Effective Interaction Principles for User-Involved Constraint Problem Solving

Pearl Pu[1] and Boi Faltings[2]

[1]Human Computer Interaction Group
Institute of Core Computing Science
Swiss Federal Institute of Technology (EPFL)
IN-Ecublens, 1015 Lausanne
Switzerland
Pearl.Pu@epfl.ch

[2]Artificial Intelligence Laboratory
Institute of Core Computing Science
Swiss Federal Institute of Technology (EPFL)
IN-Ecublens, 1015 Lausanne
Switzerland
Boi.Faltings@epfl.ch

Abstract. This paper presents four user-involved constraint problem solving systems built at the Human Computer Interaction and Artificial Intelligence labs at EPFL. We propose a classification of the types of user input and discuss optimal contexts in which human and machine intelligence may be combined. We then formulate effective design principles on the interaction and visualization methods employed in our systems, which should also apply to other interactive systems based on a CSP model.

1 Introduction

The Constraint Satisfaction Problem (CSP) is a highly successful framework for formalizing problem-solving. It has been proven to be expressive enough for a wide range of practical problems, while at the same time being restricted enough to allow efficient general techniques for solving them. It is therefore interesting to consider the circumstances in which constraint satisfaction may benefit from or even require interaction from the user, and how the characteristics of CSP can be leveraged for efficient and general techniques for such interaction.

A constraint satisfaction problem (CSP) is defined by a set of n variables $x_1,\ldots,x_n$ that can take values in associated discrete domains $D_1,..,D_n$, and a set of m constraints $C_1,..,C_m$, which we assume to be either unary (one variable) or binary (two variables), and which define the tuples that are allowed for the variable or pair of variables.

Besides hard constraints that can never be violated, a CSP may also include *soft* constraints. These are functions that map any potential value assignment into a numerical value that indicates the preference that this value or value combination carries. Solving a constraint satisfaction problem means finding one, several or all combinations of complete value assignments such that all constraints are satisfied. When soft constraints are present, it also involves finding optimally preferred assignments.

As constraint satisfaction provides a useful abstraction for developing broadly applicable problem solving algorithms, user experience issues become important to support interactive constraint solvers. In this paper, we point out some principles for such interaction.

1.1 Motivations for User-Involved CSP

If a constraint satisfaction problem is completely and accurately specified, a variety of algorithms can be used to automatically generate the solutions. However, specifications of CSP are often incomplete or inaccurate, for reasons such as:

- It involves too many tedious details to specify all the constraints;
- Users are not aware of all constraints until they see them violated;
- Constraints change over time;
- Users want to keep certain constraints confidential;
- Too many solutions are generated and the best one is selected using additional constraints or criteria known to the user;

From analyzing applications that can be modeled in CSP framework, we have identified the following three kinds of user input:

1. *Preferences* - specifying explicit preferences of values or tuples within an individual constraint, such as giving relative preferences to the colors red, green and blue in automobile configuration tasks;
2. *Hidden Constraints* - specifying constraints unknown or not expressed in the original formulation, such as "when the car has 4 doors, the color should not be red;"
3. *Tradeoffs* - specifying the relative importance of the constraints with respect to each other, such as indicating that price is more important than the delivery date.

We do not claim that this is an exhaustive list, but useful in most practical problems and especially in the systems that we have developed. Note that the categorization excludes for example extending the set of variables or domains in the problem, since such changes are incompatible with many mainstream techniques for CSP.

1.2 Abstractions for User Interaction

One of the earliest interaction models used is based on natural language dialogs. The advantage is that a novice user is able to interact by answering questions at a high level of abstraction, unaware of the underlying problem solving paradigm. However, dialogs systems are costly to build, the set of vocabulary must be changed when the

application domain changes, and most importantly, users are less likely to take initiatives to change the course of action, even though users interact when prompted. Alternatives to dialogs are direct manipulation interfaces and visualization techniques, which are more comprehensible, predictable and controllable [12]. To avoid the cost of constructing a visual interface for each domain of application, it is beneficial to understand a large domain of tasks (task analysis), and build design principles that can be applied across the whole domain. From that point of view, CSP is a good framework because it is widely applicable. Thus, interaction designs for one CSP application can often be easily and quickly applied to build visual interfaces for other domains.

We are interested in building visual interfaces that are easy and intuitive to learn, capable of soliciting users decisions to produce quality results (prompts), and provide visual feedback on the system status so that users are able to take control of the problem solving process (perception leads to action). We apply results from our research in usability design and information visualization to achieve these goals. Two issues are essential in each of the interface design process that we have done:

1. What is the optimal level of data representation (abstraction) for visualizing information about a CSP and its solving process to allow users to make effective decisions on their input (preferences, constraints and tradeoffs in the solving process), and
2. What are right abstractions for the user to manipulate so that he can readily express the required information?

In this paper, we consider interactions with the following three types of abstractions that occur in the CSP problem-solving model:

a. *value assignments*: the user applies preferences, tradeoffs or constraints by selecting value assignments or restricting domains of individual variables.
b. *solutions*: the user obtains a set of solutions of a CSP, visualizes it in task context or tradeoff space in order to introduce additional constraints, criteria, or simply select some optimal ones satisfying his current task needs; the user does so until the result is satisfactory.
c. *problem structure*: the user manipulates constraints, variable or value orderings in order to influence the speed of resolution, the quality of solutions, and how they were obtained. This abstraction model is most appropriate for CSP users, rather than novices.

1.3 Organization of this paper

The methodology for designing user interaction with a CSP solver generally starts from a specification of what input the user has to provide and in what task contexts, and then searches for a design that provides the right mapping between perception and action. In this paper, we develop principles for this design and illustrate how they were obtained from experiences with several implemented interactive CSP systems:

1. Vacation Planner - a program for configuring vacation packages according to users' criteria;

2. ICARUS - a program for reassigning aircraft to flights;
3. ISY-travel - an interactive travel planning application;
4. COMIND - an interactive constraint-based design tool.

We conclude this paper by listing the design and usability principles thus obtained. They provide an initial catalog of these rules which we hope to expand in future work.

2. Specifying Preferences

A very common interaction need is for the user to specify preferences for individual variable assignments. A natural way to do this is for the user to choose value assignments and use the computer as a bookkeeping mechanism that maintains consistency and provides the right background information. We now show two examples of systems that allow users to manually choose values for variable assignments.

2.1 Vacation Planner: Electronic Catalogs

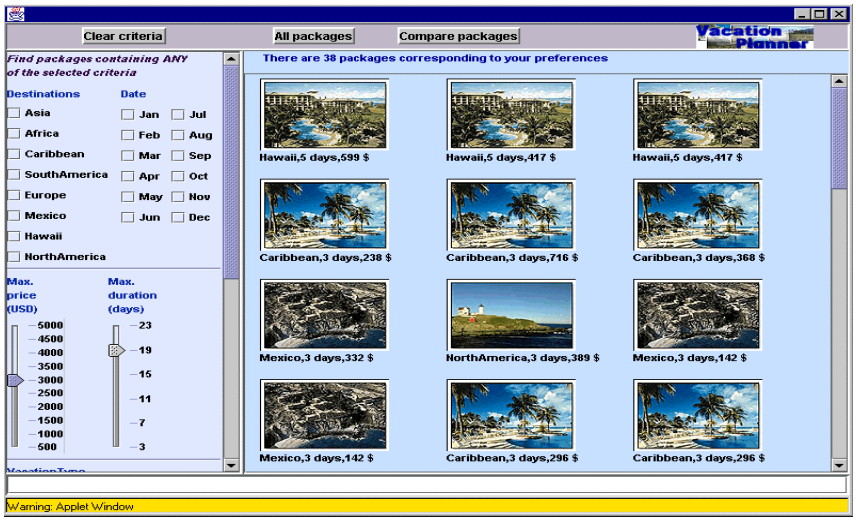

Fig. 1. Users specify any search criteria, in any order in Vacation Planner

Vacation Planner (Figure 1) is an interactive system for searching vacation packages by modeling properties of a vacation such as price, destination, duration, and date of departure as variables, and criteria as constraints in a CSP formalism [5]. A complete instantiation of all criteria without violating any constraints (user specified or system provided) specifies a vacation package. Users can start choosing the assignment of values of any criteria, in any order. For each choice made, the system dynamically displays the results by propagating chosen values to the rest of the CSP. For example, choosing a price range less than 3000 will eliminate the possibility of vacationing in Hawaii (packages all originate from a European city and

include airfare). This allows users to immediately evaluate and perceive how their choices affect the calculation of other values by the system.

Vacation Planner's approach combining CSP and usability principles stand in contrast to many other online electronic catalog systems, such as Personalogic. Users of Personalogic must first answer a set of questions in a fixed order, and when all the questions are completed, the system retrieves the corresponding products in the database. Consequences of these choices only become apparent when data is actually retrieved at the end, and backtracking to the preferences requires restarting the process.

In Vacation Planner, users can search more effectively by incrementally applying preferences and backtracking dynamically when the results are unsatisfactory. By providing the user with the same heuristics that make CSP search efficient, Vacation Planner and similar systems provide for a much more efficient overall solution process. Two important user behaviors can be observed:

1. When a criterion eliminates desirable vacation packages, users are eager to make compromises on preferences stated earlier. In our informal user study, we observe that price is the most frequently manipulated criterion depending on problem context. This confirms also findings from [6];
2. As users add more criteria, there are fewer and fewer solutions. Eventually, addition of one more criteria will produce no results in the search process. Users then backtrack in non-chronological order.

These behaviors show that user indeed use the flexibility provided by the CSP mechanism to find solutions more effectively.

2.2 ICARUS – Aircraft Reallocation

Airline companies have a tight assignment of aircraft to the flights in their schedule. During maintenance period or when an aircraft has a mechanical problem, the original assignment has to be revised to respond to the new situation. We have modeled aircraft assignment as a CSP, and reassignment of aircraft thus requires finding alternative solutions to the CSP. The most important criterion is of course that the revised schedule should be similar to the original assignment to avoid extensive perturbation, but there are numerous hidden constraints and preferences. For example, some aircraft are more fuel-efficient than others, and are preferred for flying longer routes. Aircraft are sometimes registered in different countries, and an airline may be allowed to fly only those registered in certain countries to certain destinations. These preferences and hidden constraints are highly variable, and difficult to model and maintain in an automatic system.

ICARUS ([11]) is a flight rescheduling system that uses CSP techniques to calculate resource equivalence classes when an airplane can no longer fly its designated route. It usually generates hundreds of different solutions, and the user must choose the most appropriate one.

Even though the underlying problem is a CSP, it turns out that best reassignments usually differ quite strongly in the choices that need to be made to select them, i.e. in the aircraft and flights that are being reassigned. Thus, in this case there are dependencies between the different choices, and it is not possible to let users make

choices in any order. Instead, ICARUS uses a treemap display to support users decision process. The treemap is based on a decision tree that groups solutions with similar reassignments using the ID3 algorithm. Users are able to incrementally select a set of desirable aircraft involved in the final reassignment solution by following the tree structure. For example, figure 2 shows four top-level groups of solutions, each under the heading of an aircraft for flying the set of flights SR100 and SR101 shown on the top. The more surface area there is under an aircraft (e.g., IGF), the higher the number of solutions there are involving that aircraft, indicating a high degree of compatibility of that aircraft to the one being replaced. The shading of each tile indicates the complexity of the solution. Darker squares indicate solutions involving many aircrafts in the reassignment, thus not desirable if minimal perturbation is required. To choose the "right" solution, users first perceive the overview provided by the treemap, decide on a particular area under a title, then drill down to details. Once a preference for the current reassignment is expressed, the display moves down in the tree structure to the next common reassignment.

We can observe here that while it is not possible to let user make their choices in any order, the use of consistency techniques to give an idea of the number and complexity of solutions entailed by a certain choice still give an important efficiency advantage.

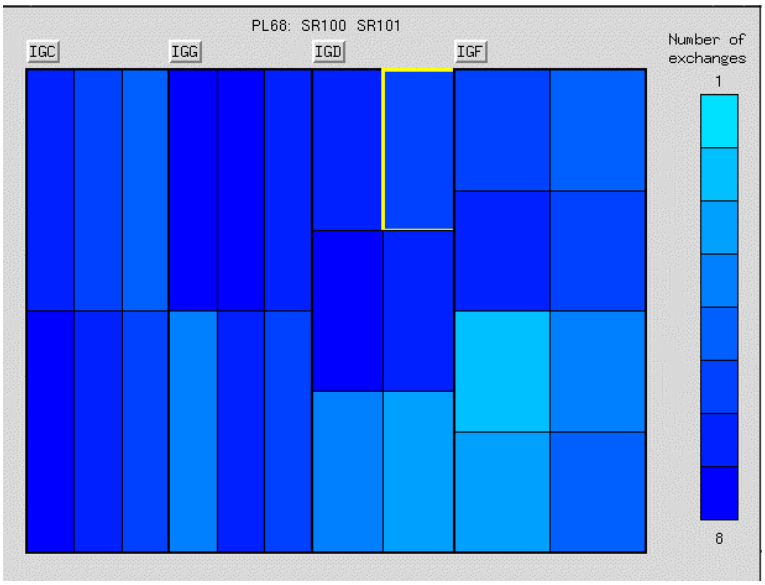

Fig. 2. Treemap used to indicate preferences for aircraft reassignment

2.3 Principles for specifying preferences

Indicating preferences is the most straightforward way for users to interact with a CSP solving process. We observe first that the process requires the user to make decisions on value assignments while considering their impact on the space of available solutions. The right abstraction for interaction is thus a mapping:

Value assignments -> solutions

We propose two principles that have contributed to the success of the systems we have implemented:

1. Order independence: a CSP formulation identifies a fixed set of variables representing decisions to be made in a search process. These decisions can be made in any order. When users' input concerns value preferences or hidden constraints, it has been shown empirically that having users choose the order is more efficient than having the system going through a rigid interviewing process.
2. Immediate feedback: Constraint propagation and consistency techniques can be used to show the immediate effects of users' choices, and thus empower them with similar advantages as provided by consistency maintenance from automated search techniques. This is particularly useful when users' inputs are preferences on individual value assignments.

Vacation Planner is an example of a program that embodies both principles, and in fact there are many configuration systems that have similar functionality. In empirical studies, it has been found that the speedup in human decision making by allowing decisions (and also backtracking) in any order can be up to 50-fold, which is enormous for human-computer interaction.

Jackson and Havens [4] describe an approach similar to Vacation Planner which allows users to interactively assign values to variables. A notion of degree of commitment has been proposed, which is a property of how users choices are retained. Algorithms such as a modified version of Ginsberg's dynamic backtracking algorithm have been shown to satisfy this property.

3. Specifying Hidden Constraints

Specifying preferences involves *selecting* individual value assignments. Conversely, users could enter hidden constraints that would *rule out* entire ranges of values or value combinations. This is complementary to entering preferences, and leads to similar design principles.

3.1 Icarus: expressing hidden constraints

Besides selecting preferences for values that are part of a solution, ICARUS also includes a mechanism for rapidly entering hidden constraints, in particular constraints on solutions that would be unacceptable.

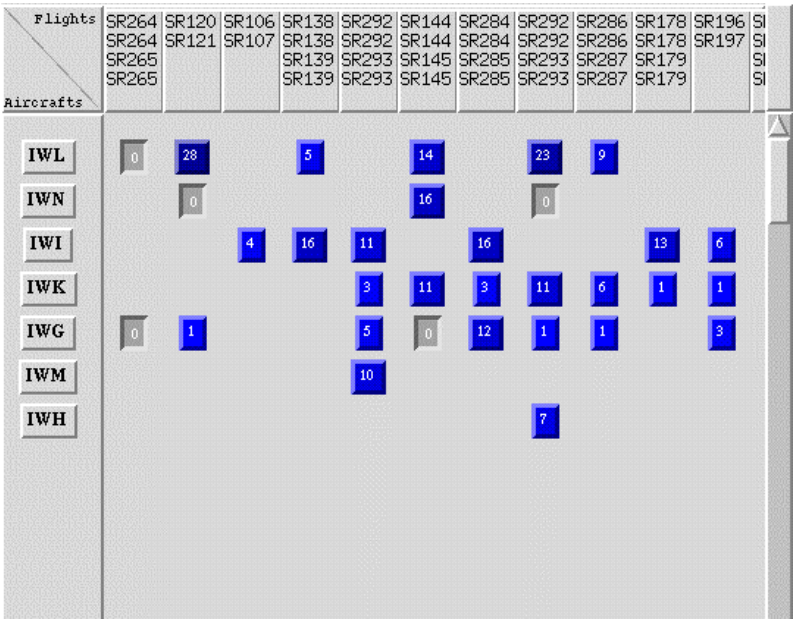

Figure 3: Matrix display for expressing hidden constraints on aircraft allocation

Figure 3 shows an example of the visualization used to express hidden constraints. It is a matrix that shows combinations of aircraft (identified by three letters of their registration) on the vertical axis and flight rotations (sequences of flights from and back to the home base) on the horizontal axis. Each entry in the matrix shows the number of solutions where the particular aircraft would end up being assigned to that particular rotation in rescheduling. By clicking on the square, the user can express a hidden constraint ruling out this particular assignment, thus reducing the number to 0 (5 entries in Figure 3 have been ruled out).

For users of this system, it was very important to see what value assignments actually occur in solutions so that examination could be limited, and also to verify that the effects that the hidden constraints had on the space of solutions.

3.2 ISY-travel: travel planning using CSP

ISY-travel based on the SmartClinent technique [8,13] formulates travel planning as a constraint satisfaction problem. Constraints are formed both by the available means of travel (flights, hotels, car rental), integrity constraints (arrival at destination before departure) and user preferences (avoid leaving at 6am). ISY-travel assigns each constraint a weight and performs an optimization that produces the 30 best solutions with the least penalty. Several visualization and interactivity methods have been designed to augment visual affordance, enable them to discover hidden constraints, express contextual constraints, and formulate tradeoff criteria in the solution space.

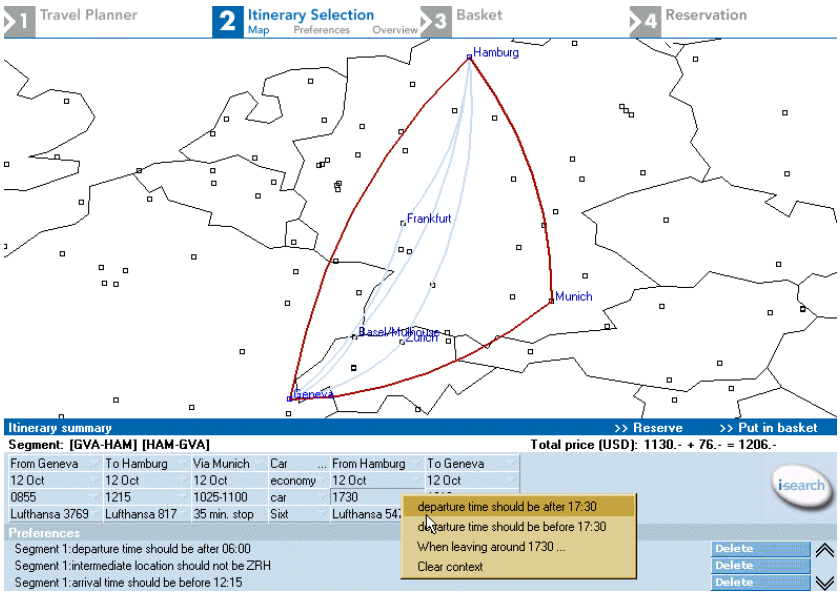

Figure 4: posting constraints to narrow the solution space

Figure 4 shows an example of adding a constraint on the departure time for the return trip at the destination. Constraints can be posted on any individual attribute (e.g.,"departure time cannot be before 6am") or on pairs of attributes (e.g.,"when departure is from Zurich, the flight cannot leave before 10:00 am"). By clicking on the "I-search" button, the CSP search can be run again to obtain a new set of 30 results that are optimal under the new criteria.

The design of ISY-travel was motivated by observations that users were unable to articulate hidden constraints up-front. For example, people would not state that they do not want to transit in a certain airport until confronted with solutions where that airport was present. At the same time, it was important to be reassured that a new constraint would not wipe out the solution space by seeing other solutions that are part of the solution space.

3.2 Principles for specifying hidden constraints

The major difficulty with hidden constraints is that users are usually not aware of them up-front. Thus, systems, which simply ask users to state their constraints, cannot obtain a complete list. In order to make constraints emerge, it is important to show users both examples of solutions where constraints could be violated, and a context of the range of available solutions.

To support such reasoning, the abstraction of interaction is a mapping

Solutions -> value assignments

showing the value assignments actually present in solutions and that might violate hidden constraints.

Therefore, we propose two further design principles particularly applicable when specification of hidden constraints is required:

3. React to examples: Most users are not aware of or cannot articulate constraints. The input of hidden constraints can be more effectively solicited by asking users to critique examples of violated constraints.
4. Show context: Most users like to judge for themselves whether their current choice of solution is the most optimal one. It is thus best not to propose a single optimal solution, but to show a set of possibilities (30 has been found to be a good number). Furthermore, overview techniques have been shown extremely effective for the visualization of solution sets, enabling users to select preferences on value assignments, or make tradeoffs.

O'Sullivan *et al* [7] describes machine learning techniques to help users formulate new constraints. The process involves asking users to provide acceptable solutions (positive examples) from which the system attempts to generalize the constraints exemplified. When the user finds a generalization not acceptable, then a negative example can be inferred to refine the version space of the constraint being acquired.

Bowen [2] suggested a formal notion of interactive CSP, motivated by the question "how can we introduce additional constraints so that only a single solution is generated." A specialization CSP is a set of these constraints. Our work can be seen as a form of specialization CSP in terms of adding hidden constraints.

4. Specifying tradeoffs

In practice, it is unlikely that all of a user's constraints can be satisfied simultaneously. Instead, it is very common that tradeoffs and compromises between criteria must be made in order to make a problem solvable. Even when a problem does have many solutions, it is necessary to convince the user that one of these solutions is the best among the alternatives. We have studied this process in two scenarios.

4.1 ISY-travel

Another way of interacting with ISY-travel is to select a single solution in the final set. From our user study, this is most optimally done in the tradeoff map shown in Figure 5, where solutions can be visualized according to two different criteria, each of which can be chosen by the user. The tradeoff map allows visualizing the tradeoff between different criteria, such as how much extra has to be paid for a convenient departure time or a shorter travel time. Details on each solution can be obtained by simply clicking on its icon in the map.

This visualization allows people to get an idea of what the costs of certain tradeoffs are and thus lets him or her specify them in an informed way. It makes it easy to see relations between parameters that are implied by the CSP. For example, in Figure 5 it

is obvious that there is little relation between travel time and price, i.e. it makes no sense to offer to pay more in order to obtain a shorter travel time. This is true because the example is for a trip within Europe; for a trip in the US, it is in general possible to obtain a shorter travel time by paying a higher fare.

By varying the parameters used for the axes of such a display, users can get an idea of how to set the requirements for different criteria. This allows them to make informed decisions on what importance to give to certain parameters, for example that it may not be useful to relax constraints on the price, but rather allow a different airline or transfer airport to obtain the right results.

Tradeoff maps are also very useful when users have to select a chosen solution from a set of possibilities, as they efficiently convince users that the choice they are making is indeed the optimal one.

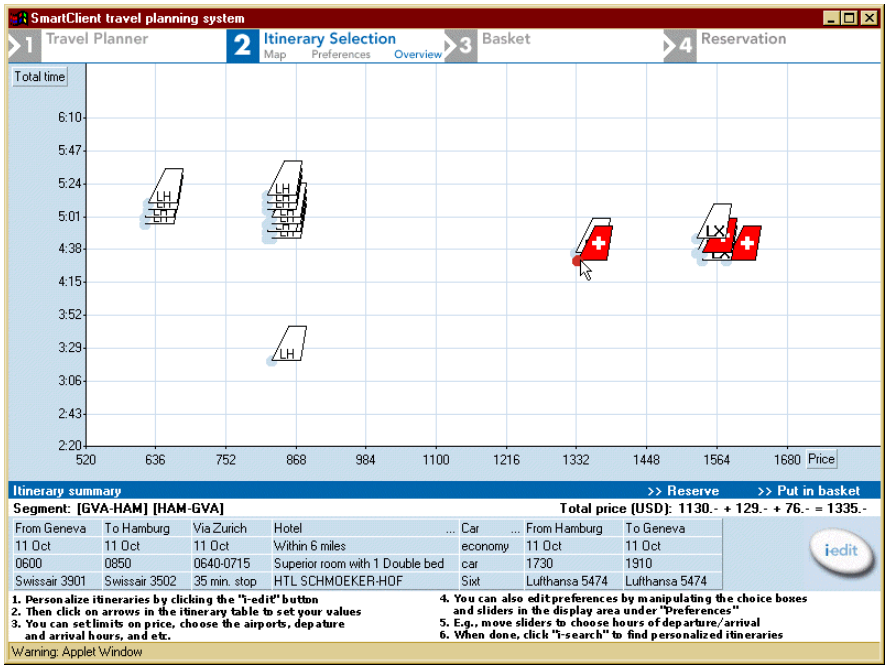

Fig. 5. Tradeoff map for two parameters: price vs. travel time

4.2 COMIND

COMIND is a generic interactive CSP tool with applications to conceptual design of industrial products [9,10]. It helps designers define and evaluate the initial design space by using CSP algorithms to generate sets of feasible solutions. COMIND treats three aspects of a CSP solver: search process visualization, interactive analysis of Pareto optimality of solution space, and user-involved conflict resolution procedures.

Figure 6 shows two different kinds of visualization provided in COMIND:

- the conflict resolution lattice indicates the degree to which certain combinations of constraints rule out potential value assignments; in addition, these potential assignments are visualized in solution space to stimulate users to reformulate these constraints;
- the tradeoff map allows visualizing a subset of the solutions with respect to two different parameters (similar to the one of ISY-travel);

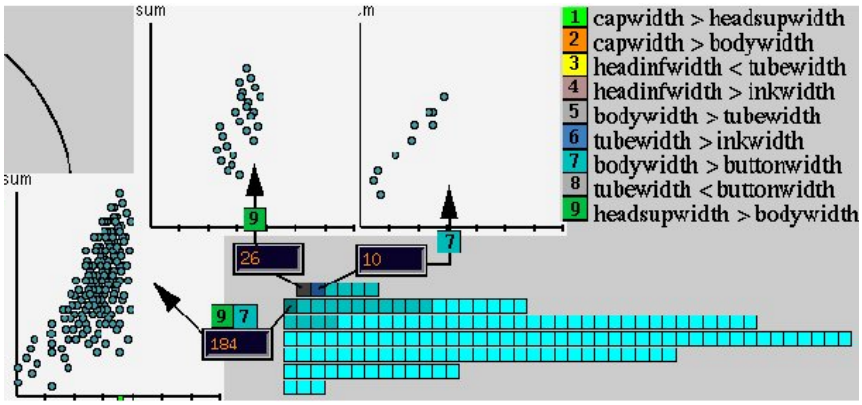

Fig. 6. Lattice's visualization is used together with the tradeoff view for browsing potential solutions.

This is a design scenario of a pen, composed of a cap, a body, a button, and a tube in addition to several other variables. There are 9 constraints that have been expressed, and even though they already allow two solutions, the user might wonder if it is possible to generate others by suitable modification of the problem.

The conflict resolution lattice, shown on the bottom right of Figure 6, shows a square for every subset of the constraints. The topmost row has 6 squares indicating individual constraints, the next has combinations of 2 constraints, and so on down to the bottommost one where each square stands for a different combination of 7 constraints. Only constraints or combinations of constraints that rule out a significant number of valuations are shown as squares, so not all combinations are present. In order to be displayable in a small area, the subset/superset relations in the lattice are only shown on demand: when the user clicks on a square, its sub- and supersets also light up.

The color of the square indicates how constraining it is: a black square indicating that the particular combination of constraints is not compatible with any solution, dark blue indicating few solutions, and light blue for a high number of solutions. By clicking on a square, the user obtains an indication of what combination of constraints it stands for, and can inspect the potential solutions ruled out by this combination of constraints. The simplest display is a box giving the number of solutions ruled out. A

more complex display is a two-dimensional tradeoff map that shows the distribution of the ruled out potential solutions according to two chosen parameters.

When there are multiple blocking sets, users naturally want to choose the easiest one to relax. The sets represented by the top row are smaller than those underneath. The smaller the set is, the easier it is to modify. Additionally, the darker the square, the more solutions can be obtained when constraints are relaxed. The general heuristic is thus to find the smallest (top most) and darkest blue set of constraints to modify. Furthermore, this visualization is linked to the map of solutions as described in the section on Tradeoff Map. Each square on the lattice will invoke secondary windows to display the potential solutions that could be obtained if the constraints would be relaxed. The advantage of this coupling is to avoid doing the entire search process unless the users are certain about the quality of solutions that they will obtain.

The conflict resolution lattice and its associated displays allow a user to quickly get an idea of what constraints are preventing the system from obtaining the solutions he is looking for, and consequently for how the set of constraints should be adjusted to improve the results. In the example of Figure 6, we realize that the sets {7} and {9} on the top most row look promising since they are small and can potentially yield 10 and 26 valuations respectively. Further, the tradeoff map shows that the valuations ruled out by the set {9} are ranking high in the tradeoff analysis. Thus one can easily try to either eliminate that constraint or relax it.

To summarize this section on over-constrained problems, we list the type of inferences representing different reasoning tasks and the corresponding results we can get from visualizing a lattice of conflict sets:

- Is the given CSP problem over-constrained: a single or several black squares in lattice
- Which one of the conflict sets to relax: either use the side window to select the most optimal one, or look up in the constraint definition to find the most appropriate one
- If certain conflicts are removed, the characteristics of the potential solutions that can be obtained are reflected in the tradeoff map, giving people a clear search criterion.

These features allow people to efficiently select the best tradeoffs, i.e. constraints or sets of constraints to relax.

4.2 Design principles for specifying tradeoff information

We have seen on the two examples that tradeoffs can be made in two forms: by explicitly selecting a solution after comparing it with others, or by manipulating the constraints that allow or do not allow certain valuations. It is clear that explicitly selecting a solution is limited to the final stages of the decision process, and does not work if many criteria need to be considered simultaneously.

We have seen in COMIND that explicit manipulation of constraints and constraint sets can result in much more powerful possibilities for making tradeoffs. Coordination with tradeoff maps is particularly useful to identify which sets of constraints are best to relax. To allow such decisions, the interaction abstraction should show the relation

Problem structure <-> solutions

For this abstraction, we formulate the following 3 design principles:

5. Visualization of tradeoff in solution space – users' notions of optimality criteria change over time, depending on the emergence of other criteria, and the characteristics of the solution space itself. An optimal way to visualize the space is to use 2D scatter plots that allow users to see a solution in relation to others in different tradeoffs, and to understand the relations implied by the CSP.
6. Tradeoff on constraints: when users have to make tradeoffs, for example to solve overconstrained problems, doing this at the level of constraints is more effective than at the level of individual solutions. Constraints affect all solutions uniformly and are more efficient than indicating preferences between the potentially huge number of individual solutions.
7. Coordination of conflict resolution in CSP with tradeoff analysis – when a CSP does not yield any valid solutions, visualization of the conflict space in the form of lattices effectively guides users to areas where it is easiest to modify the constraints responsible for conflicts. For example, a smallest set of conflicting constraints is often easiest to fix. On the other hand, a visualization of solutions in the tradeoff space can stimulate users to fix those constraints whose resolution will yield the most satisfactory solutions.

Freuder and O'Sullivan ([3]) have considered issues of modeling tradeoffs. Similar to our goal of explaining conflicts in the solution process, Wallace and Freuder ([14]) proposed a different technique for explanation. An explanation is defined as a set of constraints resulting from a set of labels (assignments of values to variables). Some discussions of semiotics were given regarding semantically equivalent content and the preciseness of the signs employed. As indicated in the article, further empirical studies will have to be conducted to verify the effectiveness of their visual representations. As another example of how human problem knowledge can guide search, Anderson et al. ([1]) show how the performance of local search algorithms can be improved by having users indicate suitable starting points.

6 Principles and Conclusions

We believe that we have covered many relevant scenarios, although by no means we claim that our categories are exhaustive. We have made a first step and hope that the set can be extended by further categories and that further principles can be uncovered in the future.

Like problem-solving, user interaction offers a vast range of possibilities that depend strongly on particular applications. Constraint satisfaction has provided a useful framework for establishing general problem-solving principles like consistency and variable ordering. In this paper, we have shown that it can define similarly strong principles for user interaction with a problem-solver, and shown several such principles derived from our research.

References

1. Anderson, D., Anderson, E., Lesh, N., Marks, J., Mirtich, B., Ratajczak, D., and Ryall., K., *Human-guided simple search*, in *Proceedings of the National Conference on Artificial Intelligence*. 2000, AAAI Press. p. 209-216.
2. Bowen, J., The (Minimal) Specialization CSP: A basis for Generalized Interactive Constraint Processing, in Proceedings of Workshop on User-Interaction in Constraint Processing at CP-2001. 2001.
3. Freuder, E. and O'Sullivan, B., Modeling and Generating Tradeoffs for Constraint-based Configuration, in Proceedings of Workshop on User-Interaction in Constraint Processing at CP-2001. 2001.
4. Jackson, W.K. and Havens, W.S., Committing to User Choices in Mixed Initiative CSPs, in Scandinavian Conference on AI. 1995. p. 239-248.
5. Jurca, A. and Pu, P., Algorithms for Online Vacation Planning. 2001, Swiss Federal Institute of Technology: Lausanne. p. 1-12.
6. Morris, J. and Maglio, P., When Buying On-line, Does Price Really Matter?, in Proceedings of the Conference on Human Factors in Computing Systems (CHI 2001). 2001, ACM Press: Seattle, WA.
7. O'Sullivan, B., Freuder, E., and O'Connell, S., Interactive Constraint Acquisition, in Proceedings of Workshop on User-Interaction in Constraint Processing at the CP-2001. 2001.
8. Pu, P. and Faltings, B., Personalized Navigation of Heterogeneous Product Spaces using SmartClient, in International Conference on Intelligent User Interfaces. 2002, ACM Press.
9. Pu, P. and Lalanne, D., Interactive Problem Solving via Algorithm Visualization, in Proceedings of the IEEE Information Visualization Symposium. 2000, IEEE Press.
10. Pu, P. and Lalanne, D., Design Visual Thinking Tools for Mixed Initiative Systems, in International Conference on Intelligent User Interfaces. 2002, ACM Press.
11. Pu, P. and Melissargos, G., Visualizing Resource Allocation Tasks. IEEE Computer Graphics and Applications, 1997. **17**(4).
12. Schneiderman, B., *Direct Manipulation Versus Agents: paths to Predictable, Controllable, and Comprehensive Interfaces*, in *Software Agents*, Bradshaw, J.M., Editor. 1997, MIT Press. p. 317-345.
13. Torrens, M., Faltings, B., and Pu, P., SmartClients: Constraint Satisfaction as a Paradigm for Scaleable Intelligent Information Systems. International Journal of Constraints, 2002. **7**: p. 49-69.
14. Wallace, R.J. and Freuder, E.C., Explanations for Whom?, in Proceedings of Workshop on User-Interaction in Constraint Processing at the CP-2001. 2001.