1994

Combining artificial intelligence and optimization in engineering design

Jonathan Cagan
Carnegie Mellon University

Ignacio E. Grossmann

John N. Hooker

Carnegie Mellon University.Engineering Design Research Center.

Combining Artificial Intelligence and. Optimization in Engineering Design

Jonathan Cagan, Ignacio £* Grossmann and John Hooker

EDRC 06-182-94

Combining Artificial Intelligence and Optimization in Engineering Design

Jonathan Cagan, Ignacio E- Grossmann and John Hooker
Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

Working paper March 1994

Introduction

Engineering design problems involve quantitative as well as qualitative aspects. This is particularly true in the case of synthesis problems in which the selection of a topology as well as its parameter values commonly requires qualitative knowledge cm design alternatives as well as quantitative models to predict the pcifmmancc of a system. Artificial intelligence and optimization provide basic frameworks for the representation and solution of problems in engineering design. However, given the qualitative and quantitative nature of these problems it is often not clear how to best formulate and solve these problems. Furthermore, the problem is compounded by the fact that very different solution methods can be applied to the same design problem.

Over the last few years Combined Al/Optimization has been CHIC of the research thrusts in the Synthesis Laboratory of the Engineering Design Research Center at Carnegie Mellon. This thrust has been complemented by an interdisciplinary graduate design course in which AI and optimization approaches to design were taught (see Fenves and Grossmann, 1991). These efforts were motivated by the fact that earlier work in the center involved the development of synthesis methods that either relied only on AI techniques (commonly expert systems) or only on optimization techniques (commonly NLP and MINLP models). It became evident that there were several shortcomings in approaching synthesis problems with only one methodology. The major problems for the Al-based methods were difficulties in integrating qualitative knowledge with analysis models that are used in engineering design, and accounting for interactions of design decisions. The major problems for the optimization-based methods were difficulties in making use of engineering knowledge that is not expressible in the form of equations, and limitations in solving large-scale problems.

Rather than presenting an overview of the research projects of the Combined Al/Optimization thrust at the EDRC, it is the objective of this paper to present some of the insights and concepts that we have learned from the interplay between qualitative and

quantitative issues in engineering design problems. Our specific objective in this work is to propose generalized representations of engineering design models, and to propose taxonomies and classifications. This is performed first in the context of design models to identify major classes of problems and solution methods that are used.. We then present a general classification of AI and OR models in terms of model attributes and establish mappings with generic solution techniques. We discuss the requirements of the solution methods and several schemes for the integration of AI and optimization to identify future research directions. Finally, we illustrate with several design problems various ways in which AI and optimization can be combined for tackling engineering design problems.

Classification of Models in Engineering Design

The development of computational models in engineering design is generally a complex activity. It normally involves an iterative procedure that is composed of the following steps:

1. Definition of search space for the design alternatives. This may be based on past experience, on qualitative knowledge or on a well-defined search space.
2. Problem formulation which involves the development of the computational model.
3. Solution of the problem through an appropriate technique.
4. Verification and critique to establish whether the solution indeed satisfies the design objectives.

Given that we have defined the search space and its corresponding representation (e.g. graph, network) in step 1, a crucial step in the above procedure is the development of a computational model. In this section we will present a classification of computational models for design that will help us to identify the solution techniques that may be applied and that will be discussed later in the paper.

First, it is useful to characterize a computational model in terms of four elements:

a) Variables or unknowns
b) Parameters or inputs
c) Relations or equations and inequalities

d) Goals or objectives

By the type of variables* a computational model may be classified as (a) Continuous, (b) Discrete, or (c) Mixed Discrete/Continuous. The continuous model will involve quantitative variables such as areas, forces, stresses, flows ami voltages. The discrete model will involve symbolic variables such as selection of a given item (a 0-1 decision) or quantitative variables that are restricted to finite values (e.g. standard sizes for equipment). The mixed model will contain both types of variables.

By die type of parameters, a computational model may be:

(a) *Deterministic* if the inputs are exactly known.
(b) *Stochastic* if the inputs axe subject to fluctuations.

By the nature of the relations or equations a computational model may be classified in three different ways:

1. (a) *Quantitative* if the equations are mathematical in nature (e.g. algebraic, differential). These in turn can be linear or nonlinear,
   (b) *Symbolic* if the relations are expressed in terms of prepositional logic (e.g. IF - THEN statements) or predicate logic.

2. (a) *Static* if there is no time dependence.
   (b) *Dynamic* if the equations are time dependent

3. (a) *Spatial* if there is a dependence with physical dimensions,
   (b) *Non-spatial*

Many design problems may involve a mixture of classes of equations (e.g. quantitative and symbolic). Most computational models in design, however, will tend to be quantitative in nature since equations are used to model physical laws and specifications that a process or artifact must obey.

By the type of goals a design model may be:

(a) *Satisficingy* if the objective is to find any values of the variables that meet the relations or equations.

(b) *Optimizing,* if the objective is to find not only any values of the variables that meet the relations or equations, but those that maximize or minimize one or several objective functions. The former case corresponds to a single criterion optimization, while the latter corresponds to a multiobjective optimization problem.

Conceptually, we can represent computational models for design as follows. Let the *continuous variables* be x and the *discrete variables bey.* We could refine our representation by partitioning the variables x into static v and dynamic variables dv/dt. For the sake of simplicity however, we will not introduce this distinction. The parameters which are normally specified are represented by d.

Equations and inequality constraints can be represented as die vectors of functions h and g, that must satisfy,

$$h(x,y,d) = 0 \qquad (1)$$
$$g(x,y,\hat{v}) \leq 0$$

It should be noted that the representation in (1) could range from a relatively simple model to a very complex model that tries to capture as much detail as possible the underlying physics of the system. The solution to the equations in (1) are often not uniquely defined since they commonly involve several degrees of freedom.

The logical relations that define symbolic relations will be given by a set of propositions that must hold true; that is,

$$L(x,y,\hat{v}) = \text{TRUE} \qquad (2)$$

Finally, the design goal (or goals) can be expressed as the objective function F(x,y,d). This function is a scalar for a single-criterion optimization, and a vector of functions for a multiobjective optimization.

With the above definitions for a design model, the general computational problem for a design can be formulated as follows:

GivetMi possibly with a description of its fluctuations (e.g. distribution functions), find values for x and y in order to satisfy:

$$\begin{aligned} &h(x,y,d)=0 \\ &g(x,y,\vartheta) \leq 0 \\ &L(x,y,\vartheta) = \text{TRUE} \end{aligned} \qquad (3)$$

while possibly optimizing the goal or goals as given by die objective function F(x,y,d).

Classes of Design Problems

The problem formulation given in (3) can be used as **a** basis to represent major classes of design problems. For this let us partition the continuous variables x into the z state variables and the design parameters u which **are** to be chosen. The space of design alternatives can then be represented by the sets Y and U, the former representing the space of topologies and the latter the space of design parameters.

If the sets Y and U, and the equations h, g, L, and variables y,u,x, are given explicitly, together with the design goals in F, this gives rise to a *declarative* model. In this case it is possible to make a clear separation between the model and the solution technique. In fact modeling systems such as GAMS, AMPL, ASCEND, and expert systems such as VPExpert and EXSYS are based on this principle. In the context of engineering design, explicit models are commonly simplified representations.

In contrast, if very detailed models are used (e.g. based on finite elements), this will give rise to implicit or *procedural* models. Typically this mode of computation is based on specifying the variables y and u for a fixed d, with which the states z are computed as an implicit function; that is,

$$h(u,z,y,d) = 0 \quad \text{-»} \quad z = z(u,y,d) \qquad (4)$$

This has the effect that the inequalities, logic relations and goals, all become implicit functions as well; that is,

$$\begin{aligned} &g(u, z(u, y, \vartheta), y,,\vartheta) \leq 0 \\ &L(u, z(u, y, fl), y,,d) = \text{TRUE} \\ &F(u, z(u, y, d), y,,d). \end{aligned} \qquad (5)$$

Finally, the sets Y and U may be implicitly defined. This is commonly the case in discrete design problems in which the set Y = {yj_f i=l,2,...n)is generated within a tree search with the recursion of the form,

$$yi+i - f(yi .yi-i \bullet> yi). \quad us \; U(yi+i) \tag{6}$$

Based on the above, some of the classes of design problems that we can identify are the following:

1, Heuristic design

Given explicit Yand U, and fixealf find y,u,z, such that as large a subset of

$$L(u,z,y,d)*TRUE \tag{7}$$

is satisfied One way to approach this problem is through rule-based systems.

2. Enumeration of feasible designs.

Given Y and U in implicit form as in (6), and for fixed d, find all y, with its corresponding u and z such that

$$\begin{aligned} &h(u,z,y,0) = 0 \\ &g(u,z,y,fl) \leq 0 \\ &L(u,z,y,d)=TRUE \end{aligned} \tag{8}$$

is satisfied. One way to approach this problem is through a tree search based on hierarchical decomposition.

3. Superstructure optimization.

Given Y and U in explicit form as part of a superstructure of alternatives, and fixed9, find y,u,z, such that,

$$\begin{aligned} \min \quad & F(x,y,O). \\ s.t \quad & h(x,y,\$)=0 \\ & g(x,y,d) \leq 0 \end{aligned} \tag{9}$$

One way to approach this problem is through mixed-integer nonlinear programming.

4. Innovative Design

Given fixed&, find Y and U such that for instance the conditions in any of the three above problems be satisfied.

White die classification given above is rather general, it docs serve as a unified framework for the representation of design problems. In addition it also serves to identify difficulties in design problems. For instance, handling of uncertain parametars d and handling of multiobjectives are not easily expressible in closed form. Also, die generic problem in (3) as well as some of its specific variations as in (8) gives rise to the intriguing possibility of posing and solving problems that contain both quantitative as well as qualitative information. Finally, a major question is how to actually solve the design problems once these are formulated as a particular case of (3) as was shown above.

Overview of solution techniques

In this section we will briefly overview some of the major solution techniques that can be used to solve particular cases of problem (3). The intent is not to provide a comprehensive review, but rather give a brief sketch of algorithmic tools that are available to designers.

Two major problems that arise in design are analysis or evaluation and synthesis or optimization. The solution techniques for analysis can be generally classified as:

a) Equation solving
b) Symbolic analysis

For synthesis and optimization they can be generally classified as:

a) Mathematical programming
b) Heuristic search techniques

Equation solving techniques are widely used and implemented in most of the design tools. Here, the particular model:

Given d and y, find x to satisfy

$$h(x,y,d) = O \qquad (10)$$

gives rise to a system of algebraic equations for the static case. Linear models can be effectively solved with matrix factorization methods (e.g. L/U decomposition) which can exploit the inherent sparsity in large scale design problems (Carnahan and Wilkes, 1980; Dahlquist and Anderson, 1974; Pissanetzky, 1984). Nonlinear models are considerably more difficult to solve, and in general require the iterative solution of linearized equations. This is for instance the case in Newton's method and it variants known as. Quasi-Newton methods. The former requires analytical derivatives for the jacobian matrix of (10), while the latter will predict jacobian approximations based on function values (Dahlquist and Anderson, 1974). First order methods, such as successive substitution, can also be used. They have the advantage of not requiring derivative information, but are slower to converge (Carnahan and Wilkes, 1980). Methods far solving algebraic equations are available in many computer codes (e.g. IMSL, 1987; Piela et al, 1991; Rice, 1983). It should also be noted that for the case of large and complex nonlinear equations, which for instance arise in simulators (e.g. ASPEN in chemical engineering (Aspen-Technology, 1991), SPICE in electrical engineering (Banzhaf, 1989)), most of the equations are treated as a "black box" routine which is converged externally as an implicit function.

For the dynamic case, the problem in (10) gives rise to differential/algebraic systems of equations. In the simplest case (only non-stiff ODE's) explicit methods such as Euler and linear multistep methods can be used (Carnahan and Wilkes, 1980; Dahlquist and Anderson, 1974). For stiff ODFs implicit methods such as backward differences and collocation methods are required. These can be extended to the case when algebraic equations are included, but care must be exercised to handle the so called "index problem" which may introduce large errors even if small integration steps are used with implicit methods. Codes implementing methods for integrating differential equations include IMSL (1987) and DASSL (Petzold, 1982).

When spatial equations are also included in a model this will often give rise to partial differential equations which are commonly solved by finite element methods (Becker et al, 1982). These have become a major tool for design analysis and are implemented in codes such as ANSYS (1992).

Symbolic analysis tools are used mostly on design problems that are expressed in qualitative terms. Here the particular model has the form:

Given d, find x and y to satisfy

$$L(x,y,\hat{0}) = \text{TRUE} \qquad (11)$$

Most of the computational techniques for logic are restricted to the case when the variables x are absent or prespecified Symbolic methods for solving these problems include theorem proving and resolution techniques such as the ones implemented in PROLOG (Dodd, 1990). For simpler cases (e.g. Horn clauses), forward and backward chaining methods can be used (Barr and Fcigenbaum, 1981). The latter are implemented in a number of expert systems shells (e.g. VP-Expert (1989), EXSYS (1990)). Quantitative approaches to solving problem (11) have also been developed (e.g. see Hooker, 1988). They rely on the idea that it is possible to systematically transform (11) into a linear programming problem with which the inference problem is solved as an optimization problem.

As for synthesis and optimization problems mathematical programming models have the general form:

Given d, find x and y to

$$\text{minimize } F(x,y,d)$$

subject to

$$h(x,y,\hat{0}) = 0 \qquad (12)$$
$$g(x,y,\hat{0}) \leq 0$$

Here F(x,y,d) is assumed to be a scalar function. The case when F(x,y,d) is a vector of functions gives rise to multiobjective optimization problems, which as opposed to the scalar case, have in general an infinite number of solutions. These are given by trade-off or pareto-optimal curves in which the various objectives cannot be improved simultaneously.

Computationally multiobjective optimization problems are always reduced in one form or another into a scalar optimization problem. This is done by either taking a weighted sum of the function, or by placing all functions except one as a constraint whose value is varied parametrically (Nemhauser et al, 1989).

The mathematical programming problem in (12) gives rise to the following major cases in which F, h and g are algebraic functions:

(a) Linear program (LP): F,h,g, are linear and discrete variables y are not present

(b) Mixed-integer linear program (MILP): F,h,g are linear and both variables x and y are present

(c) Nonlinear program (NLP): F,h,g are generally nonlinear and only continuous variables x are present

(d) Mixed-integer nonlinear program (MINLP). F&gare generally nonlinear and both variables x and y are present

There are of course some more specific cases than the ones cited above. For instance a problem with only discrete variables y corresponds to an integer programming problem. An NLP with quadratic objective function and linear constraints corresponds to a quadratic program. Also, for the case when some of the equations are ODFs or PDE's, these problems are either reformulated as NLP problems by using algebraic approximations to the differential equations (e.g. using collocation techniques), or else they are solved in a "black box" and treated as implicit functions (Biegler, 1990). The latter option, by the way, is also used for the case mentioned previously for simulators in which the algebraic equations are large and complex.

For LP problems the most common method is the simplex algorithm (Hillier and Lieberman, 1986) which is implemented in many computer codes (SQCONIC (1986), OSL (IBM, 1991), LINDO (Schrage, 1986), ZOOM (Marsten, 1986)). Interior point methods have been recently developed which have shown to outperform simplex in problems involving many thousand of constraints (Marsten et al, 1990). MILP techniques rely commonly on branch and bound methods that use the simplex algorithm for LP as subproblems in each node of the tree (Nemhauser et al, 1989). Therefore, these are commonly available in the same simplex codes. It should also be noted that the structure in special cases of LP and MILP problems can be greatly exploited, such as is the case in network flow problems. Here specialized algorithms can efficiently solve very large scale versions of these problems as opposed to the general purpose LP and MILP methods.

The most common techniques for NLP are the reduced gradient method (Reklaitis et al, 1983), which is available in the computer codes MINOS (Murtagh and Saunders,

1985), GINO (Licbman ct al, 1986), CONOPT (Drud, 1991), and the successive quadratic programming (SQP) method (Nemhauser et al, 1989), which is available in the cxxksNPSOL(Gmctal, 1983) and OPT (Vasantharajanetal, 1990) The former method tends to be better suited for models with explicit equations and the latter for "Made box" models. Finally, MINLP techniques which are only recently starting to be applied in design problems, include the Generalized Benders decomposition (Geoffrion, 1972) and the Outer-Approximation method (Grossmann, 1990) which is implemented in DICOPT++ (Viswanathan and Grossmann, 1990). All these nonlinear optimization methods can only find a local optimum solution, unless the problem is convex in which case a local optimum is also a global optimum (Bazaiaa and Shetty, 1979).

It should be noted that modelling tools such as GAMS (Brooke et al, 1988) and ASCEND (Piela et al, 1991) interface automatically with several of the computer codes cited above, greatly facilitating die formulation and solution of optimization problems.

Heuristic search techniques are methods, which do not necessarily have a rigorous mathematical basis, but have the advantage of being simple to implement Furthermore, for most cases they do not make any special assumption on the form of the equations or functions which can make them useful for complex or poorly understood design problems.

Most heuristic search techniques are aimed at solving discrete optimization or symbolic problems. For instance the A* method (Nilsson, 1980) is in essence a branch and bound enumeration method such as the one discussed in section 2. The major difference is that only a partial set of nodes is commonly examined since heuristics rather than rigorous bounds are used to prune nodes in the tree. Although this will in general not guarantee optimality, the advantage is that only few nodes may be examined in a search tree.

Other heuristic search techniques include simulated annealing (Aarts and van Laarhoven, 1985) which is a statistical method motivated by the physical equilibrium that is attained in the cooling of substances. This technique has had success in solving several combinatorial optimization problems, although the method can be computationally expensive due to its probabilistic nature.

Heuristic search techniques for symbolic logic problems rely on the use of simplified enumerations to perform inferences. An example are forward and backward chaining schemes that are used in expert systems (Bair and Feigenbaum, 1981).

Finally, all the above cited methods are aimed at deterministic problems in which the parameters are exactly known. When these are subjected to fluctuations a number of techniques have been developed which build on the deterministic methods. Common techniques include Monte Carlo simulation (Hillier and Lieberman, 1986) and multidimensional integration methods to estimate expected values when die parameters are described by distribution functions. Deterministic methods are also available for analyzing parameter variations and these commonly give rise to minimax optimization problems. It should be noted that one of the difficulties when dealing with parameter uncertainties is the choice of an appropriate design objective. For instance, we may minimize the expected cost, or alternatively, minimize the largest cost that occurs at the "worst" parameter point The former requires the evaluation of a multidimensional integral, while the latter require the solution of a minimax optimization problem.

From the brief overview that we have presented in this section, it is clear that there is a large number of computational techniques that are available for analysis and synthesis problems in design. As was noted before the choice of a given technique is highly dependent of the nature of the computational model which in turn depends on the representation that is used. The importance of problem formulation cannot be overemphasized. It is not enough to have powerful solution tools if they are not properly used. The next section provides a classification of solution techniques.

Classification of AI and OR Models

Having presented a classification of models and solution methods from an engineering design perspective, we now address the more general question as to what makes a model an OR model or an AI model? One way to answer this question is to classify models along three dimensions, each of which has a pole associated with OR and one associated with AI. This analysis clarifies the various senses in which OR and AI can be combined. It also helps one to say more precisely what sort of research thrusts could profitably combine AI and OR styles of modeling. The latter will be addressed in the next section.

The three axes along which models can be classified are:

1. numeric/symbolic
2. specific/general
3. structured/unstructured

In each case, the first attribute is normally associated with OR models, and the second with AI models. We begin with a brief description of these polarities*

L Quantitative vs. symbolic models*

In the schema presented earlier, quantitative models **have** only constraints of the form $h(x,y,d) = 0$, $g(x,y,0) \leq 0$, and symbolic models have only constraints of the form $L(x,y,0)$ =TRUE. A symbolic model might also be called **a** *logic model,* particularly if the propositions in L£t,;y,0J belong to a formal logical language.

To avoid classifying all models as symbolic models, we can assume that the propositions in $L(x,y,0)$ do not assert quantitative relations $h(x,y,9) = 0$, $g(x,y,0) \leq 0$. Nonetheless it is useful to acknowledge that, in a significant sense, all models are logic models. They consist of a problem description in a formal language (or a language that can in principle be formalized), from which one can deduce facts about the solution. To solve a model is to cany out the deduction.

Quantitative models are those in which numerical predicates play a major role. The deduction techniques are specialized to be efficient for arithmetical predicates. They often take the form of numerical computation or algebraic manipulation.

The realization that numerical computation is a form of logical deduction, credited to Leibniz and exploited by Boole, was a breakthrough. It allowed people to exploit the power of automatic computation without presupposing mathematical structure. This is still the primary rationale for logic programming. It also suggested that computers can think, at least to the extent that thinking is logical deduction. It was primarily this suggestion that inspired the field of artificial intelligence in the 50's and 60^{f}s, and the classical AI approach to problem solving has been to use symbolic computation. AI has more recently moved toward numeric models of neural networks, but it remains much more strongly identified with symbolic reasoning than OR.

2. Structured vs. unstructured models.

Although it is hard to define what is meant by a structured model, it is a concept widely employed by modelers. Consider, for instance, an assignment model, which is a very special case of a linear programming problem (Nemhauser and Wolsey 1988). All instances of an assignment model are very similar. They all exhibit the same, fairly simple pattern. This makes an assignment model highly structured. A linear

programming model need not in general be as structured If one examines a wide range of linear programming models, there is not much similarity aside from their linearity. (Some linear programming models, of course, exhibit a high degree of structure, not only assignment models but network flow models, models with staircase or block diagonal structure, etc.) Nonlinear and integer programming models can be even less structured.

Structured models tend to have two advantages. One is that their relative simplicity is more conducive to understanding. If one can fit a structured model to a situation, he is likely to improve his grasp of it One reason for the popularity of network flow models is that they are easy to understand and seem to illuminate the subject matter:

Another advantage of structured models is that they *tend* to be easier to solve. Network flow models, for instance, are much easier to solve than general linear programming problems (Bazaara *etal.* 1990).

In feet, it is tempting to define a structured problem as one that has ^exploitable" structure—structure that is conducive to fast solution—since otherwise it is unclear what kind of patterns count as structure. But an occasional structured problem, such as the famous traveling salesman problem, is very hard to solve despite its highly structured nature. In fact this very anomaly seems to intrigue mathematicians, as witnessed by the attention given to the traveling salesman problem. So we must continue to define structure as a Supreme Court Justice once defined pornography: we know it when we see it

3. Specific vs. general models.

A specific model is one that applies only to a narrow range of problems, whereas a general model fits a wide variety of problems. A specific model "presupposes structure in the problem." An assignment model is very specific; problems are rarely so neat. Logic programming, on the other hand, presupposes little structure in the problem. It applies to any problem that is expressible in its logical formalism. This results in considerable generality, particularly if one considers's W. V. Quine's assertion that first order predicate logic is adequate to formulate all of science (Quine 1961).

A highly structured *model* need not presuppose structure in the *problem;* i.e., it need not be specific. A neural network model, for instance, is structured because because it is a particular type of nonlinear regression model in which the gradient of the error function can be computed recursively via back propagation (Rumelhart *et al.* 1986). A math programmer would regard this as a highly structured nonlinear programming model.

But it is very general because it can formulate problems ranging from traveling salesman problems to visual recognition problems.

Figure 1 attempts to classify some models along the specific/general and numeric/symbolic axes. One can imagine a structured/unstructured vertical axis, with the structured pole at the upper end OR models should occur on the left side of the diagram, with preference for the upper left; they should also have a high elevation on the vertical axis. AI models should gravitate toward the right, with a preference for the lower right (although neural networks, genetic algorithms, ctc_M are moving the center of gravity toward die upper right). They should occur mainly at low elevations on the vertical axis.

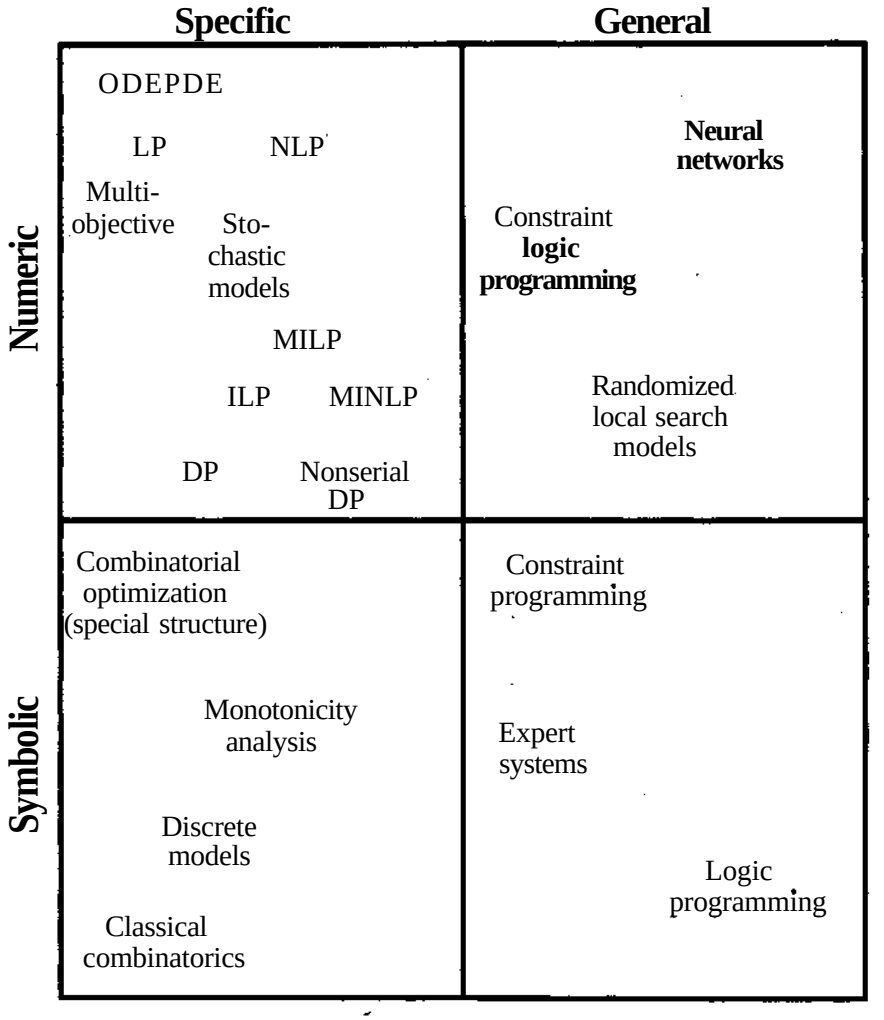


Fig. 1. Classification of models.

What We Want In a Model

We want general models because they presuppose less structure in the problem. They are more likely to fit messy, real-world situations. One software package will have wide application.

We want structured models because they reveal structure that helps us to understand the problem, and because they are more likely to be tractable.

The ideal would seem to be models that are as general as possible while being as structured as possible. This is reasonable but needs qualification, because it suggests, implausibly, that neural networks are close to the ideal. Despite their structuredness, neural networks do not reveal problem structure and often present a difficult "training" problem.

To obtain a more adequate analysis, we must a) develop a more nuanced understanding of how a model models, and b) acknowledge an additional desideratum for models: ease of calibration.

a) A neural network does not model a problem in the same way that an assignment model does. An assignment model *mirrors* the structure of an assignment problem, whereas the neural network model mirrors a neural netwoik! In other words, an assignment model models by reflecting the structure of the problem, whereas a neural network models by reflecting the structure of a *problem-solving device.* (We will see shortly that this sort of second-order modeling is characteristic of AI.) We should not infer, however, that a neural network does not actually model a problem. There are philosophical difficulties with saying that a model can model only by mirroring. It is safer to grant that a neural network models an assignment problem in the full sense of the word, because it can be used to solve the problem. But it models in a way that sacrifices one type of explanatory power.

b) An ideal model is easily calibrated, where "calibration* is used in the general sense of tailoring a model to a specific problem. It might involve adjusting parameters and coefficients, formulating constraints for a mathematical programming model, or designing a solution space for a local search algorithm (more on this later). In the case of the assignment model, both calibration and solution are easy. But calibration of the neural network is hard, since it must be "trained" on a large test set that one never knows quite how to design. In fact, in this case nearly all of the calculation involved is dedicated to calibrating the model. Even once the calculation is done, the result may be a poor calibration.

Just as less structured models *tend* to be harder to solve, more general models *tend* to be harder to calibrate, or at least harder to calibrate in a way that makes a good solution possible. In a genetic algorithm, for example, it is relatively easy to define a problem representation, a crossover operation, mutations, etc. But it is harder to define these so that good solutions evolve (Goldberg 1989).

So we want models that are a) general, b) structured, and c) easy to calibrate, all the while acknowledging that a structured model may fail to be tractable and may fail to have explanatory value.

To check the validity of this analysis, we can examine whether it accounts for die merits and demerits of several models, such as those listed in Table 1. Each model is scored on the three criteria from -6 to +4. Remarks on the right indicate when the structuredness of the model has unexpected implications. Models with higher scores should be more widely used (or at least worthy of wider use), except when the score is mitigated by the remark.

• *Linear programming* (Dantzig 1963). Its success is based on two key advances. One was Dantzig's discovery that this rather structured model is surprisingly general; it applies to a remarkably wide variety of problems. The other was his invention of a solution method (simplex) that exploits the structure. Calibration is usually not hard, since once one knows a few modeling tricks, he can represent a wide variety of problems.

• *Network flows with gains* (Bazaara *et al.* 1990). This highly structured special case of LP is very easy to solve but retains a surprising generality. The reticulate structure tends to illuminate the problem. The model is therefore popular in practice.

• *Economic order quantity model for inventory management* (Lee and Nahmias 1993). This ridiculously simple model is used rather widely because it is quite general *as an approximation.* It applies to a wide variety of inventory problems in which only a rough approximation is needed. It is hard to calibrate because it requires estimation of holding and stockout costs, but this seems to be offset by its use as a rough approximation. A number of back-of-the-envelope models are general as approximations and receive wide application for that reason. (In fact, the notion of model generality should be generality with respect to a stated degree of accuracy.)

Table 1. Evaluation of Some Models

Model	*Generality*	*Ease of calibration*	*Degree of structure*	*Total*	*Remarks*
Linear programming	0	**42**	**+2**	**44**	
Network flows with gains	-1	+2	**+3**	**44**	Very explanatory
EOQ, etc.	+1	0	+2	**43**	
Traveling salesman	-4	+2	+3	**41**	Hard to solve
Assignment, etc.	-6	+2	+3	-1	
Integer nonlinear programming	+1	+2	**-4**	-1	
Neural networks	+4	+2	-3	**43**	Not explanatory
Randomized local search	+4	-3	-1	0	Easy to solve suboptimally
Logic programming	+6	0	**-4**	+2	
Horn expert system	+1	-3	**44**	**42**	

• *Traveling salesman and other NP-hard combinatorial optimization models* (Nemhauser and Wolsey 1988). The traveling salesman, classical vehicle rooting, and other highly structured combinatorial models have seen limited application because they are too specific. Worse, they aie hard to solve despite their structure. (Their positive score is offset by the difficulty of solution.) From this point of view it seems questionable to commit so much attention on such highly structured, hard problems as the traveling saksmnn problem, although this kind of study may teach lessons that are useful elsewhere-

• *Assignment and other easy combinatorial models* (Lawlcr 1976). These include bipartite matching, problems with matroid structure, etc. They have received much attention because their high degree of structure permits fast algorithms. But they model only a few highly structured problems and so are seldom useful.

• *Nonlinear integer programming* (Hansen *et al.* 1994). This model has seen fairly limited application because, although quite general, it seems to be too unstructured mathematically to permit a good solution algorithm.

• *Randomized local search models.* Simulated annealing (Aarts and Korst 1989), tabu search (Glover 1989,1990), and genetic algorithms are similar to neural networks in that they model (in the mirroring sense) the problem-solving *process.* This allows them to remain fairly structured even while achieving a good deal of generality. These algorithms do local search in the sense that they go from one solution to a "neighboring" solution, except for probabilistic jumps. (In tabu search the latter are known as "diversification," in genetic algorithms as mutations.) Modeling the solution process requires that one define the problem space, and in partiuclar what counts as a neighboring solution. Local search models tend to be fairly simple and therefore to have a fairly high degree of structure. Although their simplicity does not make it easy to find an optimal solution, it often makes it easy to find good suboptimal solutions (for reasons that are not well understood). This factor alone would probably raise the above score of zero to +3 or +4, because their tractability is the main attraction of local search models. Their drawback is that modeling the solution space is itself a nontrivial task on which the success of the heuristic depends.

•*Neuralnetworks.* These are discussed above.

• *Logic programming* (Lloyd 1993). Logic programming follows a strategy similar to that of randomized local search heuristics: it gives up the idea that a model should "mirror" the problem. But rather than mirror a problem solving process, logic

models move to a discursive, as opposed to a "pictorial," model of the problem. This is equally effective at modeling a wide range of problems, but the resulting "declarative" models tend to be complex and unstructured, despite the small number of language elements involved. (This is a characteristic of discursive representations: a verbal description of an assignment problem uses only 26 letters, but the letters evince no particular pattern.) So logic models can be hard to solve. An extreme case are full first older logic models, which are incredibly general but often intractable. PROLOG is more restricted but currently has no efficient solution algorithm. European experience has shown that logic programming, thanks to its generality, can find a fair degree of application despite the lack of good algorithms- But it remains little used in the U.S., less than its score of +2 would justify.

• *Horn expert systems.* These highly special cases of logic programs, however, are widely used in the U.S., largely for historical reasons- Their generality is surprising in view of their very limited expressiveness (Hooker 1988). They main attraction is that the inference problem is extremely easy for Horn formulas.

Research Directions

OR emphasizes structured, numeric models. They have narrow application but provide a good deal of explanatory power when they do apply. They may also benefit from effective solution algorithms. AI has emphasized unstructured models, with a historical preference for symbolic representations. An extreme example is one of die first, H. Simon's General Problem Solver (Newell and Simon 1972). But these models sacrifice explanatory power when they overlook structure. Solutions may be computationally elusive and of poor quality when obtained.

There is merit in combining OR and AI, because many problems call for models that lie between these extremes. They have both numeric and symbolic elements. They have less structure than classical OR models but enough to be exploited by algorithms more specialized than those typical of AI.

The specific research strategies that could combine OR and AI seem to depend on which quadrant of Fig. 1 one is looking at We therefore survey die quadrants.

Specific structured models.

These typically OR models can move toward AI by becoming more general, and one route to greater generality is to become more symbolic.

One way to make a quantitative model more symbolic is to mix symbolic constraints with quantitative ones. Constraint programming does this (Van Hentenryck 1989). In this field the goal is generally to find a feasible solution, one of which may be a bound on the objective function. A job shop scheduling problem, for instance, can be described by a mixture of logical rules and numerical inequalities, plus a lower bound cm themakespan. The object is to find a feasible schedule that meets this bound Constraint programming has found some acceptance in Europe but less in die U.S. It is largely ignored by the OR community. Perhaps one reason for this is that, except to the extent it is undergirded by the theory of logic programming, constraint programming does not generally have the kind of theoretical grounding and deep analysis enjoyed by mathematical programming. OR people who have heard of it tend to dismiss it as little more than a collection of heuristics. But it is unclear that mixed logical/numerical constraint sets cannot be systematically analyzed in the way that numerical constraint sets have been. So here is a research program, to which OR people can contribute.

A related thrust is the use of logical methods in the solution of mathematical models, particularly those with combinatorial complexity. Prepositional logic, for instance, is useful in the solution of mixed integer programming problems. In fact one can develop logic-based concepts and theory parallel to those of branch-and-bound, cutting planes, facet-defining cuts, etc. (Hooker 1994).

Another related idea is to replace some quantities with qualitative descriptions and to try to deduce properties of the solution. This occurs in symbolic differentiation, monotonicity analysis, etc. To date these approaches have been used primarily to guide numerical algorithms, but they may have potential as more general models.

We have already noted an interesting way to make quantitative models more general without making them more symbolic: they can be used to model (in the mirroring sense) the problem solving process, rather than the problem itself. The influence of AI is clear here, particularly in the case of neural networks, which crudely model a problem-solving brain. Genetic algorithms (which are somewhat less numeric) model a problem-solving evolutionary process. Simulated annealing algorithms and some neural networks model nature's way of minimizing energy. Tabu search models a search process with short-term memory; the early literature even suggested 7 as the ideal length of a tabu list, because human short-term memory generally has room for 7 chunks of information. An obvious research direction is to continue to model problem-solving processes, and it is being actively pursued. Such new search strategies as scatter search

are being invented, such social problem-solving processes as asynchronous teams are being investigated, and so on.

Specific symbolic models.

There are two related classes of problems in this quadrant: a) problems from classical combinatorics (stable marriage problem, matching, etc.), as well as some combinatorial problems associated with OR (set covering, clique partition, etc.); b) highly structured logic programming problems (arranging queens on a chessboard, etc.). They are already symbolic to a large degree and need to be generalized.

One effort toward generality has been toward abstraction: matroids, greedoids, submodularity, etc. (Nemhauser and Wolsey 1988, Korte and Lovasz 1984). This has not significantly increased the range of application. Abstraction produces an even simpler model that is equally unlikely to apply.

Another approach studies structured problems with "side constraints,"[9] such as network flow problems with a few additional constraints. This gives rise to relaxation and decomposition strategies. The former enumerates relaxations of the problem (i.e., the hard constraints are thrown out), as is done in *A** search. The latter views a problem as consisting of structured subproblems coupled by a few additional constraints, which again leads to an enumeration of relaxations. These approaches have been applied most often in mathematical programming; decomposition, for instance, is used in the the methods of Dantzig-Wolfe, Benders, and Lagrangian relaxation. A possible research direction is to try to apply them to "purer" combinatorial problems.

General quantitative models.

Here, as noted earlier, the bottleneck is model calibration. This suggests that research on probabilistic local search heuristics should evolve into research cm solution space models. Rather than dividing the field into tabu search, annealing, etc., we can divide the field into solution space models of this sort, models of that sort, etc.

Another research direction is to study empirically the behavior of heuristics on certain classes of problems. Good experimental design may be able to isolate some problem characteristics that predict the performance of heuristics. For instance, simulated annealing may work best on vehicle routing problems when it creates a Markov chain having certain properties, and certain problem traits may give rise to these properties. Empirical algorithmic science therefore lies somewhere between the

specificity of typical OR models and the generality of local search metaheuristics. It relies cm some structural knowledge about a problem in order to help design a heuristic for it, but it need not presuppose the high degree of structure required by typical OR models. There is a good deal ofmformalwoik man e^pirical mode, but research canons have not yet evolved

General symbolic models.

These are in essence logic models* They can be moved closer to OR by making them more structured, perhaps by making them more numeric.

The project of making logic models more numeric goes by the name of constraint logic programming (not quite the same as constraint programming). It generally adds a few specifically numeric predicates to a logic programming language, usually related to linear inequalities, and tries to develop partially numeric algorithms to conduct logical inference. A typical problem is to answer such a query as, Tor some *xl*> x2, *xl* £x2," in a logic program that contains a linear constraint set Ax £ *b.* A response to such a query would be a description of possible values of *xlpc2* consistent with *Ax £b* and *xl£ x2.* The problem is solved as a polyhedral projection problem. (The main difference with constraint programming is that constraint logic programming works within the framework of a formal logic model, whereas constraint programming need not) This is a wide-open research area, particularly as logic programmers in this area often do not have equal training in operations research.

Examples

In this section we examine some examples of recent research which combines AI and Optimization. Although a complete sampling of the different models and methods is not presented, this section should give the reader a flavor of the merging of the two approaches. We first examine the combination of Logic in MHJP methods. Next we examine monotonicity analysis, a symbolic approach abstracted from formal optimization criteria (the Karusch-Kuhn-Tucker conditions). Monotonicity analysis is then used in conjunction with a set of mathematical heuristics to expand a design space (modifying the topology). Finally, an approach to generate the optimal topology of a network flow problem is discussed.

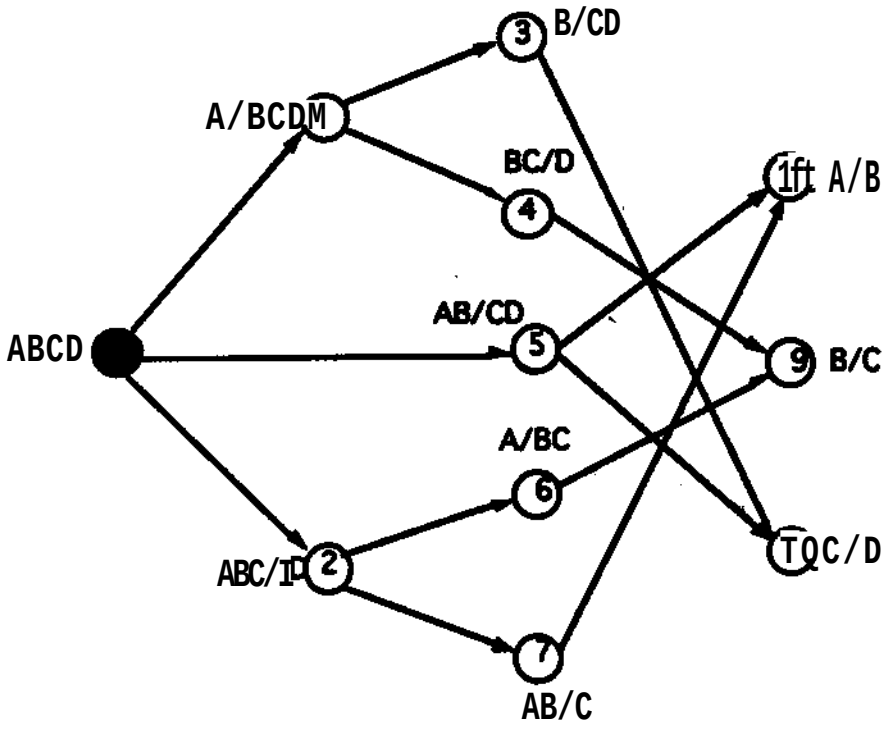
Integration of Logic in Mixed-Integer Optimization Models for Synthesis

Raman and Grossmann (1993) have proposed methods for the symbolic integration of prepositional logic relations between potential units in a superstructure to aid the branch and bound search in MILP synthesis models. The objective of this integration is to reduce the number of nodes that must be enumerated by using the logic to decide on the branching of variables and to determine by symbolic inference whether additional variables can be fixed at each node. A unique feature of this approach is that it does not require additional constraints in die MILP since die logic relations arc expressed and processed symbolically. Figure 2 presents an **example of a** separation network superstructure for which the logic of relations between the units is postulated in the form of prepositional logic.

Two different strategies for performing the symbolic integration of logic in the LP-based branch and bound method can be used. In one strategy the prepositional logic is expressed in conjunctive normal form (CNF); i.e. a conjunction of clauses containing only disjunctions. The logic inference for fixing additional variables at each node is performed with unit resolution. The other strategy consists in converting the logic into the disjunctive normal form (DNF); i.e. a disjunction of clauses containing only AND operators. The logic inference for fixing 0-1 variables is performed by solving the resulting boolean equations. This strategy is suitable for synthesis problems in which the superstructures have significantly fewer design configurations than the total number of 0-1 combinations. An interesting theoretical result in this approach is that the number of nodes required in the proposed branch and bound method is not more than twice the number of clauses in the DNF form. It should also be noted that a formalization of the above ideas has been accomplished by Hooka: et al. (1993) who introduced the concept of logic cuts for process networks with fixed cost charges.

The algorithms described above were automated using OSL both as stand alone software and within the GAMS modelling environment. As an example consider the problem in Figure 2 of synthesizing a four component separation system that in addition involves heat integration by allowing for heat matches between reboilers and condensers of the various separation columns in order to reduce the utility cost for operation. Binary variables are required to model the selection of columns and for the matches between reboilers and condensers. For the four component system, 100 binary variables are needed - 10 to model the existence of the distillation columns and 90 to model the existence of heat exchange matches between the reboilers and condensers of the various

columns. The results for this problem are shown in Table *2.* The first column corresponds to the original MILP model, the second to the MILP in which the logic is added in the form of inequalities, and the third to the MILP in which the logic is integrated symbolically with the DNF approach. Note that in the case when die logic is added as inequalities the number of constraints is almost doubled, although in this case this has no effect on the LP relaxation. The branch and bound search with logic inequalities ami with the symbolic integration shows a considerable improvement over the original MILP model. While ZOOM, SQCONIC and OSL were not able to solve the problem even after 100,000 nodes and after more than 30 minutes on the IBM RISC/6000, the DNF based approach solves the problem to rigorous optimality in 18S nodes with ZOOM, 29 nodes with SQCONIC and 20 nodes with OSL, and in all cases in less than 10 seconds! Using the formulation with logic inequalities the reductions in CPU time are also very significant, although not as large as with the symbolic DNF approach. This example then shows how the integration of logic in quantitative optimization models can have a great impact



(a) Superstructure

Yl ^ Y4 v Y5	Y6 = * Y3 A Y9
Y2 = > Y8 A Y10	Y7 ^ Y3 A Y8
Y3 = *Y6vY7	Y8=*Y2vY7
Y4 = > Y1 A Y10	Y9 ^ Y5 v Y6
Y5 ^ Yl A Y9	Y10=>Y2vY4

(b) Logic relations

Figure 2. Superstructure for 4 component separation sequence

Table 2. Computational results for different MILP models

	Original Formulation	Formulation with Logic	DNF based Approach
SIZE			
constraints	258	473	258
variables	291	291	291
binary	100	100	100
LP relaxation	1117.72	1117.72	1117.7
MILP solution	1900.58	1900.58	1900.58
ZOOM			
nodes	could not solve	78	185
CPU time	> 2,000	20.78	8.5
SCICONIC			
nodes	> 100,000	18	30
CPU time *	> 2,000	4.8	1.6
OSL			
nodes	> 100,000	74	20
iterations	> 1,000,000	540	238
CPU time *	> 5,000	8.37	2.76

* sec IBM R6000 / 530

Qualitative Optimization

Monotonidty analysis (Papalambros and Wilde, 1988), an abstraction of the Karush-Kuhn-Tucker (KKT) Conditions, can be used as a fonn of qualitative optimization. Developed as a pie-optimization technique to guarantee well-boundedness for numerical optimization, the technique is based on two rales:

Rule One!// ***the objective function is monotonic with respect to (w.r.t.) a variable, then there exists at least one active constraint which bounds the variable in the direction opposite of the objective. A constraint is active if it acts at its lower or upper bound.***

Rule Two:/fa ***variable is not contained in the objective function then it must be either bounded from both above and below by active constraints or not actively bounded at all (Le., any constraints monotonic WJT.L that variable must be inactive or irrelevant.)***

Choy and Agogino (1986) and Agogino and Almgren (1987) use monotonicity analysis as a fonn of qualitative reasoning. Monotonidty analysis identifies sets of constraints that, when active, satisfy the first order necessary conditions of optimality (KKT). For monotonic problems, their SYMON/SYMFUNE programs generate symbolic solutions to the optimization problems; for constraint-bound problems these solutions can be closed-form, leading to parametric design charts (see Agogino and Almgren, 1987). Michelena and Agogino (1990), Rao and Papalambros (1987), Azram and Papalambros (1984), and Hansen, *et* a/., (1989) have also implemented heuristic optimization programs employing some form of monotonicity analysis.

Design Space Expansion/Design Innovation

The lstPRINCE methodology was introduced by Cagan and Agogino (1987) to generate innovative designs where innovative designs are defined as those designs in which new variables or features are introduced relative to a known set of variables or features, expanding the useful design space. The design problem is formulated as an optimization problem of engineering first principles. lstPRINCE uses optimization information, via monotonicity analysis when possible, to identify sets of active constraints which model optimally directed designs, and to determine how to expand the design space to introduce the new variables or features. Expansion techniques manipulate the mathematics of the problem formulation, thereby introducing new design variables. One of these techniques, called Dimensional Variable Expansion - DVE (Cagan and Agogino, 1991a) expands the design space in a serial fashion; a different

technique called Input Variable Expansion - IVE (Aclion, *et. al,* 1991b) expands the design space in a parallel fashion. Expansion is not done arbitrarily; rather critical variables are identified from optimization information and an appropriate expansion technique is then selected. Once the design space is expanded, it is searched by monotonicity analysis or numerical techniques. Induction techniques are also introduced to examine, from optimization information, the constraint activity over generations of design expansions to determine the limiting solution of the design process (Cagan and Agogino, 1991b. Because the optimization technique uses monotonicity analysis, the method woiks best with monotonic problems.

l[st]PRINCE has been used to derive solutions to problems in various domains. A class of hollow tubes and composite rods were derived from a solid round rod under torsion load to minimize weight; the same rod under flcxural load leads to tapered beam solutions and I-beam solutions. A wheel was invented from a rectangular block to minimize resistance to spinning. Aclion, *et. al*$_9$ (1991a) derived the solution for a plug flow reactor (PFR) from the initial design of a well-mixed reactor (CSTR) to maximize conversion; under different conditions, a sequence of reactors in parallel were generated also to maximize conversion. Other applications generated a powder catalyst from an initial sphere to maintain surface area while minimizing weight, and a sequence of columns were designed form a single column in order to generate a *feasible* design from an initially *infeasible* design. l[st]PRINCE is thus able to generate new design topologies by using optimization information for both design space search and expansion.

An application of l[st]PRINCE to the design of chemical reactors is shown in Figure 3. The goal of this design is to maximize a first-order chemical reaction, subject to a maximum volume constraint. The initial design is that of a *well-mixed reactor.* The well-mixed reactor is an ideal conception in which all the contents are perfectly mixed. The design becomes:

The primitive prototype is given by:

minimize C\\

Subject to £AfL.i+k£L (hi)

C_{A1} V_0

V\ * 0 (gi)

V_{max} (g2)

where CAO * inlet concentration of species A [moles/liter],
CA1 = outlet concentration of species A [molcs/litcr],
k * reaction rate constant [sec*[1]],
UO = volumetric feed rate [litcrs/scc],
Vl = reactor volume [liters],
Vmax = maximum allowable reactor volume [liters]*

The problem minimizes exiting reactant concentration subject to a well-mixed reactor mass balance and an upper bound cm reactor volume.

A detailed analysis of this and similar problems can be found in [Action, *et al.*, 1991 and 1992]. Applications of DVE on reactor volume and IVE on input flow produce serial and parallel reactor systems. Of particular interest is the limit of infinitely many and differentially small well-mixed reactors in series produced from DVE* which produces the identical behavior of a *plug-flow reactor.* Here, * monotonicity analysis of the input optimization formulation identifies that ccmstraint hi must be active to balance the increase of C A I in the objective function; this requires that ccmstraint g2 become active to avoid an unbounded increase of the reactor volume. DVE is then applied across volume Vi producing the new optimization problem of two reactors. Successive application of DVE leads to successive series of reactors which, in the limit, produce die plug-flow reactor through inductive techniques. This reactor type is another theoretical conception, where there is no axial mixing. There is only radial mixing in every differential slug traversing the reactor. l[st]PRINCE is able to innovate this design based purely on variable and constraint typing and the application of DVE.

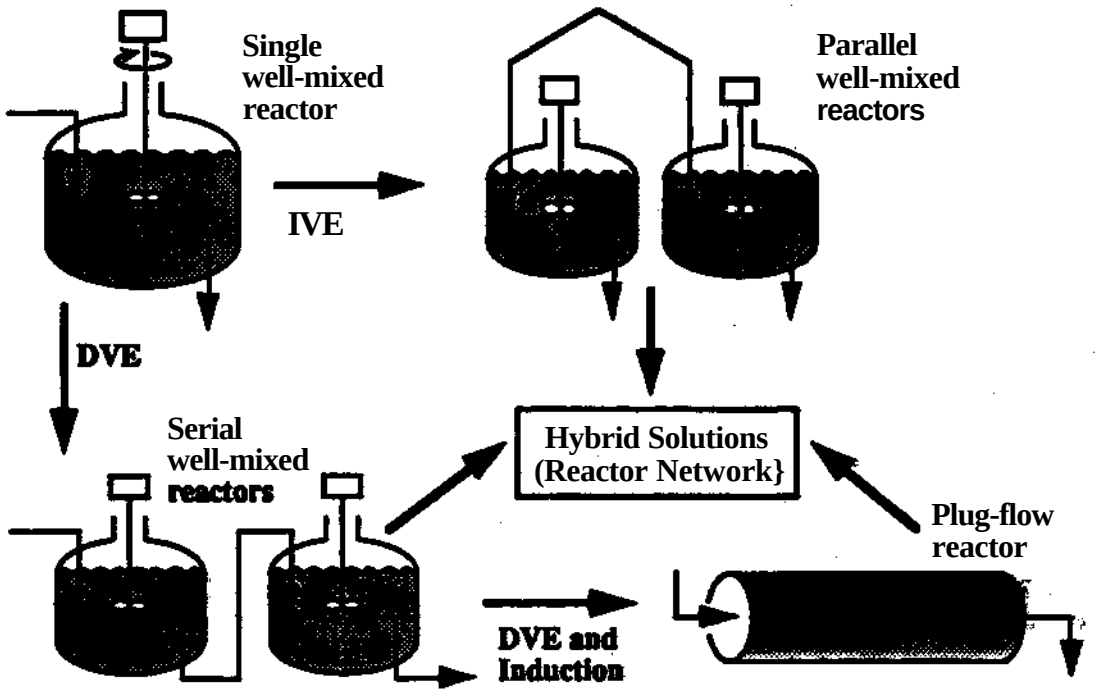


Figure 3. Application of 1st PRINCE to chemical reactor design

Shape Annealing

As **a** different example of how optimization can generate design topologies by breaking away from local minima, we examine the design technique of *shape annealing* introduced by Cagan and Mitchell (1993). Here a design problem is again formulated as an optimization problem. However, the problem knowledge is modeled as a *shape grammar* with all the properties of shapes described by Stiny (1980). Concepts of simulated annealing are used to create a technique which generates optimally directed solution shapes.

The shape annealing algorithm executes by applying a shape rule to an initial design. If the modification improves the design based on an objective it is accepted. If it generates an inferior design then it can still be accepted with a certain probability which is a function of the number of iterations executed; toward the beginning of the process almost all inferior solutions are accepted and as the algorithm progresses, only those solutions which improve the objective are accepted. In shape annealing, previous designs can be re-gained if they are superior; for every rule which modifies a shape, there is a complementary rule which removes that modification. Thus, shape annealing is an *evolutionary* design technique to explore the exponentially large number of possible

design configurations. Inferior solutions are pursued to get out of local minima, and optimally directed design topologies are derived

Applications of the shape annealing technique include solutions to constrained gecmietric knapsack problems - knapsack problems constrained by geometry and relative component orientations - (Cagan, 1994), generation of truss structures (Reddy and Cagan, 1993) and component layout (Szykman and Cagan, 1993). As an illustration of shape annealing, Figure 4 shows a shape grammar that manipulates triangles. Reddy and Cagan use this grammar to modify the topology of truss structures. Each time a truss topology is modified, the structure is shape optimized with gradient based methods interfacing with finite element methods. This modification continues until the final configuration is achieved. A variety of constraints can be incorporated through the finite element analysis. Figure 5 shows two structures generated from shape annealing; one without buckling constraints (Sa) and one that includes buckling constraints (5b). Curtrent weak is further integrating the AI and optimization technologies; a new implementation uses simulated annealing for both topology modfications and shape optimization (Reddy and Cagan, 1994).

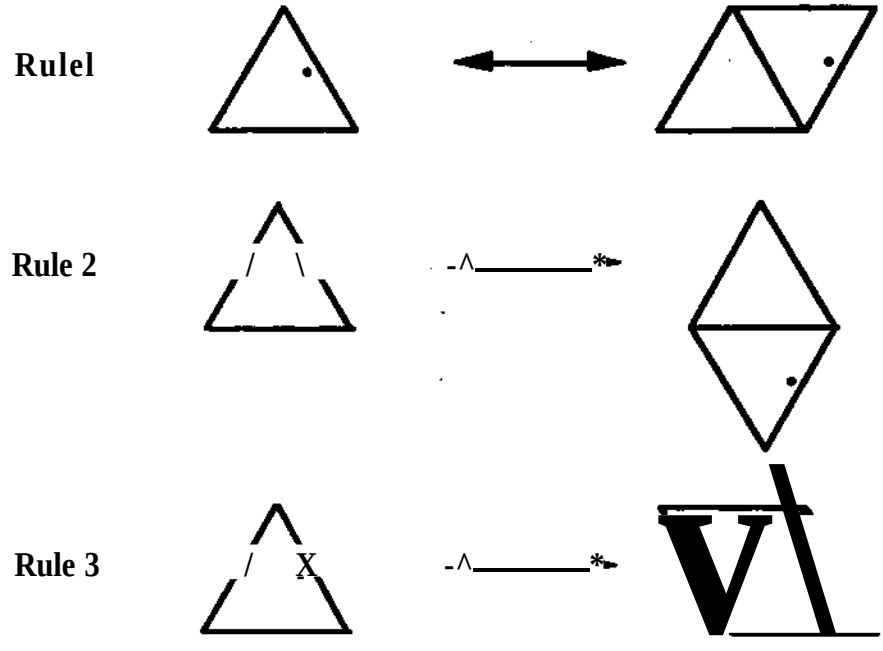


Figure 4: Shape grammar for truss generation (from Reddy and Cagan, 1993)

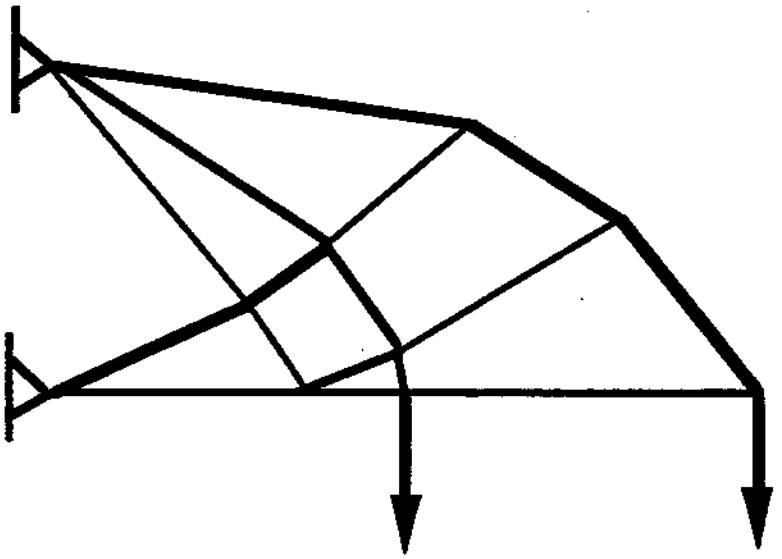


Figure 5a

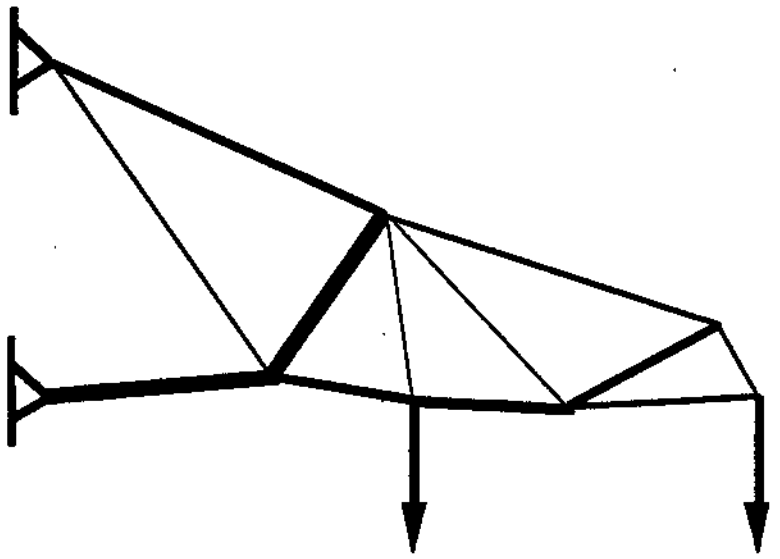


Figure 5b

Figure 5: Truss structures generated by shape annealing topology without buckling constraints (5a) and with buckling constraints (Sb) (from Reddy and Cagan, 1993)

Conclusions

This paper is meant to both provide a conceptual overview of the problem of combining artificial intelligence and optimization, and to motivate the need for and suggest directions of future research. AI and optimization together make sense: few problems can be solved by structured optimization methods alone, and AI techniques are limited in their search for general designs. Rather, the combination of AI and optimization can provide problem specific reasoning, symbolic representations, and powerful numerical optimizing search. The framework is here; future work must continue to marge and expand the focus of combined AI and optimization problem solving.

Acknowledgments

The authors would like to acknowledge financial support from the Engineering Design Research Center-

References

Aarts, E., **and** J. Korst (1989), *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing* (Wiley, New York).

Aarts, E.H.L. and van Laarhoven, PJ.M. (1985). Statistical Cooling: A General Approach to Combinatorial Optimization Problems. *Phillips J. Res.* 40.

Aelion, V., J. Cagan, and G. Powers (1991), "Inducing Optimally Directed Innovative Designs from Chemical Engineering First Principles", *Computers and Chemical Engineering,* 15(9):619-627.

Aelion, V.9 J. Cagan, and G. Powers (1992), "Input Variable Expansion - An Algorithmic Design Generation Technique", *Research in Engineering Design,* 4:101-113.

Agogino, A.M., and A.S. Almgren (1987), "Techniques for Integrating Qualitative Reasoning and Symbolic Computation in Engineering Optimization", Engineering Optimization, 12:117-135.

Amarger, R., L.T. Biegler and IJE. Grossmann (1991), "REFORM- An Intelligent ModellingSystem for Design Optimization", EDRC Report.

ANSYS *(\992), Version 5.0,* Swanson Analysis Systems Inc.

Aspen Technology (1991), "Aspen-Plus User Guide. Release 8.3", Cambridge, Mass.

Azram, S., and P. Papalambros (1984), "An Automated Proceedurc for Local Monotonicity Analysis", Transactions of the ASME, Journal of Mechanisms, Transmissions, and Automation in Design, 106:82-89.

Banzhaf, W. (1989), "Computer-Aided Circuit Analysis using SPICE", Prentice Hall, Englewood Cliffs, NJ

Bait, A. and E.A. Feigenbaum (eds.) (1981), *Handbook of Artificial Intelligence,* 3 vols., William Kaufinann, Inc., Los Altos, CA.

Bazaraa, M£. and CM. Shetty, *Nonlinear Programming,* John Wiley (1979).

Bazaara, M. S., J. J. Jarvis and H. D. Sherali (1990), *Linear Programming and Network Flaws* (Wiley, NewYoric).

Becker, E., J.T. Oden and G.F. Carey (1982), "Finite Elements: An Introduction", Prentice Hall, Englewoods Cliffs, NJ.

Biegler, L.T. (1990), "Strategies for Simultaneous Solution and Optimization of Differential-Algebraic Systems", Proceedings Third Int. Conf.Foundations of Computer-Aided Process Design (eds. Siirola et al), Elsevier, New York.

Brooke, A., D. Kendrick and A. Meeraus (1988), "A GAMS User's Guide", Scientific Press, Palo Alto.

Cagan, J. (1994), "A Shape Annealing Solution to the Constrained Geometric Knapsack Problem", in press: *CAD.*

Cagan, J. and A.M. Agogino (1987), "Innovative Design of Mechanical Structures from First Principles", *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing,* 1(3): 169-189.

Cagan, J. and A.M. Agogino (1991a), "Dimensional Variable Expansion - A Formal Approach to Innovative Design", *Research in Engineering Design,* 3:75-85.

Cagan, J. and A.M. Agogino (1991b), "Inducing Constraint Activity in Innovative Design", *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing,* 5(1):47-61.

Cagan, J., and WJ. Mitchell (1993), "Optimally Directed Shape Generation by Shape Annealing", *Environment and Planning B,* 20:5-12.

Carnahan, B. and Wilkes, J.O. (1980), "Applied Numerical Methods," Wiley New York.

Choy, J.K., and A.M. Agogino (1986), "SYMON: Automated SYMbolic MONotonicity Analysis for Qualitative Deisgn Optimization", *in: ASME Computers in Engineering,* (Gupta, G., ed.), 1: 207-212.

Dahlquist, *A3.* and N. Anderson (1974), "Numerical Methods," Prentice Hall, Engtewood Cliffs, NJ.

Dantzig, G. (1963), *Linear Programming and Extensions* (Princeton Univ. Press, Princeton, NJ).

Dodd, A. (1990), *PROLOG: A Logical Approach,* Oxford University Press, New Yak.

Drud, *AS.* (1991). *CONOPT - A Large Scale GRG Code,* ARKI Consulting and Development Denmark

EXSYS Inc. (1990X *EXSYS User's Manual,* Albuquerque, NM.

Geoffikn. *AM.* (1972), "Generalized Benders Decomposition". JOTA 10(4), p. 237-260.

Fenves, SJ. and LE. Grossmam (1992), "An Interdisciplinary Courae in Enfueering Synthesis." *Research in Engineering* Design, 3,223-231.

Gill. P.E., W. Murray, MA. Saunden and *hLA.* Wright (1983). *User's Guide for SOUNPSOL: A FORTRAN Package for Nonlinear Programming,* Dept Optus. Res. Stanford University, Technical Report SOL 83-12.

Glover. F. (1989), *"TabaSearch-Pml", ORSA Journal on Comptaing* 1:190-206.

Glover, F. (1990). "Tabu Search-Part IT, *ORSA Journal on Computing* 2:4-32.

Goldberg, D. E. (1989), *Genetic Algorithms in Search, Optimisation and Machine Learning* (Addison-Wesley, Reading, MA).

Hansen, P., B. Jaumard and V. Mathon (1994), "Constrained nonlinear 0-1 programming", *ORSA Journal on Computing,* to appear.

Grossmann, I.E. (1990), "Mixed-Integer Nonlinear Programming Techniques for the Synthesis of Engineering Systems," *Res. Eng. Des.,* l(0):205-228.

Hooker, J.N. (1988), "Resolution vs. cutting plane solution of inference problems: Some computational experience". *Operations Research Letters,* 7:1-7.

Hooker, J. N. (1988), "A quantitative approach to logical inference". *Decision Support Systems* 4:45-69.

Hooker, J. N. (1994), "Logic-based methods for optimization: A tutorial", *manuscript. GSIA,* Carnegie Mellon University, Pittsburgh, PA 15213 USA.

Hooker, J.N., H. Yan, *IE.* Grossmann, and R. Raman (1993), "LogicCuts for Processing Networks with Fixed Charges," *Computers and Operations Research,* accepted for publication.

Hillier, *FS.* and G J. Lieberman (1986), *Introduction to Operations Research,* Holden Day.

Hansen, P., B. Jaumard, and S.H. Lu (1989), "An Automated Proceedure for Globally Optimal Design", *Transactions of the ASME, Journal of Mechanisms, Transmissions, and Automation in Design,* 111:361-367.

IBM (1991), *Optimization Subroutine Library. Guide and Reference-Release 2,* Kingston, NY.

IMSL Math. Library (1987), *FORTRAN Subroutines for Mathematical Applications,* Houston, TX.

Kirkpatrick, S., C D. Gelatt, Jr., and MJ>. Vecchi (1983), "Optimization by Simulated Annealing", *Science,* 220<4598):671-279.

Kortc, B., and L. Lovasz (1984), NGreedoids~A structural framework for the greedy algorithm", in W. R. Pulleyblank, ed.. *Progress in Combinatorial Optimization,* Academic Press, pp. 221-244.

Lawkr, E. (1976), *Combinatorial Optimization: Networks and Matroids,* Holt, Rinehart and Winston, New York.

Lee, H. L., and S. Nahmias (1993), "Single-product, single-location models", in S. C. Graves, A. H. G. Rinnooyy Kan and P. H. Zipkin, eds.. *Logistics of Production and Inventory* , North-Holland, Amsterdam, pp. 3-58.

Levary, RJt. (Ed) (1988), *Engineering Design-Better Results Through Operations Research Methods,* North Holland, Amsterdam.

Liebman, J., L. Lasdon, L. Schrage and A. Warren (1986), *Modelling and Optimization with GINO,* Scientific Press, Palo Alto.

Lloyd, J. W. (1993), *Foundations of Logic Programming,* Springer Verlag, Berlin.

Marsten, R. (1986), *User's Manual for ZOOMfXMP,* Department of Management InformationSystems, University of Arizona.

Marsten, R., M. Saltzman, J. Lustig and D. Shanno (1990), "Interior Paint Methods for Linear Programming: Just Call Newton, Lagrange and Fiacco and McCormick!" *s20 Interfaces,* 20(4), pp. 105-116.

Michelena, N. and A.M. Agogino (1990)*****

Murtagh, B.A. and M.A. Saunders (1985), *MINOS User's Guide,* Systems Optimization Laboratory, Department of Operations Research, Stanford University.

Nemhauser, GX., A.H.G. Rinnory Kan and MJ. Todd (Eds) (1989), "Optimization", Vol. 1, *Handbook in Operations Research and Management Science,* North Holland, Amsterdam.

Nemhauser, G. L., and L. A. Wolsey (1988), *Integer and Combinatorial Optimization,* Wiley, New York.

Newell, A., and H. Simon (1972), *Human Problem Solving,* Prentice-Hall, Englewood Clilffs, NJ.

Nilsson, N J. (1990), *Principles of Artificial Intelligence,* Tioga Publ. Co., Palo Alto.

Papalambros, P.t and DJ. Wilde (1988)9 ***Principles of Optimal Design.*** **Cambridge University Press, Cambridge.**

Petzold, LJL (1982), A *Description ofDASSL: A DifferentiallAlgebra* System Solw.Smfo* **Tech. Rep. 82-8637.**

Pieta,P-C,mEH>eriyflLM.We8lw^ and A.W. Westerberg (1991), "ASCEND: An Object-oriented Computer Environment for Modeling and Analysis", ***Comp.Chem.Eng.*****, 15,53-72.**

Pismetzky, S. (1984),*Sparse Matrix Technology,* Academic Press, London.

Quine,W. V.(1961), *From a Logical Point of View: Nine Logico-Philosoptical Essays,* **Harvard** Univ. **Press, Cambridge, MA.**

Reddy, G., and J. Cagan (1993), "Optimally Directed Truss Topology Generation Using Shape Annealing", *in: Advances in Design Automation, Proceedings of the ASME Design Automation Conference.* **Albuquerque, NM, September 19-22,1:749-759.**

Reddy, G., and J. Cagan, "An Improved Shape Annealing Algorithm For Truss Topology Generation", ***EDRC Report 24-108-94,*** **Engineering Design Research Center, Carnegie Mellon University, Pittsburgh, PA 15213,1993.**

Raman, R. and IJE. Grossmann (1993), "Symbolic Integration of Logic in Mixed Integer linear Programming Techniques for Process Synthesis,[1]* ***s20 Computers and Chemical Engineering,*** **17, 909.**

Rao, LRJ., and P. Papalambros (1987), "Implementation of Semi-Heuristic Reasoning for Bounded Analysis of Design Optimization Models", in: *Advances in Design Automation -1987, proceedings ofthe ASME Design Automation Conference,* Boston, Sept 27-30, pp.59-65.

Reklaitis, G.V., A. Ravindran and K.M. Ragsdell (1983), ***Engineering Optimization - Methods and*** *Applications,* John Wiley.

Rice, JJL (1983), *Numerical Methods. Software, and Analysis: IMSL Reference Edition,* McGraw Hill, **New York.**

Rumelhart, D. E., G. E. Hinton and R. J. Williams (1986), "Learning internal representation", in D. E. Rumelhart and J. L. McClelland, eds., *Parallel Distributed Processing: Exploration in the Microstructure ofCognition. Vol. 1: Foundations,* MIT Press, Cambridge, MA, pp. 318-362.

Schrage, L. (1986), *Linear, Integer and Quadratic Programming with UNDO,* ScientificPress, Palo Alto.

SCICONICIVM User Guide (Version 1.4) (1986), Scicon Ltd. Milton Keynes.

Stiny, G. (1980), "Introduction to Shape and Shape Grammars", ***Environment and Planning B,*** **7:343-351.**

Szykman, S., and J. Cagan (1993), "Automated Generation of Optimally Directed Three Dimensional Component Layouts", in; *Advances in Design Automation, Proceedings of the ASME Design Automation Conference,* Albuquerque, NM, September 19-22,1:527-537.

Van Hentenryck, P., (1989), *Constraint Satisfaction in Logic Programming,* MIT Press, Cambridge, MA.

Vasantharajan S.$_9$ J Viswanathan and L.T. Biegler (1990), "Reduced SQP Implementation for Large-scale Optimization Problems", *Computers chem. Engng* 14, In press.

Viswanathan, J. and LE. Grossmaim (1990), "A Combined Penalty Function and Outer-Approximate Method for MINLP Optimization", *Computers and Chem. Eng.,* 14,769-782.

VP-Expert (1989), Paperback Software, Berkeley, CA.

Williams, HP. (1978), *Model Building in Mathematical Programming, Wilcy-lntarscicncc.*