# Constrained Heuristic Search

Mark S. Fox, Norman Sadeh and Can Baykan
Robotics Institute & Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

## Abstract

[1]

We propose a model of problem solving that provides both structure and focus to search. The model achieves this by combining constraint satisfaction with heuristic search. We introduce the concepts of topology and texture to characterize problem structure and areas to focus attention respectively. The resulting model reduces search complexity and provides a more principled explanation of the nature and power of heuristics in problem solving. We demonstrate the model of Constrained Heuristic Search in two domains: spatial planning and factory scheduling. In the former we demonstrate significant reductions in search.

## 1. Introduction

We propose a model of problem solving that provides both structure and focus to search in the problem space. The model achieves this by combining the process of constraint satisfaction (CSP) with heuristic search (HS). The resulting model both reduces search complexity and provides a explanation of the nature and power of heuristics in problem solving. Our model focuses on reasoning within a problem space, and can be viewed as being complementary to the Soar architecture [Laird, Newell & Rosenbloom 87].

Our problem solving model, called *Constrained Heuristic Search* (CHS), retains heuristic search's synthetic capabilities and extends it by adding the structural characteristics of constraint satisfaction techniques. In particular, our model adds to the definition of a problem space, composed of states, operators and an evaluation function, by refining a state to include:

1. Problem Topology: Provides a structural characterization of a problem.
2. Problem Textures: Provide measures of a problem topology that allows search to be focused in a way that reduces backtracking.
3. Problem Objective: Defines an objective function for rating alternative solutions that satisfy a goal description.

This model allows us to (1) view problem solving as constraint satisfaction, thus taking advantage of these techniques, (2), incorporate the synthetic capabilities of heuristic search, thus allowing the dynamic modification of the constraint model, and (3) extend constraint satisafaction to the larger class of optimization problems. In the following, problem topology and textures are defined, followed by examples of their use in the domains of spatial planning and factory scheduling.

## 2. Problem Topology

Our intent is to define a problem's topology so that search can be performed more efficiently; this is related to the notion of a problem being "well structured" [Simon 83].

Within the heuristic search model, a variety of techniques for structuring problems have been investigated. ABSTRIPS [Sacerdoti 74] demonstrated how hierarchical reformulation of the problem via omission of variables reduces search complexity, Hearsay-II [Erman et al 80], MOLGEN [Stefik 81], and OPIS [Smith, Fox & Ow 86] demonstrated how hierarchical reformulation via aggregation and abstraction reduces search complexity. ISIS [Fox 87] demonstrated how hierarchical reformulation via omission of constraints reduces search complexity. These structuring techniques are more engineering guidelines than formal characterizations.

On the other hand, constraint satisfaction research has begun to formalize the concept of well structured constraint graphs, but their techniques can be applied only to a narrow set of problems. Constraint satisfaction techniques, as described in [Mackworth 77, Haralick & Elliott 80, Freuder 82, Dechter & Pearl 87], approach problem solving

[1] This research hag been supported, in part, by the Defense Advance Projects Agency under contract #F30602-88-C-0001, and in part, by grants from McDonnell Aircraft Company, Boeing Computer Services, and Schlumberger Corp.

by constructing a constraint graph where nodes are variables with discrete domains and arcs are n-ary constraints among the values the variables may be assigned. Problem solving is performed by sequentially choosing a variable and a value to assign to it that satisfies all constraints incident upon it. Backtracking occurs when an assignment cannot be found. Research has gone into methods for structuring the network so that the amount of backtracking can be reduced. Arc-consistency is one such technique that achieves local consistency between groups of variables via the elimination of incompatible values [Montanari 74, Mackworth 77, Davis 87]. Width 1 networks that are arc consistent are backtrack free [Freuder 82].

Solving a CSP involves finding an assignment of values to the variables that satisfy a set of constraints. From a heuristic search perspective, the initial state contains all the variables and their domains and constraints, the operators select a variable and a value to assign it, and the evaluation function is composed of constraints, some distance metric and an objective function. The sequence of states generated in the problem space represent alternative orderings of variables and values to assign to them. Backtracking results in new braches in the search tree. *The important insight that we wish to draw from CSP research is that by manipulating the constraint graph, the ordering of variables and values can be optimized. That is, the constraint graph can be viewed as providing a structure for the problem.*

We define problem topology as a graph G, composed of vertices V and edges E:

$$V = N \cup C \cup S$$

where

$N$ is a set of variables $\{n_1, n_2, ..., n_m\}$

$C$ is a set of constraints $\{c_1, c_2, ..., c_n\}$

$S$ is a set of satisfiability specifications $\{s_1, s_2, ..., s_o\}$

Each variable in N may be a vector of variables whose domains may be finite/infinite and continuous/discrete. Constraints are n-ary predicates over variables vertices. A satisfiability specification vertex groups constraints into sets of type AND, OR, or XOR. An XOR satisfaction set denotes that only one constraint in the set must be satisified. Edges link constraint vertices to variable vertices, and satisfiability specifications to constraints.

We distinguish between two types of problem topologies:

> Definition 1: A *completely structured problem* is one in which all non-redundant vertices and edges are known a priori.

This is true of all CSP formulations.

> Definition 2: A *partially structured problem* is one in which not all non-redundant vertices and edges are known prior to problem solving.

This definition tends to be true of problems in which synthesis is performed resulting in new variables and constraints (e.g. the generation of new subgoals during the planning process).

Operators in CHS have many roles: refining the problem by adding new variable and constraint vertices, reducing the number of solutions by reducing the domains of variables (e.g., assigning a value to a variable vertex), or reformulating the problem by relaxing constraints or omitting constraints and/or variables.

Features of the problem topology are the types of variables and constraints (and their associated propagation algorithms). Davis [Davis 87] mentions two classes of what we view as topological features, namely the types of values the domain of a variable may contain, such as variables whose domains are discrete and finite (label and value inference), are intervals, have belief for each member (relaxation labelling), and are expressions (expression inference). The second classs of features focus on the types of constraints, such as constraints that are unary predicates, order relations, bounded differences (e.g. *x-y>c),* linear equations with unit (i.e. -1, 0, 1) coefficients, linear equalities and inequalities with arbitrary coefficients, boolean combinations of constraints, algebraic equations, and transcendental equations. Additionally, domains may or may not have preferences for values (e.g. preferences for due dates of a job).

What value do we derive from viewing a problem space state as a constraint graph? First, we have provided a more refined definition of a problem space state thereby reducing the looseness of its definition and allowing the definition of general measures of problem structure, i.e., textures. Second, properties can be proved about the nature of the problem, e.g., width-1 constraint networks that are arc consistent are backtrack free [Freuder 82]. Third, the process of problem reformulation can be viewed as transformations of problem topological primitives. A possible negative, is that the number of problem types that can be represented in the form of a constraint graph is limited. But this set is growing larger; in the factory scheduling example, we show how the representation can be extended to handle optimization. By adding the power of heuristic search, we believe that we can apply the model to a broader class of problems.

## 3. Problem Textures and Objective

Focus of attention in search is concerned with the ability of the search algorithm to opportunistically decide where the next decision is to be made [Erman et al 80]. In CHS, for search to be

well focused, that is to decide where in the problem topology an operator is to be applied, there must be features of the topology that differentiate one subgraph from another, and these features must be related to the goals of the problem. We have identified and are experimenting with seven such features that we call problem *textures* [Sadeh & Fox 88]. Below we define these textures for CHSs where all solutions are equally preferred, i.e., the Problem Objective rates all solutions to the constraints equally acceptable.

(Variable) Value Goodness: the probability that the assignment of that value to the variable leads to an overall solution to the CHS (i.e. to a fully consistent set of assignments). This texture is related to the value ordering heuristics [Haralick & Elliott 80] which look for the least constraining values. Value ordering heuristics are meant to reduce the chance of backtracking. In the case of discrete variables, the goodness of a value is the ratio of complete assignments that are solutions to the CHS and have that value for the variable over the total number of possible assignments.

Constraint Tightness: Constraint tightness refers to the contention between one constraint or a subset of constraints with all the other problem constraints. Consider a CHS A and a subset C of constraints in A. Let B be the CHS obtained by omitting C's constraints in A. The constraint tightness induced by C on A is defined as the probability that a solution to B is not a solution to A. In the case of discrete variables, this is the ratio of solutions to B that are not solutions to A over the total number of solutions to B.

Variable Tightness with respect to a set of constraints: Again consider a CHS A, a subset C of constraints, and the CHS B obtained by omitting C in A. A variable Vs tightness with respect to the set of constraints C is defined as the probability that the value of V in a solution to B does not violate C\ In the case of discrete variables, this is simply the ratio of solutions to B in which Vs value violates C (i.e. at least one of the constraints in C) over the total number of solutions to B.

Constraint Reliance: This measures the the importance of satisfying a particular constraint. Consider a constraint $c_i$. We defined CHS B as being CHS A - ($c_i$). Given that constraints can be disjunctively defined, the reliance of CHS A on a constraint $C_i$ is the probability that a solution to CHS B is not a solution to A. In the case of discrete variables, constraint reliance is defined as the ratio of the number of solutions to CHS B that are not a solution to CHS A to the number of solutions to CHS B. The larger the value, the greater the reliance the problem has on satisfying the particular constraint.

Variable Tightness: Consider a variable v in a CHS A. Let C be the set of constraints involving v and B be the CHS obtained by omitting *C* in A. Vs tightness with respect to C is simply called v's tightness. Hence the tightness of a variable is the probability that an assignment consistent with all the problem constraints that do not involve that variable does not result in a solution. Alternatively one can define *variable looseness* as the probability that an assignment that has been checked for consistency with all the problem constraints, except those involving that variable, results in a fully consistent assignment. Notice that if one uses a variable instantiation order where v is the last variable, v's tightness is the *backtracking probability.* Variable looseness/tightness can be identified with variable ordering heuristics [Haralick & Elliott 80, Freuder 82] which instantiate variables in order of decreasing tightness.

Variable Contention: It estimates the degree of contention that exists among a set of constraints in assigning a value to a variable. Given a CHS A, a set C of constraints incident at variable v, and CHS B = CHS A - C, one measure of contention is to take the ratio of the number of elements c' of the powerset of *C'* that do not have a solution to CHS B + c', to the total number of elements in the powerset of C' In essence, the more combinations of constraints in *C'* for which there is not a solution, the greater the contention.

Constraint Arity: the number of variables involved in a constraint or more generally in a group of constraints.

These textures generalize the notion of constraint satisfiability or looseness defined by [Nadel 86] and apply to both CHSs (and CSPs) with discrete and continuous variables. Notice that, unless one knows all the CHS's solutions, the textures that we have just defined have to be approximated. Textures may sometime be evaluated analytically [Sadeh & Fox 88]. A brute force method to evaluate any texture measure consists in the use of Monte Carlo techniques. Such techniques may however be very costly. In general, for a given CHS, some textures are easier to approximate than others, and some are also more useful than others. Usually the texture measures that contain the most information are also the ones that are the most difficult to evaluate. Hence there is a tradeoff. Each domain may have its own approximation for a texture measure.

We have extended these textures to take into account the Problem Objective where the objective is expressed as a sum of functions of one variable, using Bayesian probabilities to approximate the likelihood that a variable results in an optimal value [Sadeh & Fox 88].

Textures provide a more principled view of attention focusing. As such, they can explain the power of heuristic knowledge used in search. We have already mentioned variable and value ordering heuristics respectively based on variable looseness and value goodness. Another example is in factory

scheduling, where a useful heuristic is to schedule the bottleneck resource first. In our factory scheduling example we show that the concept of resource bottleneck analysis is motivated by constraint arity considerations and illustrates the concept of constraint tightness.

## 4. CHS Problem Solving Process

The CHS model of problem solving is a combination of constraint satisfaction and heuristic search. Search is performed in the problem space where each state contains a problem topology. The problem solving model we propose contains the following elements:

- An initial state is defined composed of a problem topology,
- Constraint propagation is performed within the state,
- Texture measures and the problem objective are evaluated for the state's topology,
- Operators are matched against the state's topology, and
- A variable node/operator pair is selected and the operator is applied.

The application of an operator results in either adding structure to the topology, further restricting the domain of a variable, or reformulating the problem (e.g., relaxation).

The next two sections demonstrate the application of the CHS model to the problems of spatial planning and factory scheduling.

## 5. Spatial Planning

WRIGHT [Baykan & Fox 89] is a spatial planning system that generates two-dimensional layouts consisting of configurations of rectangles. It is currently being applied to the design of kitchens. WRIGHT formulates space planning as a hierarchical CHS. Each level of the hierarchy consists of a set of variables and constraints. Knowledge of the design domain is represented by a class hierarchy of prototype design units and constraints on them expressing desired spatial relations and limits on dimensions, areas and distances. Inputs for generating a layout are an existing configuration which may be an empty space with dimensions specified as a range of values, and a set of design unit instances to be located and/or dimensioned. At the second level, the variables are the design unit instances. At the third level, variables are the locations of horizontal and vertical lines, dimensions, areas and orientations of the rectangular design unit instances. Constraints are unary, binary or ternary algebraic relations. The constraint graph uses satisfiability specifications to denote conjuncts and disjuncts of constraints. Spatial relations between design units are mapped onto algebraic relations between their component variables. Prototype design units that have more than one instance may cause disjuncts in the constraint graph between levels 1 and 2, and spatial constraints that can be satisfied in different ways introduce disjuncts to the constraint graph between levels 2 and 3. The top level of the graph is in *conjoint normal form.* A new state is generated for satisfying each disjunct of the selected constraint. WRIGHT finds all significantly different solutions that are pareto optimal.

The spatial planning problem demonstrates the following characteristics of the CHS problem solving model:

- the problem topology is hierarchical, nodes, i.e., design units are composed of multiple variables whose domains are continuous intervals or discrete numbers,
- constraints define spatial limitations using size bounds and relative positioning,
- texture measures are used to identify the constraint to satisfy,
- the problem objective is used to rate alterantive states, and
- operators generate new states by either assinging values to variables or further constraining or relaxing the problem.

WRIGHT uses three texture measures for selecting constraints. The first measure is from a variable perspective, and the last two are from a constraint perspective. The texture measures are:

- Variable Tightness: is approximated by the number of remaining conjunctive constraints on each design unit instance.
- Constraint Reliance: is approximated by *1/number of disjuncts.* Constraints with fewer disjunctive cases remaining are selected. If the number of disjuncts is 1, the constraint is satisfied without needing to generate a new state for each disjunct.
- Constraint Tightness: is approximated by the reduction in the domains of continuous variables involved, as a result of satisfying the constraint. Constraints that result in large reductions are favored. Types of algebraic relations that will be added to the CSP due to each of the competing constraints are also taken into account.

Texture measures are applied lexicographically. Active constraints are assigned ratings with respect to a metric, and constraints with lower values are eliminated from contention. If there is a single constraint with the best measure, it is used. If more than one constraint remains, the next texture measure is applied, or a constraint is selected randomly.

Figure 5-1 shows the number of search states required for finding all solutions to five kitchen layout problems, under different combinations of texture measures. The combinations tested are: *method 0:*

selects a constraint at random, *method 1:* uses variable tightness, *method 2:* uses constraint reliance, *method 3:* combines variable tightness and constraint reliance, and *method 4:* uses all texture measures defined above. When a combination of more than one measure is used, they are applied in the order they are defined above. Each measure eliminates some constraints from consideration. If more than one constraint remains after applying the texture measure(s), specified by the method, a constraint is selected at random. The number of states given for each problem-method combination is the average of three runs. In the second problem, method 4 reduces search by more than 80% compared to method 0, and in the third problem by 35%.

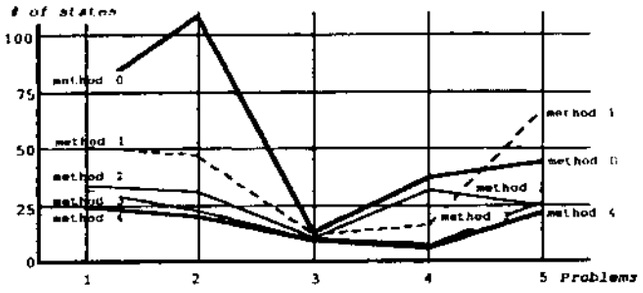


**Figure 5-1: Effectiveness of texture measures in reducing search**

In order to compare our approach to spatial planning with generate and test, WRIGHT is compared with two space planning programs, DPS [Pfeffercorn 71] and LOOS [Flemming 851. Their performances are compared in terms of the number of states generated when finding the first solution and when finding all 24 solutions. The problem used in the comparison is arranging six fixed size blocks (DPS can only deal with polygons of fixed size) in a rectangular envelope, such that no blocks overlap. Exactly the same objects and constraints can be used by all three programs due to the simplicity of the problem. WRIGHT finds the first solution using 80% fewer states compared to both DPS and LOOS, and finds all solutions using 50% fewer states compared to LOOS (no data available for DPS).

## 6. Factory Scheduling

Factory scheduling involves the assignment of start times and resources to a set of activities. Each activity belongs to an order (i.e. job). Activities within the same order are subject to precedence constraints as specified by a process plan. Additionally no two activities are allowed to use the same resource at the same time (we assume resources of unary capacity). Each order has a release date and a latest acceptable completion date (which may be later than the due date), that can be used to determine an earliest start time and a latest start time for each activity in the order. Additionally each activity may require one or several resources, for each of which there may be several alternatives. For each activity, utility functions map each possible start time and each possible resource alternative onto a utility value (preference). The sum of these utilities over all the activities to be scheduled defines an objective function to be maximized. Th ese utilities [Fox 87, Sadeh & Fox 881 arise from organizational goals such as reducing order tardiness, reducing order flowtime, using accurate machines, performing some activities during a specific shift, etc.

We view the scheduling problem as an optimization version of the CHS model, where each activity is an aggregate variable whose values are reservations. A reservation consists of a start time and a set of resources to be allocated to the activity. Each activity constitutes a variable vertex in the problem topology. Activity precedence constraints are binary constraints represented by constraint vertices connected to two activity variable vertices. A capacity constraint vertex is associated to each physical resource of the domain and connected to all the variablevertices representing activities that can possibly use the resource. Each capacity constraint ensures that the corresponding resource will not be allocated to more than one activity at any given time. Accordingly we distinguish between two types of constraint interactions:

- the *intra-order* interactions defined by the precedence constraint vertices between activities belonging to a same order, and
- the *inter-order* interactions induced by the capacity constraint vertices between activities contending for a same resource.

Both types of interactions contribute to the tightness of each activity.

Search in our CHS model begins with a single state where all activities still have to be scheduled and all resources are available Scheduling an activity in a state with a reservation results in the creation of a new search state where new constraints resulting from the assignment of the reservation to to the activity are propagated. The propagation consists in udpating the domain of start times and resources that remain possible for each unscheduled activity [Sadeh & Fox 88]. If an inconsistency is detected the system backtracks. Next the scheduler computes a tightness measure for each unscheduled activity. The activity with the highest tightness measure is selected to be scheduled next. A value goodness measure is computed to select the first reservation to be tried for that activity (among the reservations that are still possible). In this paper we assume that the goodness of a reservation is simply given by its combined utility, i.e. the sum of its start time utility and the

utilities of each of the resources selected for the activity in that reservation (more sophisticated measures of value goodness are discussed in [Sadeh & Fox 89] along with experimental results). The process goes on until all activities have been scheduled or until all search states have been visited.

In order to evaluate the performance of our measure of activity tightness, we designed a set of 45 scheduling problems. The problems contained between 3 and 5 orders, for a total number of activities ranging between 10 and 20. Each activity required only one resource, for which there could be alternatives. In some examples the resource alternatives all had equal preferences, while in others they had different preferences. The scheduling problems were built to reflect a variety of demand profiles: localized bottlenecks at the beginning, middle, and end of the problem span, global bottlenecks spanning the whole duration of the scheduling problems, and auxiliary bottlenecks were all considered. Three different types of start time utility functions were allowed: all start times (between the earliest and latest start times) are equally preferred , late start times are preferred, and triangular start utility functions with a peak corresponding to the due date (minus the duration of the activity). Triangular utility functions were only assigned to the last activities of some orders. Time was discretized and a granularity equal to the third of the smallest activity duration was used.

The experiments involved two variants of the same scheduler. In one variant the scheduler picked the next activity to be scheduled according to the tightness measure described in the previous subsection, while in the other variant the next activity to be scheduled was picked randomly among the remaining activities to schedule.

The performances of the two variants of the scheduler were measured along two dimensions: *search efficiency* (i.e. number of activities to schedule over number of search states generated) and *global utility* of the solution as defined by a normalized objective function. The normalized objective functions were built so that the best possible schedules that could be built without checking for constraint violation would have a global value of 1. In general the best feasible schedule had a global utility that was smaller than 1, hence the measures of global utility should only be used to compare the relative performance of the two variants, rather than assess their absolute performance. In the best case search was performed without backtracking, thereby resulting in an efficiency of 1.

The table in Figure 6-1 reports the average search efficiencies and schedule values obtained for the two variants of the scheduler. Standard deviations are provided between parentheses. RAND denotes the random variable ordering variant, and TIGHT the variant using our measure of variable criticality. The search was stopped when it required more than 50 search states. For RAND, this cutoff rule had to be used in 18 of the 45 experiments. It did not have to be used for TIGHT. The average search efficiency of RAND is therefore even worse than 0.47. The average schedule values listed in the table correspond to the 27 experiments for which the RAND variant found a schedule in less than 50 search states.

The results clearly indicate the increase in search efficiency obtained by using our measure of variable tightness. The micro behavior induced by this texture measure resulted in a macro behavior that focused search attention on bottleneck resources, when appropriate. Additionally they show that the schedules obtained when using the tightness measure are also slightly better than the ones obtained with the RAND variant.

| Experimental Results | | |
|---|---|---|
| | RAND | TIGHT |
| Search Efficiency | < 0.47 (> 0.22) | 0.96 (0.05) |
| Schedule Value | 0.66 (0.05) | 0.69 (0.05) |

**Figure 6-1: Average search efficiencies.**

## 7. Conclusion

The creation of general models for problem solving has been of continuing interest to Artificial Intelligence researchers. The process is evolutionary, elaborating and/or creating new search methods and richer representations of knowledge. The SOAR architecture, for example, combines both the problem space and production system models and extends them with universal subgoaling and chunking, thus achieving a model with powerful learning capabilities. But within this model, there are two aspects of the problem space that remain ill-defined: the notion of structure and means of focusing attention within a structure. Our model, Constrained Heuristic Search, extends the problem space model in these directions. Problem topology provides a definition of structure in the form of a constraint graph. Problem textures provide a probablistic, graph theoretic definition of the complexity and importance of decisions within a topology. Problem objective defines an objective function that rates states that satisfy their constraints. Together they enable the problem solver to direct search more economically towards a higher quality solution.

We demonstrated the model in two domains: spatial planning and factory scheduling. In spatial planning, we demonstrated that HCS is more efficient in finding solutions than other comparable

systems. In factory scheduling, we generalized constraint graphs to account for preferential temporal constraints, making it possible to represent the general job shop scheduling problem for the first time. Texture measures, based upon these preferences, enabled the scheduler to opportunistically select the next best decision to make. They also provided an explanation of the power of domain heuristics like bottleneck analysis.

Currently, we are exploring additional constraint representations and propagation techniques in order to represent a broader set of problems. We also hope to explore the creation of methods for automatically reformulating problem space topology.

## References


[Baykan & Fox 89] Baykan, C, and Fox, M.S., "Constraint Satisfaction Techniques for Spatial Planning,'' *Third Eurographics Workshop on Intelligent CAD Systems,* 1989.

[Davis 87] Davis, E., "Constraint Propagation with Interval Labels," *Artificial Intelligence,* Vol. 32,1987, pp. 281-331.

[Dechter & Pearl 87] Dechter, R. and Pearl, J., "Network-based Heuristics for Constraint-Satisfaction Problems," *Artificial Intelligence,* Vol. 34, No. 1,1987, pp. 1-38.

[Erman et al 80] Erman, L.D., F. Hayes-Roth, V.R. Lesser, & D. Raj Reddy, "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," *ACM Computing Surveys, Vol.* 12,1980, pp. 213-253.

[Flemming 85] Flemming, U., "On the representation and generation of loosely packed arrangements of rectangles," Tech. report DRC-48-05-85, Carnegie-Mellon University Design Research Center, 1985.

[Fox 87] Fox, M.S., *Constraint-Directed Search: A Case Study of Job-Shop Scheduling,* Morgan Kaufmann Publishers, Inc., 1987.

[Freuder 82] Freuder, E.C., "A Sufficient Condition for Backtrack-free Search," *Journal of the ACM,* Vol. 29, No. 1,1982, pp. 24-32.

[Haralick & Elliott 80] Haralick, R.M., and Elliott, G.L., "Increasing Tree Search Efficiency for Constraint Satisfaction Problems," *Artificial Intelligence,* Vol. 14, No. 3,1980, pp. 263-313.

[Laird, Newell & Rosenbloom 87] Laird, J.E., Newell, A., Rosenbloom, P.S.„ "SOAR: An Architecture for General Intelligence," *Artificial Intelligence,* Vol. 33, No. 1, September 1987, pp. 1-64.

fMackworth 77] Mackworth, A.K., "Consistency in Networks of Relations," *Artificial Intelligence,* Vol. 8, No. 1,1977, pp. 99-118.

[Montanari 74] Montanari, U., "Networks of Constraints," *Proc. IFIP Congress,* 1974, pp. 727-732.

[Nadel 86] Nadel, B.A., "The General Consistent Labeling (or Constraint Satisfaction) Problem," Tech. report DCS-TR-170, Department of Computer Science, Laboratory for Computer Research, Rutgers University, 1986.

[Pfeffercorn 71] Pfeffercorn, C, *Computer Design of Equipment Layouts Using the Design Problem Solver,* PhD dissertation, Carnegie-Mellon University, May 1971.

[Sacerdoti 74] Sacerdoti, E.D., "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence,* Vol. 5, No. 2,1974, pp. 115-135.

[Sadeh & Fox 88] Sadeh, N., and Fox, M.S., "Preference Propagation in Temporal Constraints Graphs," Tech. report, Intelligent Systems Laboratory ,The Robotics Institute, 1988.

[Sadeh & Fox 89] Sadeh, N., and Fox, M.S., "Focus of Attention in an Activity-Based Scheduler," *Proceedings of the NASA Conference on Space Telerobotics,* NASA, 1989.

[Simon 83] Simon, H.A., "Search and Reasoning in Problem Solving," *Artificial Intelligence,* Vol. 21,1983, pp. 7-29.

[Smith, Fox & Ow 86] Smith, S., Fox, M.S., and Ow, P.S., "Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems," *AI Magazine,* Vol. 7, No. 4, Fall 1986, pp. 45-61.

[Stefik 81] Stefik, M., "Planning with Constraints (MOLGEN: Part 1)," *Artificial Intelligence,* Vol. 16,1981, pp. 111-140.