Explanation and Implication for Configuration Problems *

Eugene C. Freuder, Chavalit Likitvivatanavong, Richard J. Wallace
Constraint Computation Center, Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
ecf-,chavalit-, rjw@cs.unh.edu

Abstract

In this work we explore the problem of generating explanations for configuration problems using on the constraint satisfaction (CSP) framework. In addition, we are concerned with deriving implications from user choices in order to guide selection of later choices. We show that the consistency methods used in connection with constraint processing can be used to generate inferences that support both functions. In this work we use the n-queens problem as a testbed. The system we have developed is interactive and allows the user to make selections and perform arc consistency on the current problem, as well as retracting selections, a typical arrangement with current configurator systems. At the same time it generates explanations for value deletions and current choices as well as implications in terms of the amount of future domain reduction that will follow certain choices and whether these choices will lead to solutions or non-solutions. Explanations take the form of trees which show the basis in terms of previous choices for current choices and deletions. Together, these methods suggest ways in which the process of solving combinatorial problems can be made more perspicuous and more interactive.

1 Introduction

Constraint-based technology has become a major tool for solving configuration problems [Sabin and Weigel, 1998]. Present-day applications allow users to work interactively, to create representations in the form of constraint satisfaction problems and to instantiate these representations to obtain solutions. In these systems, arc consistency algorithms are used to maintain a degree of local consistency by ruling out values that are not supported by the choices the user has already made.

In an interactive process like this, comprehensibility of the problem solving process becomes an increasingly important issue. In particular, the process may be facilitated if users can get *explanations* for results, e.g. "the frammus must be blue because red will clash with the gingus". This is especially true if the result is: "your problem is unsolvable"; users want to know why, and ideally to get advice as to how to modify the problem to make it solvable.

Another facet of comprehensibility when users are incrementally making choices as they define a problem or try to solve it interactively, is knowing the *implications* of a choice, e.g. "if you make the frammus red, then you can't have a red gingus". When the problem is unsolvable the user will want to change the problem and view the implications of the changes.

Current applications provide some facilities to meet these requirements. Specifically, they allow users to attach verbal "explanations" to specific constraints. When one of these constraints is violated, this explanation is shown to give some indication of the basis for the conflict. Obviously, such 'reminders' are limited in the information they convey and the situations they can handle, and they do not deal at all with the question of implications of user choices. Moreover, they are entirely extraneous to the actual process of problem solving.

Explanations for constraint-based systems would appear to be intrinsically difficult because such systems generally rely on combinatorial search. An obvious response to the need for explanation or implication information - tracing the solution process - does not work well for search. However, the consistency processing that distinguishes the AI approach to constraint solving and is already used in commercial configuration systems is an inference process. In this case, inferences lead to domain restrictions, and in the extreme case, when a domain is reduced to one or zero elements, the inference process limits us to assigning a specific value or shows us that the previous assignments produce a non-solution. This means that solution search can provide at least partial explanations for why an assignment was made or why a solution is not possible under the circumstances.

The present work builds upon this insight, and is concerned with automating the process of providing information about explanation and implication. With respect to the first part of the problem, providing explanations, our goal is to help the user understand the following situations:

- why did we get this as a solution?
- why did this choice of labels lead to a conflict?
- why was this value chosen for this variable during pro-

*Supported by Calico Commerce.

cessing?

- why was this domain restricted?

Knowing about implications of current choices will help the user make intelligent choices during the subsequent course of problem solving. For implications our goal is to provide the user with information about the following:

- is there a basis for choosing among values in a future domain?
- specifically, what effect will different selections have on the number of values remaining in other domains?
- are there values whose choice will lead to conflict, even though they are consistent with the present domains?

In short, we want to be able to offer suggestions about how best to proceed, especially if a conflict has been encountered.

When we consider how to generate explanations, we face two important issues: What should serve as an explanation? and How can we produce better explanations? The latter presumes that we can measure the "goodness" of an explanation. There are clearly many possible answers to these questions. One approach, that follows naturally from our methods, is to measure goodness in terms of explanation size, assuming that, other things being equal, smaller is better.

In the next section we describe the basic features of the demo. In Section 3 we introduce the notion of an "explanation tree" to as a framework for automatically generating explanations in a dynamic environment. In Section 4 we demonstrate how constraint-based inference methods can be used prospectively to determine implications of user choices. In Section 5 we discuss problems associated with interactive use, namely, retracting choices and conflict handling, that require undoing choices. Section 6 reviews related work. Section 7 gives conclusions and discusses some extensions of this work.

2 Testbed

In this paper we illustrate the generation and use of explanations and implications with the n-queens problem. In this problem, n queens must be placed on an n by n chessboard in such a way that no queen can attack another. (Recall that a queen can move in a straight line horizontally, vertically or diagonally on the board.) Figure 1 shows an example of this problem when n = 9. In this case, if, for example, a queen is placed in the second cell (counting from the left) of row 1, as shown in this figure, a second queen cannot be placed in the same row, nor can it be placed in column 2, nor can it be placed in cells 1 or 3 of row 2, cell 4 of row 3, etc. This problem differs from the logic puzzles that we used earlier in this connection [Freuder *et al.*, 2000] in that the problem cannot be solved through inference alone.

The n-queens problem can be represented as a constraint satisfaction problem, which consists of a set of variables, sets of values, where a member of each set must be assigned to a specific variable, and a set of constraints, where each constraint is a relation on a Cartesian set of domains that is associated with some subset of the variables. Here, we represent the queens problem in the conventional manner, where the rows are variables and the domain of each variable is the set of cells in that row. Constraints hold between pairs of rows and are based on the rules of attack described above. For example, the constraint between rows 1 and 2 includes cell 1 of row 1 and cell 3 of row 2, which can be denoted as ((1,1), (2,3)) but not cell 1 of row 1 and cell 2 of row 2. Because the constraints are all binary, the entire network of constraints forms a constraint graph, where the variables are the nodes and each constraint is represented as an edge between two variables. For the queens problem the constraint graph is complete. In these terms, the problem becomes one of choosing a cell in each row for the placement of a queen, so that there is no violation of any of the constraints that are due to the rules of attack.

The interface itself shows the problem in terms of domains rather than as a constraint graph (Figure 1), which is a popular alternative for constraint satisfaction problems. This is quite reasonable in this case, where every variable is constrained by all the others, and it may be a generally useful approach, as we will show later.

In this work we solve the problem using arc consistency. This is a simple form of inference in which the domains of each pair of variables linked by a constraint are made fully consistent. This means that for every value in the domain of one of these variables there is at least one value in the other domain such that the two together form a tuple that is a member of the constraint between these variables. In this case the tuple is said to *satisy* this constraint. Values that do not satisfy a relation in this manner are discarded because they cannot form a part of any solution.

As indicated above, arc consistency by itself cannot solve the n-queens problem. In fact, it cannot delete any values at all if no assignments have been made. Therefore, at least one user selection (i.e. an assignment of a queen to a cell in one row, which reduces that domain to one value) is required before any values can be deleted, and a certain number of user selections are required to solve the problem completely. In this respect, our testbed is analogous to the usual configurator, which as described before also uses arc consistency as its sole inference engine.

Figure 1 shows the testbed interface, with a partly instantiated problem. The user clicks on a cell on the n by n board to place a queen there. At any time he or she can perform arc consistency by clicking the top button on the right (labeled "implications"). In the Figure, two queens have been placed on the board, in row 1, cell 2 and row 3, cell 6, (i.e. cells (1,2) and (3,6)), and arc consistency has been performed. Cells that represent values deleted by arc consistency are shaded. At the same time, the remaining cells are labeled with numbers indicating how many more values (cells) would be deleted by arc consistency if a queen were place on that cell.

In addition to the representation of variables (rows) and values (columns) in the center panel, other panels below and to the right of this are used to present explanations and other commentary, as discussed in later sections of this paper. Buttons in the upper right corner of the layout are used in the course of constructing and solving a problem. Beginning at the top, they are for (1) performing arc consistency, (2) undoing the last alteration, (3) starting a new problem, (4) show-

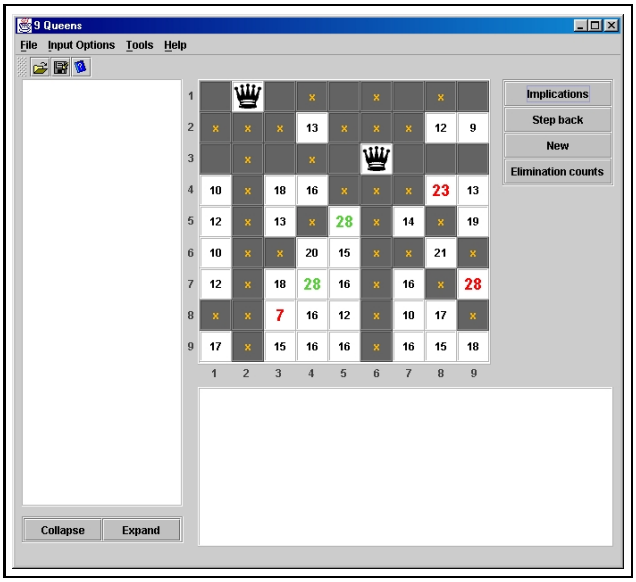

Figure 1: n-queens interface. Greyed-out cells have been eliminated by the two queens placed on the board. White cells indicate remaining values. Counts in the latter cells indicate how many more values will be eliminated if a queen is placed in that cell and full arc consistency performed. The meaning of the x's on the greyed-out cells is left as an exercise for the reader. Other features of this interface are discussed in the text.

ing elimination counts (currently shown by default, as in the figure). Explanations for the state of a given cell are shown in the panel to the left. (This is discussed at greater length in the next section.) A running commentary on the progress of search is given in the panel at the bottom. (This is not discussed further in this paper, although examples can be seen in Figures 2 and 3.) The testbed is implemented in Java. Running on a Pentium III machine, these and all other functions described in this paper are performed instantaneously, from the user's point of view, on the present 9-queens problem.

3 Deriving Explanations

3.1 Explanations

In the context of searching for a solution, the idea of explanation turns on the notion of sufficiency. That is, an explanation is a set of elements that is sufficient to deduce another element whose selection is to be explained. These elements are members of the basic sets from which a CSP is composed, in particular, domain values and constraints. Another key concept here is selection: we are always trying to explain a choice or selection from what was given in the original problem, and we must compose an explanation from elements of the problem that are pertinent to this selection.

By itself, this definition does not tell us how this information is to be communicated to the user. Questions of presentation form another part of the overall problem of comprehensibility, and these are handled for the present in a largely intuitive fashion, using what seems to us to work. Thus, in the description of the testbed given above, we have indicated how critical actions and outcomes are presented to the user via icons that represent value selections and changes in color that represent deletions.

3.2 Explanation Trees

As already indicated, this definition of explanation ties in well with constraint-based reasoning. When a value is deleted or an assignment made because all values in a domain except one have been eliminated, then the basis for these outcomes is given by the selections already made in the course of search. In fact, from the present set of assignments and deletions we can obtain *immediate explanations* for such outcomes that meet the sufficiency condition described above. But the elements in the immediate explanation may have their own explanations (unless they were either chosen by the user or given in the original problem description), and this process can be iterated. This means that explanations can be unwound to form a network of elements or an *extended explanation*, which in its fully extended form, where all its elements are either themselves explained or are givens, is an ultimate or complete explanation of the selection in question.

From this it might appear that we face potential tractability problems if we allow the user to call for extended explanations *ad libitum*. Fortunately, there are several ways to avoid incorporating cycles into our extended explanations. In the first place, whenever a value is deleted, information about the (earlier) assignment that led to the deletion can be stored in connection with the deleted value. Similarly, whenever a variable is assigned a value, we can use this stored information to derive a set of existing assignments that form a sufficient basis for assigning this new value (i.e. for deleting all the other values in the domain of this variable). Obviously, such a set exists; otherwise the assignment would not have been made. Now, since the process of storage follows the order of search, and at any time during search there is a current search path that is, of course, acyclic, then in forming an extended explanation from this information we are guaranteed not to encounter cycles. Because the explanations formed in this way are acyclic, we call them *explanation trees*. Such trees are always rooted at the element to be explained.

This approach has other convenient features. In particular, since it follows the course of search it is eminently suited for generating explanations dynamically. In fact, it is hard to see how this can be accomplished except by building explanations dynamically as well.

There is also a minimal degree of redundancy in the stored information, since when a deletion occurs only one of the current assignments is stored in association with the deletion. This appears to be a much more satisfactory strategy than trying to avoid redundancy by specifying explanatory links ahead of time. (And it is not clear how such a strategy could be used to generate explanations dynamically.)

Of course, there is a cost incurred for updating: in particular, if an assignment is retracted by the user (and possibly altered at the same time), information must be discarded from that point in the current search path, and at least partly regenerated. The procedures used for this purpose in the present testbed are described in Section 5 below. In practice, this has

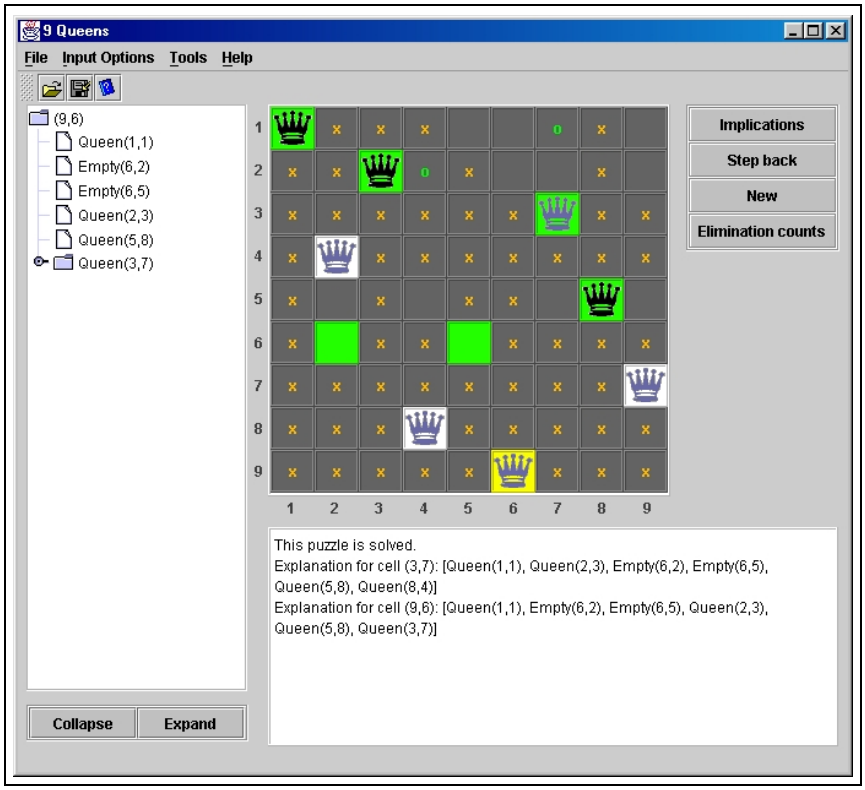

Figure 2: n-queens interface with explanation for queen in cell (9,6). Lefthand panel shows explanation tree restricted to one level, representing an 'immediate' explanation for this assignment. Cells in black are given; the rest were derived from arc consistency processing. Note inclusion of empty cells in explanation;- see text for further discussion.

proven to be as efficient as processing after a new assignment, as described in the last section.

An explanation tree for one assignment in a solution to the queens problem is shown in Figures 2 and 3. The assignment to be explained is the queen in cell (9,6). The immediate explanation is shown in the left-hand panel in Figure 2. (At the same time, the elements in the immediate explanation are highlighted on the board.) The user obtains this explanation by right clicking on cell (9,6) on the board layout. The cells listed in the explanation are the set of elements that together eliminate all the cells in this domain except the one where the queen has been placed. Note that in this case the set includes two empty cells in addition to attacking queens; the former, cells (6,2) and (6,5), are the only remaining elements in their domain, and neither supports a queen in cells (9,2) or (9,5).

Explanations of greyed-out cells can also be obtained in the same manner; these are restricted to a set of cells that rules out that value. Often this is a singleton cell to which a queen has been assigned, but as implied in the last paragraph, it can also be a domain of cells none of which supports the designated cell.

In Figure 3, the explanation tree has been expanded to give a fully extended explanation for the same designated cell [(9,6)] as in Figure 2. This is done by clicking on the right-hand button below the left panel, that says "expand". (The cells in the immediate explanation remain highlighted on the board.)

There are alternatives to the present approach. If we want explanations for labels, when each label is assigned we can search for a set of values that *could* eliminate all the other values from this domain. With this approach we may be able to find better explanations according to some criterion of goodness. Another alternative in this case is to wait until we have a complete solution and then search for an explanation. However, since we are interested in having explanations at each stage in the search process, here we store information immediately after each significant event, such as value loss.

Since our explanations take a well-defined form in this situation, this allows us to describe them quantitatively and to establish criteria for goodness in this domain based on simple properties like number of nodes or average levels in a tree. These features were studied in some detail in previous work [Freuder *et al.*, 2000]. We have not yet done similar studies in the present testbed, although it may be possible to generate more compact trees using improved heuristics for AC algorithms, as described in [Wallace and Freuder, 1992].

4 Deriving Implications

Each successive value assignment alters the status of values in the rest of the problem in various ways that are often not obvious. Using arc consistency, we can determine many of these implications of user choice. In the first place, we can run arc

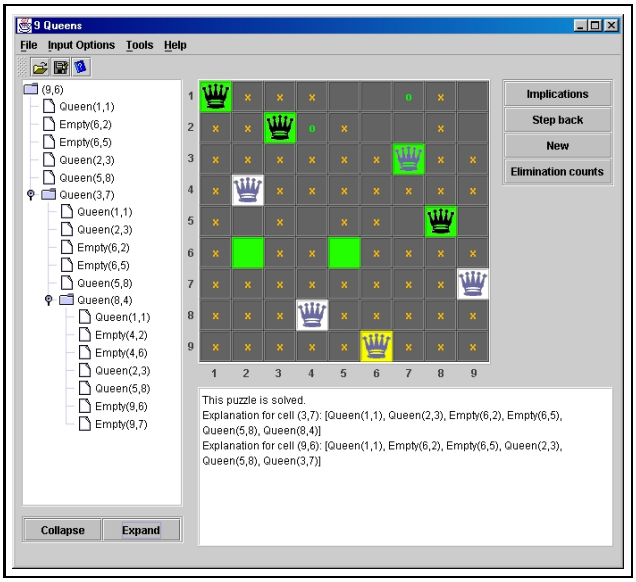

Figure 3: n-queens interface with explanation for queen in cell (9,6). Lefthand panel shows the extended explanation tree, representing a complete explanation for this assignment. In the left-hand panel, assignments that were derived, and for which the explanation can be extended, are indicated by gray boxes.

consistency with each future value selected for assignment to determine the reduction in domains that will ensue. In the course of doing this, we can sometimes determine that a given value if selected will lead to a solution in the next round of arc consistency or, conversely, that it will lead to failure in the form of a situation in which all the values in some domain have been deleted.

These capacities are illustrated in Figure 1 above. There, each empty cell is labeled with a count of the number of cells that will be deleted if a queen is placed there, and full arc consistency is then performed. In the course of performing arc consistency in association with one of these presumptive assignments, it may be found that there is no solution; in this case the count is given in red. (Examples in this figure are cells (4,8) and (7,9).) Conversely, if the problem is solved (by deducing positions for the queens in the now-empty rows) when the prospective problem is made arc consistent, then the count is given in green. (Examples in this figure are cells (5,5) and (7,4).)

5 Supporting Interactive Use

5.1 Dynamically Altering Explanations

Since the systems we are concerned with operate in interactive mode, we must generate explanations and derive implications interactively. As part of this, the user must be allowed to retract assignments, and the explanations must be updated accordingly. This is, of course, important for coping with unsolvable problems and situations in which all the problem features are not available in advance.

To achieve this form of interaction, we must be able to undo effects of inferences from the assignment being modified, since explanations of other values that include this value are no longer valid. In addition, we must undo inferences that affect the modified value. Suppose that a cell c in row k has been greyed out; this means that there is a queen at cell c' in row k' that can attack that space. Therefore, if this queen is removed or moved to another cell in the same row, then the inferences based on that queen, including the explanation for cell c, must be undone as well.

In the present testbed, this is done by rerunning arc consistency from scratch after a change has been made and rebuilding the explanations and implications for the new board. For this purpose, the program keeps track of all user input and the cells affected by each input. (This can be shown by using "Display input sequence" in the file menu.) If the user steps back, the last input in the input sequence is removed and the states of affected cells are reversed. If the user moves a queen to another position, the program modifies the corresponding input in the input sequence, clears the board, and reruns the entire input sequence from the beginning. In this way it always comes up with the same explanations if they are not affected by the change.

5.2 Using Explanation Trees to Handle Conflicts

An important case where decisions must be retracted occurs when these decisions have led to a conflict or domain wipeout. For wipeout, the simplest approach is to generate the same type of explanation tree as for necessary assignments. In other words, an immediate explanation is a set of assignments that serves to eliminate all the values for a given domain.

However, for our testbeds we have developed an alternative approach that appears to have considerable value as an intuitive way of flagging a deadend condition. During arc consistency processing, all domains are checked to see if there are any cases where a queen is not supported by any element in another row. Obviously, under such conditions further processing will lead to wipeout. Instead of allowing this, the cells that are 'in conflict' are highlighted. This serves to emphasize the fact that processing has reached a stage where a subset of variables has only incompatible values. The user can then check the explanations for each of these domains to determine what to alter.

In fact, we can go further in aiding the user at this point. For each conflicted cell, we can determine the set of all possible explanations (an inexpensive operation [Freuder *et al.*, 2000]), and then decompose this set into elements common to all explanations and those not. Then, if an element in the former set is changed, this will allow removal of the label in conflict. In contrast, for the subsets of non-common elements, one element must be changed in each subset to allow the same removal. This has not yet been implemented for the queens testbed, although it was included in the earlier testbed based on the 9-puzzle [Freuder *et al.*, 2000].

For the n-queens problem, conflicts can involve more than one value in a domain. However, this poses no problem for visualization because the different values in a domain can be shown together in a natural way, as the squares in a row of the chess board.

6 Related Work

Explanation trees are related to truth maintenance systems (TMS's, [Forbus and deKleer, 1993]), in that they provide a form of justification for particular facts such as variable assignments. In fact, a full explanation tree for an assignment is simply the transverse closure of its justifications, which corresponds directly to this feature in justification truth maintenance systems. Explanation trees are more restricted in scope in that they are not used to enhance search by supporting backjumping strategies (dependency-directed backtracking) as TMS's often are; with CSPs this capacity is, of course, usually bundled with the search algorithm. Of greater importance is the fact that with explanation trees, justifications are directly tied to search paths. As a result they are always enlarged in a certain order, one that guarantees a tree structure. In addition, this provides us with the opportunity to 'manage' the explanation by choosing more efficient search orders or by choosing among possible explanations at each step if there are more than one.

Recently [Bowen, 1997] has discussed the generation of explanatory glosses by a constraint-based system, Galileo4, that supports concurrent engineering for solving configuration problems. This system allows the user to add constraints or variables to a problem incrementally. Domains can be annotated with explanations in the form of relevant constraints that have been added to the problem and that have resulted in domain restrictions. Explanations in the form of verbal descriptions of constraints are also given on request when variables have been added to the problem by the system. This can occur in response to the invocation of conditions in the form of constraints that may not be apparent to the user. Finally, explanations can be minimized by removing redundant, less restrictive constraints. In one respect, this is a generalization of the present work in that the 'element' being explained is the current set of viable domain values. (Some need of this in the present context is indicated by the use of empty cells in our immediate explanations.) Another important difference is one of focus, or presentation: our focus is on assignments as explanatory elements, which leads naturally to iterated inferences in the form of explanation trees. As a result, our system captures the historical aspect of explanations (how did I get here?) in a perspicuous fashion. In Galileo4 the focus is on constraints; this is important when the constraints are heterogeneous in character.

A different approach to comprehensibility in the context of configuration problems has been taken by [Felfernig *et al.*, 2000]. These authors use a version of model-based diagnosis to aid the user in discovering errors in a configuration knowledge base or errors in the user specification for a problem. In the former case the knowledge base is shown to be inconsistent with positive problem-cases; in the latter the specifications are shown to be inconsistent with a valid knowledge base. It is unclear whether the system prototype supports the incremental, dynamic features of our system, although it may be possible to extend it in this fashion. Like Galileo4 this system reports conflicts as explanations without going further into the basis for these conflicts. (Since Galileo4 is incremental in nature, this in itself provides some focus for determining this.)

The present work is also related to the problem of "analytic debugging" [Meier, 1995], which is designed to answer questions similar to those posed in the Introduction in the context of program evaluation. However, the system described in that paper does not formulate explanations, but instead presents information pertinent to the current state of search in the spirit of debugging systems.

In summary, this work together with ours indicates that there are important questions regarding extensions to other environments for all these systems. In the future it will also be helpful to understand how the numerous features embodied in these different approaches are related to overall comprehensibility of the solution-finding process.

7 Conclusions and Prospects

Our work to date has shown how explanations can be built automatically for a complex inferential process and how implications can be derived based on the same forms of consistency processing. Explanations take the form of trees that are created during the course of problem solving and can then be used as a commentary for each step of the process. Implications can also be derived on the basis of simple consistency processing.

Some further directions have already been indicated in the discussion in the previous section. Currently, we are developing a system and interface using ideas developed in connection with the n-queens problem that can be applied to typical configuration problems such as assembling a computer system. A sample configuration problem in this form is shown in Figure 4. (This problem is derived from the sample given in [Calico, 2000].)

This prototype system uses the same matrix approach for representing the problem, where the values in each domain are shown as squares on a board. Since domain sizes are unequal, the total number of squares in a row must equal the largest domain size; if the squares are 'unused', they are shown in gray. When the user places the cursor on a row, the attribute or value names are shown in a panel to the right near the top of the display. Values that are currently available are shown in white with those selected by the user marked with check marks; deleted values are darkened.

Unlike the queens testbed, this system must be able to handle n-ary as well as binary constraints. We are working on ways to handle these in an efficient manner, while retaining the comprehensibility of the queens testbed.

References

[Bowen, 1997] J. Bowen. Using dependency records to generate design coordination advice in a constraint-based approach to concurrent engineering. *Computers in Industry*, pages 191–199, 1997.

[Calico, 2000] Calico. *Calico Advisor and Visual Modeler Training Guide*. 2000.

[Felfernig *et al.*, 2000] A. Felfernig, G. E. Friedrich, D. Jannach, and M. Stumpter. Consistency-based diagnosis of configuration knowledge bases. In *Proc. 14th Europ. Conf. Artif. Intell., ECAI-2000*, pages 146–150, 2000.

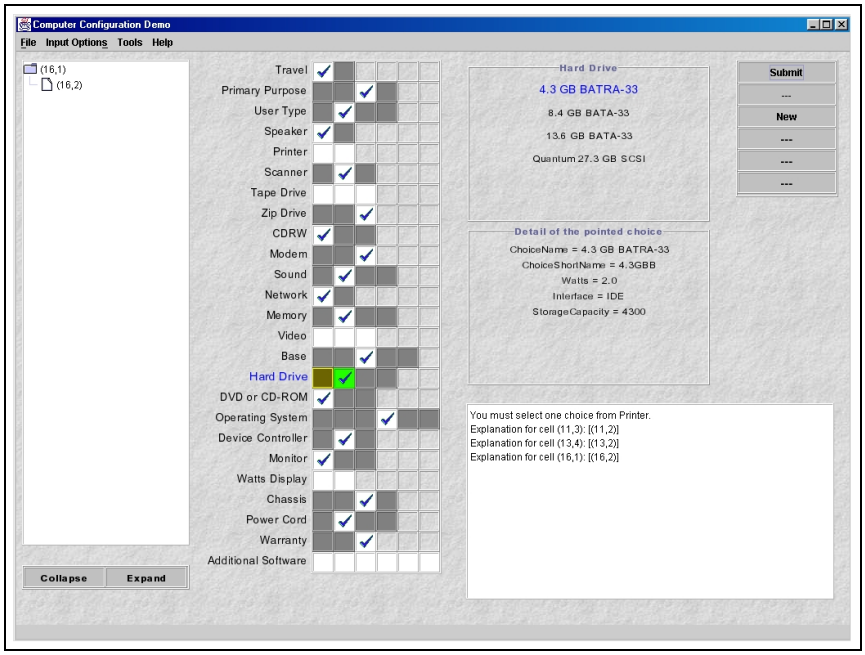

Figure 4: Computer configuration problem represented in terms of the present system. Current domain values are shown in white, deleted values are darkened, and check marks indicate current user choices.

[Forbus and deKleer, 1993] K. D. Forbus and J. deKleer. *Building Problem Solvers*. MIT, Cambridge, MA, 1993.

[Freuder *et al.*, 2000] E. C. Freuder, C. Likitvivatanavong, and R. J. Wallace. A case study in explanation and implication. In *CP2000 Workshop on Analysis and Visualization of Constraint Programs and Solvers*, 2000.

[Meier, 1995] M. Meier. Debugging constraint programs. In U. Montanari and F. Rossi, editors, *Principles and Practice of Constraint Programming - CP'95. Lect. Notes in Comp. Sci. No. 976*, pages 204–221. Springer, Berlin, 1995.

[Sabin and Weigel, 1998] D. Sabin and R. Weigel. Product configuration frameworks - a survey. *IEEE Intelligent Systems and Their Applications. Special Issue on Configuration*, pages 42–49, 1998.

[Wallace and Freuder, 1992] R. J. Wallace and E. C. Freuder. Ordering heuristics for arc consistency algorithms. In *Proc. 9th Canad. Conf. Artif. Intell.*, pages 163–169, 1992.