Making Partial Choices in Constraint Reasoning Problems

Sanjay Mittal and Felix Frayman
Intelligent Systems Laboratory, Xerox PARC,
3333 Coyote Hill Rd., Palo Alto, CA. 94304

Abstract

Constraint problems derived from design and configurations tasks often use components (structured values) as domains of constrained variables. Most existing methods are forced into unnecessary search because they assign complete components to variables. A notion of partial choice is introduced as a way to assign a part of a component. The basic idea is to work with descriptions of classes of solutions as opposed to the actual solutions. It is shown how this idea can reduce search and in the best case eliminate search. A distinction is made between a partial commitment (a partial choice that would not be retracted) and a partial guess. A particular way to implement partial choice problem solvers is discussed. This method organizes choices in a taxonomic classification. Use of taxonomies not only helps in pruning the search space but also provides a compact language for describing solutions, no-goods, and representing constraints. It is also shown how multiple hierarchies can be used to avoid some of the problems associated with using a single hierarchy.

I. Introduction

A central component of many design tasks is a constraint satisfaction problem (CSP) as defined by Mackworth [Mackworth, 1977], i.e., finding consistent assignment of values for a set of variables that together define the artifact which is the output of the design task. These variables are constrained by expressions derived from structural, functional, and performance requirements (see [Mittal and Araya, 1986], and [Araya and Mittal, 1987] for a more detailed articulation of this approach). An important characteristic of such constraint problems, i.e., the ones formulated for design tasks, is the use of structured values to represent domains of variables. Simply stated a structured value has internal structure in terms of additional variables with corresponding values. It is not hard to see why this is convenient. In design tasks, one often has pre-defined components that are used to define the domain of some of the design variables. Use of certain pre-defined resistors in discrete circuit design or the use of fixed sets of components in computer configuration are some examples. Unfortunately, most of the general-purpose methods for constraint satisfaction work at the level of making a complete choice for a variable and rely on some form of least commitment to defer making a guess to minimize search. In this paper, we introduce the idea of making a partial choice, which is especially appropriate for variables that have structured values (henceforth components).

The paper is organized as follows. We start with a simple example involving the use of components and show how the existing methods are forced into unnecessary search because they have to choose a complete component. Next, we introduce the notion of partial choice and how it reduces search. The basic idea of partial choice is to operate on descriptions of sets of solutions as opposed to actual solutions. In some situations, a partial choice can be viewed as a commitment that will not be retracted. A partial commitment may thus be viewed as an extension of the least commitment principle. We also discuss situations in which no partial commitment can be made and one has to resort to a partial guess. Making a partial choice affords benefits similar to hierarchical search, i.e., pruning of choices without having to examine all choices. In the next section, we show that taxonomies are one way to implement partial choices. We also show how multiple taxonomies can be simultaneously used to avoid having a particular order in which partial choices are made, something that results from using a single hierarchy. Some of these ideas have been implemented in two design expert systems, Pride [Mittal et al., 1986] and Cossack [Frayman and Mittal, 1987].

II. Example of a constraint problem using components

In the first part of the paper we will use the following very simple constraint problem.

Example 1. There are two variables X and Y. Each has two components as possible choices (in other words their domains). The components in turn can be viewed as having two distinct fields: p1 and p2 for components in the domains of X and Y. We shall use curly braces ({}) to represent the choices for a variable and square brackets ([]) to represent a component. The choices for X and Y are:

X: {[p1:a, p2:b] [p1:a, p2:c]} and Y: {[p1:d, p2:e] [p1:d, p2:f]}

We shall further use the dot notation to refer to nested variables. Thus, X.p1 means the value of the p1 field of the component assigned to X. The constraints on X and Y are:

C1: X.p1 = a iff Y.p2 = e;   C2: Y.p1 = d iff X.p2 = c

The constraint C1 reads, "X.p1 has value a if and only if Y.p2 has value e". This problem has four candidate solutions (2 choices for X x 2 choices for Y) and only one solution:

X = [p1:a, p2:c] and Y = [p1:d, p2:e]

III. Analyses of existing methods

In this section, we shall briefly analyze the performance of some of the general-purpose methods for doing constraint search on the above problem.

A. Generate and test

Using generate and test (G&T), one would build a generator that generates candidate solutions by making all possible assignments to X and Y. The constraints would be used to test the acceptance of the candidates. It is easy to see that in this case, the generator would produce 4 candidates, only one of which will be acceptable.

B. Hierarchical Generate and test

We define hierarchical G&T as follows. Assign values to the variables in some order to define partial candidates. Prune a partial candidate if any constraints can be applied. This improves over G&T in general. But in our example, the constraints can only be tested after both variables have been assigned values and thus no benefit results. Note that chronological backtracking is a standard way to implement hierarchical G&T.

C. Constraint propagation

Another common technique is constraint propagation as described in [de Kleer and Brown, 1986]. This is like hierarchical G&T with the constraints folded into the generator in such a way that they can be used to "look-ahead" for making more constrained choices for the other variables. In our example, constraints can be used inside the generator in the following way. Once a choice is made, say for X, constraint C1 can be used to make a choice for Y that satisfies that constraint. Clearly, this has advantages if the constraints can be so folded into the generator. However, only one of the two initial choices for X or Y is the correct one so in half the cases one would still have to search. However, as we shall show one can still do better.

D. Least Commitment Approaches

One approach that has sometimes been very effective is the use of some kind of least commitment problem solver as implemented in Molgen [Stefik, 1981] or more recently in the Pride expert system [Mittal and Araya, 1986]. In both of the above systems, least commitment is practiced by employing techniques for ordering the constraint problem in such a way that arbitrary choices can be minimized. In simple terms, one can think of least commitment as a technique for deferring a variable assignment as long as possible if multiple choices are possible and enough constraints are not known that would allow one to commit to the right choice. In other words, avoid making a guess as long as possible in order to minimize backtracking. In our example, the choices and constraints are such that no particular order of assigning values or checking the constraints helps in reducing the search.

IV. Use of partial choice to reduce search

The idea of making a partial choice is deceptively simple. However, instead of just stating it we will motivate it by the following exercise.

A. "Flattening" the components

The basic common cause of search in the methods discussed in the previous section can be traced to the fact that each component really represents a pre-packaged assignment of values to many variables. This means that a problem solver that assigns such a component to a variable ends up with a larger commitment than is warranted. Consideration of later constraints may lead to a contradiction for some of these assignments, causing the problem solver to search. This is easy to see if we eliminate the use of components, i.e., "flatten" them, and transform the above problem to the following problem. There are four independent variables: X.p1, X.p2, Y.p1, Y.p2 with the following domain of values:

X.p1: {a}; X.p2: {b, c}; Y.p1: {d}; Y.p2: {e, f}.

One can easily build a least commitment problem solver that would minimize search and in our example find the correct solution without any search. In general, however, the above problem transformation would be incorrect. Consider the problem defined in figure 1 (example 2, sec. IV.B.2). Simply "flattening" the components changes a 2-variable problem with a search space of 16 to a 4-variable problem with a search space of 64. This is because the use of components already represents the result of doing constraint satisfaction on this flattened set of variables with their own domains. *In essence, components represent solutions to a set of constraints which can then be ignored because they are already implicit in the components.* Alternately, one can view components as the result of doing some kind of compilation of the constraints on sub-sets of variables, which now correspond to components. By going back to the flatter set of variables, we will have to re-introduce those constraints, undoing past work. Real design problems often have hundreds of variables with tens of components per variables.

In summary, flattening such problems affects the search efficiency in two ways. One, flattening the components causes the search space to grow exponentially. Remember that the increase in the search space comes both from increasing the number of variables as well as by, possibly, increasing the domains for the variables. Two, flattening brings back the constraints which had been compiled away, increasing the time complexity at least linearly.

B. Partial choices: the best of both worlds

It is easy to state the notion of partial choice now. Simply, it is a way to make a commitment to only a part of a component in the expectation that such a partial choice would allow constraints to be considered to enable a better choice[1] for the rest of the component at a later point in problem solving. The idea is that where one would resort to guessing, one now selects an appropriate description for a class of components. This description may not uniquely apply to a single component but should be specific enough to enable further inferences to be made. At the same time it should not be so specific that it has to be retracted, at least avoidably so. We differentiate between partial commitment - a partial choice that does not have to be retracted and partial guess - a partial choice that may have to be removed later in the processing. Partial choice refers to both partial commitment and partial guess.

1. Partial Commitment

Consider example 1 of section II. A partial commitment for X would be to commit to p1 = a because that is common to all choices of X. Similarly, for Y commit to p1 = d because that is common to all choices for Y. With these partial commitments, the constraints C1 and C2 can be used to select the correct component for X and Y. In effect, by allowing partial commitments we get the best of both worlds. By continuing to operate on components, albeit partially, we preserve the advantages of components, i.e., prior solutions to other constraints and a reduced search space. At the same time we get the benefits of being able to consider the fields of a component as independent variables which allows finer-grain commitments to be made (or deferred).

At any moment during problem solving, when there is more than one alternative to be selected from and least commitment has to resort to guessing, the notion of partial commitment is applicable. Partial commitment involves examining the set of alternatives and determining a common part present in all the alternatives or the facts which will be true no matter which alternative is selected. Partial commitment is a monotonic inference, i.e., it never has to be retracted. Another way of viewing partial commitment is as means of calculating the entailments of the previously made decisions or making explicit the facts already available in an implicit form.

A problem solver that uses partial commitment would progressively commit to more fields of a component until all of them are assigned values. The major benefit of partial commitment approach is reducing search (backtracking) since a potentially retractable full commitment is replaced by a safe partial commitment. Another benefit of partial commitment has to do with finding all solutions of the constraint satisfaction problem. Since making a partial commitment does not make any unnecessary commitments that have to be retracted later, the state of the computation implicitly carries all the solutions to the CSP problem.

[1] By "better choice" we mean a choice that minimizes backtracking.

2. Partial Guess

The notion of partial commitment crucially depends on the existence of a common part for a set of alternatives. Often, there is nothing common among all the choices considered. We introduce the notion of partial guess to make the same idea work in this case. Let's consider the following example.

Example 2. There are two variables X and Y having two fields p1 and p2 with identical domains of possible values:

X: {[p1:a, p2:b] [p1:a, p2:c] [p1:d, p2:e] [p1:d, p2:f]}

Y: {[p1:a, p2:b] [p1:a, p2:c] [p1:d, p2:e] [p1:d, p2:f]}

The domains of the variables internal to these components, p1 and p2, are:

p1: {a, d}; p2: {b, c, e, f}

The constraints on X and Y are:

C1: X.p1 = a iff Y.p1 = d; C2: Y.p2 = e iff X.p2 = c

C3 X.p1 = d iff Y.p1 = a; C4 Y.p2 = c iff X.p2 = e

Figure 1 shows all 16 candidate solutions with four solutions marked:

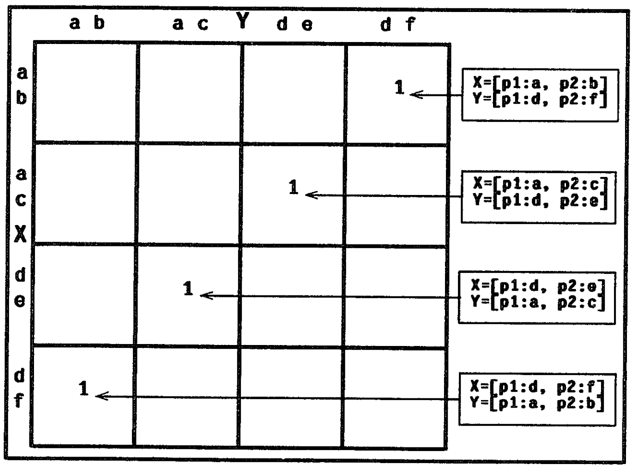
a b
a c
Y
d e
d f
a b
a c
X
d e
d f
1
1
1
1
X=[p1:a, p2:b] Y=[p1:d, p2:f]
X=[p1:a, p2:c] Y=[p1:d, p2:e]
X=[p1:d, p2:e] Y=[p1:a, p2:c]
X=[p1:d, p2:f] Y=[p1:a, p2:b]

Figure 1. Choices and solutions for example 2.

It is not possible to use the partial commitment approach as it has been introduced earlier, since there is no common part for any of the 4 possible choices for X and Y. It is necessary to introduce some modifications for the partial choice idea to make it work in this case. The modified approach will be called *partial* *guess approach*. The partial guess approach is based on the idea that it is possible to introduce subsets of the original set that have common parts and make commitments to the introduced sets, instead of committing to individual components. The subsets with common parts for our example are shown in Figure 2.

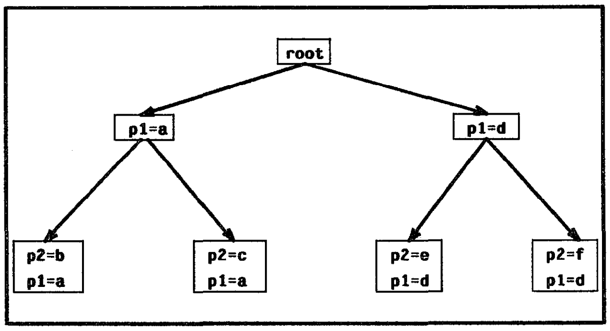

Figure 2. Choices for X and Y in example 2 organized as a tree.

Since the common part decomposition is based on p1 we will use p1 for the first decision. There are multiple alternatives for both X.p1, Y.p1 and we arbitrarily[2] select assignment of value to X.p1 first. There are 2 possible values X.p1 = a or X.p1 = d. We will choose arbitrarily X.p1 = a. Assuming that the problem solver can use constraints in the generation[3] of alternatives, the first constraint C1 can be used to assign Y.p1 = d. Making a partial guess of X.p1 = a effectively reduced the search space by eliminating four choices in the top-left quadrant in Figure 1 from viable solution candidates. Thus, making partial guesses effectively cuts down the search space by providing the benefit of a hierarchical search. A solution for the problem in Example 2 can be found easily by further examining the set of solution candidates left.

It is necessary to point out an important difference between making partial guesses with the introduced subsets and making a partial commitment as illustrated by Example 1. Guesses with the introduced sets are retractable, while partial commitments are monotonic and do not involve any guessing.

The performance of the partial guess approach in the worst case is no better than the performance of the least-commitment approaches, while in the best case may eliminate the backtracking completely. On the average, it would seem to offer the most advantage in underconstrained problems with large number of alternatives, especially when the alternatives can be grouped by some common descriptions. Partial choice strategy is also preferred to the flattening approach in cases when components have large number of properties or when the properties have large domains of values. In such cases flattening approach will increase the search space of possible values describing an individual component to the product of the domain sizes for every component property, while partial choice strategy will operate on the search space defined by a number of distinct component alternatives.

[2] To simplify the discussion, the problem was structured symmetrically in order to eliminate the reasoning in choosing the preferred order of making commitments. Selecting Y.p1 at this point will work similarly.

[3] The ideas are still applicable in case the problem solver can not use constraints in the generation phase.

So far we have presented these ideas in terms of assigning a partial component to a variable and thereby reducing search. One can also think of partial choice as a method for working with a description of classes of solutions, instead of working with actual solutions. In the examples we used for illustration the common part (either as a committment or as a guess) represented a description of a set of choices which could be instantiated by filling in the other variables of the components to obtain the complete components. Viewed this way, partial committment is the special case where a description applies to all members of a set. As a result, the description is a monotonic inference. Partial guess is the general case where the description may cover only a sub-set and may have to be retracted as the problem solving proceeds. We elaborate on this view in the next section where we present the use of taxonomies as a particular way of representing the descriptions of solution sets.

V. Use of taxonomies for making partial choices

There is a long history in AI of representing a set by abstracting the common parts and organizing the set in a taxonomic classification. The same basic idea can be used as a way of organizing choices for constraint problems. A set of component choices can be organized in a hierarchy. Intermediate nodes in this hierarchy represent a subset of the choices characterized by some common description of all the choices. For example, in a computer configuration problem, the set of printers may be organized in a taxonomy of high speed, medium speed, and low speed printers. The ***high speed*** printer node represents the subset of printers that all have speeds greater than say 100cps. Notice that at this level of description of the set, nothing is said about other properties of printers such as technology of printing, quality of printing, cost, interface, etc.

A. Searching with taxonomies

We briefly sketch a method for using taxonomic grouping of choices for making partial choices. The basic ideas are as follows. First, the choices for variables are organized into taxonomies. Second, instead of committing to a complete component as the value of a variable, we allow a more abstract description to be assigned to a variable. This description is a node in a pre-defined taxonomy. Third, the constraints have to be written in such a way that they can operate on these taxonomic descriptions. Finally, the search methods operate by moving down the taxonomy. If the correct taxonomic node cannot be committed to and a guess has to be made, it is made by selecting one of the nodes. If the choice later proves to be incorrect, then the problem solver backs up and selects another node. If a choice at some level does not allow all the applicable constraints to be processed, the problem solver descends to the next level and makes further choices. Notice that a retracted node allows a whole set of choices to be marked as no-goods (in ATMS terms [de Kleer, 1986]) without having to examine them individually. Furthermore, the chosen node represents a partial choice that would allow further inferencing. Space limitations do not allow detailed consideration of the algorithmic choices for building such problem solvers.

We summarize some of the advantages of using taxonomies for organizing choices. One, they allow partial choices to be made which leads to more efficient problem solving. Two, they provide a compact description of the no-good sets. For example, the no-goods determined from the constraint, "Class Foo of word-processing programs need letter-quality printers", can be compactly expressed as:

(((Word-Processing Foo) (Printer Dot-Matrix)),
((Word-Processing Foo) (Printer Thermal)))

In other words, no-goods can be represented compactly by pairs of inconsistent classes. Without the use of taxonomies, one would need to enumerate the actual printers and word-processing programs that are now represented by the classes *Dot-Matrix*, *Thermal*, and *X*. Finally, taxonomies provide a compact language for describing the solutions to the constraint problem. For example, instead of enumerating the sets of components that are consistent, one might be able to express them more succinctly in terms of these taxonomies. For example, ((Word-Processing X) (Printer Letter-Quality)) compactly describes a potentially very large set of solutions.

B. Natural taxonomies

In practice, it may not be easy to automatically compute an appropriate way of organizing a set of components in a hierarchy that is most effective during search. There may be competing ways of decomposing a component, some offering search advantages over the others. Furthermore, common part decomposition involve multiple component properties. The decomposition search space for computing all possible ways of finding common parts is a power set over the set of properties in a component. Natural taxonomies that evolve over a period of use by different users in some domains provide some relief from those problems. Essentially, taxonomies can be used to represent pre-determined decisions about how to abstract common parts. The different levels of the taxonomy reflect a decision about which variables are more useful to commit to earlier. As we pointed out previously, this decision is intrinsically tied to the nature of the constraints and choices. One can conjecture that the evolution of these taxonomies reflects a continuing experimentation with ways to make the proper choices with respect to features to abstract. A stable taxonomy, as obtained from domain experts, represents a compilation of these prior experimentations.

Another kind of decision embedded in a taxonomy is the grouping of variables in a node. Consider the case where a set of choices can be abstracted along different dimensions by finding different common parts. The constraint relationships might be such that instead of walking down a hierarchy, making progressively larger choices (i.e., involving more variables), it might make more sense to directly commit to two or more variables. These meta-level choices are again reflected in how the taxonomies are created. Note, that instead of viewing these kinds of properties of taxonomies as a given, one can use them actively in developing suitable taxonomies. Thus, there is a duality between characteristics of existing taxonomies and problem solving criteria for forming new ones. Use of taxonomies also helps in acquiring and maintaining the knowledge-base of components and constraints (see [Frayman and Mittal, 1987] for an extended discussion).

C. Using multiple taxonomies

Use of taxonomies suffers from one potentially fatal flaw. Consider what happens if the choices and constraints are such that a suitable hierarchical classification cannot be made or is not made. In the worst case, all the choices may be grouped under a single class. Taxonomies would not only not help but may lead to much wasted effort while the other nodes are rejected. Another, and more common, situation can occur when there are alternate ways of classifying some choices and it cannot be pre-determined which of those ways is more useful. Figure 3 shows three alternate ways of classifying printers, each of which is useful in resolving some of the constraints some of the time. Any single hierarchy which merges these three will lead to the wrong partial choice some of the time.

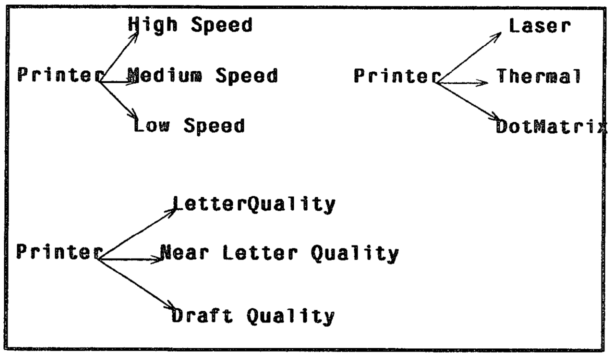

Figure 3. Alternate ways of classifying printers.

We briefly sketch a method whereby one can search using multiple alternate ways of organizing a set of choices. The basic idea is to extend the problem solver to simultaneously make partial choices in each of the alternate hierarchies. For example, instead of just selecting a node in a single hierarchy, the problem solver can select a node in one of the hierarchies that is most relevant to the constraint being considered. Each such partial choice would lead to certain constraints being processed, which in turn would allow further inferencing. Once all the constraints have been satisfied, then the choices described by the chosen nodes in each of the hierarchies can be intersected to find the actual solutions. For example, the problem solver could be simultaneously making partial choices from each of the hierarchies shown in Figure 3. At the end of constraint satisfaction, the printer choices might be described by: {HighSpeed & LetterQuality & Laser}. By intersecting the sets of printers described by these classes one gets the actual set of printers which constitute all consistent solutions. Note that the same intersection technique can be used during the intermediate stages of problem solving to quickly determine if the current partial choices are mutually consistent. By allowing multiple hierarchies one can avoid the problems associated with a single taxonomy.

VI. Conclusion

This paper introduced the notion of partial choice for constraint satisfaction problems in structured domains. Partial choice represents an improvement over the least commitment strategy when least commitment has to rely on guessing to proceed. There are two flavors of partial choices - *partial commitments* are partial choices that do not have to be retracted and *partial guesses* which are partial choices that may be retracted. Partial commitments are applicable in case alternate choices have a common part and involve computing the entailments of the previously made decisions. Partial guesses are applicable in case alternative choices do not have a common part, but can be divided into non-intersecting subsets with common parts. Partial guesses select such subsets which allows the whole subsets to be ruled out without considering all their elements. A common thread through all the ideas presented here is the use of description of classes of solutions as opposed to the actual solutions. Working with such descriptions, in appropriate cases, can both help reduce the search as well as find multiple solutions.

Our use of hierarchies for structuring a set of choices is similar in some ways to the use of hierarchical domains for arc consistency, i.e., removal of local inconsistencies for binary constraints [Mackworth et al., 1985]. In some ways the notion of partial choice as applied to structured values is complementary to the use of hierarchies in [Mackworth et al., 1985]. There hierarchies are an effective way to organize the choices for a variable without any consideration to the internal structure of the choices. We are particularly concerned about situations where the choices have internal structure in terms of additional variables and constraints. Thus, hierarchies are a way of organizing a set of variables that effectively allows a set of constraints (i.e., ones internal to the set of variables comprising a component) to be ignored. We suspect that both of these ideas can be effectively used together. Another difference between the two works is that while arc consistency algorithms only remove some of the inconsistencies, partial choice methods are also effective in finding consistent solutions to the complete CSP problem. A continuing area of investigation for us is to extend some of the existing constraint reasoning methods in a way that incorporates partial choice ideas, especially the use of multiple taxonomies. We are also investigating if our ideas can also be incorporated in the various network consistency algorithms [Mackworth, 1977; Mackworth et al., 1985].

Acknowledgements

We are grateful to Agustin Araya and Mark Stefik for useful discussions of some of the ideas presented in this paper. Danny Bobrow, Johan de Kleer, and Mark Stefik gave insightful comments on earlier draft of this paper. Danny pointed out the relationship to the work on minimization of boolean terms.

References

[Araya and Mittal, 1987] A. Araya and S. Mittal. Compiling design plans from description of artifacts and problem solving heuristics. To appear in *Proc. IJCAI-87*, Milan, Italy, International Joint Committee for Artificial Intelligence, August 1987.

[de Kleer, 1986] J. de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28(2):127-162, March 1986.

[de Kleer and Brown, 1986] J. de Kleer and J. S. Brown. Theories of Causal Ordering. *Artificial Intelligence*, 29(1):33-61, July 1986.

[Frayman and Mittal, 1987] F. Frayman and S. Mittal. Cossack: A constraints-based expert system for configuration tasks. To appear in *Proc. 2nd Intl. Conf. on Applications of AI to Eng.*, Boston, MA., August 1987.

[Mackworth, 1977] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8: 99-118, 1977

[Mackworth et al., 1985] A. K. Mackworth, J. A. Mulder, and W. S. Havens. Hierarchical arc consistency: exploiting structured domains in constraint satisfaction problems. *Computational Intelligence*, 1(3-4):118-126, August - November 1985.

[Mittal et al., 1986] S. Mittal, C. L. Dym, and M. Morjaria. PRIDE: An Expert System for the Design of Paper Handling Systems. *Computer*, 19(7):102-114, July 1986.

[Mittal and Araya, 1986] S. Mittal and A. Araya. A Knowledge-Based Framework for Design. In *Proc. AAAI-86*, pages 856-865, Philadelphia, PA., American Association for Artificial Intelligence, August 1986.

[Stefik, 1981] M. J. Stefik. Planning with constraints. *Artificial Intelligence*, 16(2):111-140, 1981.