Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern

SEKI - REPORT

Optimizing Expert Systems

Peter Spieker

SEKI Report SR-90-18

Optimizing Expert Systems

Peter Spieker
Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
D–6750 Kaiserslautern
FRG

Abstract

Until now, expert systems only have been little successful in planning and construction domains. Especially, when optimal solutions are required they fail. On the other side, algorithmic methods proposed in the field of Operations Research only work on single problems, or if designed for a wider range of problems, are highly inefficient. In this paper we show that some of this problems can be avoided by combining the techniques of expert systems and Operations Research.

The system proposed in this paper generates an optimal solution to a construction problem in two phases. In the first stage the given concrete problem will be mapped to a mathematical problem; in the second stage an algorithm is applied to this problem and the result is transformed back by the expert system and presented to the user.

1 Introduction

The construction process in general has a determining influence on the costs of a future product. For this reason already many different approaches using optimization algorithms were presented to improve the results of this process. In the field of Operations Research several solutions to different domains like scheduling and storekeeping were suggested. Well–known algorithms like "linear programming" and "dynamical optimization" yield optimal solutions to formally specified problems. Nevertheless, several serious restrictions prevent them from successful applications:

- Mostly, reasonable efficient methods insufficiently reflect reality; additional expressive power, e.g. to ensure integer solutions, discards this efficiency. Additionally, the majority of the OR–algorithms are already at least NP–complete in their unmodified version.
- Human experts in construction and planning hardly use algorithms of this kind themselves and therefore refuse to cooperate with systems they cannot understand.

Expert systems represent an alternative way of solving such problems. Like a human expert, expert systems use individual properties of a given problem to reduce the overall complexity of this kind of problems. In contrast to the OR–Systems where the knowledge is coded implicitly in the particular algorithms, expert systems contain an explicit representation of knowledge which is more comprehensive and allows to cover a broader range of problems. However, the system has to extract the procedure of solving a particular problem from this knowledge, which can result in various, perhaps rather inefficient methods. Therefore the performance of expert systems heavily depends on the knowledge about the reduction of the complexity. Expert systems are successful in domains where such kind of knowledge can easily be obtained.

Additionally, expert systems yield explanations of their behavior which correspond to the methods the experts use themselves. This improves the cooperation between the expert and the system.

Frequently, these two approaches are regarded competing and mutual exclusive. First, this point of view cannot be supported by their original intention, since OR–methods are used to cope with problems man

cannot handle very well, while expert systems simulate human behavior, and therefore, in a first step, cannot exceed them. Second, it prevents from a perhaps beneficial amalgamation.

The following sections present a system for construction and conception of mechanical devices. The system is divided in two parts: an expert system and an optimizing algorithm. The expert system part has to extract wishes and demands from the engineer to build up an optimizing problem which can be given to the second part. To perform this task the expert system uses different types of knowledge: general technical knowledge and knowledge about the special problem as well as knowledge about the algorithms used by the second part.

2 Conceptional tasks in technical applications

The process of construction in mechanical engineering can be subdivided in the following four phases:

- First, during **planning**, the requirements on the product which is to be constructed, and its overall function have to be defined.
- Based on this analysis, during **conception**, primitive functions and their realizations have to be determined.
- During **design** further decisions like geometrical order have to be made.
- During the last phase, **detailing**, all informations which are necessary for production are added to the design.

At present, only the last two stages of this process are partly supported by the computer. Since many important decisions relevant for the whole process of construction have to be made in the first stage, this is especially regrettable.

2.1 The problem

Designing a complex technical device means to propose a technical realization for a given set of **requirements**. Additionally, as much as possible certain **wishes** have to be considered. In technical domains requirements mostly are stated as a functional description with given **properties** or **constraints**. These requirements necessarily have to be fulfilled by the final product. Wishes, however, are **goals** of any optimization performed during construction. Therefore the system has to perform two main tasks: first, all requirements and wishes have to be ascertained from the user; second, a satisfactory realization has to be generated.

2.2 The solution

To solve this problem properly the system proceeds in the following manner: Starting at the initial set of functions, the abstract representation of the requirements stated above, a stepwise refinement of the functions is performed recursively. The decomposition stops whenever a technical device can be found which yields the function considered currently. During this process all necessary information, like wishes and properties, are asked from the user.

To proceed in that way the system needs several types of knowledge:

- Knowledge about the decomposition of functions.
  A function may be related to one or more subfunctions, which, if assembled in a proper way, will exactly yield the original function. Additionally, some new constraints may be added to the subfunctions.

- Knowledge about the technical realizations of functions.
  Every primitive function is related to several technical devices which perform this function. These so-called **technical realizations** can be differentiated by the properties they have and that are required by the functions.

  Every function either has a technical realization or can be decomposed into subfunctions.

- Knowledge about incompatibilities of technical realizations.
  Some technical devices are mutual exclusive. To avoid inconsistent solutions the systems has to know about them.

- Knowledge about objects and their properties.
  Besides the relations between the objects the system has to know the objects themselves and their individual properties.

The method sketched in this section basicly reflects the expert system IDA (Intelligent Design Assistant) for which a more detailed description can be found in [Kratz 89].

2.3 The conception of fixture elements

An example for a problem like described above is the construction of fixture elements. During processing a working piece has to be hold in a certain position to allow the necessary operations; simultaneously, resulting forces have to be absorbed. Fixtures in general consist of several fixture elements which fix the working piece at different positions.

The general functionality is presented in figure 1. It can be subdivided into six, partially optional subfunctions. Goals of optimization are size, weight and costs; properties which have to be met by the chosen technical realizations are precision, forces which have to be absorbed, etc. Every function is related to several technical realizations as it is shown in figure 2.

3 The expert system approach

As a rule, problems with synthetical characteristics are formalized as search problems defined on a state space of (partial) solutions. The main task is either to find any complete solution or the optimal solution[1].

Therefore systems designed to solve such problems can be characterized by the following points:

- The knowledge is mainly attached to so-called objects. This can be regarded as true for structural knowledge which describes the "world" as well as for procedural knowledge which describes the methods to proceed within this world.

- Dependencies between objects can be represented by so-called constraints; constraints constitute valid combinations of parameter values or subcomponents.

- To allow retraction of decisions which have been shown to be wrong, either in the sense of consistency or optimality, a "truth maintenance system" has to maintain dependencies between values and objects. Presently, most systems employ some restricted mechanism, which only allows chronological backtracking, rather than dependency directed backtracking [DeKleer 86].

In general it can be said that until now the representation of the domain knowledge, i.e. the structural knowledge, has been regarded as essential. The procedural knowledge is mainly hidden in the inference engine, which mostly is a general purpose engine. Therefore serious problems concerning efficiency occur.

[1] In this context one usually has to discuss the notion of optimality. We will not go into this discussion here for two reasons. First, in technical domain many of the problems like evaluating of properties do not arise; second, remaining problems like contrary optimization goals have not really been tackled by expert systems so far.

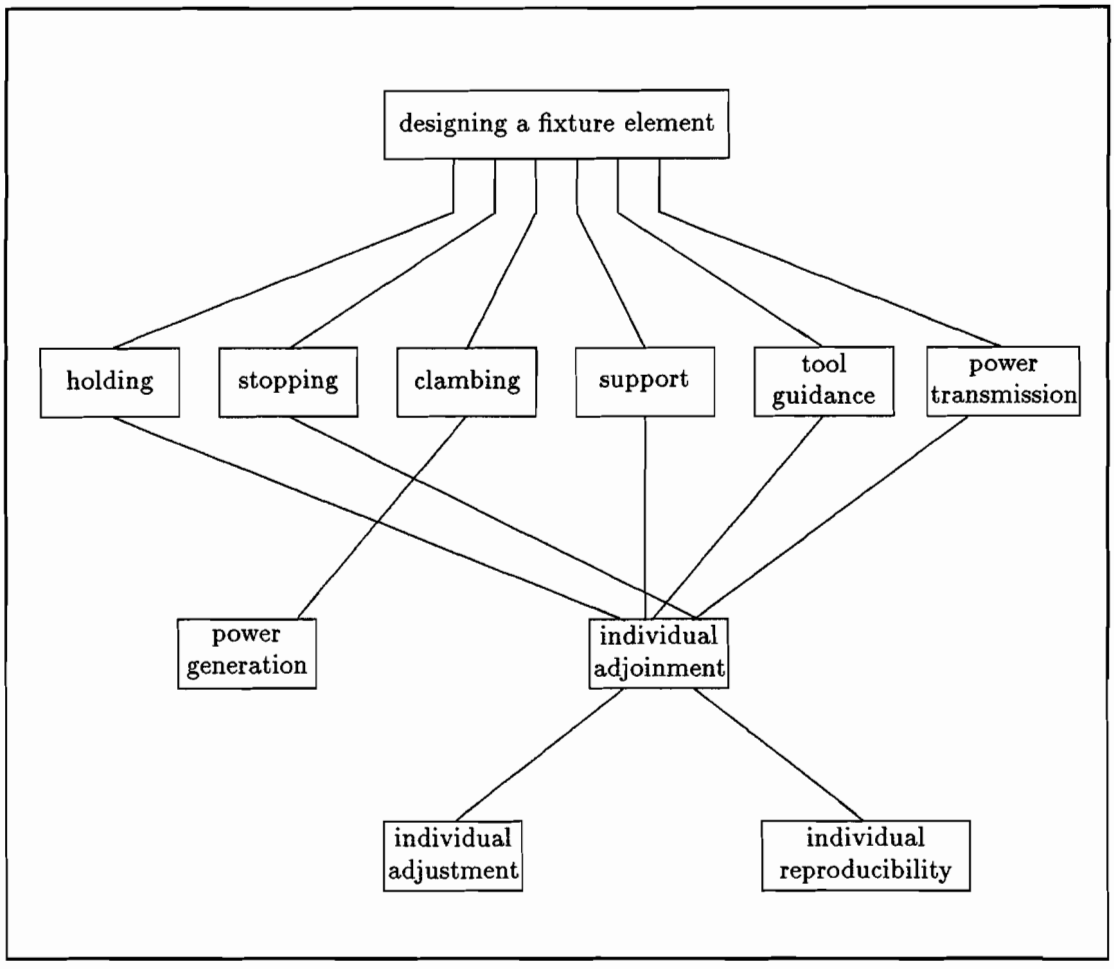

Figure 1: Functional decomposition of a fixture element

3.1 Construction viewed as constraint satisfaction

Most of todays systems are concerned with searching for "any" complete and permissible solution [Owsnicki–Klewe 88]. Especially in the early stages of the expert system research systems which employ deterministic search were realized; XCON (R1) [McDermott 80] was one of the first of this kind.

Unfortunately, in reality most problems require an indeterministic search method, because interactions between different parts of the solution cannot be predicted whenever it is necessary. Therefore all alternatives concerning every past decision have to be maintained to allow backtracking whenever a failure occurs. To identify the real cause of such a failure, the system also has to maintain all dependencies and preconditions of any object or value.

3.2 Construction viewed as an optimization problem

To solve problems of the second type, i.e. problems which require some kind of optimal solution, systems we described above are "augmented" twice:

- If a decision has to be made, the system chooses the more promising alternative. Since almost every system employs a decompositional method of generating the solution, this only yields an optimal solution if it is the sum of the optimal solutions of the partial problems.

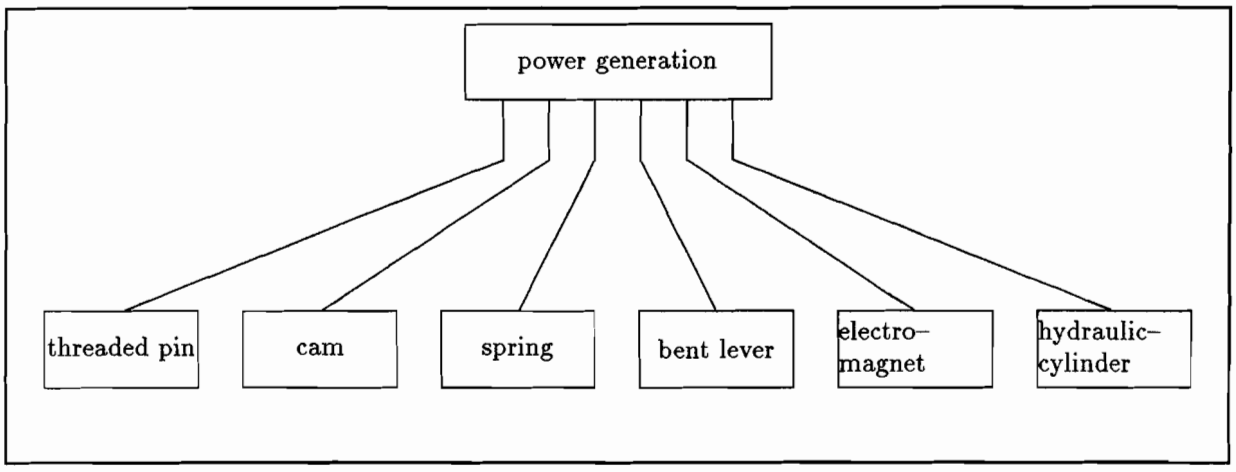

Figure 2: Technical Realizations of "power transmission"

- Backtracking also becomes activated whenever a (partial) solutions is reached which seems to be worse than another more promising one. This is some kind of top level description of the well–known A^*–Algorithm [Nilsson 80].

Systems which behave in that way are only of little success. The assumptions made by them are only seldomly justified by real problems. Especially the second one mostly fails, because it requires, that there are no interactions between any partial problems at any level. If there are any interactions which is the rule in technical construction problems, predictions made by the system tend to be wrong. As a consequence, either suboptimal solutions are provided or an exhaustive search has to be performed.

4 Optimizing systems in construction

Looking at the facts stated so far, successful expert system applications in this domain seem rather unlikely. But these facts only form the surface of the underlying problem; the problem of the lack of real expert knowledge. Therefore expert systems which exclusively base on expert knowledge obviously have to be shaky. As we stated above, the main deficiency is the lack of procedural knowledge, which at first, results in an oversimplified behavior of the experts, and finally led to poor expert systems.

The lack of expert knowledge was one reason for introducing the methods of Operations Research in the past. Based on mathematical models of real world problems algorithms were developed which, theoretically, yield an optimal solution to these problems. The reasons for the little success of this approach until now are the following:

- In general, most potential users of OR–algorithms lack the sound mathematical knowledge which would be necessary to understand them as a whole. This leads to several consequences:
  - OR–methods are used rarely;
  - the potential of OR–methods is left unused.

  Finally, results cannot been explained in any natural way like humans explain to each other.
- Variations of any problems need new or modified models which results in clumsy tools, even for a restricted range of applications.
- OR–methods often show unexpected behavior w.r.t. efficiency when confronted with slightly modified problems, e.g. linear programming vs. integer linear programming.

The aim of any use of OR–methods within expert systems is to free the user from deciding whether and how to use them for solving a particular (sub–) problem. This means that the expert system has to be a technical expert and an OR expert.

4.1 The process of conception

Analyzing the process of conception yields the following subprocesses:

- Elucidation of the relevant data;
- Generation of the functional structure;
- Mapping this structure onto a set of technical realizations;
- Search for an optimal solution w.r.t. to possible constraints.

The first three stages represent the deterministic part of the search process described above. Conflicts based on multiple possible subfunctions for decomposing a particular function can be resolved by means of the properties which are already known. Therefore this process is chiefly determined by structural knowledge about the domain. Since this type of knowledge is of declarative nature, expert system techniques are well suited to cope with this part of conceptional tasks. As a result a set of technical realizations is delivered without being concerned with any technical constraints between them. On the other hand, all constraints which stem from the properties of the functions are taken into account.

Finally, as a last step, we have to find an optimal solution regarding to the wishes and the constraints neglected so far.

4.2 The optimization problem

In this section we describe how the problem of searching for an optimal realization can be viewed as an linear programming problem and we provide a mapping from the structures of the technical domain onto the terms of this formalism:

- Maximize the benefit of the construction
  The particular benefit of a construction results from summing up the benefit of any single technical realization used in the construction. The benefit value of a technical realization can be computed from the degree of it's appropriateness w.r.t. a single property represented by numbers from 0 to 10 multiplied with the significance represented by numbers, too. This products are summed up for every individual property to form the benefit of the technical realizations w.r.t. to certain wishes of the user. (The benefit values of each technical realization are represented by the c_i below.)
- Satisfy every subfunction
  Every solution has to supply exactly one technical realization for every primitive function. (This demand is represented by the array A'.)
- Avoid incompatibilities
  Every solution has to obey each mutual restriction between every technical realization added to the final construction. (These restrictions are represented by the array A''.)

In the subsequent formulation of our problem the following assumptions hold:

1. Each construction consists of n subfunctions.
2. Each subfunction i ($i = 1 \ldots n$) can be realized by a set $real_i$ of technical realizations. Each set $real_i$ consists of num_i elements. Therefore we have to choose from $m = \sum_{i=1}^{n} num_i$ technical realizations. An index j is attached to each technical realization; identical technical realizations in different sets $real_i$ are named differently.

3. Each technical realization j is related to a set $confl_j$ $(j = 1 \ldots m)$ which contains the index of every technical realization which is incompatible with j. This results in $k = \sum_{j=1}^{m} confl_j$ pairs of incompatibilities. ($incomp_l$ $(l = 1 \ldots k)$.)

The problem we have to solve is the following integer linear programming problem:

maximize: $c^t x$,

where
$c \in \Re^m, x \in B^m, B = \{0, 1\}$

subject to:

$A'x = b'$,
$A''x \leq b''$,

where
$b' \in \Re^n, A' \in \Re^{n \times m}, b'' \in \Re^n, A'' \in \Re^{k \times m}$,
$b'_i = b''_j = 1 \; \forall i \in \{1 \ldots n\}, \forall j \in \{1 \ldots k\}$.

$$a'_{i,j} = \begin{cases} 1 & \text{if } i \leq \text{n and technical realization } j \in real_i \\ 0 & \text{otherwise} \end{cases}$$

$$a''_{i,j} = \begin{cases} 1 & \text{if } i = l \text{ and technical realization } j \in incomp_l \\ 0 & \text{otherwise} \end{cases}$$

4.3 Implementation

To build a system which employs the method suggested so far we modified the expert system IDA in the following manner:

The first stages described in section 4.1 were left unchanged, so IDA builds up the functional structure of the fixture element the user wants. Rather than selecting a particular technical realization for each subfunction IDA creates a LP–problem shown in section 4.2 and hands it over to the optimization part.

Since the meaning of the variable values is selecting the corresponding technical realization (1) or not (0) only integer (binary) values are allowed. As practical experiments using the simplex algorithm have shown us, only a few variables have non–integer values; exceptionally, all variables have integer values. So we could employ the "branch & bound" method to ensure proper solutions.

In our domain, the construction of fixture elements, we have to cope with up to 22 individual subfunctions, each with an average number of technical realizations of eight. Hence, the number of variables is between 50 and 200. Our algorithm, a version of the revised simplex method [Best, Ritter 85], only requires a few seconds to deliver the results we need.

The results of this step are presented graphically to the user and enables him to modify it and return it for further optimization.

5 Summary

In this paper we argued that in constructive applications, expert systems in a "pure" style only perform poorly. Alternatively, we presented an approach in which we integrated different techniques to overcome the difficulties of lacking expert knowledge. The example we selected in this paper is the conception of fixture elements in mechanical engineering, a less complex application than configuration which is another widely investigated application. Therefore the question arises whether this approach is well suited for more complex tasks requiring a large amount of (binary) variables. In our future work we will concentrate on configuration tasks and more sophisticated methods to solve large integer linear programming problems like genetic algorithms or simulated annealing.

References

[Best, Ritter 85] Michael J. Best; Klaus Ritter:
Linear Programming
Prentice–Hall; 1985

[DeKleer 86] Johan de Kleer:
An Assumption–Based TMS
Artificial Intelligence 28; 1986

[Kratz 89] Norbert Kratz:
An Approach to Knowledge–Based Design
FAW Ulm; 1989

[McDermott 80] John McDermott:
R1: An Expert System in the Computer System Domain Proc. of AAAI; 1980

[Nilsson 80] Nils J. Nilsson:
Principles of Artificial Intelligence
Springer; 1980

[Owsnicki–Klewe 88] Bernd Owsnicki–Klewe:
Configuration as a Consistency Maintenance Task
Hamburg, Philips GmbH Forschungslaboratorium; 1988