making the set X_0 empty). A typical advice in such situations is to specify two types of constraints, so called hard and soft constraints (cf Sec. 2.4.1) which correspond to *must* and *should* types of conditions, respectively. But, in fact, dealing with soft constraints can easily be done within multiobjective model analysis (cf [Mak94b] for details).

Therefore, the specification of a model that defines X_0 should not include any relations that reflect conditions for acceptability of a solution by a user or a preferential structure of a DM. In the RWQM, both costs and water quality standards are treated as goals (criteria). This provides flexibility of examining trade-offs between costs and water quality. Hence, the so-called *core model* accounts only for logical and physical relations between all the variables that define the set X_0 of feasible solutions. The specification and parameters of the core model are not to be changed after a verification and validation of the model is done. All other constraints and conditions that implicitly define acceptability of a solution by a user and those that represent a preferential structure of a DM will be included into an interactive procedure of the model analysis. Therefore a mathematical programming model that is solved is composed of two parts:

- A constant and usually large core model. This part is built and verified before an actual analysis of a problem starts.
- A part that corresponds to a current specification of goals and conditions set by a user. This specification is interactively being changed, often drastically, by a DM.

A proper implementation of such an approach makes it possible for a DM to analyze feasible solutions that correspond to a his/her preference structure. Changing this structure is the essence of the model analysis and of the model-based decision support. An additional bonus is due to the fact that there always exists a feasible solution of the underlying

mathematical programming problem, which is a prerequisite for an analysis of complex models.

Finally, we should point out that building of a model requires collection, processing and verification of data. The famous saying *"garbage in garbage out"* implies that the problem of data accuracy should not be underevaluated. A user need not worry about possible ranges of quantities (which usually have an impact on computational problems) because this should be accounted for by the DSS. A designer of a DSS should make sure that only all substantial elements are included into the model (see a more detailed discussion on a specification of a model e.g. in [Mak94b]).

The value of a mathematical model as a decision aid comes from its ability to adequately represent reality. Therefore, there is always a trade-off between the requested accuracy (realism) of the model and the costs (also time) of its development and providing the model with data. Hence the requested accuracy should be consistent with the accuracy really needed for the model and with the quality of the available data.

2.3.3 Simulation and optimization for decision support

There is a general agreement (cf e.g. [Sim90]) that model-based DSS are generally of two types that correspond to the two ways of a model analysis: descriptive (sometimes called predictive) and prescriptive (normative). The descriptive DSS are used for prediction of the modeled system behavior without an attempt to influence it. The prescriptive DSS are aimed at providing information about controls (in managerial situations called decisions) which can result in a desired behavior of the modeled system. This desired behavior is usually evaluated with the help of goals (objective values, performance indices, etc.). In other words, a descriptive DSS helps to answer a question such as *"what will happen if"* whereas a prescriptive DSS supports answers for questions like *"what decisions are likely to be the best"*. For the prescriptive type of DSS, optimization techniques are widely considered (especially in the OR community) as good tools for selecting (out of an admissible set) a solution (part of which is composed of the above mentioned controls) which is considered *"the best"*. The term *"best"* corresponds to a solution that provides best value of a performance index (goal function, objective, criterion) or a set of such indices that are used for evaluation of the expected consequences of implementing corresponding decisions (controls). A commonly accepted (by OR community) approach is to assume that a mathematical model corresponds well to reality and it is possible to define a performance index that reflects well the preferences of a DM; hence, an application of an optimal solution for the mathematical model will cause in reality *"the best"* results. However, as we will show later in Section 2.4, such a conclusion is often erroneous and optimization alone usually does not provide *"the best"* solution.

Therefore it is desirable to use a model-based DSS in both (i.e. descriptive and prescriptive) modes interchangeably. For example, before even trying to find prescriptions, one should verify the model also in the descriptive mode. The model should not only conform to the formal specification but also all discrepancies between intuitive judgment of a DM and analytic results obtained from the model must be resolved. In other words, such inconsistencies show that either the model (assumptions, specification, data) or the DM's intuition is wrong. Conflicts between results provided by the model and what is perceived by a DM must be resolved before the DM may trust the model, which is obviously a necessary (but often forgotten) condition for the actual use of a DSS. This applies, in particular, if the model used in a DSS is not restricted to substantive aspects of the decision situation but incorporates also some preferential aspects, that is, it tries

to implicitly represent the preferences of the DM.

Simulation is one of the tools useful for running a model in a descriptive mode. For simulation, one may use random values for variables (cf [GoW91] for different techniques and examples) or assign values basing either on DM's intuition or on a heuristic (possibly based on information from a knowledge base). So-called inverse simulation (cf [Wie92a]) is a very useful technique for examination of decisions in situations, when specification of a set of feasible decisions is not easy. Obviously, a mixture of these techniques can be used for two groups of variables since values of the variables selected (into the group of simulated variables[7]) can be temporary fixed. Various simulation techniques applied in the descriptive mode may provide information not only for model verification but also may lead the DM to modification of selected constraints or goals.

The usage of simulation and optimization can be compared as follows:

- In simulation mode decision variables are inputs and goals are outcomes. Therefore this technique is good for exploring intuition of a DM, not only for verification of the model but also for providing a DM with information about consequences of applying certain decisions (for example, what would be the value of goals and constraints). One can also consider simulation as an alternative-focused method of analysis that is oriented to identify (examine) the alternatives.
- Optimization can be considered as a goal-oriented (value-focused) approach that is directed towards creating alternatives. Optimization is driven by hope to reach a set of goals[8] (objectives). Therefore goals are a driving force and the values of decision variables are outcomes. This is very appealing. However, one should not forget about disadvantages that are consequences of the two facts. First, usually not all real objectives are included into the goals formulated for the optimization (cf Section 2.4). Second, an optimal solution is found on a set of feasible solutions and a DM might prefer to change (usually implicitly) this set, if he knew that, for example, a substantial increase of a goal considered to be of secondary importance may result from a relatively small change of one of goals that is represented in the model as a constraint.

Therefore, an interchangeable use of both (simulation and optimization) techniques has obvious advantages, especially in the learning phase of using a DSS. In particular, a DSS should be armed with options of changing selected constraints into goals, fixing values of some variables, etc.

2.4 Optimization in DSS

Let us concentrate in this Section on the role of optimization in DSS. It has been observed (cf e.g. [Ack79]) that many DSS are driven by optimization techniques, which means that a user has only partial control of the way in which analysis of the model is being done. This observation justifies not only the demand of using techniques of descriptive DSS as an equally important option in a prescriptive DSS, but also calls for the rethinking of problems related to using optimization within a DSS. To simplify the discussion, let us assume for a moment that we deal only with single-objective optimization. This is not a restriction since all the following arguments are also valid for multi-objective optimization.

From the (traditional) OR perspective, it is natural to formulate a mathematical

[7] Variables for which values are set by a user.

[8] Not in the sense of goal programming - cf Section 2.4.

programming problem in the form:

$$\hat{x} = \arg\min_{x \in X_0} \mathcal{F}(x) \tag{1}$$

and solve it. A concise formulation of (1) may be misleading for those who don't know that very often solving a mathematical programming problem is a challenging task. One should be aware of both the scientific values and of the resources required for finding and implementing an algorithm that can provide (using possibly small computer resources) a (correct !) solution $\hat{x}$ from a set X_0 that minimizes the objective $\mathcal{F}(x)$.

However, a DM has a completely different perspective. Let us briefly summarize some elements of this perspective (typical for non-engineering applications of DSS) which differ to the traditional OR way of formulating and solving a mathematical programming problem:

- A unique specification of both a mathematical model and of one performance index is very difficult, if at all possible, for most real-life situations. Therefore, a series of cycles composed of analysis of the results provided by a solution of the model and modification of the preference structure of the DM (expressed, for single criterion optimization, in the form of one objective and constraints for selected quantities representing other goals) is a typical desired activity, not only for the initial stage of using a DSS.
- Models are simplifications of reality, and optimization is limited to the model that includes an objective that is always a simplification of a preferential structure of the DM. Therefore, optimal solution of a model may not necessarily be optimal in reality, as perceived by a DM. It may be desirable to modify an "optimal" solution, in order to take into account some factors yet not accounted for in the underlying model (very often some of them are not included deliberately). A DMP is composed of subproblems analyzed/solved independently; therefore, overall optimum is usually not composed of optima computed for each sub-problem separately.
- DM usually prefers to be sure that his sovereignty in making decisions (for which he is responsible) is not jeopardized by a computer. The main reason is psychological[9]. Therefore it is important that a DM – who rarely is also a computer guru nor does he/she want to devote substantial amount of time to dig into hundreds of pages of software documentation – understands well enough all important functions of the DSS.

Adding this perspective to the discussion presented in the previous section, it is clear that optimization in DSS should have quite a different role than the function of optimization in some engineering (especially real-time control problems) applications or in very early implementations of OR for solving well-structured military problems. This point has been clearly made by Ackoff in [Ack79].

Optimization would be better accepted outside the OR community, if users would be able to treat optimization as a tool for selecting a number of solutions that have certain properties and if a support for comparing (from various perspectives preferred by a DM) such solutions would be widely available. This is, however, not only contrary to a traditional way of using OR methods (cf e.g. [Cha92]) for solving a problem in the following five stages: describe the problem, formulate a model of the problem, solve the model, test the solution, implement the solution. It is also contrary to another traditional OR perspective that implies that an optimal solution is also the best available solution which in turn implies that there is neither much need nor room for human decision making.

[9]It is a commonly known fact that even the developers of DSS's supporting choice (out of a given set) of an alternative do not necessarily follow optimal solutions suggested by their own DSS when solving a personal problem. However, they use the DSS for an analysis of the problem.

Also text books on DSS addressed to managers (e.g. [Eme87]) often treat optimization merely as a tool for providing *the* solution. To make the situation even worse, many of such books still present only single-objective optimization, whereas multi-objective optimization, when properly used, remarkably softens this perspective.

Let us try to summarize some basic conditions that can make optimization techniques more useful in DSS:

- Optimization should be implemented in such a way that a user considers it as just an option that facilitates an analysis of the problem by selecting solutions having certain properties represented by the, interactively modified, goals or preferences. Designers of a DSS should never forget that the aim of a DSS is not at all to discover an optimum solution, but to help DM to improve the DMP. This implies that optimization should be just an element of a DSS, which sometimes may play a central role, sometimes is only an additional tool and sometimes is not used.
- Simulation should be at least as important an option as optimization. A DM should not be "driven" towards an optimization option until he/she clearly recognizes a need for using it.
- Optimization should be well combined with simulation. This should include possibilities of fixing values of variables and/or goals, modification of a set of goals (both treating goals as soft constraints and vice versa as well as changing definitions of goals), and looking for a suboptimal solution with certain additional properties. Fixing values should not be implemented as constraints. Rather the *regularization* or the so-called *inverse simulation* techniques should be used (cf Section 2.4.1 for more details).
- A DSS should provide information and tools for making it possible and easy to understand consequences of a decision. A decision might be a result of a choice made by a DM, but it might also be a suggestion computed as an answer to an explicit or implied selection of goal(s) used as the driving force for optimization.
- In some applications, it might be reasonable to refrain from providing an optimization option. For example, in situations where a DM has better intuition in selecting values of decision variables (for simulation mode) than in selecting goals.

2.4.1 Extensions of the traditional formulation of LP problems

We have selected for the discussion the classical LP problem but a similar approach can easily be adapted and used for non-linear and mixed-integer problems. The purpose of the discussed extensions is to make the examination of various properties of an LP model easier, especially those related to suboptimal solutions with additional properties and those related to feasible set of solutions.

Let us consider some possibilities of extending the traditional formulation of a linear programming problem (LP):

$$\min cx \tag{2}$$

$$b - r \leq Ax \leq b \tag{3}$$

$$l \leq x \leq u \tag{4}$$

to the following framework form:

$$\min w_0 * < c, x > + \sum_{i=1}^{k} w_i * f_i(x, v) \tag{5}$$

$$\tilde{b} - r \leq Ax \leq \tilde{b} \tag{6}$$

$$l \leq x \leq u \tag{7}$$

where $\tilde{b} = b + Iv$, I - is a diagonal selection matrix, $v \geq 0$ is a vector of auxiliary variables. The selection matrix I contains binary elements with values set to one for elements corresponding to the constraints that are treated as soft constraints. Therefore, the vector $\tilde{b}$ corresponds to the right hand side vector where the auxiliary variable v enters a modified objective function in order to provide a mechanism for minimizing the differences between the original right hand side values b of constraints and the values $\tilde{b}$ which are generated by converting *hard* into *soft* constraints. Note that the problem (5)-(7) boils down to the original LP problem (2)-(4), if the selection matrix I has all elements set to zero and $w_0 = 1$, $w_i = 0 \ \forall i > 0$. The problem (5)-(7) is formulated in a way that is convenient for discussion. In an actual implementation, one can obviously generate only appropriate variables and constraints that correspond to a particular *soft* constraint or to a group of *soft* constraints that should be treated in a similar way. One can also convert bounds (4) into *soft* bounds that would be a modification of (7) by the same approach as is used for constraints.

We will now consider several possibilities of actual specifications of the objective template (5) and will briefly discuss the interpretation of the corresponding mathematical programming problem. For the sake of brevity we will use the following notation: C is a selected positive number and $\| \cdot \|$ is L^2 or L^∞ norm (possibly for a subvector of x, which would correspond to a group of variables that should be dealt with using a corresponding modification). Using the L^2 norm is for many applications more appropriate than using the L^∞ norm, but then the resulting problem is no longer an LP problem. However, some LP solvers (e.g. HYBRID - cf [MaS93b]) are also capable of solving the resulting linear-quadratic programming problems.

Let us consider four possible approaches that can often be useful during an examination of an LP problem. The following modifications can easily be implemented by extending capabilities of the software that is used for generating an LP problem. For the sake of simplicity, we will not consider possible mixtures of the presented modifications.

1. Quite often, especially in the initial stage of an examination of an LP problem, a solver reports that no feasible solution exists. It is a commonly known fact that the corresponding information (about inequalities that are infeasible at the point when the algorithm is stopped) is usually not sufficient to trace the real source of the problem (mainly because most LP algorithms have no robust way to specify a set of infeasible constraints and in addition, for many problems such a set is not unique). In order to make the examination of the problem feasibility easier, the objective defined for an LP problem may be specified using the following definition of parameters and functions in the template (5):
$$w_0 = 0, \quad w_i = 1, \quad f_i = \| (Ax - b)_+ + (b - r - Ax)_+ \| \tag{8}$$
where $(u)_+$ denotes the vector composed of components $\max(0, u_i)$. One can use this approach also for preselecting candidates for a set of *soft* constraints.
2. One can easily generate a series of problems that correspond to different selections of a group of constraints that should be treated as *soft* constraints by assuming
$$w_0 = 1, \quad w_i = C_i, \quad f_i = \| v_i \| \tag{9}$$
and using different selection matrices I. The values of C_i (which might be different for different groups of constraints) provide control of the trade-off between the original objective and the other objective of narrowing the gap between the values that correspond to the *soften* and the original hard constraints, respectively.

3. An LP problem may have non-unique optimal solutions. Although this is theoretically rare, in practice many problems actually have a large set of widely varying basic solutions for which the objective values differ very little (cf e.g. [Mak80, Sos81, MaS89a]). In most cases, an optimization algorithm stops when a current solution is recognized as optimal for a given set of tolerances. For problems with a non-unique optimum, the first optimal solution found is accepted, so that one may not even be aware of the non-uniqueness of the solution reported as optimal. Thus, in practical applications, we are faced with the problem of choosing an optimal (or, in most cases, to be more accurate, a suboptimal) solution that possesses certain additional properties required by the user. This problem may be overcome by applying an approach called *regularization.* Regularization (Tikhonov's type) is a way of finding the optimal solution with either minimum norm or minimum distance from a given reference point. This can be achieved by setting in (5):
$$w_0 = 1, \quad w_i \rightarrow 0_+, \quad f_i = \| x - \alpha \| \tag{10}$$
where α will be replaced by one of the following three terms: $\bar{x}$, 0., x^j, respectively. The choice depends on the desired properties of the solution. If one knows the desired values of decision variables, then those values should be used for defining the vector $\bar{x}$ and the first choice should be applied. The second choice corresponds to a preference of the minimum norm solution, whereas the third choice serves mainly for numerical stabilization of some algorithms without any preferences for additional properties of the solution. The notation " $\rightarrow 0_+$ " means that during the computation one uses a sequence of positive numbers that converges to zero. For the interpretation of final results (and also for some applications) this sequence can be replaced by " ϵ " (which is by convention a small positive number).
4. In some situations it might be desirable to look for a solution which - besides a minimization of the given objective - would have values of selected variables possibly close to given "reference" values. The corresponding problem can be generated by setting in (5):
$$w_0 = 1, \quad w_1 = C, \quad f_1 = \| x - \bar{x} \| \tag{11}$$
where $\bar{x}$ is composed of the corresponding reference values and the parameter C serves for the control of the trade-off between the original objective and the other objectives (i.e. finding a solution possibly close to a given point). This option is equivalent to the so-called *inverse simulation* (cf [Wie92a]), if $w_0 = 1$, $C = 1$ and the vector x is composed of all decision variables. The inverse simulation is very helpful in simulation of models that are likely to have no feasible solutions for values of decision variables that a DM would like to examine.

The above presented ways of extending the traditional formulation of mathematical programming problems towards a better meeting of the requirements of using a mathematical model within a DSS are not new to many of those who are involved in real-life applications. However making these approaches more popular and combining them with multi-objective optimization would widen the acceptance of OR methods. Optimization can and should be a very useful component of a DSS provided that it does not pretend (nor make an impression of doing so) to be a governing way for selection of *"the best"* solution.

2.4.2 Multi-objective model analysis

Many commonly known shortcomings of single-objective optimization have led to the development of methodology and techniques for application of multi-objective program-

ming (MOP) to analysis of a model used for decision support. Different MOP techniques and experiences from applications (cf eg [LeW89a, HoW93, KaZ89, KeR76, KMW92, KLW91, LAP94, MiS92, Sak93, SeS88, Ste92b, Vin89, WeW93, Yu85]) make it possible to remarkably increase the usefulness of optimization in DSS. We will briefly summarize only one MOP technique, which seems to be the most natural method that best corresponds to a real-life DMP. This is the method based on the *aspiration level* (sometimes referred to as *reference point*) concept. Following [OgL92] we will use for this method the acronym ARBDS (Aspiration Reservation Based Decision Support). The ARBDS method is an extension of the reference point method proposed by Wierzbicki in [Wie80]. Several other extensions or similar approaches have been proposed and implemented (cf [LeW89a, LAP94, Nak94, SNT85, SeS88, Ste86]). The differences between the approaches presented in those publications are discussed and a modular implementation of ARBDS approach is presented in more details in [Mak94b]. Here we only outline the ARBDS method.

The basic concepts of multiple criteria optimization are well described in the above listed publications. We recall only those that are needed for characterization of the ARBDS:

- Consequences of a decision $x \in X_0$ (cf Sec. 2.3.1) are evaluated by values of criteria $q \in R^N$, where N is a number of criteria selected by a user. Each criterion can be maximized, minimized or stabilized (a deviation from a given target value is minimized for each stabilized criterion).
- A solution $x \in X_0$ is called *a Pareto-optimal solution*, if there is no other feasible solution for which one can improve the value of any criterion without worsening the value of at least one other criterion. A Pareto-optimal solution is also called *an efficient* solution. A solution $x \in X_0$ is called a weakly Pareto-optimal solution, if there exists no other feasible solution that has better values of all criteria. Weakly Pareto-optimal solutions are easier to be computed than an efficient solution, but often have no practical meaning. Most practical in applications are properly Pareto-optimal solutions with a prior bound on trade-off coefficients (see [Wie86]). Therefore, a proper method should be implemented to avoid reporting a weakly or not properly Pareto-optimal solution as an efficient solution. This is a purely technical problem (see [Mak94b] for details). Therefore, for the sake of brevity we will refer to properly Pareto-optimal solutions as to Pareto or efficient solutions (unless otherwise mentioned). For N minimized criteria one can define $\hat{x} \in X_0$ as a Pareto-optimal (efficient) solution by the following condition:

$$\neg\exists x \in X_0 \neq \hat{x} : \ \{q_i(x) \leq q_i(\hat{x}) \ \forall i \in [1,...,N] \text{ and}$$
$$\exists k \in [1,...,N] : q_k(x) < q_k(\hat{x}) \ \} \qquad (12)$$

- A Pareto-optimal point is composed of values of all criteria for a corresponding Pareto-optimal solution.
- The utopia and nadir points are composed of best and worst values out of the set of all Pareto-solution for each criterion respectively. One should note that finding an utopia point can be done from a pay-off table, however finding a nadir point maybe a difficult problem (cf e.g. [IsS87, Mak94b]). Utopia and nadir (or a good approximation of a nadir) provide valuable information about ranges of values (for all efficient solutions) of each criterion.

The ARBDS approach may be summarized in the form of the following stages:

1. The DM selects, out of the potential objectives, a number of variables that will serve as criteria for evaluations of feasible solutions. In typical applications there are 2–7 criteria. For an LP problem, a criterion is often a linear combination of variables but criteria may also have a specific form for an actual application (cf e.g. [MaS87]).

2. The DM specifies (with a help of an interactive tool) an aspiration and reservation levels $\bar{q} = \{\bar{q}_1, \ldots, \bar{q}_N\}$, $\underline{q} = \{\underline{q}_1, \ldots, \underline{q}_N\}$, where N is a number of criteria and $\bar{q}_i$ and $\underline{q}_i$ are desired values and worst acceptable values for each criterion, respectively.
3. The problem is transformed by a DSS into an auxiliary parametric single-objective problem. Its solution gives a Pareto-optimal point. If a specified aspiration level $\bar{q}$ is not attainable, then the Pareto-optimal point is the nearest (in the sense of a Chebyshev weighted norm) to the aspiration level. If the aspiration level is attainable, then the Pareto-optimal point is uniformly better than $\bar{q}$. Properties of the Pareto-optimal point depend on the selection of the reference point (aspiration and reservation levels) and on the resulting weights[10] associated with criteria.
4. The DM explores various Pareto-optimal points by changing the aspiration and/or reservation levels $\bar{q}$ and $\underline{q}$, respectively. The underlying (done by a DSS) formulation of the problem is minimization[11] of a (piece-wise linear) *achievement function* that can be interpreted as an ad-hoc non-stationary approximation of the DM's value function depending on the currently selected aspiration and reservation levels.
5. The procedures described in points 2, 3 and 4 are repeated until a satisfactory solution is found. Additionally, the user can temporarily remove a criterion (or a number of criteria) from analysis or convert selected criteria into stabilized criteria. This option results in the computation of a Pareto optimal point in respect to remaining "active" criteria, but values of criteria that are not active are also available for review. The utopia and nadir points help to define reference points in the procedure outlined above because it is reasonable to expect values of each criterion to lie between utopia and nadir points.

The ARBDS method can be considered as an extension of the commonly known technique called Goal Programming (GP) originally proposed by Charnes and Cooper in [ChC67]. One can compare GP to using all goals as soft equality constraints, while ARBDS approach includes also soft inequalities of both signs (for maximized or minimized objectives) as well as soft equality constraints (for stabilized objectives). Thus, the ARBDS approach produces Pareto-optimal solutions, while the traditional GP fails to produce them, if aspirations are attainable. The detailed comparison of the two methods is provided by Ogryczak and Lahoda in [OgL92]. All functionality of the GP can be provided by the ARBDS method and computational complexities of both methods are comparable. Therefore the ARBDS seems to be a good replacement for the GP method.

A more formal presentation of the mathematical background of the ARBDS method can be found e.g. in [Wie92b]. The reference point and ARBDS methods have proved to be useful in many applications (cf [LeW89a, Mak91b] for a summary) in different areas. The software packages described in [MaR91] implement this methodology for different types of mathematical models. A modular software tool LP-MULTI, which is an implementation of the aspiration led multiple-criteria model analysis, is documented in [Mak94b].

[10]Weights in the achievement scalarizing function should not be confused with using weights as a tool for conversion of a multi-criteria optimization problem into a single-objective problem. Different ways of definition of those weights (which are in fact scaling coefficients for criteria) are the main differences between the various approaches listed above (cf [Mak94b] for the discussion of related problems).

[11]It can be also formulated as maximization problem, depending on the interpretation of the achievement function.

2.5 Applications of Artificial Neural Nets in decision support

Artificial Neural Net(s) (ANN) are being recently applied in many fields such as engineering, computer sciences, management. Therefore ANN are more widely applied also to decision support. One can distinguish two main groups of such applications:

- Applications of ANN for solving different types of mathematical programming problems. In such cases, an ANN replaces a traditional implementation of an algorithm for solving a well-defined mathematical model of the underlying decision problem. ANN can be used for solving different types (linear, nonlinear, discrete, combinatorial) of optimization problems, for linear algebra problems, for estimation, identification and prediction problems. For example, an overview of implementations of ANN in those fields is given by Cichocki and Unbehauen in [CiU93], implementation of ANN to exact classification can be found in [AZW92], and theoretical foundations for sorting with ANN are presented in [ZAW91].
- Applications of ANN to solving problems that are too complicated to be adequately well formulated and/or solved as a corresponding mathematical programming problem. In such cases, an ANN is a kind of *black-box* (cf Aarts et.al. [AWZ93]) that can be tuned (trained) in order to mimic a rational behavior. Problems of this type include production planning (cf [ZvKAW91]), spatial analysis (cf [Fis92]), land use planning (cf [YiX91]), elicitation, representation and utilization of the decision maker's preference information (cf [SSS93]).

We will briefly discuss only one new possibility of ANN application for decision support. The interactive multicriteria model analysis outlined in Section 2.4.2 requires, for each specification of aspiration levels, solution of an underlying optimization problem. An efficient LP solver is capable of solving a medium size problem in a time still acceptable for an interactive procedure. However, times required by best available solvers for solving large problems (cf the example in Section 3.5) make it practically impossible to apply an interactive procedure. In such a case an implementation of an ANN would be useful because an ANN would be capable to solve the corresponding LP problem in a much shorter time. In order to illustrate this statement, let us consider a possibility of using ANN for the solution method used by the HYBRID solver (cf [MaS93b]).

For solving a linear programming problem, HYBRID uses a non-simplex algorithm – a particular implementation of the proximal multiplier method. This method differs from the ordinary multiplier method by adding a quadratic form to the augmented Lagrangean function and making the function strongly convex with respect to primal variables. The minimized function is strongly convex and provides a unique minimizer.

In the proximal multiplier method, the LP problem is solved by minimizing a sequence of piece-wise quadratic, strongly convex functions, subject to simple constraints (lower and upper bounds). This minimization is achieved by using a method that combines the conjugate or preconditioned conjugate gradient method and an active constraints' set strategy. As an option for solving the problem of minimizing piece-wise quadratic function a numerically stable method that combines the proximal multiplier method and an active set algorithm with QR factorization or Cholesky factorization is implemented. In these methods we solve a sequence of minimization quadratic functions without constraints. After a finite number of steps, a set of indices of constraints, which are active in the solution, is found.

The LP problem that has to be solved for each selection of aspiration levels during the analysis of an LP model can be treated as a special case of the following general linear programming problem (cf [Mak94b] for details on solving a multiple criteria LP problem

using the achievement scalarizing function):

$$\min cx \tag{13}$$

$$A_1 x = b_1 \tag{14}$$

$$A_2 x \leq b_2 \tag{15}$$

$$l \leq x \leq u \tag{16}$$

We introduce the following notation: $\|x\|$ denotes L_2–norm of x and $(u)_+$ denotes the vector composed of components $\max(0, u_i)$.

The following scheme describes a modification of the proximal multiplier method for the linear programming problem (13)–(16):

1. Select initial vector of multipliers $y^0 = (y_1^0, y_2^0)$ $(e.g., y^0 = 0)$ and $\rho^k, \gamma^k \in R$, $\rho^k, \gamma^k > 0$.
2. For $k = 0, 1, \ldots$ determine successive x^{k+1}, $y^{k+1} = (y_1^{k+1}, y_2^{k+1})$ where

$$x^{k+1} = \arg\min_{l \leq x \leq u} L(x, y^k) + \frac{1}{2\rho^k \gamma^k}\|x - x^k\|^2 \tag{17}$$

and

$$y_1^{k+1} = y_1^k + \rho^k(A_1 x^{k+1} - b_1) \tag{18}$$

$$y_2^{k+1} = (y_2^k + \rho^k(A_2 x^{k+1} - b_2))_+ \tag{19}$$

where

$$L(x, y^k) = (1/\rho^k)cx + \frac{1}{2}\|((1/\rho^k)y_1^k + A_1 x - b_1)\|^2 + \frac{1}{2}\|((1/\rho^k)y_2^k + A_2 x - b_2)_+\|^2 \tag{20}$$

until a stopping criterion is satisfied.

The theoretical justification of this algorithm can be found in [MaS93b]. The tests made with over hundred examples (including all commonly used examples from the Netlib collection) show that the algorithm rarely needs more than 3 multiplier iterations. Therefore the crucial point of the method is the problem of minimization of piece-wise quadratic functions (17). The quadratic term in (17) is needed for application of a numerically stable minimization method for piece-wise quadratic strongly convex function. However, it can be dropped, if high accuracy of solution is not required. In such cases one can minimize (in k-th multiplier iteration) the function

$$f^k(x) = c^k x + \frac{1}{2}\|A_1 x - b_1^k\|^2 + \frac{1}{2}\|(A_2 x - b_2^k)_+\|^2 \tag{21}$$

which is a trivial modification of (20).

The ordinary or preconditioned conjugate gradient algorithms can be used for finding the minimum of (21). The details of such algorithms can be found in [MaS89a]. An implementation of such algorithm by an ANN can be done in a way similar to the implementation proposed in [CiU93] for solving LP problems using the classical (ordinary) penalty function. The basic advantage of the shifted penalty function method implemented in HYBRID (over the ordinary penalty function) is due to the fact, that the later one requires the penalty coefficient $\rho^k \to \infty$ while the shifted penalty function guarantees an exact solution for a finite (usually smaller than 1000) value of ρ. Therefore the tuning of an ANN designed for the proposed algorithm will be much easier. However, such an application of ANN will become rational when neural nets with hundreds of thousands' perceptrons will be available.

3 Design and implementation of a DSS

The problem of a DSS design and application has been well covered by many conferences, books and articles. The topic itself is far too broad and complex to be fully presented and discussed within a paper. An overview of three groups of related problems, namely requirement analysis, features of a DSS, and an implementation process, as well as a large list of references is provided in [Mak91b]. Below we briefly summarize a few issues related to hardware and software considerations and to development of reusable software tools.

3.1 Hardware considerations

Appropriate selection of the computer hardware should be made by taking into consideration the hardware currently being used by a DM and taking into account both the computational requirements[12] and the software tools provided by specific computer hardware (one should however be aware of portability issues - cf Section 3.2). Prices for computer hardware are rapidly decreasing, and computer hardware capabilities are rapidly increasing. In order to illustrate this trend, let us present in Table 1 a summary of a panel discussion[13] on the future of hardware. The summary has a form of the forecast of the

	1991	1996	2001
Memory	16-64 MB	64-256 MB	256-1 GB
CPU speed	20 MIPS	200 MIPS	1000 MIPS
Disk space	200 MB	1 GB	??
Price	10000$	3000$	below 1000$
Network throuput	10 MB/sec	100 MB/sec	1GB/sec

Table 1: Forecast (made in 1991) of capabilities of a typical workstation.

basic characteristic of a computer that poses the computing capabilities corresponding to those of a typical (in 1991) workstation. The prices refer to a configuration that will be an equivalent of the 1991 configuration given in the Table 1.

Many researchers confronted in 1991 with the forecast did not consider it to be realistic. Today the development of hardware seems to be even faster than that forecasted three years ago.

The decision about selection of hardware (although is crucial) can be, for most applications, easily reached as a result of cooperation of future users and experienced software developers. Usually the hardware related costs are just a small fraction of all costs related to the development of a DSS. Therefore hardware is nowadays no longer a limitation for development and using of a DSS.

3.2 Software considerations

Software related problems are key components of any decision related to design and implementation of any DSS. This is due to three mutually related reasons:

[12]This should definitely include inevitable extensions of requirements for DSS functions; therefore, one should make sure that the requirements are not underestimated.

[13]The discussion was held during the Seminar on *Symbolic Mathematical Computation* organized by IBM Europe Institute in Oberlech, Austria, on 29 July – 2 August, 1991.

Costs: Software development and maintenance is a major cost component of probably[14] any DSS. Direct costs of development of any dedicated software are higher than those of purchasing a *ready from the shelf* package whereas the relation of their functionality are the opposite. Direct costs of software development may be remarkably reduced by two factors: first, using appropriate software tools (both as parts of a DSS and for software development), second, by reusability of the developed software. There are two key principles that allow for the re-use of software. First, the software should have a *modular structure* and should be well tested. This implies that the popular *"quick and dirty"* programming technique should never be used in DSS development (no matter how far behind schedule the project is). Second, the software should be developed in such a way that it is *portable* between many possible architectures (this will almost be necessary, if one considers arguments presented in Section 3.1).

Implementation: The general belief is that software is "never" ready on time. This is expressed both by users (cf e.g. [Eme87]) and by software developers (the latter formulated in the so called *5% principle* which says that the last 5% of the scheduled development time usually takes about 50% of the real development time. Very often the testing phase of software development is under evaluated despite the commonly known fact that there is probably never enough software testing. Therefore quite often either the software is not ready on time or that an inadequately tested version of the software is released. Emery (cf [Eme87]) provides a very critical overview of quality of software and mismanagement in software development for MIS (Management Information Systems). Ackoff (cf [Ack67]) even suggests that MIS stands for Management MISinformation Systems.

Credibility: There is probably not one bug-free software package[15], therefore it is important to use software engineering techniques that ease the software development and result in better end products like the modular structure of each DSS component and the application of object oriented programming languages, as well as utilization of well-tested software tools. Reliability of a DSS is obviously a key condition for its usability.

The above arguments can be summarized with general advice that one should prefer both the programming languages and tools that allow for fast development of software that is *robust (reliable), reusable and portable.* If one considers also arguments presented in the previous sections, then the software should allow for fast prototyping, modifications, testing and also be used easily and efficiently.

Since there are such diversified objectives for software development, there exists no single set of rules that should be followed. However, it is sensible to make a list of issues that should be considered in all cases. Let us first briefly summarize software trends formulated together with the hardware forecast presented in Section 3.1:

- Slow programming languages' drift will continue with increasing share of object oriented programming (OOP) languages. This implies a rapid increase of usage of the C++

[14]Probably refers only to situations when a ready software package can be purchased and provided that this package fits well enough to the problem. Well enough means that costs for learning, customizing and upgrading the package and indirect costs of using it are negligible. Indirect costs are due to the difference of time that is required from a user for using a *ready from the shelf* package and that required by learning a dedicated DSS.

[15]Therefore almost all software packages sold on the market contain various disclaimer statements within the license agreement.

language[16].

- Network and data share will become universal. This will result in making popular *computing on the move* and - with usage of the cellular communications - collaborative computing will also be more popular.
- There are significant challenges of integration of symbolic and numerical (parallel) computation and visualization in order to provide tools that will ease development of software in different computing environments and for specific problems.
- There will be more powerful tools (especially for symbolic computation). However, one should be aware that the more powerful a tool is, the less probable is its full control, and tracing of a bug is more difficult.
- There will be a rapidly increasing demand for *guaranteed* software, i.e. a software that is guaranteed to function according to a given (much more rigorously than nowadays) specification.

One can consider (cf [Sil91]) the DSS development as composed of a three layers' structure. On the first layer, there are software tools that are developed by a "tool smith". On the second layer, a DSS builder uses those tools and a DSS generator generates a problem specific DSS. On the third layer, the DSS is used by a user. It is usually difficult and not practical to separate the first two layers because the generation of a DSS often requires development of a new tool. However, one should try to make such tools reusable whenever sensible.

Software tools are generally of three types: programming languages, general purpose tools (like utilities for development of the user interface or for handling typical data structures) and tools specific for a given category of problems (either mathematical programming problems or substantial problems).

Let us first briefly discuss a few problems related to software tools. Usually it is sensible to use a set of general purpose tools that fit well to a software development environment and which conform to a requirements set for a DSS. Tools specialized for specification and generation of a specific type of model can also be very useful, but here a more careful analysis is needed. One can distinguish two extreme situations. First, when a general purpose or a specialized tool is available for a developer who is (or can easily become) familiar with this tool. Sometimes, a reasonable approach to generate a core model is to use an *integrated modeling environment* (e.g. AIMMS [BiE93], AMPL [FGK93], GAMS [BKM88]). In other situations, using a *problem oriented tool* remarkably improves the design and implementation of a DSS (cf [Wes91, Wes93]) and thus allows for saving substantial amount of resources. ExSpecT, which is a specialized tool for production planning problems (cf [vdA92]), is a good example a such a tool. Second situation occurs, when the time needed for mastering a *general purpose model generator* (to the extent needed for more complex models and problems related to the model verification and update within the integrated environment of the DSS) justifies implementation of a *specialized model generator*, which can easily be written by a system analyst. Also requirements for data handling and user interface quite often make usage of existing general purpose modeling tools not practicable, especially for many complex, real-life applications when specific requirements (like access and processing of data, user interface) are difficult to be fulfilled by an integrated modeling environment. Development of specialized tools is a resource and a time demanding task, but the example of ExSpecT shows that it might be sensible to do it in a situation when a similar type of DSS will be generated many times.

[16]Note that this forecast was made in 1991. The number of programmers who use C++ in 1994 may serve as confirmation of this forecast.

One specific type of tools are 4GL (fourth generation languages). 4GL are non-procedural languages that are often considered to be more productive than traditional (procedural) programming languages. The main argument is that for 4GL it is enough to define *"what to do"* whereas procedural languages require additionally programming of *"how to do"*. This, however, does not necessarily mean that 4GL are always a better choice than a procedural programming language. A 4GL is in fact a prespecified specialized (for certain task) tool. Therefore, an extension of its functionality (beyond tasks for which it was designed) is usually difficult, if at all possible. An example of 4GL is the SQL (Structured Query Language) which is typically provided by most DBMS. Development of a typical data handling application is much easier with an SQL tool but incorporation of such a tool into a model-based DSS might be more resource-consuming than using another tool for data handling.

It is important to use software engineering techniques that ease the software development and result in better end products. These techniques include the modular structure of DSS components and utilization of well-tested software tools. All DSS software components that have to be developed should be developed preferably by using object-oriented programming languages. The consequences of selecting a programming language usually last for years. However, it should be pointed out that software and data structures which are designed properly make it possible to use different programming languages even in one program. A more detailed discussion of topics related to the development of software for DSS is far beyond the scope of this paper. Guidelines for development of such software are proposed in the separate paper (cf [Mak91a]). Here we will shortly comment only on the following five issues:

Software robustness and reuse: Software robustness is understood as its ability for functioning according to the specification and to the needs of the user. This also includes proper behavior for possibly any data supplied for the program, any action of the user and for typical hardware problems. Software for DSSs should obviously have a high level of robustness. The software robustness is also the necessary condition for the software reuse, and justifies the careful design and testing of all software components. A large part of the software should be reusable both in different applications and by different teams without the necessity of any modification of the available software. This requires some additional programming effort, but it is enough to consider how much time is necessary to debug and test new software to justify this effort. This reusable software should be carefully designed, tested and converted into libraries that are well documented. The reuse of some specific software (e.g. solvers) requires an additional agreement on the common structure of data for a specific class of mathematical programming problems.

Software portability: The portability is an important issue, especially for the software developed within the DOS environment. In the past, DOS may have been considered as a good operating system[17] for prototyping and small scale applications, however, for many real-life applications, more powerful operating systems (mainly Unix) are more appropriate. Unfortunately, too many DSS software packages developed under DOS are hardly portable to any other operating system because non-portable extensions of programming languages were widely used. Nowadays, there exist many

[17]Mainly because, in the last few years, it has been one of the very few widely available operating systems. However, because of the recent change in the price structure for computer hardware, the Unix-based workstations are becoming more readily available and are likely to replace PC's for many DSS applications.

portable tools[18] for software development. Therefore, portability is now mainly a question of a careful design of the software and of a proper selection of software tools used for implementation of a DSS.

Software modification: Well-designed and implemented software should be easily modifiable for specific purposes. This includes e.g. modification of the user interface for a specific decision making problem or translation to another spoken language or extension for supplying additional options.

Software efficiency: A choice of an algorithm for solving a problem is critical for software efficiency. The classical example of simple improvements of a simple algorithm (for selecting prime numbers) that finally result in the cutting execution time by a factor of several thousands can be found in [Ben88]. A well known *"rule of thumb"* says that 20% of code accounts for 80% of execution time. This critical part of the code may be easily identified with the help of a profiler. Those two arguments (together with continuous improvements of code optimization provided by compilers) suggest that implementing various programming tricks (which often decrease maintainability of programs) is not recommended.

New approaches: Combining knowledge based and algorithmic approaches can be a sensible method also for design and implementation of tool for generation of both models and scenarios. Also application of artificial neural nets (cf Section 2.5) is a relatively new, in DSS area, but promising field of research.

3.3 Modular software tools

Several modular software tools are being developed by the Methodology of Decision Analysis (MDA) Project and within the scientific cooperation between the MDA Project and different research institutions in Poland. The software has been developed for IBM compatible personal computers (running MS-DOS) and for Sun Workstations (running Solaris 2.3). Currently, the modular software tools include:

FT – A prototype implementation of the methodology proposed by Granat and Wierzbicki in [GrW94] for interactive specification of user preferences in terms of fuzzy sets (cf [GrM95] for details). Currently this tool is operational only under MOTIF running on SunOS and on Solaris 2.3. However, a portable and distributable version for MS Windows and for Solaris 2.3 should be available in beginning of 1995.

LP-DIT – Data Interchange Tool for Linear Programming Problems (cf [Mak94a] for details) is a prototype implementation for handling data that define a MIP or LP problem. LP-DIT provides an easy and efficient way for definition and modification of MIP problems and interchange of data between a problem generator, a solver and software modules that serves for problem modification and analysis of a solution.

LP-MULTI – Modular tool for multiple criteria problems (cf [Mak94b] for details) is a prototype implementation of a tool for generation and interactive modification of a multiple criteria problem. It currently uses LP-DIT for data handling and FT for interaction with a user. Therefore a distributable version of LP-MULTI will be available together with the FT Tool.

[18]Especially for Graphical User Interface (GUI), which was the main cause for non-portability of the software developed under DOS.

HOPDM – An efficient implementation of the Interior Point method for solving LP problems (cf [AlG93, Gon94]). HOPDM is especially efficient for solving large LP problems and was applied at IIASA as a solver for land-use models (cf [GoM95]) and energy models (cf [NGI+93]).

Hybrid – A modular LP solver (cf [MaS89a, MaS89b, MaS93b]), which uses a non-simplex method that combines the proximal multiplier method and an active set algorithm. Hybrid is used as a solver in the currently distributed version of the RAINS model (cf [ASH90]).

Simplex – A pilot implementation of a modular solver based on the simplex method (cf [Swi94]). This package is used for finding an optimal solution of a relaxed MIP problem thus providing a good starting point for the Momip solver.

Momip – Modular Optimizer for Mixed Integer Programming (cf [OgZ94] for details). Momip is an efficient solver that is being successfully used in the RWQM DSS (cf [MSW95]).

DIDAS-N – is a new modular implementation of modular modeling and optimization system for nonlinear problems (cf [GKPS94]). A pre-release version of DIDAS-N is being currently tested and the package will be ready for distribution in the beginning of 1995.

HOPDM, Hybrid, Momip and Simplex use the LP-DIT Tool for efficient data handling but each of these packages can also input data in the standard MPS format.

The MDA Project distributes the software (both the above listed packages and the software described in [MaR91]) under the following conditions, which correspond to the proposal formulated by Steuer in [Ste92a]:

1. The software will only be used for research or educational purposes provided that the usage is non-commercial. For any commercial usage of the software a separate written agreement is necessary.
2. The software can be used only by individuals or institutions who were issued a license either by IIASA or by the authors of the software; redistribution of the software and/or manuals requires written permission from IIASA or from the authors. No part of the software can be modified or incorporated into other software without written permission from the authors.
3. The user agrees to cite the software, if used, by giving references to the documentation that accompanies the software package and by mentioning that the software has been developed in cooperation with IIASA.
4. In no event shall IIASA or the authors of the software or the institutions that employee the authors be liable for any damages whatsoever (including, without limitation, for loss of profits, business interruption, loss of information, or other pecuniary loss) arising out of the use of or inability to use the distributed software.
5. The user will contact the authors directly in case of any problems with the software (except of possibly damaged diskettes) or if the software requires modification for her/his application. The MDA Project would appreciate it, if – at some point – the user will share with us his/her critical evaluation of the software.

The above listed software is currently distributed free of charge. A great deal of effort has been made to make the distributed software robust and reliable, but the programs cannot be – outside IIASA – serviced or supported in any way. However, comments, suggestions and bug reports are welcome.

3.4 Linear Programming Data Interchange Tool: LP-DIT

Many model-based research and applications require formulation, solution, analysis and modification of mathematical programming problems. This is especially important for model-based DSS, where a sequence of related problems is generated, solved and analyzed. The typical model-based DSS is composed of several software modules that serve for a generation of a core model, an interactive analysis of the model and a solver. These modules are complex pieces of software (which also typically have modular structure) that are usually developed by different research teams. Each module processes large amounts of *"private"* data, but the amounts of data that have to be exchanged between modules are quite often also large. Internal (*private*) data structures are different, quite often also for solvers that use a same method for solving a given type of a mathematical programming problem. In order to couple those modules, an efficient way of data interchange is needed for providing efficient access to data without restricting actual implementation of the internal data structure. Finding a commonly accepted way of data interchange is an important issue for practical applications of Operations Research tools developed by teams specialized in different fields.

One of the attempts to stimulate activities in the direction of establishing a widely accepted way of data interchange was the proposal for a Data Interchange Tool for Mathematical Programming (cf [MaS93a]). Since applications of LP (Linear Programming) and MIP (Mixed Integer Linear Programming) problems constitute a substantial part of optimization problems, a pilot implementation of a data interchange tool LP-DIT has been made for LP problems.

The LP-DIT, which is documented in [Mak94a], serves two mutually related purposes:

- To propose data structures and declarations of functions that can easily provide efficient data processing commonly needed for interchange of LP problem data between different modules.
- To provide a pilot implementation of those data and function specifications in a form of a public domain tool (fairly easy to implement and efficient to use) which allows for LP problem data interchange between different software modules.

In other words, LP-DIT provides an alternative for using the MPS format to access to, and modification of an LP problem data. Additionally, LP-DIT provides efficient and flexible functions for a full definition (which includes information contained both in the MPS format [including commonly used extensions] and in a specification file) of an LP problem, its modifications and solutions.

3.5 Modular solvers of mathematical programming problems

Software tools described in Sec. 3.3 include four solvers of LP problems. There exists also a number of commercial and public domain LP solvers. Solving an LP problem is not a problem nowadays. However, if our aim is not to find an optimal solution but to support a model analysis, then there are several issues worth to be discussed. We will not deal here with issues of robustness and of numerical stability of solvers, since we assume that a solver used in a DSS is well tested and robust.

We will summarize experiences from several applications of different solvers in order to provide some suggestions for improvement of solvers' design and implementation. Most of the following suggestions are of technical nature that are minor (and therefore often overlooked) problems for a stand-alone solvers, but are important for applications of a solver as a part of a DSS:

Preprocessing: It has been observed that a presolve analysis can substantially reduce the size of an LP problem. Therefore efficient presolvers are implemented in contemporary solvers (cf e.g. [Gon94, LMS94]). As the result a smaller problem, having the same solution as the original one, is solved. A good presolver can also often detect infeasibility or unboundness of a problem. However, a design and implementation of a modular presolver that would fit as a part of a DSS is still an open issue.

Uniqueness of optimal solution: For many LP problems there exist many solutions for which the goal function values differ very little from the optimum, while corresponding solutions are quite different. Therefore it is reasonable to introduce a way of selecting a (possibly sub-optimum) solution that has additional properties. For this, an old technique called regularization can be applied. Such an implementation has been done in HYBRID (cf [MaS89a]) but it can also be easily applied in most of other LP solvers.

Dual solution: Many researchers apply dual solution for sensitivity analysis. However, it should be pointed out that the results of such analysis might be misleading. The reasons of problems are threefold. First, many problems have no unique dual solution. Second, some solvers do not provide robust dual solutions. Third, the dual solution has a well-defined interpretation only in a neighborhood of the optimal solution. This neighborhood is not directly available from the standard output of a solver and commonly known observations show that users tend to extend the interpretation of dual solution (shadow prices for LP problems) far beyond the region in which it is valid. This issue is far beyond the scope of this paper. A reader interested in problems related to dual solutions may consult e.g. [GdHR+93, JdJRT93].

Infeasibility: A properly generated problem should have a feasible solution, but in practice problems are often formulated in such a way that there is no feasible solution. Solvers rarely provide all available information that can help in tracing source of infeasibility.

Input of the model formulation: Generation and processing of medium size LP problems in the form of the MPS format input file consume a lot of computer resources (disk space and time). Therefore solvers should have well separable functions for input the model formulation that can easily be modified for accepting more efficient ways of a problem formulation, such as LP-DIT.

Hot start for modified problem: A solver is used within a DSS for solving a sequence of modified problems. The modifications quite often involve generation of new, and/or deletion of old, variables and/or constraints. In order to shorten the time required for interactive analysis of the problem, appropriate techniques for an efficient use of a previously obtained solution should be implemented.

Access to results: Typically only a small fraction of a solution is used by a DSS. Therefore a selective and efficient access to a solution is an important practical problem (cf an example below). LP-DIT (cf Section 3.4) provides an efficient way for handling a solution of an LP problem.

Other technical issues: Information provided by solvers during the optimization process is often of very technical nature and therefore it is useless for a casual user.

Messages are designed and implemented (often printed directly on a screen) in a way that corresponds to needs of solver developers. However, those needs differ from the expectations of a DM. Some solvers generate a core dump in case of problems. Such a dump is useless for anyone but the developer and often is a source of confusion and technical problems.

As an illustration of issues typical to large LP problems, we provide a characteristic of the *world* energy model (cf [NGI+93]). The problem has (depending on the scenario) about 36,000 rows, 33,000 columns and 221,000 elements. The MPS format file of this model has about 10 MB, a text solution file has about 6 MB. It requires about 10 hours of cpu time (of Sun SparcCenter 1000) to find an optimal solution by Cplex (cf [CPL93]) while HOPDM solver [GoT94] (with processing input data by the LP-DIT) takes about 40 min. of cpu. The presolver included in HOPDM has found an infeasibility (in one of the first versions of the model) in few minutes whereas an attempt to solve this infeasible problem by Cplex took several hours of cpu. This is by no means a claim that HOPDM is a better solver than Cplex; it is just an argument for trying, especially for large problems, different solvers and to use efficient tools for handling the model formulation and its solutions.

4 Conclusions

The developments of DSSs create new challenges in many fields – mathematical programming, game theory, decision theory, psychology, computer science, etc. There are many theoretical and methodological problems in those scientific fields that have been stimulated by the development of DSS for real-life applications. A number of these problems still remain to be solved by scientists collaborating in these fields despite of many advances in mathematical programming in recent decades. However, there is a common agreement that many of those advances have resulted from needs created by applications.

A DSS is a problem specific software tool that requires a careful design, implementation and testing. This process has to be done in close collaboration between developers and users of a DSS. Although each DSS is a unique system of software modules, the development of a DSS is much easier if reusable software modules are available. Therefore reusability and portability of software are one of critical issues.

Acknowledgment

The first versions of the two parts of this paper were originally prepared for the Symposia organized by the Japan Institute of Systems Research in Tokyo in 1992 and in 1994. A large part of the paper is based on the results of scientific collaboration between the Methodology of Decision Analysis Project and several projects at IIASA. References are given whenever practical, but the author would like to take this opportunity to thank colleagues from the Food and Agriculture Project, the Transboundary Air Pollution Project, the Water Resources Project, the Institute of Automatic Control of Warsaw University of Technology, the Institute of Informatics of Warsaw University, and from the Systems Research Institute of the Polish Academy of Sciences.

The author would like also to acknowledge the detailed comments on the draft of this paper that were provided by Jaap Wessels and Andrzej P. Wierzbicki.

References

[Ack67] R. L. Ackoff, *Management misinformation systems*, Management Science **14**, no. 4 (1967).

[Ack79] ———, *The future of operational research is past*, Journal of OR Society **30** (1979).

[AlG93] A. Altman and J. Gondzio, *HOPDM–a higher order primal-dual method for large scale linear programming*, European Journal of Operational Research **68**, no. 1 (1993) 158–160.

[And89] S. Andriole, *Handbook of Decision Support Systems*, TAB Professional and Reference Books, Blue Ridge Summit, PA, 1989.

[ASH90] J. Alcamo, R. Shaw and L. Hordijk, eds., *The RAINS Model of Acidification*, Kluver Academic Publishers, Dordrecht, Boston, London, 1990.

[AWZ93] E. Aarts, J. Wessels and P. Zwietering, *Applicability of neural nets for decision support*, Working Paper WP-93-05, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1993.

[AZW92] E. Aarts, P. Zwietering and J. Wessels, *Exact classification with two-layered perceptrons*, Working Paper WP-92-49, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1992.

[Ben88] J. Bentley, *More Programming Pearls, Confessions of a Coder*, Addison-Wesley, New York, 1988.

[BiE93] J. Bisschop and R. Entriken, *AIMMS, The Modeling System*, Paragon Decision Technology, Haarlem, The Netherlands, 1993.

[BKM88] A. Brooke, D. Kendrick and A. Meeraus, *GAMS, A User's Guide*, The Scientific Press, Redwood City, 1988.

[Cha88] C. B. Chapman, *Science, engineering and economics: OR at the interface*, Journal of Operational Research Society **39**, no. 1 (1988).

[Cha92] ———, *My two cents worth on how OR should develop*, Journal of Operational Research Society **43**, no. 7 (1992).

[ChC67] A. Charnes and W. Cooper, *Management Models and Industrial Applications of Linear Programming*, J. Wiley & Sons, New York, London, 1967.

[CiU93] A. Cichocki and R. Unbehauen, *Neural networks for optimization and signal processing*, J. Wiley & Sons, Chichester, New York, 1993.

[CPL93] CPLEX Optimization, Incline Village, *Using the CPLEX Callable Library and CPLEX Mixed Integer Library*, 1993.

[Dav88] M. Davis, *Applied Decision Support*, Prentice Hall, Englewood Cliffs, 1988.

[EKO90] H. Eschenauer, J. Koski and A. Osyczka, eds., *Multicriteria Design Optimization: Procedures and Optimization*, Springer Verlag, Berlin, Heidelberg, New York, 1990.

[Eme87] J. Emery, *Management Information Systems, The Critical Strategic Resource*, Oxford University Press, New York, 1987.

[FGK93] R. Fourer, D. Gay and B. Kernighan, *AMPL, A Modeling Language for Mathematical Programming*, The Scientific Press, San Francisco, 1993.

[Fis64] P. C. Fishburn, *Decision and Value Theory*, J. Wiley & Sons, Chichester, New York, 1964.

[Fis79] W. Fisher, *Utility models for multiple objective decisions: Do they acurately represent human preferences?*, Decision Sciences **10**, no. 3 (1979) 451–479.

[Fis92] M. Fischer, *Expert systems and artificial neural networks for spatial analysis and modelling: Essential components for knowledge-based geographical information systems*, Discussion Paper WSG-17-92, Vienna University of Economics and Business Administration, Vienna, Austria, 1992.

[FlJ91] R. Flood and M. Jackson, *Critical Systems Thinking*, J. Wiley & Sons, Chichester, New York, 1991.

[Gal67] J. Galbraith, *The New Industrial State*, Houghton-Mifflin, Boston, 1967.

[GdHR+93] O. Güler, D. den Hertog, C. Roos, T. Terlaky and T. Tsuchiya, *Degeneracy in interior point methods for linear programming: a survey*, Annals of Operations Research **46** (1993) 107–138.

[GKPS94] J. Granat, T. Kreglewski, J. Paczynski and A. Stachurski, *IAC-DIDASN++ modular modeling and optimization system: Theoretical foundations*, Technical report, Institute of Automatic Control, Warsaw University of Technology, Warsaw, Poland, 1994.

[GoM95] J. Gondzio and M. Makowski, *Solving a class of LP problems with primal-dual logarithmic barrier method*, European Journal of Operational Research **80**, no. 1 (1995) 184–192.

[Gon94] J. Gondzio, *Presolve analysis of linear programs prior to applying the interior point method*, Technical Report 1994.3, Department of Management Studies, University of Geneva, Geneva, Switzerland, 1994.

[GoT94] J. Gondzio and T. Terlaky, *A computational view of interior point methods for linear programming*, in Advances in Linear and Integer Programming, J. Beasley, ed., Oxford University Press, Oxford, England, 1994. (to appear).

[GoW91] P. Goodwin and G. Wright, *Decision Analysis for Management Judgment*, J. Wiley & Sons, Chichester, New York, 1991.

[GrM95] J. Granat and M. Makowski, *FT, A tool for interactive specification of user preferences in terms of fuzzy sets*, Working Paper WP-95-xx, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1995. (To be published).

[GrW94] J. Granat and A. P. Wierzbicki, *Interactive specification of DSS user preferences in terms of fuzzy sets*, Working Paper WP-94-29, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1994.

[Hop88] G. W. Hopple, *The State of the Art in Decision Support Systems*, QED Information Sciences, Inc., Wellesley, Massachusetts, 1988.

[HoW93] C. Holsapple and A. Whiston, eds., *Recent Developments in Decision Support Systems*, NATO ASI Series F: Computer and Systems Sciences, vol. 101, Springer Verlag, Berlin, New York, 1993.

[HuJ90] I. Huntley and D. James, eds., *Mathematical Modelling, A Source Book of Case Studies*, Oxford University Press, Oxford, New York, Tokyo, 1990.

[IsS87] H. Isermann and R. E. Steuer, *Computational experience concerning payoff tables and minimum criterion values over the efficient set*, European J. Oper. Res. **33** (1987).

[Jan92] R. Janssen, *Multiobjective Decision Support for Environmental Management*, Environment & Management, vol. 2, Kluwer Academic Publishers, Dordrecht, Boston, London, 1992.

[JdJRT93] B. Jansen, J. de Jong, C. Ross and T. Terlaky, *Sensitivity analysis in Linear Programming: Just be careful !*, Technical Report AMER.93.022, Shell Internationale Research, Maatschappij B.V., The Netherlands, 1993.

[KaZ89] B. Karpak and S. Zionts, eds., *Multiple Criteria Decision Making and Risk Analysis Using Microcomputers*, NATO ASI Series F: Computer and Systems Sciences, vol. 56, Springer Verlag, Berlin, New York, 1989.

[Kee92] R. Keeney, *Value Focused Thinking, A Path to Creative Decisionmaking*, Harvard University Press, Harvard, 1992.

[KeR76] R. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, J. Wiley & Sons, New York, 1976.

[KLW91] P. Korhonen, A. Lewandowski and J. Wallenius, eds., *Multiple Criteria Decision Support*, Lecture Notes in Economics and Mathematical Systems, vol. 356, Springer Verlag, Berlin, New York, 1991.

[KMW92] P. Korhonen, H. Moskowitz and J. Wallenius, *Multiple criteria decision support – a review*, European J. Oper. Res. **63** (1992) 361–375.

[KOCZ93] G. Klein, J. Orasanu, R. Calderwood and C. Zsambok, eds., *Decision Making in Action: Models and Methods*, Ablex Publishing Co., Norwood, NJ, USA, 1993.

[KoW90] P. Korhonen and J. Wallenius, *Multiple objective linear programming decision support*, Decision Support System Journal **6** (1990) 243–251.

[LAP94] F. Lootsma, T. Athan and P. Papalambros, *Controlling the search for a compromise solution in multi-objective optimization*, Report 94-12, Department of Mechanical Engineering & Applied Mechanics, The University of Michigan, Ann Aborn, Michigan, USA, 1994.

[LeW89a] A. Lewandowski and A. Wierzbicki, eds., *Aspiration Based Decision Support Systems: Theory, Software and Applications*, Lecture Notes in Economics and Mathematical Systems, vol. 331, Springer Verlag, Berlin, New York, 1989.

[LeW89b] A. Lewandowski and A. Wierzbicki, *Decision support systems using reference point optimization*, in Aspiration Based Decision Support Systems: Theory, Software and Applications, A. Lewandowski and A. Wierzbicki, eds., Lecture Notes in Economics and Mathematical Systems, vol. 331, Springer Verlag, Berlin, New York, 1989.

[LMS94] I. Lustig, R. Marsten and D. Shanno, *Interior point methods for linear programming: Computational state of art*, ORSA Journal on Computing **6**, no. 1 (1994) 1–14.

[Mac85] D. Maclean, *Rationality and equivalent redescriptions*, in Plural Rationality and Interactive Decision Processes, M. Grauer, M. Thompson and A. Wierzbicki, eds., Lecture Notes in Economics and Mathematical Systems, vol. 248, Springer Verlag, Berlin, New York, 1985, pp. 83–94.

[Mak80] M. Makowski, *Modeling the expansion of the water system in the Upper Noteć region*, in Proceedings of Joint Task Force Meeting on Development Planning for the Noteć (Poland) and Silistra (Bulgaria) Regions, A. Albegov and R. Kulikowski, eds., Collaborative Paper, vol. CP-80-09, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1980, pp. 267–295.

[Mak91a] ———, *Guidelines for software development for decision support systems*, Working Paper WP-91-15, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1991.

[Mak91b] ———, *Selected issues of design and implementation of decision support systems*, Working Paper WP-91-16, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1991.

[Mak94a] ———, *LP-DIT, Data Interchange Tool for Linear Programming Problems, (version 1.20)*, Working Paper WP-94-36, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1994.

[Mak94b] ———, *Methodology and a modular tool for multiple criteria analysis of LP models*, Working Paper WP-94-102, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1994.

[MaR91] M. Makowski and T. Rogowski, *Short software descriptions*, Working Paper WP-91-18, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1991.

[MaS87] M. Makowski and J. Sosnowski, *Multicriteria dynamic linear optimization methods applied in* HYBRID *package*, Arch. Automat. Telemech. **32**, no. 4 (1987) 247–276.

[MaS89a] ———, *Mathematical programming package* HYBRID, in Aspiration Based Decision Support. Theory, Software and Applications, A. Lewandowski and A. Wierzbicki, eds., Lecture Notes in Economics and Mathematical Systems, vol. 331, Springer Verlag, Berlin, New York, 1989, pp. 106–144 and 376–377.

[MaS89b] ———, *Solving dynamic multicriteria linear problems with* HYBRID, in Methodology and Software for Interactive Decision Support, A. Lewandowski and I. Stanchev, eds., Lecture Notes in Economics and Mathematical Systems, vol. 337, Springer Verlag, Berlin, New York, 1989, pp. 171–180.

[MaS93a] M. Makowski and M. Savelsbergh, *MP-DIT Mathematical Programming Data Interchange Tool*, Mathematical Programming Society COAL Bulletin no. 22 (1993) 7–18.

[MaS93b] M. Makowski and J. Sosnowski, HYBRID*: Multicriteria linear programming system for computers under DOS and Unix*, in User-Oriented Methodology and Techniques of Decision Analysis and Support, J. Wessels and A. Wierzbicki, eds., Lecture Notes in Economics and Mathematical Systems, vol. 397, Springer Verlag, Berlin, New York, 1993, pp. 223–233.

[Mei76] D. Meister, *Behavioral Foundations of System Development*, A Wiley-Interscience Publication, New York, London, Sydney, 1976.

[MiS92] W. Michalowski and T. Szapiro, *A bi-reference procedure for interactive multiple criteria programming*, Oper. Res. **40**, no. 2 (1992).

[MSW95] M. Makowski, L. Somlyódy and D. Watkins, *Multiple criteria analysis for regional water quality management: A Nitra river basin case study*, Working Paper WP-95-xx, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1995. (To be published).

[Nag90] S. Nagel, ed., *Computer-aided Decision Analysis*, Quorum Books, London, 1990.

[Nak94] H. Nakayama, *Aspiration level approach to interactive multi-objective programming and its applications*, Working Paper WP-94-112, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1994.

[NGI+93] N. Nakicenovic, A. Gruebler, A. Inaba, S. Messner, S. Nilsson, Y. Nishimura, H.-H. Rogner, A. Schaefer, L. Schrattenholzer, M. Strubegger, J. Swisher, D. Victor and D. Wilson, *Long-term strategies for mitigating global warming*, Energy – The International Journal, Special Issue **18**, no. 5 (1993) 409–601.

[OgL92] W. Ogryczak and S. Lahoda, *Aspiration/reservation-based decision support — a step beyond goal programming*, Journal of Multi-Criteria Decision Analysis **1**, no. 2 (1992) 101–117.

[OgZ94] W. Ogryczak and K. Zorychta, *Modular optimizer for mixed integer programming,* MOMIP *version 2.1*, Working Paper WP-94-35, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1994.

[Rad94] F. Radermacher, *Decision support systems: Scope and potential*, Decision Support Systems **12**, no. 4/5 (1994) 257–265.

[Rap89] A. Rapoport, *Decision Theory and Decision Behaviour, Normative and Descriptive Approaches*, Theory and Decision Library, Mathematical and Statical Methods, vol. 15, Kluwer Academic Publishers, Dordrecht, Boston, London, 1989.

[Sak93] M. Sakawa, *Fuzzy Sets and Interactive Multiobjective Optimization*, Plenum Press, New York, London, 1993.

[SaN91] Y. Sawaragi and Y. Nakamori, *An interactive system for modeling and decision support - Shinayakana system approach*, in Advances in Methodology and Applications of Decision Support Systems, M. Makowski and Y. Sawaragi, eds., Collaborative Paper, no. CP-91-17, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1991.

[SeS88] F. Seo and M. Sakawa, *Multiple Criteria Decision Analysis in Regional Planning: Concepts, Methods and Applications*, D. Reidel Publishing Company, Dordrecht, 1988.

[Sil91] M. S. Silver, *Systems that Support Decision Makers, Description and Analysis*, J. Wiley & Sons, Chichester, New York, 1991.

[Sim57] H. A. Simon, *Models of Man*, J. Wiley & Sons, Chichester, New York, 1957.

[Sim58] ———, *Administrative Behavior, a Study of Decision Making Process in Administrative Organizzation*, Macmillan, New York, 1958.

[Sim90] ———, *Prediction and prescription in systems modeling*, Operations Research **38**, no. 1 (1990).

[SNT85] Y. Sawaragi, H. Nakayama and T. Tanino, *Theory of Multiobjective Optimization*, Academic Press, New York, 1985.

[Sos81] J. Sosnowski, *Linear programming via augmented Lagrangian and conjugate gradient methods*, in Methods of Mathematical Programming, S. Walukiewicz and A. Wierzbicki, eds., Polish Scientific Publishers, Warsaw, Poland, 1981.

[SpW93] R. Sprague and H. Watson, eds., *Decision Support Systems: Putting Theory into Practice*, Prentice Hall, Englewood Cliffs, N.J., 1993.

[SSS93] M. Sun, A. Stam and R. Steuer, *Solving interactive multiple objective programming problems using artificial neural networks*, Working Paper 93-364, College of Business Administration, the University of Georgia, Athens, GA, US, 1993.

[Ste86] R. E. Steuer, *Multiple Criteria Optimization: Theory, Computation, and Application*, J. Wiley & Sons, New York, 1986.

[Ste92a] ———, *On the academic exchange of research software: an opportunity for MCDM leadership*, Computers Operational Research **19**, no. 7 (1992).

[Ste92b] T. Stewart, *A critical survey on the status of multiple criteria decision making theory and practice*, OMEGA, International Journal of Management Science **20**, no. 5/6 (1992) 569–586.

[Swi94] A. Swietanowski, *SIMPLEX ver. 2.17: an implementation of the simplex algorithm for large scale linear problems – user's guide*, Working Paper WP-94-37, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1994.

[Thi88] R. J. Thierauf, *User-Oriented Decision Support Systems: Accent on Problem Solving*, Prentice Hall, Englewood Cliffs, N.J., 1988.

[Thi93] ———, *Creative Computer Software for Strategic Thinking and Decision Making – A Guide for Senior Management and MIS Professionals*, Quorum Books, London, 1993.

[Tur93] E. Turban, *Decision Support and Expert Systems: Management Support Systems*, Macmillan Publishing Company, New York, 1993.

[TvK85] A. Tversky and D. Kahneman, *The framing of decisions and philosophy of choice*, in Behavioral Decision Making, G. Wright, ed., Plenum, New York, 1985, pp. 25–42.

[vdA92] W. van der Aalst, *Timed Coloured Petri Nets and Their Application to Logistics*, Eindhoven University of Technology, Eindhoven, The Netherlands, 1992.

[vG91] J. P. van Gigch, *System Design Modeling and Metamodeling*, Plenum Press, New York, London, 1991.

[Vin89] P. Vincke, *Multicriteria Decision-aid*, J. Wiley & Sons, Chichester, New York, 1989.

[Wes91] J. Wessels, *Tools for the interfacing between dynamical problems and models within decision support systems*, Working Paper COSOR 91-29, Department of Mathematics and Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 1991.

[Wes93] ———, *Decision systems: The relation between problem specification and mathematical analysis*, in User-Oriented Methodology and Techniques of Decision Analysis and Support, J. Wessels and A. Wierzbicki, eds., Lecture Notes in Economics and Mathematical Systems, vol. 397, Springer Verlag, Berlin, New York, 1993, pp. 2–20.

[WeW93] J. Wessels and A. Wierzbicki, eds., *User-Oriented Methodology and Techniques of Decision Analysis and Support*, Lecture Notes in Economics and Mathematical Systems, vol. 397, Springer Verlag, Berlin, New York, 1993.

[Wie80] A. Wierzbicki, *The use of reference objectives in multiobjective optimization*, in Multiple Criteria Decision Making, Theory and Applications, G. Fandel and T. Gal, eds., Lecture Notes in Economics and Mathematical Systems, vol. 177, Springer Verlag, Berlin, New York, 1980, pp. 468–486.

[Wie82] ———, *A mathematical basis for satisficing decision making*, Mathematical Modelling **13** (1982) 5–29.

[Wie84] ———, *Interactive decision analysis and interpretative computer inteligence*, in Interactive Decision Analysis, M. Grauer and A. Wierzbicki, eds., Lecture Notes in Economics and Mathematical Systems, vol. 229, Springer Verlag, Berlin, New York, 1984, pp. 2–19.

[Wie86] ———, *On the completeness and constructiveness of parametric characterizations to vector optimization problems*, OR Spectrum **8** (1986) 73–87.

[Wie92a] ———, *Multi-objective modeling and simulation for decision support*, Working Paper WP-92-80, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1992.

[Wie92b] ———, *Multiple criteria games. theory and applications*, Working Paper WP-92-79, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1992.

[Wie92c] ———, *The role of intuition and creativity in decision making*, Working Paper WP-92-78, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1992.

[Wil90] H. P. Williams, *Model Building in Mathematical Programming*, J. Wiley & Sons, Chichester, New York, 1990.

[WiM92] A. Wierzbicki and M. Makowski, *Multi-objective optimization in negotiation support*, Working Paper WP-92-07, International Institute for Applied Systems Analysis, Laxenburg, Austria, 1992.

[YiX91] Y. Yin and X. Xu, *Applying neural net technology for multi-objective land use planning*, Journal of Environmental Management **32** (1991) 349–356.

[Yu85] P. L. Yu, *Multiple-Criteria Decision Making: Concepts, Techniques, and Extensions*, Plenum Press, New York, London, 1985.

[Yu90] ———, *Forming Winning Strategies, An Integrated Theory of Habitual Domains*, Springer Verlag, Berlin, New York, 1990.

[ZAW91] P. Zwietering, E. Aarts and J. Wessels, *Sorting with a neural net*, Working Paper COSOR-91-10, Eindhoven University of Technology, Eindhoven, The Netherlands, 1991.

[ZvKAW91] P. Zwietering, M. van Kraaij, E. Aarts and J. Wessels, *Neural networks and production planning*, Working Paper COSOR-91-15, Eindhoven University of Technology, Eindhoven, The Netherlands, 1991.