Research in Engineering Design (1997) 9: 20–34

Research in
Engineering
Design

A Conceptual Framework for Combining Artificial Intelligence and Optimization in Engineering Design

Jonathan Cagan, Ignacio E. Grossmann and John Hooker
Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, USA

Abstract. *Research in artificial intelligence and optimization (OR) has had significant impact on the formulation and solution of computational methods in engineering design. This paper presents a conceptual framework for understanding a more powerful technology that is evolving from a combination of these approaches. The paper first proposes generalized representations of engineering design models that involve quantitative and qualitative aspects. Second, it presents a general classification of AI and OR models in terms of model attributes, in order to establish mappings with generic solution techniques. Third, the requirements of solution methods are discussed, as well as several schemes for the integration of AI and optimization to identify future research directions. Several specific approaches are included to illustrate various ways in which AI and optimization can be combined for tackling computational design models.*

Keywords. Artificial intelligence; Design; Operations Research; Optimization

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 on design alternatives as well as quantitative models to predict the performance 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 them. 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 there has been great interest in developing solution approaches to design problems that rely on combining AI and Optimization (e.g., [1–6], as well as other references given below). These efforts have been motivated by the fact that most work in engineering design and synthesis has involved the development of computational methods that either rely only on AI techniques (commonly expert systems), or only on optimization techniques (commonly NLP and MINLP models). It has become evident that there are several shortcomings in approaching synthesis problems with only one methodology (e.g., [7]). The major problems for the AI-based methods are 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 are using engineering knowledge for exploring the space of design alternatives, and limitations in solving large-scale problems.

It is the objective of this paper to present an overview of emerging concepts and ideas in the combination of AI and optimization techniques for dealing with qualitative and quantitative issues in engineering design. Our thesis is that, due to the symbolic and numeric nature of design problems, the merging of the AI and optimization fields will have strong impact on the capabilities of computational design tools. The scope of the paper is restricted to the steps in the design process that involve formulation and solution of a computational design model. Our specific objectives in this work are first to propose symbolic representations and classifications of design models and show their association to AI on the one hand and to optimization (and more generally to operations research, OR) on the other; for future reference we also provide an overview of solution techniques in the Appendix. Having shown that a large

Correspondence and offprint requests to: Jonathan Cagan, Department of Mechanical Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA.

class of design problems can be formulated as AI problems, OR problems, or a combination of the two, we next present a general classification of AI and OR models in terms of model attributes and establish mappings with generic solution techniques. We then discuss the requirements of the solution methods and several schemes for the integration of AI and optimization in order to characterize present and future research directions. The aim is to identify ways in which AI and optimization methods can be truly synthesized, rather than simply mixed as when an AI-based system merely calls optimization routines to solve well-structured subproblems. Finally, we briefly describe some specific illustrations of this kind of synthesis. Our main goal is to characterize recent research trends, and to present a conceptual framework within which they can be understood, rather than to provide an exhaustive survey of recent work.

Classification of Models in Engineering Design

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

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. If this is not the case, return to steps 1 or 2.

Given that we have defined the search space and its corresponding representation (e.g., graph, network) in step 1, two crucial steps in the above procedure are the development of a computational model, and its solution through an appropriate technique. This paper is aimed at the integration of AI and optimization techniques in these two steps of the computational design process. 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. In particular, within these two steps, it is often feasible (and practical) to represent design problems as logic, constraint satisfaction, or symbolic reasoning problems that are solved by AI techniques, or optimization problems that are solved by OR techniques. However, combined symbolic and numeric (or quantitative) models, which capture broader aspects of a design problem, cannot, in general, be solved with current approaches. Therefore, we argue that the combination of AI and OR techniques will lead to important advances in design computation since they will considerably expand the capability for solving a broader class of design models.

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,
(c) mixed discrete/continuous.

The continuous model will involve quantitative variables such as areas, forces, stresses, flows, and 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). The mixed model will contain both types of variables.

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

(a) *deterministic* if the inputs are exactly known,
(b) *stochastic* if the inputs are subject to fluctuations.

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

1. (a) *Numeric* 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 propositional 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., numeric and symbolic). Most computational models in design, however, have tended to be numeric 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) *constraint satisfaction*, 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.[1] Let the *continuous variables* be x and the *discrete variables* be y. The parameters which are normally specified as fixed values are represented by ϑ.

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

$$\begin{aligned} h(x, y, \vartheta) &= 0 \\ g(x, y, \vartheta) &\leq 0 \end{aligned} \tag{1}$$

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 in 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,

$$L(x, y, \vartheta) = \text{TRUE} \tag{2}$$

Finally, the design goal (or goals) can be expressed as the objective function $F(x, y, \vartheta)$. 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: given ϑ, possibly with a description of its fluctuations (e.g., distribution functions), find values for x and y in order to satisfy:

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

while possibly optimizing the goal or goals as given by the objective function $F(x, y, \vartheta)$.

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 variables 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 alternative topology configurations 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 [8], AMPL [9], 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 (decision variables or degrees of freedom) for a fixed ϑ, with which the states z are computed as an implicit function; that is,

$$h(u, z, y, \vartheta) = 0 \rightarrow z = z(u, y, \vartheta) \tag{4}$$

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

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

Finally, the sets Y and U may be implicitly defined. This is commonly the case in discrete design problems in which the set $Y = \{y_i, i = 1, 2, \ldots, n\}$ is generated

[1] We could refine our representation by partitioning the variables x into static v and dynamic variables dv/dt. For the sake of simplicity we do not introduce this distinction.

within a tree search with the recursion of the form,

$$y_{i+1} = f(y_i, y_{i-1}, \ldots, y_1\}, \quad u \in U(y_{i+1}) \tag{6}$$

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

1. Heuristic Design

Given explicit Y and U, and fixed ϑ, find y, u, z, such that as large a subset of

$$L(u, z, y, \vartheta) = \text{TRUE} \tag{7}$$

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

2. Superstructure Optimization

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

$$\begin{aligned} \min \quad & F(x, y, \vartheta) \\ \text{s.t.} \quad & h(x, y, \vartheta) = 0 \\ & g(x, y, \vartheta) < 0 \end{aligned} \tag{8}$$

One way to approach this problem is through mixed-integer nonlinear programming. Also, if the discrete variables y are fixed, the above gives rise to parametric design optimization problems that are commonly tackled with nonlinear programming techniques.

3. Implicit Generation of Designs

Given Y and U in implicit form as in (6) and for fixed ϑ, find a subset or all feasible combinations of the discrete variables y, with its corresponding u and z such that

$$\begin{aligned} h(u, z, y, \vartheta) &= 0 \\ g(u, z, y, \vartheta) &\leq 0 \\ L(u, z, y, \vartheta) &= \text{TRUE} \end{aligned} \tag{9}$$

is satisfied. One way to approach this problem is through a tree search based on hierarchical decomposition. The other is through the use of random search techniques on implicit representations of the design space of alternatives. In principle the problem in (9) can also be extended to perform optimization.

While the classification given above is rather general, it does serve as a unified framework for the representation of design problems. In addition, it also serves to identify several difficulties in modeling design problems. For instance, in the above models parameters ϑ are assumed to be fixed, although in practice a major issue is to find designs than can effectively cope with the uncertainty of these parameters. Also, the generic problem in (3) as well as some of its specific variations as in (9) 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. In the Appendix we present an overview of specific solution techniques. The next section provides a classification of models and solution techniques from the viewpoints of artificial intelligence and operations research.

AI vs OR Models

Having presented a classification of models from an engineering design perspective, we now address the more general question as to what makes a design model such as in (7), (8), or (9) 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, particularly in the context of heuristic and optimization solution methods. 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.

Numeric vs Symbolic Models

In the schema presented earlier, numeric models such as (8) have only constraints of the form $h(x, y, \vartheta) = 0$, $g(x, y, \vartheta) \leq 0$, and symbolic models such as have only constraints of the form $L(x, y, \vartheta) = \text{TRUE}$. A symbolic model might also be called a *logic model*, particularly

if the propositions in $L(x, y, \vartheta)$ belong to a formal logical language.

To avoid classifying all models as symbolic models, we can assume that the propositions in $L(x, y, \vartheta)$ do not assert quantitative relations $h(x, y, \vartheta) = 0$, $g(x, y, \vartheta) \leq 0$. 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 carry out the deduction.

Numeric 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 [10] and exploited by Boole [11], 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 1950s and 1960s, and the classical AI approach to problem solving has been to use symbolic computation. AI has more recently moved toward numeric models, as in constraint programming and neural networks, but it remains much more strongly identified with symbolic reasoning than OR.

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 [12]. 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. First, their relative simplicity is more conducive to understanding. If one can fit a structured model to a situation, one's grasp of it is likely to improve. A 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 [13].

In fact, it is tempting to define a structured model as one that has an "exploitable" structure—a structure that is conducive to fast solution—since otherwise it is unclear what kind of patterns count as structure. But an occasional structured model, 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.

The idea of structure discussed here should be distinguished from that of Newell and Simon [14]. A well-structured problem in their sense is one in which the objective and the constraints are relatively clear. Ill-structured problems are the sort that face us in everyday life: it is not even clear what the questions are. Because it is a salient characteristic of human intelligence to be able to deal with ill-structured problems, AI has historically been concerned to automate this process, whereas OR has always assumed that a human being clarifies and formulates the problem. This has led AI to an interest in models that are unstructured in our sense, because it is easier to build a solver of ill-structured problems when the models it formulates are not required to be highly structured.

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 W. V. Quine's assertion that first order predicate logic is adequate to formulate all of science [15].

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

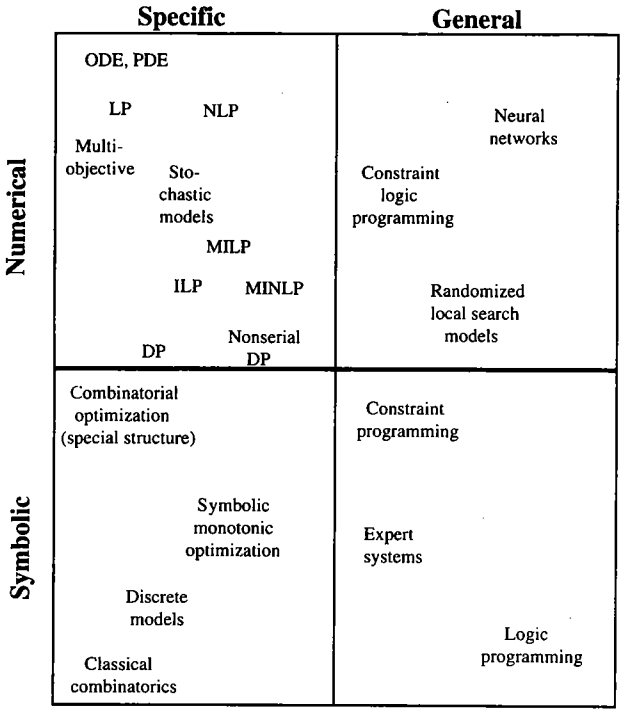


Fig. 1. Classification of models.

structured 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 [16]. 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, etc., are moving toward the upper right). They should occur mainly at low elevations on the vertical axis.

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 a 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 that neural networks alone 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 network! 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.) A neural network models a problem in some sense, 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 [17].

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.

Research Trends in the Integration of AI and OR Techniques

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 the first, Herb Simon's General Problem Solver [14]. But these models sacrifice explanatory power when they overlook structure. Solutions may be computationally elusive and of poor quality when obtained.

Both communities need somehow to borrow the strength of the other, and both seem to be aware of it. Over the last 15 years or so the OR community, in search of robustness, has built the sophisticated general-purpose mixed-integer solvers cited earlier. The AI community has been looking more closely at problem structure, particularly in the context of constraint programming [18]. It has also increased the rigor and depth of its analysis, for instance in the investigation of search strategies, where it is substantially ahead of OR [19–22]. But both communities have made limited progress toward achieving the advantage of the other, perhaps because both have remained largely within the bounds of their traditions without undertaking a true synthesis of the fields.

The specific research trends that move toward asynthesis of OR and AI seem to depend on the quadrant of Fig. 1 in which one begins. We therefore survey the quadrants. Some examples are provided in each case. Each example is a technique that is obtained by beginning with ideas associated with the quadrant under which it is classified and moving toward one of the other quadrants.

Specific Numeric Models

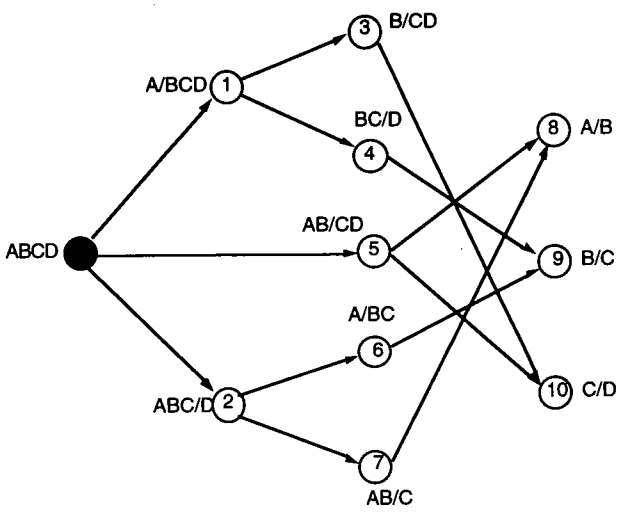
Numeric models have been pushed in directions of greater generality in at least two ways: (a) by using them to model (in the mirroring sense) the problem solving process, rather than the problem itself, and (b) by making them more symbolic.

Strategy (a) permits the models to remain structured, because they reflect the structure of the process, as noted earlier. 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 [17]. Simulated annealing algorithms [23, 24] and some neural networks model nature's way of minimizing energy. Tabu search [25, 26] models a search process with short-term memory; the early literature even suggested seven as the ideal length of a tabu list, because human short-term memory generally has room for seven 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 [27] are being invented, such social problem-solving processes as asynchronous teams [28] and ant colonies [29] are being investigated, and so on.

Another route to greater generality is to introduce symbolic elements into or find symbolic analogs of numeric methods:

- *Logic-Based Methods for Optimization.* Logic, particularly propositional logic, is useful in the formulation and solution of discrete and mixed continuous/discrete optimization problems. In fact one can develop logic-based analogs of many of the concepts used in optimization, such as branch-and-bound, cutting planes, facet-defining cuts, duality, Benders decomposition, etc. [30–33]. These parallels allow one to apply some of the structure-exploiting ideas of classical mathematical programming in a broader context.
- *Mixed Logical/Linear and Nonlinear Programming (MLLP and MLNLP).* Raman and Grossman [34] have proposed methods for the symbolic integration of logic relations between potential units in a design superstructure to aid the branch and bound search in MILP synthesis models. The objective is to reduce the number of nodes that must be enumerated by using logic inference to decide whether additional variables can be fixed at each node. This solution approach illustrates how a numeric model for superstructure optimization such as in Eq. (8), can move in the symbolic direction by incorporating logic information as in Eq. (7). This allows one to exploit the problem structure in a way that is not possible in a purely numeric setting, and without compromising optimality.

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 propositional logic (see also [33]). When heat integration is considered as part of the synthesis of the separation system 100 binary variables are needed—10 to model the existence of distillation columns, and 90 for the potential heat exchanges between condensers and reboilers. The algorithms, which can process the logic in either conjunctive



(a)

$Y1 \Rightarrow Y4 \vee Y5$	$Y6 \Rightarrow Y3 \wedge Y9$
$Y2 \Rightarrow Y8 \wedge Y10$	$Y7 \Rightarrow Y3 \wedge Y8$
$Y3 \Rightarrow Y6 \vee Y7$	$Y8 \Rightarrow Y2 \vee Y7$
$Y4 \Rightarrow Y1 \wedge Y10$	$Y9 \Rightarrow Y5 \vee Y6$
$Y5 \Rightarrow Y1 \wedge Y9$	$Y10 \Rightarrow Y2 \vee Y4$

(b)

Fig. 2. Superstructure for 4-component separation sequence: (a) superstructure, (b) logic relations.

normal form (CNF) or in disjunctive normal form (DNF), were automated using the simplex based code OSL [35] within the GAMS [8] modeling environment. OSL was not able to solve the problem even after 100,000 nodes and after more than 30 min on the IBM RISC/6000. In contrast, the DNF based approach solves the problem to rigorous optimality in only 20 nodes with OSL, and in less than 3 s! This illustrates how the integration of symbolic logic in numeric optimization models can have a great impact.

- *Qualitative Optimization.* Williams and Cagan [6] introduced a qualitative abstraction of the Karush–Kuhn–Tucker (KKT) conditions of optimality [36, 37] called the qualitative KKT (QKKT) conditions. By using a hybrid algebra combining signs and reals [38], QKKT uses signs (direction of monotonicities) of the objective function and constraints to determine whether a point can be stationary (called *qstationary*) or whether a point is nonstationary. By using a square bracket to denote "sign",[2] QKKT states that a feasible point x^* is stationary only if:

$$[\nabla f(x^*)] + [\lambda]^T[\nabla h(x^*)] + [\mu]^T[\nabla g(x^*)] \supseteq 0^T, \quad \text{(QKKT1)}$$

subject to

$$[\mu]^T[g(x^*)] = 0^T, \quad \text{and} \quad \text{(QKKT2)}$$

$$[\mu_i] \neq \hat{-} \quad \text{(QKKT3)}$$

where f is the objective function, x are the variables, and μ and λ are the inequality and equality Lagrange multipliers, respectively.

KKT says that to be stationary there must exist a weighted sum ($\vec{w}$) of ∇g and ∇h that exactly cancels ∇f. QKKT says a point is *nonstationary* unless each component of $\vec{w}$ lies in the opposite directions from that which contains ∇f. QKKT1 results in a set of equations (one for each variable) consisting of the signs of the Lagrange multipliers and other sign terms. QKKT, by combining AI and optimization, gives a much simpler condition than KKT to rule out nonstationary points. By abstracting the quantitative KKT condition to the qualitative QKKT condition and using symbolic reasoning techniques, a potentially powerful merging between the AI and optimization fields emerges which moves a technique from the specific numerical (upper left) quadrant of Fig. 1 to the specific symbolic (lower left) quadrant; the same is true of the following two techniques.

- *Activity Analysis.* Williams and Cagan [6] developed a symbolic monotonic optimization technique called *activity analysis* that manipulates QKKT. The approach partitions a design space by removing the largest subspaces which are provably nonstationary, resulting in smaller spaces which satisfy the QKKT conditions. These qualitatively stationary spaces (called qstationary spaces) correspond to smaller optimization problems, at least one of which contains the true optimum.
- *Monotonicity Analysis.* A different symbolic monotonic optimization method called *monotonicity analysis* was developed by Papalambros and Wilde [39, 40]; and, Wilde [41] as a pre-optimization technique to determine if an optimization problem is well-bounded prior to numerical optimization. The technique is based on two rules:

Rule 1. If 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 2. If a 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 (i.e., any constraints monotonic w.r.t. that variable must be inactive or irrelevant).

Choy and Agogino [3] and Agogino and Almgren [1] used monotonicity analysis as the basis of a symbolic reasoning system. Note that the monotonicity rules can be derived from QKKT.

Specific Symbolic Models

There are two related classes of problems in this quadrant: (a) problems from classical combinatorics (e.g., stable marriage problem, matching, etc.), as well as some combinatorial problems associated with OR (set covering, clique partition, etc., discussed in Lawler [42]); (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. [12, 43]. 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", 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 branch-and-bound. These approaches have been applied most often in mathematical programming; decomposition, for instance, is used in the methods of Dantzig–Wolfe, Benders, and Lagrangian relaxation. A possible research direction is to try to apply them to nonnumeric combinatorial problems.

General Numeric Models

Several strategies are available to move general, quantitative models in the direction of greater structure.

- *Structured Heuristics.* One strategy is to design heuristic methods that focus on the structure of the problem rather than the problem-solving device (just the opposite of the strategy described earlier). This is in effect what is typically done when generalized local or randomized search strategies are applied to a specific problem. Various search spaces and parameter settings are considered, often in a trial-and-error fashion, until the algorithm works well on the problem at hand. This can be done more systematically if there is some knowledge of the effect of problem structure on the behavior of heuristic methods. For example, Sorkin [44] has studied the influence of search space characteristics on the behavior of simulated annealing. Although deductive analysis does not often yield results of this kind, systematic empirical study can be useful [45–47]. For instance, empirical work helped to establish connections between the behavior of simulating annealing algorithms and certain entropy properties of the Markov chains they define [48].
- *Exploiting the Structure of Learning Problems.* Learning problems, a central concern of AI, can themselves be highly structured and therefore suited for optimization models. For instance, partially observed Markov decision models have been applied to learning problems, particularly in the context of neural networks [49, 50]. The optimal calibration of a neural network, for a given training set, is itself also a highly structured problem that can be solved with combinatorial optimization techniques [51].
- *Design Space Expansion.* Another strategy is to combine a heuristic algorithm with a structured symbolic approach, mapping from the upper right to the lower left quadrants of Fig. 1. The 1st-PRINCE methodology [52–56] combines the symbolic optimization technique of monotonicity analysis with heuristics to manipulate the mathematical representation of the optimization problem. The technique uses monotonicity analysis to identify sets of active constraints which model optimally directed designs, and determine how to expand the design space to introduce the new variables or features. The expansion techniques involve the recognition of dimensional variables and their expansion into serial designs (DVE), the recognition of input variables and their expansion into parallel designs (IVE), and induction techniques that determine unconditional activity and inactivity of constraints across expansions. An application of 1stPRINCE to the design of a chemical reactor to maximize a first-order chemical reaction, subject to

[2] A sign is one of four values, namely $\hat{+}$ if the quantity is positive, $\hat{-}$ if the quantity is negative, 0 if the quantity is zero, and $\hat{?}$ if the quantity is unknown ($[x]$ for $x > 0$ is $\hat{+}$, while $[x - y]$ is $\hat{?}$ unless additional information is known). Signs are added based on sign algebra (e.g., $[3] + [4] = \hat{+}$; for $x > 0$, $[x] + [3] = \hat{+}$; for x unknown, $[x] + [3] = \hat{?}$).

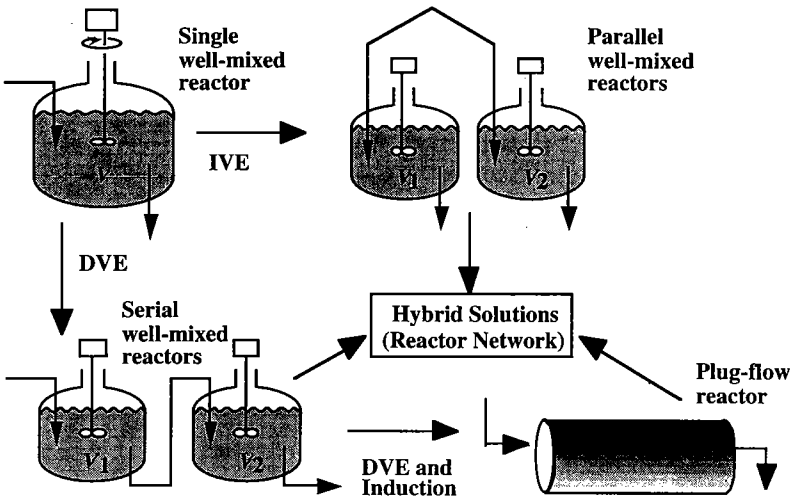


Fig. 3. Application of 1stPRINCE to chemical reactor design.

a maximum volume constraint, is shown in Fig. 3, where a *well-mixed reactor* is expanded into various preferred designs including a *plug-flow reactor*. See [55, 56] for details.

- *Constraint Logic Programming.* Still another strategy is to exploit numeric structure in constraint logic programming, which is a form of logic programming with numeric elements [57]. Constraint logic programming adds specifically numeric predicates to a logic programming language such as Prolog, which permits the use of linear inequality constraints. A typical problem is to answer such a query as, "for some x_1, x_2, $x_1 \leq x_2$," in a logic program that contains a linear constraint set $Ax \leq b$. A response to such a query would be a description of possible values of x_1, x_2, consistent with $Ax \leq b$ and $x_1 \leq x_2$. The problem is solved as a polyhedral projection problem. There is much potential for the use of linear and nonlinear programming technology here.
- *Shape Annealing.* The *shape annealing* technique introduced by Cagan and Mitchell [2] is a different example of how optimization can generate design topologies by breaking away from local minima. Here a design problem is again formulated as an optimization problem. However, the problem knowledge is modeled as a *shape grammar* with the properties of shapes described by Stiny [58]. Concepts of simulated annealing are used to create a technique which generates optimally directed solution shapes. One application of shape annealing for structural design can be found in Reddy and Cagan [59] where the technique is used in conjunction with finite element analysis to generate optimally directed topologies and optimal shapes of the truss structures. Figure 4 shows two structures

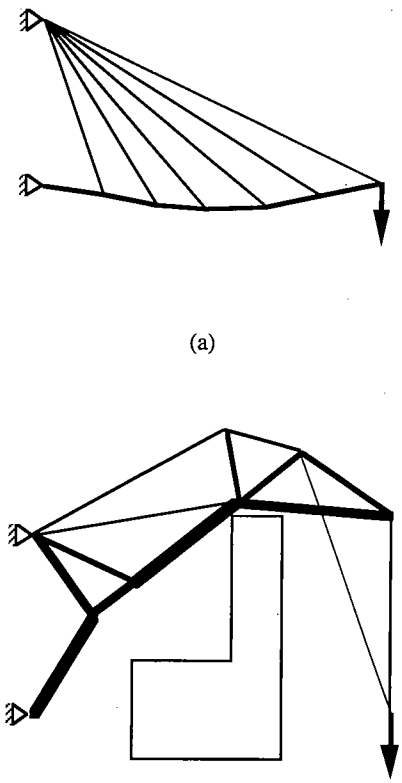


Fig. 4. Truss structures generated by shape annealing (from Reddy and Cagan [59]).

generated from shape annealing which are constrained by stress and Euler buckling criteria; Fig. 4(b) also includes a geometric obstacle. This general method illustrates a combination that moves both

toward a symbolic formulation with expert systems/logic models (using shape grammars) and randomized local search models (through simulated annealing), with specific numeric models to evaluate the design progress, merging the general numerical and symbolic approaches on the right of Fig. 1.

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.

One way to make a numeric model more symbolic is to mix symbolic constraints with numeric ones. This is done in constraint programming, which builds on constraint satisfaction techniques designed primarily for symbolic constraints [60, 61] but incorporates numerical equations and inequalities as well. For example, constraint propagation ideas have been specialized to obtain interval methods for finding all roots of equations and globally optimal solutions of nonlinear programming problems [62–64].

Constraint programming is not quite the same as constraint logic programming. The latter works within the framework of a formal logic model, whereas constraint programming need not. Technically, constraint logic programming extends the unification step of inference algorithms to encompass general constraint solving.

A number of constraint programming packages have been developed, particularly in Europe. These include CHIP [65, 66], CHARME [67–69], ILOG Solver [70, 71], PECOS, CHLEO, ConstraintLisp and other packages [72, 73]. Constraint programming is less accepted in the USA and until recently was 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. But mixed logical/numerical constraint sets can be systematically analyzed in the way that numerical constraint sets have been.

Concluding Remarks

This paper provides a conceptual overview of the problem of combining artificial intelligence and optimization, and motivates the need for and suggests 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 exists; future work must continue to merge and expand the focus of combined AI and optimization problem solving.

Acknowledgements

The authors would like to acknowledge financial support from the Engineering Design Research Center, an NSF Engineering Research Center at Carnegie Mellon University.

References

1. Agogino AM, Almgren AS. Techniques for integrating qualitative reasoning and symbolic computation in engineering optimization. Engineering Optimization 1987; 12:117–135
2. Cagan J, Mitchell WJ. Optimally directed shape generation by shape annealing. Environment and Planning B, 1993; 20:5–12
3. Choy JK, Agogino AM. SYMON: automated SYMbolic mONotoniciy analysis system for qualitative design optimization. In: Proceedings of ASME 1986 international computers in engineering conference, Chicago. 1986. pp 305–310
4. Cohen RM, May JH. Incorporating OR into an AI design environment to facility planning, working paper number 91–08. Boston University, School of Management 1991
5. Kusiak A. Process planning: a knowledge-based and optimization perspective. IEEE Transactions on Robotics and Automation 1991; 7:257–266
6. Williams BC, Cagan J. Activity analysis: simplifying optimal design problems through qualitative partitioning. Engineering Optimization (in press)
7. Fenves SJ, Grossmann IE. An interdisciplinary course in engineering synthesis. Research in Engineering Design 1992; 3:223–231
8. Brooke A, Kendrick D, Meeraus A. A GAMS user's guide. Scientific Press, Palo Alto, CA 1988
9. Fourer R, Gay DM, Kernighan BW. A modeling language for mathematical programming. Management Science 1990; 36:519–554
10. Leibniz GW. In: Gerhardt CI (ed.). Die philosophische Schriften von G. W. Leibniz, v. 7 Berlin 1890
11. Boole G. The mathematical analysis of logic. Oxford University Press, Oxford 1948
12. Nemhauser GL, Wolsey LA. Integer and combinatorial optimization. John Wiley, New York 1988
13. Bazaara MS, Jarvis JJ, Sherali HD. Linear programming and network flows. John Wiley, New York 1990
14. Newell A, Simon H. Human problem solving. Prentice Hall, Englewood Cliffs, NJ 1972
15. Quine WV. From a logical point of view: nine logico-philosophical essays. Harvard University Press, Cambridge, MA 1961
16. Rumelhart DE, Hinton GE, Williams RJ. Learning internal representation. In: Rumelhart DE, McClelland JL (eds). Parallel distributed processing: exploration in the microstructure of

cognition. vol. 1: Foundations. MIT Press, Cambridge, MA. 1986. pp 318–362
17. Goldberg DE. Genetic algorithms in search, optimization and machine learning. Addison-Wesley, Reading, MA 1989
18. Freuder EC. Exploiting structure in constraint satisfaction problems. In: Mayoh B, Tyugu E, Penjam J (eds). Constraint programming. Springer-Verlag, Berlin 1993. pp 50–74
19. Nadel B. Constraint satisfaction algorithms. Computational Intelligence 1989; 5:188–224
20. Jiang Y, Richards T, Richards B. No-good backmarking with min-conflict repair in constraint satisfaction and optimization. PPCP94, 1994. pp 36–47
21. Ginsberg ML. Dynamic backtracking. Journal of Artificial Intelligence Research 1993; 1:127–162
22. Ginsberg ML McAllester DA. GSAT and dynamic backtracking. PPCP94 1994. pp 216–225
23. Kirkpatrick S, Gelatt CD Jr, Vecchi MP. Optimization by simulated annealing. Science 1983; 220:671–679
24. Aarts EHL, Korst J. Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing. John Wiley, New York 1989
25. Glover F. Tabu search—Part I ORSA Journal on Computing 1989; 1:190–206
26. Glover F. Tabu search—Part II. ORSA Journal on Computing 1990; 2:4–32
27. Glover F. Genetic algorithms and scatter search: unsuspected potentials. Statistics and Computing 1994; 4:131–140
28. Talukdar S, DeSouza P, Murthy S. Organizations for computer-based agents. International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications 1993; 1:75–87
29. Colorni A, Dorigo M, Maniezzo V. Distributed optimization by ant colonies. In: Varela F, Bourgine P (eds). Proceedings of ECAL91—European conference on artificial life. Elsevier, Amsterdam, 1994. pp 134–142
30. Hooker JN. Logic-based methods for optimization. In: Borning A (ed). Principles and practice of constraint programming. Lecture Notes in Computer Science 1994; 874:336–349
31. Hooker JN. Logic-based Benders decomposition. GSIA, Carnegie Mellon University, available at http://www.gsia.cmu.edu/afs/andrew/gsia/jh38/jnh.html. 1995
32. Hooker JN. Inference duality as a basis for sensitivity analysis GSIA, Carnegie Mellon University, available at http://www.gsia.cmu.edu/afs/andrew/gsia/jh38/jnh.html. 1996
33. Hooker JN, Yan H, Grossmann IE, Raman R. Logic cuts for processing networks with fixed charges. Computers and Operations Research 1994; 21:265–279
34. Raman R, Grossmann IE. Symbolic integration of logic in mixed integer linear programming techniques for process synthesis. Computers and Chemical Engineering 1993; 17:909
35. IBM. Optimization subroutine library. Guide and reference-release 2. Kingston, New York 1991
36. Karush W. Minima of functions of several variables with inequalities as side conditions. MS Thesis, Department of Mathematics, University of Chicago, Chicago, IL. 1939
37. Kuhn HW, Tucker AW. Nonlinear programming. In: Neyman J (ed.). Proceedings of the second Berkeley symposium on mathematical statistics and probability, University of California Press, Berkeley, CA 1951
38. Williams BC. A theory of interactions: unifying qualitative and quantitative algebraic reasoning. Artificial Intelligence Special Volume on Qualitative Reasoning About Physical Systems II 1991; 51:39–94
39. Papalambros P, Wilde DJ. Global non-iterative design optimization using monotonicity analysis. Transactions of the ASME, Journal of Mechanical Design 1979; 101(4): 645–649
40. Papalambros P, Wilde DJ. Principles of optimal design. Cambridge University Press, Cambridge 1988
41. Wilde DJ. Monotonicity and dominance in optimal hydraulic cylinder design. Trans ASME, Journal of Engineering for Industry 1975; 94(4):1390–1394
42. Lawler E. Combinatorial optimization: networks and matroids. Holt, Rinehart and Winston, New York 1976
43. Korte B, Lovasz L. Greedoids—a structural framework for the greedy algorithm. In: Pulleyblank WR (ed.). Progress in combinatorial optimization. Academic Press, New York 1984 pp 221–244
44. Sorkin GB. Efficient simulated annealing on fractal energy landscapes. Algorithmica 1991; 6:367–418
45. Hooker JN. Needed: an empirical science of algorithms. Operations Research 1994; 42:201–212
46. McGeoch C. Toward an experimental method for algorithm simulation. INFORMS Journal on Computing 1996; 8:1–28
47. Hooker JN. Testing heuristics: we have it all wrong. Journal of Heuristics 1996; 1:33–42
48. Fleischer M, Jacobson SH. The entropy of inhomogeneous Markov chains with application to simulated annealing. Operations Research Dept., Case Western Reserve University, Cleveland OH 1992
49. Dean T, Kaelbling LP, Kirman S, Nicholson A. Planning under time constraints in stochastic domains. Artificial Intelligence 1995; 76:35–74
50. Kaelbling LP, Littman ML, Moore AW. Reinforcement learning: a survey. Journal of AI Research (in press)
51. Chandru V, Vidyasagar M, Vinay V. Tractable theories for the synthesis of neural networks. In: Proceedings, 5th INFORMS conference on computer science and operations research: recent advances in the interface. Kluwer Academic Publishers, Dordrecht 1996
52. Cagan J, Agogino AM. Innovative design of mechanical structures from first principles. Artificial intelligence in Engineering Design, Analysis, and Manufacturing 1987; 1(3):169–189
53. Cagan J, Agogino AM. Dimensional variable expansion—a formal approach to innovative design. Research in Engineering Design 1991; 3:75–85
54. Cagan J, Agogino AM. Inducing constraint activity in innovative design. Artificial Intelligence in Engineering Design, Analysis, and Manufacturing 1991; 5(1):47–61
55. Aelion V, Cagan J, Powers G. Inducing optimally directed innovative designs from chemical engineering first principles. Computers and Chemical Engineering 1991; 15(9):619–627
56. Aelion V, Cagan J, Powers G. Input variable expansion—an algorithmic design generation technique. Research in Engineering Design 1992; 4:101–113
57. Jaffar J, Lassez J-L. From unification to constraints. Logic Programming 87. Proceedings of the 6th Conference, Springer-Verlag 1987. pp 1–18
58. Stiny G. Introduction to shape and shape grammars. Environment and Planning B 1980; 7:343–351
59. Reddy G, Cagan J. An improved shape annealing algorithm for truss topology generation. ASME Journal of Mechanical Design 1995; 117(2(A)):315–321
60. Tsang E. Foundations of constraint satisfaction. Academic Press, London 1993
61. Van Hentenryck P. Constraint satisfaction in logic programming. MIT Press, Cambridge, MA 1989
62. Hansen P, Jaumard B, Mathon V. Constrained nonlinear 0-1 programming. ORSA Journal on Computing 1993; 5:97–119