Interactive Problem Solving Via Algorithm Visualization

Pearl Pu
Database Lab, Computer Science Department
Swiss Federal Institute of Technology
LBD/DI EPFL
1015 Lausanne, Switzerland
Pearl.pu@epfl.ch

Denis Lalanne*
LIA - CERI, Université d'Avignon et des Pays de Vaucluse
339 chemin des meinajariès
BP1228, 84911 AVIGNON cedex 9, France
denis.lalanne@lia.univ-avignon.fr

Abstract

COMIND is a tool for conceptual design of industrial products. It helps designers define and evaluate the initial design space by using search algorithms to generate sets of feasible solutions. Two algorithm visualization techniques, Kaleidoscope and Lattice, and one visualization of n-dimensional data, MAP, are used to externalize the machine's problem solving strategies and the tradeoffs as a result of using these strategies. After a short training period, users are able to discover tactics to explore design space effectively, evaluate new design solutions, and learn important relationships among design criteria, search speed, and solution quality. We thus propose that visualization can serve as a tool for interactive intelligence, i.e., human-machine collaboration for solving complex problems.

1. Introduction

In Douglas Adams' novel "The hitch hiker's guide to the galaxy," the computer Deep Thought was put to work for 7.5 million years to solve "the ultimate question of Life the Universe and Everything." His answer, 42, rightfully disappointed its designers who had expected a more comprehensive explanation. Many computer users face similar situations every day: complex algorithms for scheduling, configuration and design turn out supposedly optimal answers without justification. If users could also gain an understanding of how solutions were obtained, and why they are the best, they could tune the behavior of their programs to obtain solutions of much better quality. Here we consider the case of conceptual design using the formulation of constraint problem solving and a set of automatic search methods. We examine the issues of learning, discovery and control of strategies by novice users. We focus on three important tasks in conceptual design: solution search, tradeoff analysis of alternative solutions, and discovery of new solutions.

Conceptual Design

"Design is not description of what is, it is exploration of what might be [16]." But the space quickly becomes large and the set of interacting design criteria very complex, making it difficult for a designer to effectively evaluate the different possibilities. Computational assistance can offload some of the cognitive tasks from designers so that they can concentrate on the most creative aspects.

Designers start the definition of a search space for a new product by first identifying a set of key parameters. Then they enumerate a list of possible values for each design parameter. Finally, to put everything together, they combine all values into a set of coherent and consistent design alternatives taking into consideration design rules, customers' criteria and preferences. Such a design process has been used not only for conceptual design [17], but also configuration design [6], land use design [18] and industrial product design. We employ constraint satisfaction problem solving (CSP) techniques [23] to formulate and solve conceptual design problems. CSPs are known to be NP-complete. Many efficient algorithms exist, but they are only optimal when the right context is found. Algorithm visualization is useful in presenting right opportunities to users in order for them to select the best strategy for a given problem.

2. Related works

Algorithm visualization is about presenting the workings of complex algorithms in visual forms, often with animation [20]. The benefit is to offer users with a significantly fast and intuitive understanding of algorithm

behavior by taking advantage of the high bandwidth communication channel between the display, the human fovea and perceptual inference [3]. Research work in this area has given important results in algorithm learning [1], software maintenance [8], designing and analyzing concurrent algorithms [13], and software engineering.

We use algorithm visualization not only to show the workings of the algorithms, but also to externalize the machine's strategies and the tradeoffs of these strategies. Thus our system helps users discover problem solving tactics and become an expert user of those tactics.

The use of interactive visualization tools to support design is not new. Attribute and influence explorers [21,22] use multiple linked interactive representations to extrapolate dependencies among design attributes which are often governed by a set of abstract mathematical models. Thus untrained users can set a range of values for a design criterion such as cost and ask the system to give ranges of admissible values for other design attributes such as material used and dimensions of the designed pieces. Such interval propagation techniques are useful for engineering design where parameters are continuous and mathematical models that describe their behavior are difficult to interpret for normal users. However, in conceptual design where parameters are both discrete and continuous, combinatorial search methods are needed to generate admissible solutions. In doing so, the internal search state is important and must incorporate users' strategies in choosing the right subspace.

Meanwhile in the information visualization domain, there is increasingly a strong interest in developing methods that support human and computer interaction at the problem solving level. Several examples can be found in document retrieval [11], complex information visualization [4], worldwide web navigation [9], and large data base exploration [10]. According to Duce [7], visualization tools can be generally described as consisting of three components: data access, mapping and rendering. But in order to support visualization of a much larger quantity of data, such three-component architecture is not sufficient, as pointed out by Campo et al [2]. Our work on algorithm visualization proposes to abstract problem solving knowledge using algorithm visualization to allow humans to explore in a much larger information space.

Our paper is organized as follows. We first describe constraint satisfaction problem solving in the conceptual design framework; we then introduce visualization methods for three important steps of conceptual design: design space definition, solution tradeoff analysis, and design conflict resolution; finally, we describe the user evaluation of our system followed by our conclusion.

3. Constraint satisfaction and search

A CSP formulation of conceptual design consists of a set of variables, a domain of values for each of these variables, and constraints on the values. A solution to a CSP is an assignment of values from its domain to all variables so that none of the constraints is violated. Consider a simplified watch design (see [14] for more realistic examples, such as parallel robot design, land use configuration, and industrial product design). The CSP consists of design criteria such as `beauty`, `complexity`, `implementation`, and `usability`. Each criterion takes values ranging from 1(the least) to 5 (the most). It is more suitable to use discrete values such as low, medium, high, and maximum. But designers adopt scalar values as shortcuts. Criterion `beauty` is the aesthetic appearance; `complexity` is the number of pieces needed for the watch; `implementation` is how easy it is to manufacture the final product; and `usability` is how easy it is to put on the watch and manipulate the settings. Designers can now describe solution spaces in terms of what should and what should not be included in the final solutions using rules as follows:

C1 (`beauty` > 3) -> (`implementation` > 3)
C2 (`complexity` > 3) -> (`implementation` > 3)
C3 (`usability` > 3) <-> (`complexity` < 5)
C4 `usability` == 4
C5 `complexity` == 2

Thus the more beautiful the watch is, the more difficult it is to implement (C1). The more complex it is, the more difficult it is to implement (C2). The easier it is to use the watch, the less complex it should be (C3). The watch currently being designed should have usability value 4 (C4) and complexity value 2 (C5).

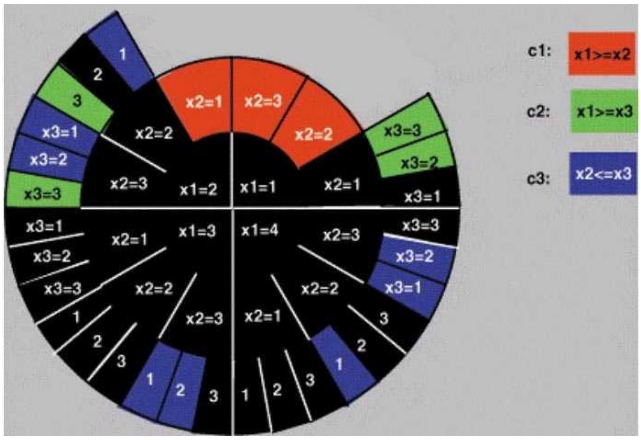

Figure 1: Kaleidoscope showing search result

3.1. Backtrack search

Backtracking is the simplest way to solve a CSP problem. It systematically instantiates one value at a time from each variable domain and checks for constraint violation. When it succeeds in instantiating all required variables, then the enumeration becomes a solution. All admissible solutions can be generated automatically. The following list shows the trace of the backtracking algorithm:

- `beauty` = 1 `complexity` = 1 search fail by C5
- `beauty` = 1 `complexity` = 2 `implementation` = 1 `usability` = 1 search fail by C4
- `beauty` = 1 `complexity` = 2 `implementation` = 1 `usability` = 2 fail by C4 …..

We call each row in the trace a valuation. Unsuccessful valuations are immediately discarded and the algorithm goes back to the last variable for which an alternative value remains to be tried. It continues in this way until all value combinations have been examined.

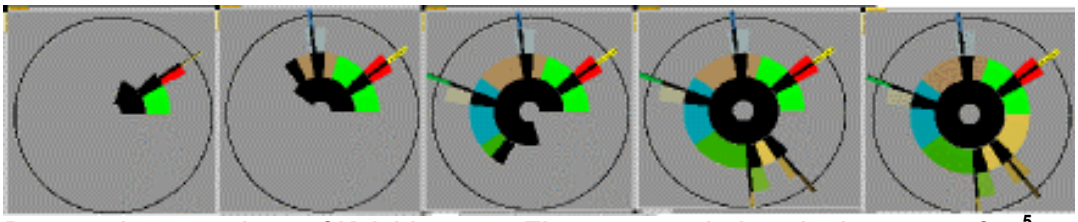

Figure 2: Progressive snapshots of Kaleidoscope. There are 5 solutions in the space of 4^5 possibilities

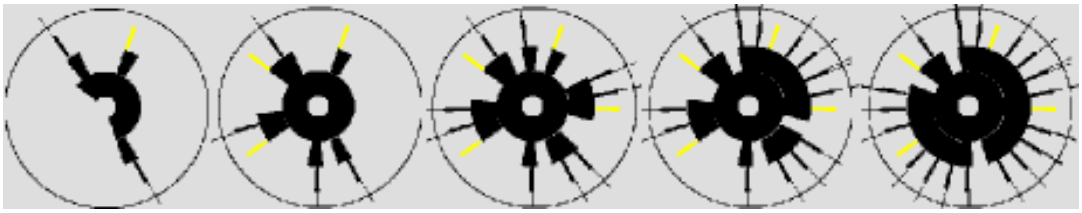

Figure 3: Visualization of a random search based on Knuth algorithm

Figure 1, which we call Kaleidoscope, is the visualization tool animating the internal search state using color patterns. These patterns change as users modify design parameters, similar to kaleidoscope's curious changing patterns as one manipulates the device. Kaleidoscope shows how solutions are obtained. The entire search space is the Kaleidoscope disk which is divided into concentric rings, one for each variable of a CSP problem. The drawing of the wedges starts from the right-hand center line and sweeps the disk in counter-clock-wise direction as shown in figure 2. Each value assignment is visually displayed by a wedge of colors in the disk. Successful assignment is colored in black while unsuccessful assignment is drawn with the corresponding color of the constraint. E.g., assignment (`x1`=1 `x2`=1 and `x3`=1 in figure 1) gives the first wedge in black. Assignment (`x1`=1 `x2`=1 and `x3`=2) results in the first portion of the wedge in black and the second portion in the color that corresponds to `x1>=x3`. Thus a thin black line reaching beyond the outer most circle represents a solution. Figure 2 are successive snapshots of the Kaleidoscope for the watch design example. Black lines represent solutions while color bands show forbidden spaces blocked by the relative constraints. Kaleidoscope can scale well to a CSP of any size. Similar to the circular forms used in [5], we also use a fisheye view technique to visualize local search spaces when a CSP becomes too large. In contrast to [5], our visual display represents a reasoning process, rather than an aggregation of hierarchically organized data.

A phenomenon, called thrashing [15], can occur when search repeatedly fails on a certain combination of values. In Kaleidoscope, the disk has a large slice chopped off by the same color (top part in figure 1). Thrashing behavior signals to a designer that certain combinations of values should be discarded to avoid future valuation. But backtracking is not intelligent enough to provide such preprocessing.

3.2. Random search by Knuth

Our example only used 4 variables, each having a range of 5 values. Unfortunately most CSP problems grow to the size of 20 variables with the number of values ranging in the 10 to 50s, thus too big for backtracking search algorithms to find answers quickly. Our second CSP algorithm is a Monte Carlo search method by Knuth [12], which can be used to explore the search space efficiently although randomly. Figure 3 is a set of snapshots showing the progress of using Knuth algorithm

for a slightly different design problem, which contains more solutions as visualized by the black lines. As depicted in the figure, the algorithm explores different areas of the circle randomly. However, by the first snapshot, the algorithm already obtained two solutions.

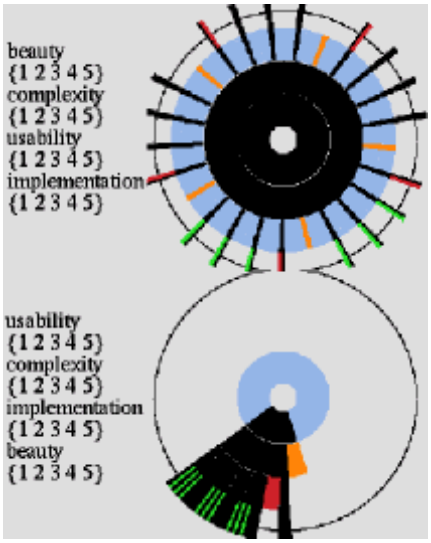

Figure 4: From top to bottom, parameters' order is more optimal in order to accelerate the search

3.3. Search with variable reordering

The third CSP algorithm is called variable reordering. It examines the constraints and picks those that are most restricting. The order of search then starts with those most restricting ones. Consider the watch design example again. The new order is `usability`, `complexity`, `implementation`, and `beauty`. Figure 4 compares the visualization of the same CSP problem by two different algorithms, simple backtracking (top) and variable reordering (bottom). In the second case, there are significantly fewer numbers of unsuccessful valuations, thus making the algorithm much faster to terminate.

3.4. What can users discover in Kaleidoscope?

The color patterns (black lines and color elements) convey important abstractions which guide humans in setting up search strategies. When search space is large and solutions are scarce (called futile space), Knuth's method is most appropriate. When a small set of variables have strong constraints, variable ordering offers significant advantages. For simple problems, backtracking is often sufficient. Not only should users become aware of the different strategies used by the algorithms, they also have to know enough to switch from one algorithm to another depending on context. The program allows humans to stop the current search and change the subspaces if results so far are not satisfactory. For example, if the search process has been blocked by the same color element (thus the same constraint), then thrashing is obviously the cause and we can detect easily the constraint responsible for the thrashing by looking up the color coding. Knuth algorithm is a good cure for thrashing behavior. Kaleidoscope also contains two visual structures that are easily learned and appreciated by users: the spaced out black lines (i.e., solutions) represent solutions of diverse characteristics while concentrated black lines represent more homogeneous solutions.

We can also quickly observe if the search space is abundant or futile by noticing whether full-length black lines are numerous or none has existed so far. In the case of a futile search space, we use color patterns to diagnose the constraints responsible for the lack of solutions.

To summarize, we list a set of strategic points that can help users perform efficient combinatorial search with the help of Kaleidoscope:

- Does thrashing occur and with what type of frequency?
- Are solutions diversified or concentrated in clusters?
- Are solutions abundant or futile?
- If variables are reordered, does solution generation become faster?

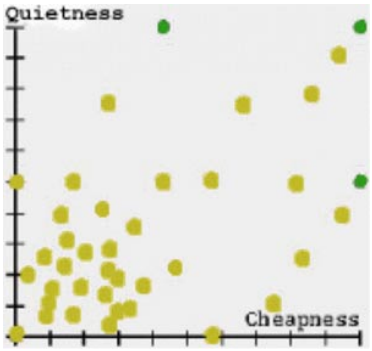

Figure 5: MAP in 2D Pareto space

4. Map and N-dimensional tradeoff

Designers can become bewildered by too many choices. We provide them with visualization assistance to evaluate the solution space in order for them to make intelligent decisions. Users first define a set of criteria, for instance the quality of material used for a product and its manufacturing cost. For two-criteria tradeoff analyses, solutions are mapped to a 2D space where x and y coordinates represent the criteria values and each solution is represented by a node.

Figure 5 shows the solution space of a city planning design problem with criteria on noise factor (`quietness`) and cost (`cheapness`). The node which performs the best on both criteria is called the dominant solution. The nodes lying on the outer rim of the solution space are called non-dominant nodes (darker in figure 5). In most cases a solution map does not contain a dominant node. Thus, tradeoff analysis is necessary in order for users to choose a winner based on their dynamic preference measure. For example, if they prefer to optimize more on noise factor and less on cost, then a point with the highest value on y will be the current winner. While this visualization, called Pareto space [19], is feasible for up to three criteria, many real-world problems call for tradeoff analysis in much higher dimensions.

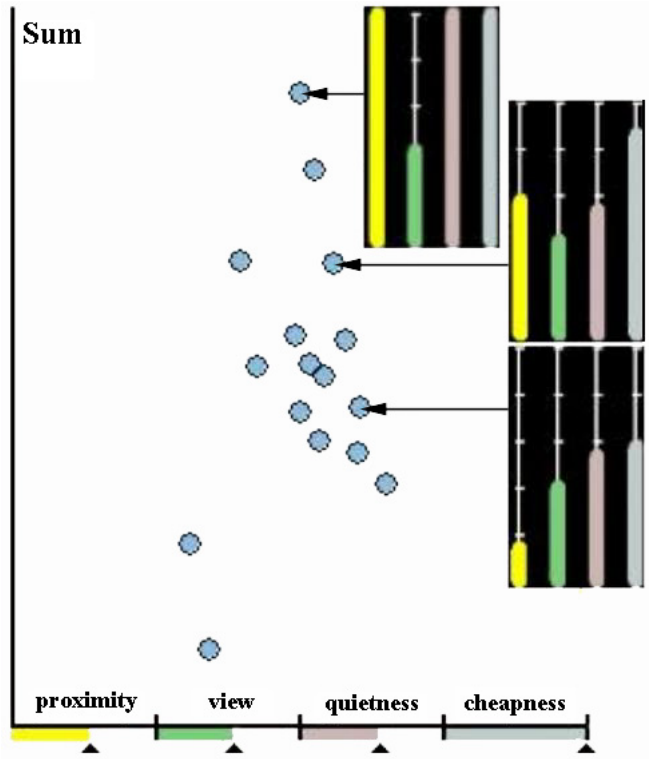

Figure 6: The balance visualization for performing tradeoff analysis in n-dimension.

Our new visualization design, MAP (**M**ultiple **A**ttribute **P**areto), overcomes this limit by combining color patterns, visual structures and interactivity. Figure 6 shows the new design where the x position represents the center of mass of the N criteria values which make up the bars on the bottom, and y position represents the sum of all criteria values. The solution that performs the best overall-speaking is the node with the largest y value. While y represents an absolute performance value, x shows the distribution of the underlying criteria much like a balance. For example, nodes located above the *view* line represent solutions having relatively higher values on *view* compared to other criteria. When tradeoff analysis is required, users can slide to the left or right from the center line depending on his current preferences. There are cases where solutions are pulled by multiple criteria values, but individually these criteria are not distinguishable. In this case interactivity solves this problem by allowing users to click on the solution nodes and examine the details. Another method is to change interactively criteria's weight so that ambiguities disappear. For example, increasing the quietness' weight would pull the optimal solutions toward the right side of the visualization.

The answers that can be obtained from MAP are:

- Is there a dominant solution?
- Are there numerous or few non-dominant solutions?
- Are solutions clustered around a certain area, or more spread out in MAP?

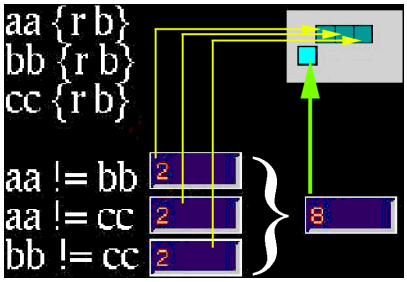

Figure 7: The map coloring problem in a Lattice shown in the upper-right corner.

5. Enemy of design: null solution sets

The opposite of an abundant solution space is the designers' enemy: null solution space. A design problem can become over-constrained very quickly due to the number of variables explored and constraints defined. An over-constrained problem contains one or several sets of constraints defined in such a way that no solution could exist. The conflict elicitation algorithms described in [14] diagnose over-constrained problems by listing a set of constraints which cannot be satisfied simultaneously, or a set of sets of constraints of which at least one set has to be "repaired" in order to give any solutions.

As a simple example of an over-constrained problem, consider coloring 3 mutually adjacent countries (`aa`, `bb`, and `cc`) with 2 colors such that all neighboring countries have a different color. There are 3 constraints which cannot be satisfied at the same time: `aa` != `bb`, `aa` != `cc`, and `bb` != `cc`, where != means that the colors of the countries are not the same. Further observe that if a CSP

contains any over-constrained CSPs, it will not have any solutions either. We define the following notions:

- a constraint set is a conflict set if it does not allow any partial solutions
- a constraint set is the minimal conflict set if no smaller set is a conflict set
- a constraint set cannot allow any solutions if and only if it contains at least one minimal conflict set

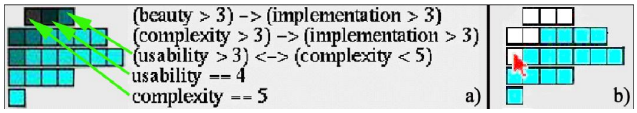

Figure 8: A Lattice and its interactive form

5.1. Interactive lattice

Conflict sets can be analyzed in a visual interactive lattice (shown in the upper right corner in figure 7). Each square is a set of constraints. Sets are further ordered from the top to the bottom by their sizes with the sets on the top-most row being the smallest. When a square is black, it is a minimal conflict set. When it is dark blue, the set blocks a certain number of solutions. The darker it is, the more potential solutions it blocks. The question is if there are multiple blocking sets, as is the case with most problems, then which one should the user focus on first? The sets lying on the top row are smaller than those underneath. The smaller the set is, the easier it is for users to modify the constraints. The general heuristic is thus to find the smallest (top most) and darkest blue set of constraints to relax.

The square on the bottom of the lattice shown in figure 7 is the minimal conflict set of the map coloring problem. By clicking on that set, three subsets are highlighted on the top. Removing the main conflict will give 8 solutions, while removing or relaxing each of the subsets will give 2 solutions, as indicated by the numbers in the small windows. This means that we either allow adjacent countries to have the same color (i.e., removing the constraint !=) or we remove or relax each of the subsets. Since it is not desirable to color neighboring countries with the same color, we opt for relaxing the subsets by adding another color to one of the domains, thus obtaining two solutions. The final map coloring problem is the same as before except there are three color choices for one of the countries.

While normal lattices contain lines to relate sets to their sub or supersets, our lattice is interactive and only displays all subsets when a set is clicked on. In figure 8b, one square becomes white when clicked, so are all subsets highlighted in white. This way we can display a large lattice without the risk of having lines crisscrossing and thus causing visual overloading.

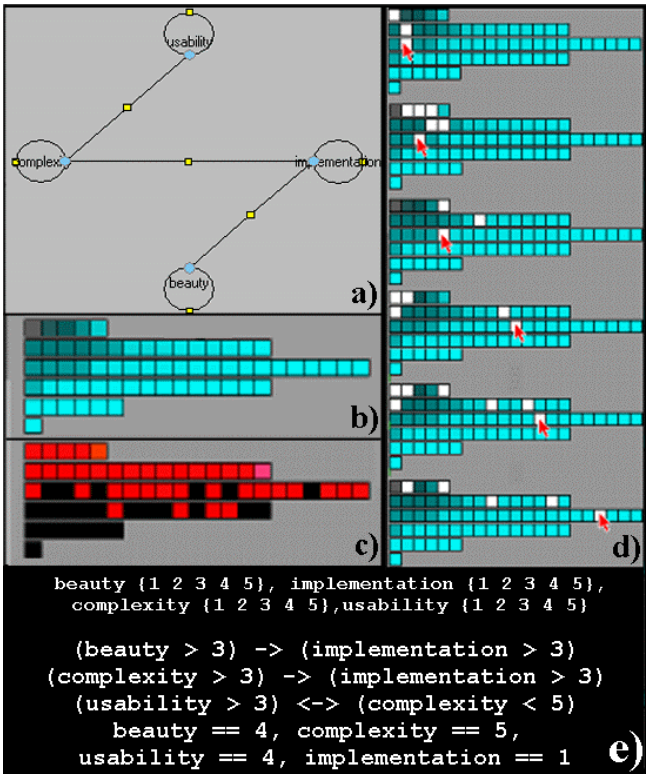

Figure 9: lattice with multiple conflict sets shown as black squares in c)

5.2. Multiple minimal conflict sets

Some problems have several minimal conflict sets as indicated by the black squares on the left part of figure 9. This is the same watch design but with different constraint sets (the constraint network is visualized in Fig 9a):

- (beauty > 3) -> (implementation >3)
- (complexity > 3) -> (implementation > 3)
- (usability > 3) <-> (complexity < 5)
- usability == 4
- complexity == 5
- beauty == 4
- implementation ==1

The black squares in figure 9c correspond to all minimal conflict sets. At least one of the minimal conflict sets has to be relaxed to generate any solutions. Further, if a square on that row is clicked on, Lattice shows the subsets (see lattices on figure 9d). The subsets define parts of the original problems for which solutions exist. This information guides the users to choose the subsets to keep.

For example, clicking on the black square on the first lattice in figure 9d, two subsets are shown. Thus either we keep the first subset or the second. Since the second subset is a larger set, keeping it automatically allows us to keep the largest original problem. By visualizing all conflicts and largest consistent subsets, we offer designers the choice of what to throw away and what to keep, which is the most difficult decision in design and requires experience, gut feeling and dynamic criteria from the designers.

To summarize this section on over-constrained problems, we list the type of queries representing different reasoning tasks and the corresponding results we can get from the lattice visualization:

- Is the CSP problem over-constrained: a single or several black squares in a lattice
- Which 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
- How potential solutions perform in MAP if corresponding constraints are removed?

6. Putting it all together

Figure 10 is a set of design parameters for a pen which consists of the following variables and domains: `capwidth` {3 3.5 4}, `headsupwidth` {2 2.5 3 3.5}, `headinfwidth` {1.5 2 2.5}, `bodywidth` {2.5 3 3.5}, `tubewidth` {2 2.5 3}, `inkwidth` {1 1.5 2}, `buttonwidth` {2 2.5 3}.

We first used simple backtracking in Kaleidoscope and discovered that there was only one successful valuation. A quick application of the Knuth algorithm gave the same answer; the CSP was *futile*. The lattice, shown in the lower right corner of figure 10, gave three dark squares indicating main conflicts responsible for the lack of solutions. The two squares on the top row represent constraints 9 and 7. If they are eliminated, there will be 26 + 10 new solutions. The square on the second row is the combination of constraint 9 and 7. That is, if these two constraints are eliminated at the same time, there will be 184 solutions liberated. Clicking on each square, MAP gives visualization of potential solutions. 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 top squares seem promising, because they are small and can potentially yield 26 or 10 new solutions respectively (+ the one already found). Further they rank fairly well in the tradeoff analysis. It is thus rational to relax the constraint 7 (`bodywidth > buttonwidth`). The design choices are thus re-evaluated with this new consideration. Instead of having the button inset in the body, we will screw it directly to the body. The button width no longer has to be smaller than the body width. The two widths are now equal. The constraint 7 is re-written: `bodywidth == buttonwidth`. In order to find new solutions, the system just re-evaluates the new constraint over the 10 solutions that the previous constraint was forbidding plus the only one that was initially found. COMIND returns 7 solutions among those 11. The largest pen among the 7 returned was the one that we finally selected because it is optimal on the tradeoff visualization (figure 10). This small scenario shows how algorithm visualization can influence the user in his design strategies. For the last few years, COMIND has been used for several design problems (parallel robot design, micro-engineering products, kitchen arrangement, etc.). It helped discover new concepts and different aspects of existing designs, and has been particularly appreciated as a reflective tool.

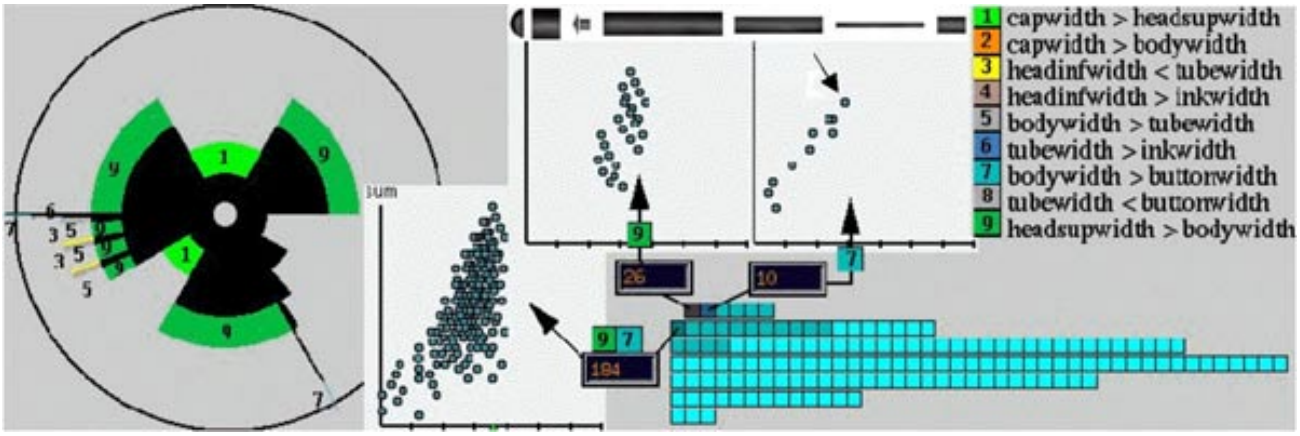

Figure 10: A design of a pen. The Lattice's visualization in the lower right corner can be used in collaboration with the tradeoff view in order to browse the potential space of solutions.

7. Evaluation

We asked 6 subjects to solve two real design problems in COMIND (figure 11), one being under-constrained, and one over-constrained. Each subject was asked to note down their initial problem solving strategies. After they have used COMIND, they were to compare their strategies with those of the machine's. Subjects were all students in our university, either in micro-engineering or computer science. We summarize the results as follows (for more detail, please see [14]): 5 out 6 used the Knuth algorithm to first find out if the problem yielded solutions or was over-constrained. Most of them with a brief explanation of MAP and Lattice could use it as a tool to navigate in the constraint editor to modify the problem and later obtain interesting and optimal solutions. All 6 used the interactive search feature in Kaleidoscope while search was underway and became aware of the role of variable order in search speed after being told the color coding. 4 out 6 solved the two design problems within 20 minutes. In general, we are satisfied with the learning speed of our subjects and the speed of their mastering of the set of search heuristics offered by the machine. They were able to distribute computationally intensive tasks to the various algorithms, and concentrate on finding innovative design solutions or relaxing design constraints to obtain compromises.

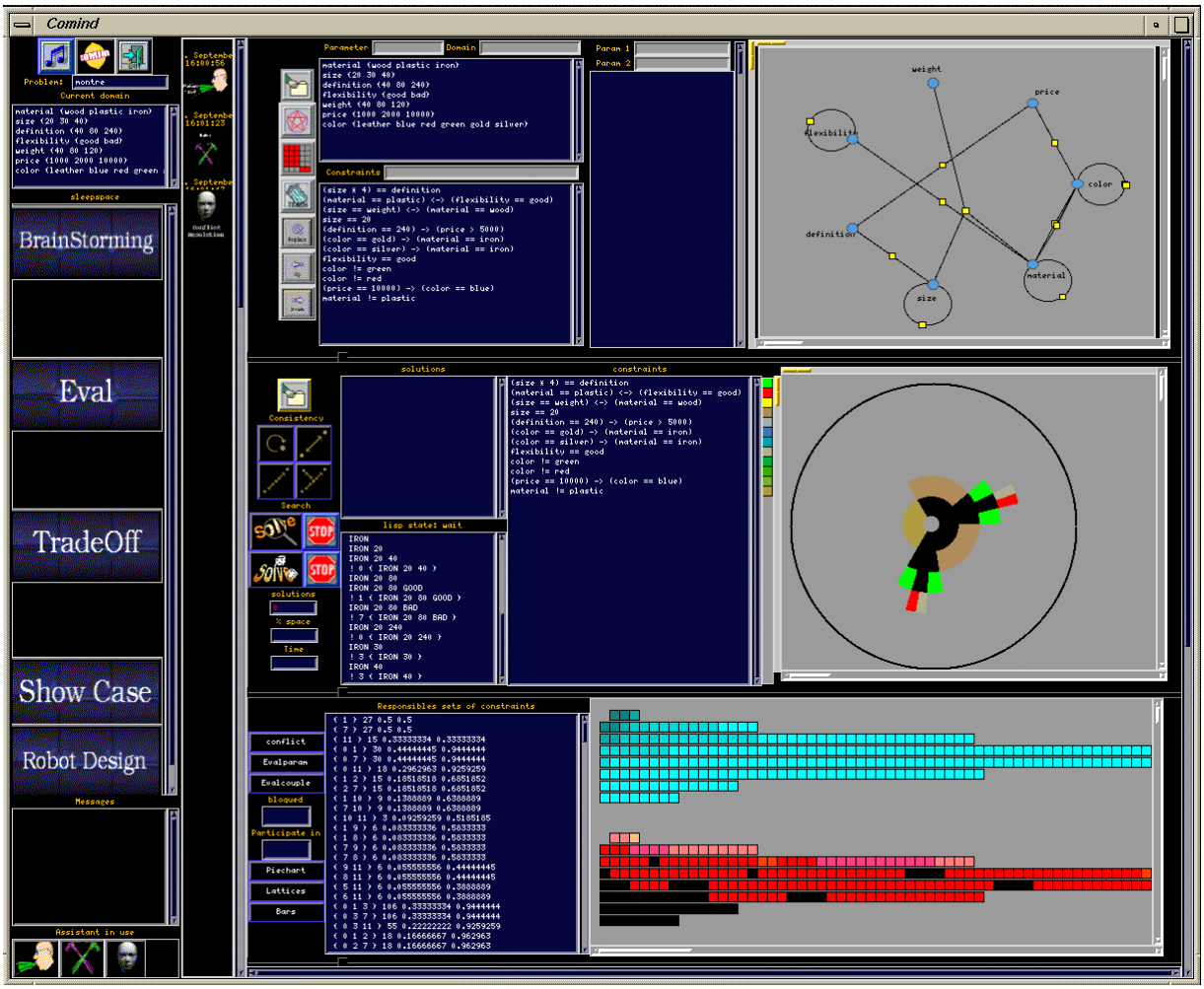

Figure 11: The COMIND system offers a set of computational tools (located on the far left) to define, solve, repair and visualize design problems. Active assistants are located on the lower left corner, while inactive ones are on the left panel. The next column presents the history of the working space. Any tool can be used at any time; there is no predefined order. The main working area on the right provides visualizations of each of the active computational assistants (three at the moment: problem definition, solve, and conflict elicitation).

8. Conclusion

We presented three visualization techniques to help designers explore design spaces, evaluate and discover new design solutions. Our "interactive intelligence" paradigm is in contrast to many automatic methods that keep themselves as intelligent black boxes, often compounding untrained users' frustration of the problem with further misery of the miscomprehension of the results, the Deep Thought analogy. Kaleidoscope, MAP, and Lattice are some of the initial steps towards applying interactive visualization to conceptual design problems where human and machine must collaborate to solve problems that are difficult for either of them. More generalization of our framework is underway for resource allocation and travel planning since both activities use the same CSP formalism. We believe that this type of user-involved intelligent systems are engaging, and that visualization is a good communication medium.

9. References

1. Brown, M.H., and Sedgewick, R. A system for algorithm animation. *IEEE Software*, 2(1): 28-39, January 1985.
2. Campo, M., Orosco, R., and Teyseyre, A. Automatic abstraction management in information visualization systems, In *Proceedings of the Visualization Conference*, 1997.
3. Card, S. Information Visualization. In *Tutorial notes*, CHI'99.
4. Chalmers, M. Design perspectives in visualising complex information, in *Proc 3rd IFIP Visual Databases Conference (VDB.3)*, 1995.
5. Chuah, M. Dynamic Aggregation with Circular Visual Designs. Proceedings of the IEEE Symposium on Information Visualization (InfoVis '98), Triangle Park, NC, October, 1998.
6. Darr, T., Klein, M., and McGuinness, D. Configuration Desing, Artificial Intelligence for Engineering Design, Analysis, and Manufacturing, 12(4), 1998.
7. Duce, D.A. Visualization, In *Proceedings of the Visualization '93 conference*, 1993.
8. Eick, S.G. Maintenance of Large Systems, in Software Visualization, Stasko et al (eds), MIT Press, 1999.
9. Gershon, N. Moving happily through the world wide web, *IEEE Computer Graphics and Applications*, 16(2):72--75, March 1996.
10. Goldstein, J., and Roth, S.F. Using aggregation and dynamic queries for exploring large data sets, In *Proceedings of the Conference on Human Factors in Computing Systems (SIGCHI '94)*, pages 23--29, 1994.
11. Hearst, M.A. Tilebars: Visualization of term distribution information in full text information access, in *Proceedings of the Conference on Human Factors in Computing Systems (CHI'95)*. ACM Press, May 1995.
12. Knuth, E. Estimating the efficiency of backtrack programs, *Mathematics of Computation*, 29:121--136, 1975.
13. Kraemer, E., and Stask, J.T. Creating an Accurate Portrayal of Concurrent Executions, *IEEE Concurrency*, 6(1), pp. 36-46, January /mar 1998.
14. Lalanne, D. Computer aided creativity and multicriteria optimization in Design, Ph.D. thesis (thèse no.1879), Swiss Institute of Technology Lausanne, 1998. Also appeared in doctoral consortium, CHI'97.
15. Mackworth, A.K. Consistency in networks of relations, Artificial Intelligence, 8, 1977.
16. Mitchell, W. Introduction: A new agenda for computer-aided design in the electronic studio. In Mitchell, MaCollough and Purcell (eds), *The Electronic Design Studio*. MIT Press, 1990.
17. Mittal, S., and Frayman, F. Towards a Generic Model of Configuraton Tasks, in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp.1395-1401, Morgan Kaufmann, August 1989.
18. Navinchandra, D. Exploration and Innovation in Design. Springer-Verlag, New York Inc., 1991.
19. Pareto, V. Cours d'économie politique, Technical report, Rouge, Lausanne, Switzerland, 1896.
20. Stasko, J.T., Domingue, J.B., Brown, M.H., and Price, B.A. (eds), Software Visualization, MIT Press, 1999.
21. Tweedie, L.A. , Spence R., Williams D., and Bhogal R. The Attribute Explorer, in *Video Proceedings CHI'94* Boston, April 24th - 28th , ACM Press 1994
22. Tweedie, L.A., Spence, R., Dawkes, H., and Su, H. The Influence Explorer -- a Tool for Design. In *Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, VIDEOS: Visualization, Vol. 2, pp. 390-391, 1996.
23. Tsang, E. Foundations of Constraint Satisfaction, In *Academic Press*, 1993.