Workshop Notes

Third International Workshop on

User-Interaction in Constraint Satisfaction

Held in conjunction with the

Nineth International Conference on
Principles and Practice of Constraint Programming, CP-2003

Kinsale, County Cork, Ireland
September 2003

Editors

Barry O'Sullivan and Eugene C. Freuder

Cork Constraint Computation Centre
Department of Computer Science
University College Cork, Ireland.

Preface

User-interaction issues arise both for users of constraint programming languages and for users of constraint-based applications. The former need assistance in building models and tailoring solvers. The latter need assistance in specifying problems and understanding results. Successful user-interaction is the key to fully exploiting advances in constraint solving abilities.

This *Third International Workshop on User-Interaction in Constraint Satisfaction* should be of interest to researchers who wish seek a deeper understanding of the technical issues associated with supporting user-interaction for constraint-based applications, languages or environments. The workshop should also be of interest to industrialists interested in the state-of-the-art in the area.

These workshop notes comprise 7 papers. Amongst the topics addressed by these papers are: visualizing explanations; question-generation in interactive constraint-based systems; explanations for global constraints; tree-driven automata for interactive constraint processing; formal models of interaction in constraint satisfaction; applications of interactive constraints to real-world problems; and tradeoff analysis for interactive constraint-based configurators.

We would like to acknowledge the Workshop Programme Committee for their assistance in reviewing submissions for suitability for inclusion in the workshop notes. Also, we would like to acknowledge the assistance of the CP-2003 Workshop Chair, Christian Bessière, the CP-2002 Conference Chair, James Bowen, and the CP-2002 Program Chair, Francesca Rossi, for their support and assistance.

August 2003

Barry O'Sullivan and Eugene C. Freuder
Cork Constraint Computation Centre

Organizing Committee

Barry O'Sullivan (Chair), University College Cork, Ireland.
Eugene C. Freuder, University College Cork, Ireland.

Programme Committee

Alan Borning, University of Washington, USA.
Ken Brown, University of Aberdeen, UK.
Hélène Fargier, Laboratoire IRIT, Universite Paul Sabatier, France.
Gerhard Friedrich, Universitat Klagenfurt, Austria.
Pascal van Hentenryck, Brown University, USA.
Ulrich Junker, ILOG, France.
Narendra Jussien, École des Mines de Nantes, France.
Pearl Pu, EPFL, Switzerland.
Francesca Rossi, University of Padova, Italy.
Helmut Simonis, Parc Technologies Ltd, UK.

Contents

Visualizing explanations to exhibit dynamic structure in constraint problems*

Mohammad Ghoniem[1], Narendra Jussien[1], and Jean-Daniel Fekete[2]

[1] École des Mines de Nantes
4, rue Alfred Kastler – BP 20722
F-44307 Nantes Cedex 3 – France
{Mohammad.Ghoniem,Narendra.Jussien}@emn.fr
[2] INRIA – Unité de recherche Futurs
Bâtiment 490, Université de Paris Sud
F-91405 Orsay Cedex – France
Jean-Daniel.Fekete@inria.fr

Abstract. In this paper, we introduce new visualization tools for explanations generated during search in a constraint program. Explanations are a very powerful tool for exhibiting dynamic interactions and relations appearing only during search. Moreover, we show that classical information that can be gathered in standard solvers does not allow retrieving this dynamic behavior thus advocating for the embedding of explanations within existing constraint solvers.

1 Introduction

Explanation-based constraint programming [9] has proven to be effective both for improving search algorithms (as in `mac-dbt` [11] and `decision-repair` [12]) and providing user interaction tools (as in the COINS systems [15], or in [6]).

In this paper, we would like to go one step further and show how adapted visualization tools used on explanations generated during search can help understand constraint solving: discovering static or dynamic structure between constraints or variables emerging during search, discovering inefficient decisions, observing hard resolution steps, etc.

We implemented the generic trace defined in the OADYMPPAC project [14] within the PALM solver [10] and developed a set of visualization tools that we introduce in this paper. We claim that using the information contained in the explanation network helps discover information that is not available when looking only at the structure of the solved problem and even when looking at domain reductions as they are performed during search. Moreover, our tools provide insights into the dynamics of search (as opposed to static exploration of conflicts or search procedures as introduced for example in [16]) and help understand the deep relations dynamically appearing between constraints or variables during search.

* This work has been partially supported by the French RNTL project OADYMPPAC [14].

This paper is organized as follows: after some definitions related to explanations, we present the philosophy of our visualization tools. Next, various examples illustrate the interest of explanations for dynamic analysis of constraint program.

2 Explanations for constraint programming

Solving constraint satisfaction problems is often based upon chronological backtracking algorithms. The main disadvantages of these algorithms are well known: the *thrashing* phenomenon due to the impossibility to remember past failure conditions and to the poor relevance, in general, of getting back to the last choice point.

2.1 Definition

To compensate thrashing, explanation-based solutions were proposed in the literature [7, 9]. An explanation contains enough information to justify a decision (throwing a contradiction, reducing a domain...): it is composed of the constraints and the choices made during the search which are sufficient to justify such an inference.

Definition 1 (Explanation) *An explanation of an inference ($\mathcal{X}$) consists of a subset of original constraints ($\mathcal{C}' \subset \mathcal{C}$) and a set of instantiation constraints (choices made during the search: $d_1, d_2, \ldots, d_k$) such that:*

$$\mathcal{C}' \wedge d_1 \wedge \ldots \wedge d_n \Rightarrow \mathcal{X}$$

$\mathcal{C}' \wedge d_1 \wedge \ldots \wedge d_n$ is called an explanation-set.

An explanation-set e_1 is said to be *more precise* than explanation-set e_2 if and only if $e_1 \subset e_2$. The more precise an explanation, the more useful it is.

2.2 Explanation-based algorithms

Thanks to information about propagation, algorithms such as dynamic backtracking (dbt [7]) know all the instantiations that imply a contradiction, and so, it is able to determine which instantiation should be undone (not necessarily the last one). The instantiation order is then modified to undo this instantiation and only this one (keeping non related inferences made in between).

A drawback of dynamic backtracking is that it does not take advantage of propagation techniques. mac-dbt is an algorithm which allows to maintain arc-consistency (mac [17]) within dbt. This algorithm offers advantages from both filtering and repairing techniques but requires that all filtering decisions are explained (contrarily to dynamic backtracking that only needs explanations for contradictions). Moreover, since the cancelled decisions are not always the last taken choice, the implementation of an explained constraint must support the removal of constraints (or the addition of values to domains) and not only backtracking.

2.3 Computing explanations

The most interesting explanations are those which are minimal regarding inclusion. Those explanations allow highly focused information about dependency relations between constraints and variables. Unfortunately, computing such an explanation can be exponentially costly. We claim that a good compromise between precision and ease of computation is to use the solver embedded knowledge to provide interesting explanations[9]. Indeed, constraint solvers always know, although it is scarcely explicit, *why* they remove values from the domain of the variables. By making that knowledge explicit, quite precise and interesting explanations can be computed.

For example, let us consider two variables v_1 and v_2 whose domains are both $\{1, 2, 3\}$.

- Let c_1 be a first decision constraint: $c_1 : v_1 \geq 3$. Let us assume that the filtering algorithm in use is 2B-consistency filtering. The constraint c_1 leads to the removal of $\{1, 2\}$ from the domain of v_1. An explanation for the new domain $\{3\}$ of v_1 is thus $\{c_1\}$.
- Let c_2 be a second constraint: $c_2 : v_2 \geq v_1$. Value 1 and value 2 of v_2 have no support in the domain of v_1, and thus c_2 leads to the removal of $\{1, 2\}$ from v_2. An explanation of the removal of $\{1, 2\}$ from v_2 will be: $c_1 \wedge c_2$ because c_2 precipitates that removal only because previous removals occurred in v_1 due to c_1.

3 Visualization tools

We introduce here some recent visualization tools that we used to exploit explanation-related information in constraint networks.

3.1 An alternate representation of graphs

So far, visualization of networks has mainly focused on node-link diagrams because they are popular and well understood. However, node-link diagrams do not scale well: their layout is slow and they become quickly unreadable when the size of the graph and link density increase.

In this paper, we present a recent technique that uses adjacency matrices instead of node-link diagrams to interactively visualize and explore large graphs, with thousands of nodes and any number of links. This technique relies on the well known property that a graph may be represented by its connectivity matrix, which is an N by N matrix, where N is the number of vertices in the graph, and each row or column in the matrix stands for a vertex. When two vertices V_i and V_j are linked, the corresponding coefficient m_{ij} in the matrix is set to 1, otherwise it is set to 0.

From a visualization standpoint, not only do we switch on or off the cell located at the intersection of V_i and V_j , but we use color coding as well when

dealing with weighted links: the heavier the weight (here the number of interactions), the darker a link. Unlike node-link diagrams, matrix-based representations of graphs do not suffer from link and node overlappings. Virtually every link (out of the $N^2/2$ links) in the graph can be seen separately (see figure 1). With this technique, we can show as many links as the display hardware resolution allows, roughly 2 million pixels on a regular 1600×1200 display. Moreover, advanced information visualisation techniques such as dynamic queries [2], fisheye lenses [3] and excentric labels [5] enhance the exploration of large graphs and push the matrix-based visualization one step further in coping with large networks.

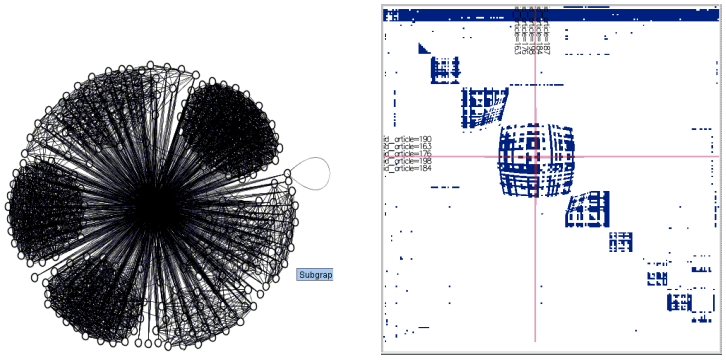


Fig. 1. Representing a graph with 220 vertices and 6291 links using a node-link classical diagram (**left**) and an adjacency matrix (**right**). The matrix view is produced by our tools. A fisheye magnifies the central part of the display. Notice that the node-link diagrams in this paper are produced by `neato`, an open-source graph layout program provided by AT&T. It relies on the force-directed layout algorithm of Kamada and Kawai [13].

The main tradeoff of such a technique lies in the fact that vertices are no longer represented by a unique graphic symbol, they are rather distributed on both axes of the matrix. This is why users may often need some training before they get familiar with the matrix metaphor. Further investigation of this technique in terms of human-computer interaction is still required though, in order to assess more formally its advantages and weaknesses compared to the traditional node-link metaphor.

3.2 Making sense of graphs

Making sense out of network data often depends on the ability to understand its underlying structure. Therefore, cluster detection has been an active topic of research for a long time. Many works have concentrated on data analysis techniques in order to aggregate the graph into its main components. From a

different standpoint, Bertin [1] has shown that the discovery of the underlying structure of a graph can be achieved through successive permutations over the rows and columns of the grid representing it. This idea relies on the fact that the rows and columns of a matrix can be ordered according to a given criterion, which is another advantage of the matrix metaphor as ordering the vertices and links in a node-link diagram is not straightforward.

In our tools, we achieve clustering through successive permutations of rows and columns according to two generic algorithms (a hierarchical agglomerative algorithm and a partition-based algorithm). Other domain-specific algorithms can be fit in our system effortlessly *e.g.* algorithms tailored for constraint programming graphs. In the following, we will present our early experiments in making use of matrix-based visualizations with constraint programming graphs.

4 Experiments

We present here our first experiments with the visualization tools described earlier. We first show how to confirm intuition or information that could be deduced from the static structure of the problem on a toy problem involving the `allDifferent` constraint. Second, solving that same toy problem for all solutions we show how some new information could be gathered. Finally, we present some early results obtained on an scheduling optimisation problem. In all the experiments, we compare our explained system with what can be deduced from a non explained system.

4.1 Visualization parameters

In the following, we display information using two main representations:

– **An undirected constraint-constraint graph**
Contraints c_i and c_j are connected in three different ways:
- representing static structure information: c_i is related to c_j if c_i reduced a variable shared with c_j. This relation represents the fact that the activity of c_i will awake c_j in the future. Only an explained constraint solver can tell if the awakening is only due to c_i and if it will add information to the constraint store (reducing domains). This graph will be denoted `cc-direct`.
- representing dynamic relations from the static structure of the problem: c_i is related to c_j each time c_i reduces a variable and if c_j and c_i share any common variable. This relation states that all constraints c_j with their past effects are helping c_i adding information to the constraint store. This is a kind of *a posteriori* explanation-based information. That is all that can be inferred from an non explanation-based constraint solver. This graph will be denoted `cc-static`.
- representing the explanation network: c_i is related to c_j if c_i and c_j appear in the same explanation during computation. This relation represents the fact that c_i and c_j concurrently worked to provide new information to the solver. It represents some dynamic structure appearing

during computation as constraints cooperate to solve the problem. This graph will be denoted `cc-explain`.

Notice that these graphs, being undirected, will result in symmetric matrices.

– **A directed variable-variable graph**

Variables v_i and v_j are connected in two different ways:

- representing static structure information: v_j has an impact on v_i if v_i has been modified by a constraint c which has been posted upon v_j. This represents the static relations between variables. No more information can be computed from a non explanation-based constraint solver. This graph will be denoted `vv-direct`.
- representing the explanations network: v_j has an impact on v_i if v_i has been modified because of the set of constraints e and v_j is a variable upon which a constraint $c \in e$ has been posted. This represents real variable impact as it can be inferred from explanations. This graph will be denoted `vv-explain`.

Notice that graphs, being directed, will result in possibly non symmetric matrices. Moreover, we will represent them in such a way that variables on top of the matrix are considered to impact variables on the left of the matrix.

All relations in those graphs are weighted with the number of times the relation can be established throughout computation. This helps modifying the static structure with dynamic information pointing out *active* relations. More precisely, we keep a full history of activity within the graph. In this way, we can dynamically query the graph for links that are active within a user-controlled time range and compare the amount of activity between links in that range. The user may visualize the activity in the graph throughout the whole resolution process or in a smaller time range whose bounds and extent are interactively parameterized. By sweeping the time range from the beginning to the end of the history, the user may play back the resolution process and see which links are established, when, and how often. Our tools also support user-defined time slices. Simply put, a time slice is a time range between two events of interest. For instance, in our experiments, we were interested in activity between pairs of successive solutions. Our system computes the relevant time slices and allows the user to jump between successive time slices through direct interaction too.

4.2 A toy problem: retrieving known information

A first example involves 13 variables: a_1, a_2, a_3, b_1, b_2, b_3, d_1, d_3 whose domain is $[1, 3]$ and c_1, c_2, c_3, c_4, c_5 whose domain is $[1, 5]$. Five constraints are posted on these variables: three `allDifferent` constraints on respectively all the a_i (constraint c_{00001}), all the b_i (constraint c_{00004}) and all the c_i (constraint c_{00002}) and two `allDifferent` constraints relating the sets of variables, respectively on d_1,c_2 and d_3 (constraint c_{00003}) and on a_1, b_1, c_1, and d_1 (constraint c_{00005}).

We are looking for the first solution to this problem. Constraint propagation is not powerful enough to exhibit a solution without any enumeration. Nine enumeration constraints need to be added in order to reach a solution. We will therefore report 14 constraints on the following graphs.

What are we looking for ? Looking at the problem itself, one can deduce that c_{00002} and c_{00003} have to interact to provide a solution because they share a variable. However, constraint c_{00001} and c_{00004} should not be hard to satisfy, they share variables with other constraints but they are quite easy to satisfy. The *hard* part of the problem should be represented by the set $\{c_{00002}, c_{00003}, c_{00005}\}$. Regarding search, obviously early choices (*i.e.* small constraint index) should have a long impact on late choices as they should be used to lead search.

Regarding variables, as constraints c_{00001} and c_{00004} should not be hard to satisfy, variables a_2, a_3, b_2, and b_3 should not have much impact during search. This is probably the only information that can be identified just by looking at the problem.

Constraint-constraint graphs We report in figure 2 the three constraint-constraint graphs that can be obtained using the trace[3] generated by our solver. According to the `cc-direct` graph, constraints c_{00012} (enumeration constraint assigning the value 2 to variable c_2) and c_{00013} (enumeration constraint assigning the value 1 to variable c_3) are strongly related to c_{00002} and c_{00003}. We observe the same apparent relation between constraints with c_{00005} and c_{0006} (enumeration constraint assigning the value 1 to variable c_4). But, none of our intuitions can be confirmed here (except for the strong links between c_{00002}, c_{00003}, and c_{00005}.

The `cc-static` graph gives some more information. Indeed, in this graph, constraints c_{00004} and c_{00001} are not related to the other `allDifferent` constraints (except c_{00005}) as expected. However, the dynamic structure that we would like to appear between enumeration constraints is not apparent here. Only the `cc-explain` graph gives the full information: links between the `allDifferent` constraints, and more importantly the strong impact of early enumeration constraints (c_{00009}) compared to late ones (c_{00014}). Moreover, some structuring information appear: obviously enumeration constraint c_{00008} (and similarly constraint c_{00010}) has not helped search (it had almost no subsequent relations with other constraints). Notice that some of this information may be inferred from the classical representation of graphs as shown in figure 3 in which c_{00005} appears with a central role whereas constraints c_{00001}, c_{00004}, c_{00008}, and c_{00010} play a peripheral role.

As we can see, the explanation graph is able to provide both structure and dynamic information about search in this toy example. Let us now have a look at the variable-variable graphs reported on figure 4.

Variable-variable graphs The `vv-direct` graph of figure 4 confirms that a_2, a_3, b_2, and b_3 have a limited impact on other variable during the search of the first solution to this toy problem. Notice that this graph also suggests that variable a_1 has no impact on a_2 which is quite odd. Moreover, the static structure of the problem suggests that decisions on c_i variables should impact b_2 and b_3 variables which is also unexpected. The `vv-explain` gives us the correct[4]

[3] Remember that we used the generic OADYMPPAC trace format (see [14]).

[4] In the sense that it confirms intuitions about the problem.

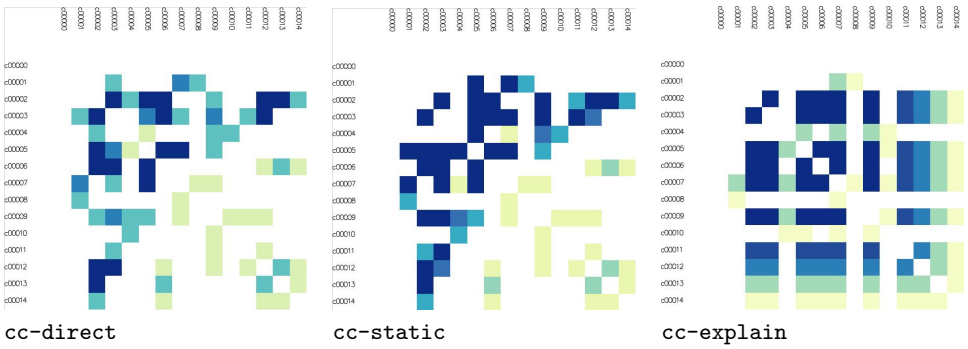


Fig. 2. A toy problem (first solution): constraint-constraint graphs. The darker the dot, the more often the constraints are related.

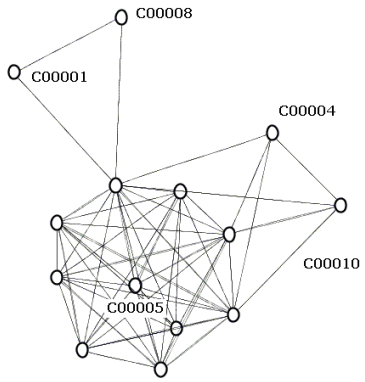


Fig. 3. A toy problem (first solution): a classical representation of the `cc-explain` graph

view about resolution: a_2, a_3, b_2, and b_3 have no impact on the other variables of the problem. Moreover, a_1, b_1, and finally d_1 seem to be the first enumerated variables as they are used to reduce all the other variables. Finally, it seems hard to satisfy the constraint c_{00003} as all the c_i interact a lot during search. Moreover, c_i variables only impact other c_i variables as expected. All the noise, on the upper part of the `vv-direct` matrix is obliterated by the precise information provided by the explanations.

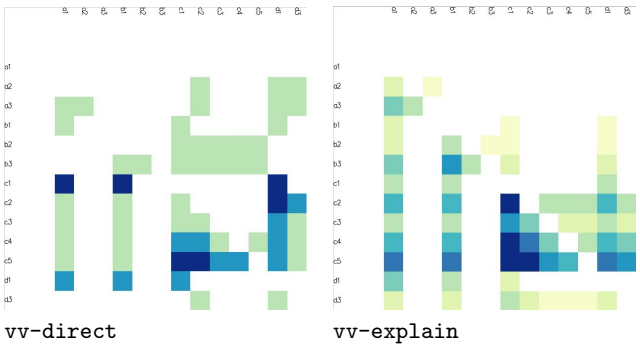


Fig. 4. A toy problem (first solution): variable-variable graphs

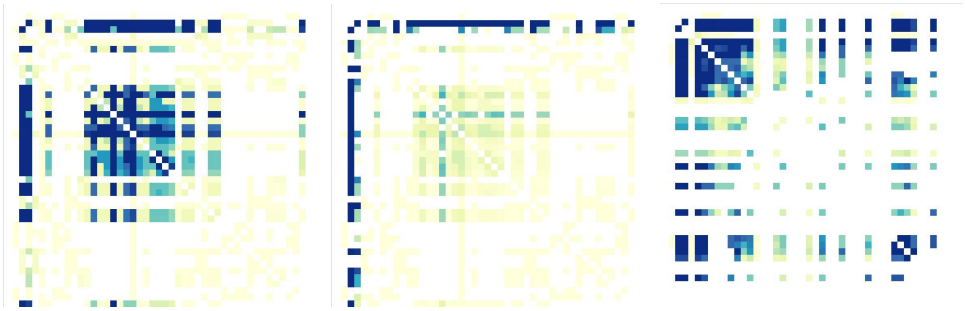
4.3 A toy problem: new information

Keeping the same problem, now we would like to discover some information about search when computing all the 1152 solutions to the problem. At the end of the process, 45 constraints have been posted *i.e.* the 5 original constraints and 40 possibly reused enumeration constraints.

4.4 Constraint-constraint graphs

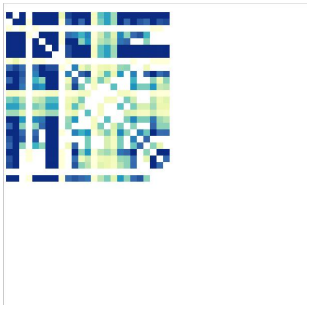
Figure 5 reports the different constraint-constraint graphs that can be obtained from the resolution trace. As we can see, the `cc-direct` graph tend to show strong relations between enumeration constraints (in the middle of the matrix) that cannot be explained. If we consider reduction-time relations, the `cc-static` graph shows that those relations do not really exist, but only the explanation graph `cc-explain` gives the correct perspective on what happens: as expected, early enumeration constraints have a great impact on all the search. Moreover, a clustered view (see figure 6) of that same matrix clearly shows that only roughly half the enumeration constraints have an important role during search: many of such enumeration constraints come from the fact that they correspond to the only choice left.

Retrospectively, the relations appearing in the `cc-direct` graph can be explained by the fact that they are enumeration constraints posted on the same variable (different values) that clearly cannot interact during search as only one of them is active at any given time of the search.



`cc-direct` `cc-static` `cc-explain`

Fig. 5. A toy problem (all solutions): constraint-constraint graphs



`cc-explain` clustered

Fig. 6. A toy problem (all solutions): a clustered matrix representation of a constraint-constraint graph

4.5 Variable-variable graphs

As for constraint-constraint graphs, explanation-based variable-variable graphs (see figure 7) both:

– reduce the noise of the static structure of the problem: see how all c_i variables seem to have an impact on all b_i variables in the `vv-direct` representation and how the `vv-explain` representation show that it is not the case at all (only decisions made on c_1 have an impact).
– exhibit hidden information: indirect impact of a_1 (and a_2) and b_1 (and b_2) on the c_i.

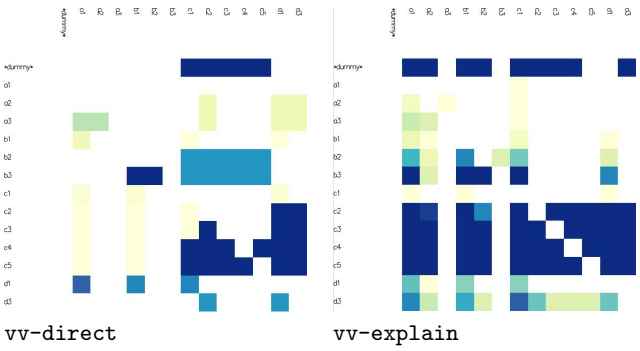


Fig. 7. A toy problem (first solution): variable-variable graphs. The `*dummy*` variable is used to generate new solutions and should not be considered as part of the problem.

4.6 Resolution dynamics

Another interesting use of our visualisation tools is to observe the dynamics of search. Figure 8 shows time slices between different solutions among the 1152 identified ones. The first column shows how the first solution is obtained. We recognize the `cc-explain` graph of figure 5 (with a different scale for intensity). As we can see in the second column, getting from solution #2 to #3 only involves working on the c_i variables (variable-variable representation: bottom) but some new enumeration constraints have been introduced to be able to produce that solution (constraint-constraint representation: top). On the third column, a completely new solution is found (modifying b_i variables) whereas no new constraint is introduced: only old enumeration constraint are used. On the fourth column, new symmetrical solutions are generated on the c_i variables whereas on the last column more variables are modified (c_i, b_i, and d_i). For that last solution, new constraints are introduced and activated.

Such a representation should give interesting insight on what is really happening during search: which constraints are active, what dynamic relations appear between variables, etc.

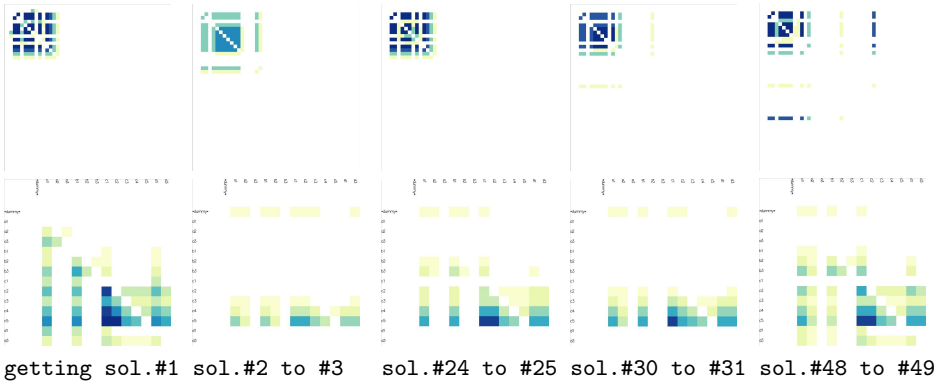


Fig. 8. A toy problem (all solutions): `cc-explain` and `vv-explain` subgraphs for different solutions during search.

4.7 Open-shop scheduling problems: exhibiting structure

Our last reported experiments were done on open-shop scheduling problems.

Open-shop scheduling as CSP Classical scheduling shop problems, for which a set J of n jobs consisting each of m tasks (operations) must be scheduled on a set M of m machines, can be considered as CSP[5]. One of these problems is called the open-shop problem [8]. For this problem, operations for a given job may be sequenced as wanted but only one at a time. We will consider here the building of non-preemptive schedules of minimal makespan[6].

The open-shop scheduling problem is NP-hard as soon as $\min(n, m) \geq 3$. This problem although quite simple to enunciate is really hard to solve optimally: instances of size 6×6 (*i.e.* 36) variables remain unsolved !

Solving open-shop scheduling problems As reported in [12], using explanation-based algorithms to solve open-shop scheduling problems can be very effective. We used here a complete version of `decision-repair` to solve a 4×4 instance of the problem. This problem is structured as follows: 32 definition constraints are posted (they related each task to the beginning and the end of the scheduling), then 264 unary-resource related constraints (each machine and job is considered as a unary resource[7] used by the relevant tasks). During search, 102 enumeration

[5] The variables of the CSP are the starting dates of the tasks. Bounds thus represent the least feasible starting time and the least feasible ending time of the associated tasks.
[6] Ending time of the last task.
[7] We use here task-intervals [4] to efficiently manage those resource.

constraints are needed in order to find the optimal solution and prove its optimality. Notice that the construction structure is clearly apparent in the `cc-direct` graph reported in figure 9: part **a** represents the definition constraints, and part **b** represents the machine-related resource management constraints, while part **c** represents the job-related resource management, and part **d** represents the enumeration constraints.

Learning from search In figure 9, we report the different constraint-constraint graphs that can be obtained from the trace of the search. As we can see, the construction structure of the problem (`cc-direct` and `cc-static`) is not the structure that is really used through computation as reported in the `cc-explain` graph. When looking at the clustered version of the graphs in figure 10, we can see that unrelated constraints during search (in `cc-explain`) are statically strongly related which helps determine the dynamic structure of the problem and possibly explains the high-performance of explanation-based algorithms. Similar information can be exhibited from the variable-variable graphs (figure 11).

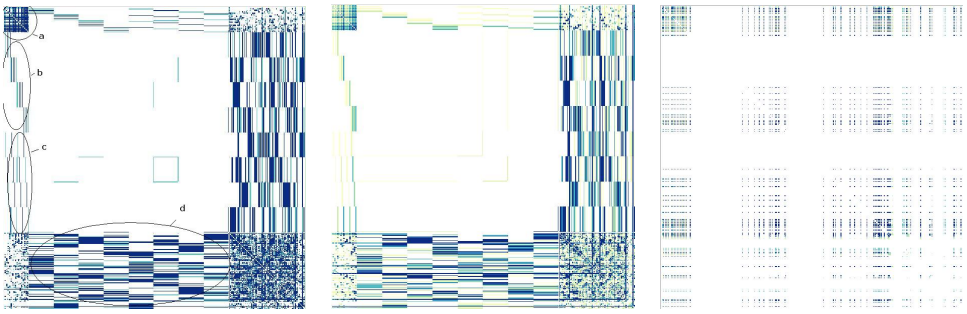


Fig. 9. Open-shop scheduling: constraint-constraint graphs

5 Conclusion

We introduced in this paper new visualization tools well suited for exploring relations between constraints and variables through explanations. We showed how explanations could provide much more insight about how search is performed in a constraint program than classical representations of the static structure of the solved problem.

A lot of work remains to be done especially on the interpretation of graphs and providing navigation tools (in the constraint network, between solutions, etc.) in our main tool.

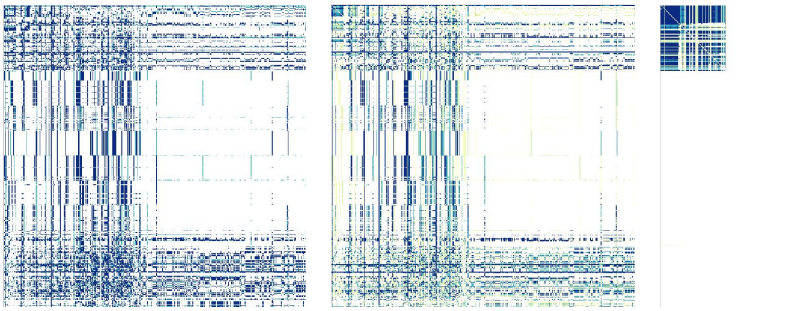


Fig. 10. Open-shop scheduling: clustered constraint-constraint graphs. We applied on `cc-direct` and `cc-static` the same row and column order as pictured on `cc-explain`.

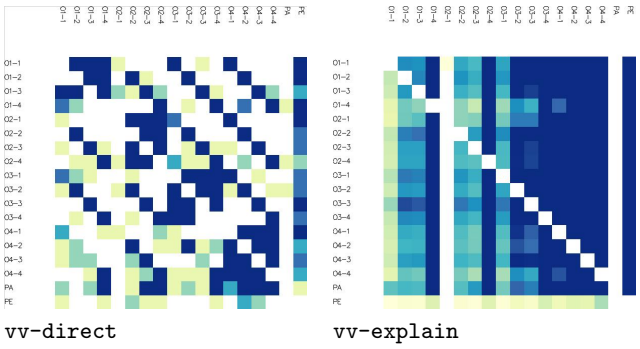


Fig. 11. Open-shop scheduling: variable-variable graphs.

References

1. J. Bertin. *Semiology of Graphics: Diagrams, Networks, Maps.* University of Wisconsin Press, 1983.
2. S. Card, J. Mackinlay, and B. Shneiderman. *Readings in Information Visualization: Using Vision to Think*, chapter Dynamic Queries, pages 235–261. Morgan Kaufmann, San Francisco, USA, 1999.
3. M. S. T. Carpendale and C. Montagnese. A framework for unifying presentation space. In *Proceedings of the 14th annual ACM symposium on User inter-face software and technology (UIST'01)*, pages 61–70, November 2001.
4. Y. Caseau and F. Laburthe. Improving clp scheduling with task intervals. In P. V. Hentenryck, editor, *Proc. of the 11th International Conference on Logic Programming, ICLP'94*, pages 369–383. MIT Press, 1994.
5. J.-D. Fekete and C. Plaisant. Excentric labeling: Dynamic neighborhood labeling for data visualization. In K. Ehrlich and W. Newman, editors, *Proceedings of the International Conference on Human Factors in Computing Systems (CHI 99)*, pages 512–519. ACM, May 1999.
6. 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.
7. M. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.
8. T. Gonzales and S. Sahni. Open-shop scheduling to minimize finish time. *Journal of the Association for Computing Machinery*, 23(4):665–679, 1976.
9. N. Jussien. e-constraints: explanation-based constraint programming. In *CP01 Workshop on User-Interaction in Constraint Satisfaction*, Paphos, Cyprus, 2001.
10. N. Jussien and V. Barichard. The PaLM system: explanation-based constraint programming. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, pages 118–133, Singapore, September 2000.
11. N. Jussien, R. Debruyne, and P. Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Principles and Practice of Constraint Programming (CP 2000)*, number 1894 in Lecture Notes in Computer Science, pages 249–261, 2000.
12. N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, July 2002.
13. T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.
14. OADYMPPAC. Tools for dynamic analysis and debugging of constraint programs. http://contraintes.inria.fr/OADymPPaC, 2001. RNTL project.
15. S. Ouis, N. Jussien, and P. Boizumault. k-relevant explanations for constraint programming. In *FLAIRS'03: Sixteenth international Florida Artificial Intelligence Research Society conference*, pages 192–196, St. Augustine, Florida, USA, May 2003. AAAI press.
16. P. Pu and D. Lalanne. Interactive problem solving via algorithm visualization. In *Proceedings of the IEEE Symposium on Information Visualization (InfoVis 2000)*, pages 145–154, Salt Lake City, Utah, October 2000.
17. D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In A. Borning, editor, *Principles and Practice of Constraint Programming*, volume 874 of *Lecture Notes in Computer Science*. Springer, May 1994. (PPCP'94: Second International Workshop, Orcas Island, Seattle, USA).

Question-Generation in Interactive Constraint Processing

J. Bowen and S. Cummins

Department of Computer Science, UCC, Cork, Ireland
{j.bowen,s.cummins}@cs.ucc.ie

Abstract. Existing work on interactive constraint processing has been based on a scenario in which the machine infers the consequences of choices made by the user; in other words, the interaction is user-driven. In this paper, we address an alternative scenario where the interaction is machine-driven: the machine requests, from the user, information that it needs to solve a problem that has previously been posed by the user. In particular, we address the task of minimizing the number of questions that the machine must ask.

1 Introduction

A constraint network is a triple $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$, in which $\mathbf{D}$ is a tuple of domains, $\mathbf{X}$ is a tuple of parameters, each of which may assume values from the corresponding domain in $\mathbf{D}$, and $\mathbf{C}$ is a set of constraints, each of which restricts the values that may simultaneously be assumed by a sub-tuple of the parameters in $\mathbf{X}$. The overall network constitutes, in effect, an intensional specification for a set of tuples; each tuple in this set has the same length as $\mathbf{X}$ and provides assignments for the corresponding parameters such that all the constraints are satisfied. The set which contains all these tuples of consistent value assignments is called the *intent* of the network. Many different forms of constraint satisfaction problem (CSP) have been distinguished. The most common one, the Exemplification CSP, will be relevant in this paper; it can be defined as follows:

> *Definition 1, The Exemplification CSP:*
> Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$: return nil if the intent of $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$ is empty; otherwise, return some tuple from the intent.

Another problem which will be relevant in this paper, the Enumeration CSP, can be defined as follows:

> *Definition 2, The Enumeration CSP:*
> Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$: return the intent of $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$.

Most research on constraint processing algorithms has been directed at consistency propagation or at search. That work can be regarded as assuming autonomous machine processing, in which the user gives a CSP to the machine and waits for it to return a solution to the given problem.

In recent years, however, there has been increasing interest in interactive constraint processing. Research in this area has been based on a scenario in which

the user gives to the machine a CSP which the user and machine then co-operate in solving [4]. For example, in some systems based on this scenario, the user-machine duo can be regarded as performing MAC-type search: the user assigns values to network parameters while the machine maintains arc-consistency; if the machine finds an inconsistency, the user decides which of his previous assignments to change, he assigns a new value to the parameter involved and the process continues.

Within this field of interactive constraint processing, various themes have evolved. For example, some research [1, 6, 4, 5], has addressed the issue of explanation generation, which involves the machine explaining to the user why a certain parameter must, or may not, have a certain value. There has also been some research on processing generalized user-inputs, where the user is allowed to assert not just a value for a parameter but an arbitrary constraint between parameters [2].

Nevertheless, all this work on interactive constraint processing has had a common characteristic: the machine infers consequences of choices made by the user; that is, the machine reacts to initiatives taken by the user. In this paper, we address an alternative scenario, one in which the initiative is taken by the machine.

To illustrate, consider the following situation, based on map colouring. The user gives the machine a constraint network in which the parameters represent countries on some map in an atlas that the user has in his hand, while the constraints represent inequality constraints between countries that are contiguous on the map. The user tells the machine what range of colours are used on the map; this becomes the domain for each parameter in the network. Then the user chooses some country on the map and challenges the machine to determine what colour is used in the atlas for this target country. The machine cannot, of course, see the atlas but it is allowed to ask the user questions. Specifically, it can ask the user what colour is used in the atlas for any country, apart from the target country. The point, of course, is that the machine is expected to ask as few questions as possible.

Consider a simple version of this situation, involving only three colours (red, green and blue) and a map which contains only four countries (Austria, Germany, Switzerland and Italy); the adjacency relations in this map are shown in Figure 1 (a). Suppose that the user has asked the machine to determine what colour is used in the atlas for Italy. How many questions would the machine have to ask the user? What country should it ask about first? Should it ask about one of the countries that touch Italy?

In fact, the machine need ask only one question – it should ask for the colour of Germany, for the following reason. Germany must have a different colour from Austria and Switzerland, whose colours must be different from each other. Italy must also have a different colour from Austria and Switzerland. Since there are only three colours, Italy must have the same colour as Germany.

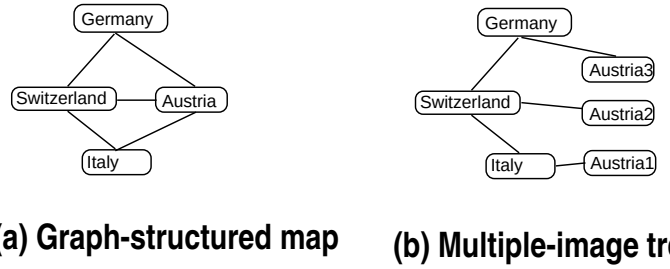


Fig. 1. Map of Central Europe

The task posed to the machine above is an example of a class of problem which we call the Optimal Questioning Strategy (OQS) problem and which can be defined as follows.

> *Definition 3, The Optimal Questioning Strategy CSP:*
> Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$ whose intent is non-empty and a tuple $\mathbf{T}$ of target parameters in $\mathbf{X}$, return the smallest tuple $\mathbf{Q}$ of the parameters in $\mathbf{X}$-$\mathbf{T}$ such that, if the values of all the parameters in $\mathbf{Q}$ were known, there would be only one possible consistent value for each of the parameters in $\mathbf{T}$.[1]

In the example task above, the set $\mathbf{T}$ contained only one target parameter, Italy, and the tuple $\mathbf{Q}$ contained only one question parameter, Germany. In general, however, there could be several target parameters in $\mathbf{T}$ and the optimal questioning strategy $\mathbf{Q}$ could contain a sequence of several questions.

Note that the projection of the network intent onto the parameters in $\mathbf{T}$ may contain multiple tuples. In the example above, the projection of the network intent onto $\mathbf{T} = \{\text{Italy}\}$ contains a set of three singleton-tuples {r, g, b}, each of which is a possible consistent value for the single target parameter in $\mathbf{T}$. If $\mathbf{Q}$ contains only one question parameter, as in the example above, the task of computing an optimal strategy is easier than where multiple questions are needed, because, in the latter case, each of the different valuations in the projection of the network intent on $\mathbf{T}$ may require a different sequence $\mathbf{Q}$. In this latter case, we have a multiple-ply gaming situation – in devising the questioning strategy, the system must consider different possible answers that the user could give to the various questions the system could pose.

In the rest of this paper, we will focus on OQS problems where there is only one target parameter. Thus, we provide an explicit definition of this special case, the single-target OQS problem:

> *Definition 4, The Single-Target Optimal Questioning Strategy CSP:*
> Given a network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$ whose intent is non-empty and a target parameter T which is a member of $\mathbf{X}$, return the smallest tuple $\mathbf{Q}$ of

[1] The sequence of parameters in $\mathbf{Q}$ is, then, the optimal questioning strategy.

the parameters in ($\mathbf{X}$ - $\langle \mathbf{T} \rangle$) such that, if the values of all the parameters in $\mathbf{Q}$ were known, there would be only one possible consistent value for T.

This kind of constraint-based problem is not a frivolous one. There are many economically or socially significant instances of the single-target OQS CSP. Consider medical diagnosis, for example.[2] A body of diagnostic expertise could be represented as a constraint network in which the target parameter represents the disease afflicting a patient, while other parameters represent the results of possible diagnostic tests that could be performed on him. Since diagnostic tests can be dangerous, expensive and/or time-consuming, an expert diagnostician tries to minimize the number of tests that he calls for. As another example of the single-target OQS CSP, consider an automated telephone-based interactive product selection and sales system.[3] Knowledge for this domain could be represented as a constraint network in which the target parameter represents the product which best meets a customer's needs, while other parameters represent salient aspects of the situation in which the customer proposes to use the product that he wishes to purchase. Since a customer is likely to hang up in frustration if he is asked too many questions, a successful system will try to ask as few questions as possible.

2 A Naive Approach to Question Generation

In this section, we will consider one approach to generating an optimal questioning strategy. While it may be practical in cases where the constraint network is small, it is unlikely to be useful for dealing with real-world situations where the constraint networks are likely to be quite big. Nevertheless, we present the approach because it illustrates some concepts which arise in the other approaches that we will consider. The approach relies on prior computation of the intent that is implicit in the constraint network.

Suppose that, in the constraint network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$ for the map-colouring example we considered above, the parameters in tuple $\mathbf{X}$ are ordered $\langle$ Germany, Switzerland, Austria, Italy $\rangle$. The intent that is implicitly defined in Figure 1 (a) is, then, $\{ \langle r,g,b,r \rangle, \langle r,b,g,r \rangle, \langle g,r,b,g \rangle, \langle g,b,r,g \rangle, \langle b,r,g,b \rangle, \langle b,g,r,b \rangle \}$. If the system knew that the contents of the intent are as just given, it could compute from this that, for each possible value of Germany, only one value is allowed for Italy. It could also compute that, for each possible value of Switzerland, two values are allowed for Italy. Similarly, it could also compute that each possible value of Austria admits two values for Italy. Therefore, the system would know that, if it wants to identify the value of Italy as quickly as possible, it should ask about Germany.

[2] Our work can be regarded as preliminary steps to developing a constraint-based approach to interactive diagnosis. While there is a large existing literature on rule-based diagnosis, the richer expressiveness of constraints poses greater processing complexity, in particular when it comes to question generation.

[3] Interactive product *configuration*, as opposed to selection, corresponds to a multiple-target OQS CSP.

To prepare ourselves to consider the later approaches, it it useful to cast this argument in terms of probabilities and expected values of domain sizes. If the system knew that the contents of the intent are as given above, it could, *before asking any question*, compute an expected domain size for parameter Italy after learning the answer to each possible first question. Let $prob(X = y)$ denote the probability that, if the system asks for the colour of some country X, it will receive the answer y. The expected domain size for Italy after asking about Germany is

$$prob(Germany = r) * 1 + prob(Germany = g) * 1 + prob(Germany = b) * 1.$$

Simplifying this, we get

$$prob(Germany = r) + prob(Germany = g) + prob(Germany = b)$$

which equals 1, since the probabilities for the different values of Germany must sum to unity. Similarly, the expected domain size for Italy after asking about Austria is

$$prob(Austria = r) * 2 + prob(Austria = g) * 2 + prob(Austria = b) * 2$$

which evaluates to 2. Finally, the expected domain size for Italy after asking about Switzerland, which is

$$prob(Switz = r) * 2 + prob(Switz = g) * 2 + prob(Switz = b) * 2$$

also evaluates to 2. Thus, the expected domain size, after asking one question, is smallest if that question is about Germany. Therefore, the system knows that, if it is interested in narrowing the possibilities for Italy as quickly as possible, its best chance of doing so is to ask about Germany first.

3 Sampling the Intent

Although the rationale for question selection given above was expressed in terms of the expected domain size for the target parameter, the same approach can be explained from a different perspective. We could have used Shannon's Information Theory [9]. That is, choosing between questions on the basis of their effectiveness in reducing entropy would lead to exactly the same questioning strategy. Indeed, our use of expected domain size in choosing questions is similar to the use of entropy reduction in Decision Tree learning [8].

In fact, an alternative approach to our task is prompted by the observation that Optimal Questioning Strategy generation in constraint networks is similar to Decision Tree Learning. This second approach is likely to be more practical than the first approach because it does not require knowledge of the intent of the constraint network; that is, it does not require prior solution of the Enumeration CSP. Instead, as we shall see below, this second approach requires solving the Exemplification CSP, which is computationally much less expensive than solving the Enumeration CSP.

A decision tree [8] is a tool for classifying instances from some population. A non-leaf node in the tree is a question whose children are connected to it by

edges which correspond to the possible answers for the question. A leaf node in the tree is a classification. When an instance from the population is to be classified, it is analysed according to the questions encountered along the path, from the root node, that correspond to the answers which the attributes of the instance give to the questions; this path eventually leads to a leaf-node which is the classification for the instance.

Answering questions posed by a machine which has solved the Single-Target OQS problem can be compared with the process by which a decision tree classifies an instance. The domain of the target parameter in the constraint network corresponds to the range of possible classifications that may result from a decision tree analysis of an instance. The non-target parameters in the constraint network correspond to the instance attributes that are examined by the question nodes in a decision tree. In fact, the population of instances which a decision tree is supposed to be capable of classifying corresponds to the intent of the constraint network: each tuple in the constraint network intent contains the description of an instance and its corresponding classification – the value, in the tuple, of the target parameter corresponds to the classification, while the values of the non-target parameters correspond to the attribute-values of the instance.

While decision trees could be constructed manually, they are normally constructed by machine learning algorithms called Decision Tree Learning algorithms [8]. Such an algorithm is given a *sample* from the population along with, for each member of the sample, its appropriate classification. We have seen that the population of instances to be classified by a decision tree corresponds to the intent of a constraint network. Since a decision tree learning algorithm needs only a sample of the population, this suggests that we should be able to generate a constraint-based questioning strategy if we know only a sample of the network intent[4].

In other words, instead of requiring prior solution of the Enumeration CSP (in order to compute the complete intent), it should be possible to learn an appropriate set of questions from a relatively small set of alternative solutions to the Exemplification CSP; an important requirement, of course, is that this set of alternative solutions should be representative of the overall intent of the network.

An investigation of this approach is part of our ongoing research. The basic idea is as follows: use repeated invocations of a search algorithm to find several randomly distributed solutions to the Exemplification CSP; then use a decision tree learning algorithm to learn a decision tree from this sample of the overall intent of the constraint network. We are investigating usage of the approaches presented in [3, 7] for generating random solutions to the Exemplification CSP, as part of this sample-based approach to generating questioning strategies.

Our work on learning a questioning strategy from randomly generated solutions to the Enumeration CSP is, however, still in its early stages. The rest of this paper reports on a different approach, where our work is further advanced.

[4] Acknowledgement: this observation was made by Pat Langley during a conversation.

4 Intent-independent Question-Generation

So far, we have outlined two approaches to question generation, one of which requires prior computation of the network intent (by solving the Enumeration CSP) while the other requires prior computation of a random sample from the network intent (by computing several randomly distributed solutions to the Exemplification CSP).

An obvious ambition, therefore, is to find some approach which does not require prior knowledge of any part of the network intent, an approach which does not require prior solution even of the Exemplification CSP, let alone prior solution of the Enumeration CSP.

Such an approach does exist. Before introducing it, is is useful to define some concepts.

> *Definition 5, Target Shadow:*
> The set of values for the target parameter that are consistent with a particular value of a non-target parameter is called the *target shadow* of that value for the non-target parameter.

To see some examples of target shadows, consider, again, the network for the map-colouring problem – see Figure1 (a). Remember that, where the parameters in this network are ordered ⟨ Germany, Switzerland, Austria, Italy ⟩, the intent of the network is { $\langle r,g,b,r\rangle$, $\langle r,b,g,r\rangle$, $\langle g,r,b,g\rangle$, $\langle g,b,r,g\rangle$, $\langle b,r,g,b\rangle$, $\langle b,g,r,b\rangle$ }. From this, we can see that, if Austria has the value r, the target parameter, Italy, can have either the value g (tuple 4 in the intent) or b (tuple 6 in the intent). Thus, the target shadow for the value r in the domain of Austria is $\{g,b\}$. The target shadows for all the values in the domain of each non-target parameter are shown in the following table:

Germany	$r \rightarrow \{r\}$, $g \rightarrow \{g\}$, $b \rightarrow \{b\}$
Switzerland	$r \rightarrow \{g,b\}$, $g \rightarrow \{r,b\}$, $b \rightarrow \{r,g\}$
Austria	$r \rightarrow \{g,b\}$, $g \rightarrow \{r,b\}$, $b \rightarrow \{r,g\}$

To compute the expected size of the domain of the target parameter that would result if the user were to instantiate some non-target parameter P, we need to know (a) the target shadow for every value v in the domain of P and (b), in most, but not all, cases, the probability[5] that the user will select the value v for P.

> *Definition 6, Expected Shadow Size* The expected size of the domain of the target parameter that results from instantiating some non-target

[5] We say "in most but not all cases" because we do not need the probability distribution if each value in a domain has the same size of target shadow. So, what about those cases where the values in a domain do have differening sizes of target shadow? For now, as a simplifying heuristic, we assume a uniform probability distribution. However, the expert knowledge in a real-world problem domain usually includes a better approximation than this – for example, in medicine, diagnosticians know that certain symptoms are more likely than others; eventually, we propose to represent this information by treating the domains of the parameters in a constraint networks as sets with associated probability distributions.

parameter, P, is called the *expected shadow size* of P. Its value is

$$\sum_{v \in domain(P)} prob(P = v) * |shad(v)|$$

where, for each value v in the domain of P, $prob(P = v)$ is the probability that the user will instantiate P to this value and $shad(v)$ is the target shadow of v.

From what we have seen so far, it appears that the best question to ask the user at any time is to ask him for the value of the non-target parameter which has the minimum expected shadow size[6]. To determine this "best" question, we need to know the target shadow for every value in the domain of every non-target parameter.

We have seen that we can compute these target shadows if we know the network intent. However, we do not need to know the network intent. In what follows, we will first show that this is true in the case of tree-structured constraint networks. Then we will extend the result to cover graph-structured networks.

5 Target-shadow Propagation in Tree-Structured Networks

Theorem 1: If a constraint network is tree-structured, only 2-consistency information is needed to compute the target shadows for all the values of all the non-target parameters in the network.

Proof: In a tree-structured network, any node can be regarded as the root; thus it is possible to treat the target node as the root. Consider a parameter whose node is a child of the root. Using only 2-consistency information, each value in domain of this parameter can be labelled with a set which contains those values of the target parameter that are supported by this value of the child parameter – that is, each value can be labelled with its the target shadow[7]. Now consider a parameter whose node is a child of the parameter for whose values we have just computed the target shadows. Using only 2-consistency information, each value in the domain of this grand-child parameter can be labelled with a set which is the union of the target shadows of those values of the child node that are supported by this value of the grand-child node; the resultant set contains the values of the target parameter that are supported by this value of the grand-child node – in other words, its target shadow. In

[6] In fact, of course, this notion of "best" question leads to a hill-climbing approach to question generation – there are instances where it will lead to question sequences that are longer than necessary.

[7] Note that the memory cost of storing these labels is only O(qd), where q is the number of parameters and d is the maximum domain size, because each of the subsets of the domain of the target parameter can be represented by a binary number; for example, if the domain of the target parameter is $\{r, g, b\}$, we can represent the subset $\{r, b\}$ of this domain by the binary number 101.

this manner, the target shadow for every value in the domain of every non-target parameter can be determined.

Consider, for example, the tree-structured map in Figure 2; suppose that, again, we have a palette of three colours, red, blue and green. Let Poland be the target parameter.

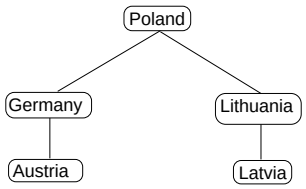


Fig. 2. Tree-structured map

The target shadows for the values in the domain of Germany are $\{r \rightarrow \{g, b\}$, $g \rightarrow \{r, b\}$, $b \rightarrow \{r, g\}\}$. The target shadows for the values in the domain of Austria can be computed from those for Germany; they are $\{r \rightarrow \{r, b\} \cup \{r, g\}$, $g \rightarrow \{g, b\} \cup \{r, g\}$, $b \rightarrow \{g, b\} \cup \{r, b\}\}$; in other words, they are $\{r \rightarrow \{r, g, b\}$, $g \rightarrow \{r, g, b\}$, $b \rightarrow \{r, g, b\}\}$. Similarly, the target shadows for the values in the domain of Lithuania are $\{r \rightarrow \{g, b\}$, $g \rightarrow \{r, b\}$, $b \rightarrow \{r, g\}\}$ and those for the values in the domain of Latvia are $\{r \rightarrow \{r, g, b\}$, $g \rightarrow \{r, g, b\}$, $b \rightarrow \{r, g, b\}\}$.

6 Target Shadow Propagation in Graph-Structured Networks

It is probably not surprising that we need only 2-consistency information to compute the target shadows in a tree-structured constraint network. What is more surprising is that, as we will prove below, only the same degree of consistency is needed to compute target shadows in graph-structured constraint networks. The reason that we need only 2-consistency information to compute the target shadows in a graph-structured network is that we can use hypothetical reasoning to temporarily break cycles, so that we can use the approach given above for propagating target shadows in tree-structured networks. To simplify the proof below, we will first define some terms.

Definition 7, Non-target Cycle-Cutset:
In a graph-structured network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$, there is at least one subset of the non-target parameters that are in the tuple $\mathbf{X}$ such that, if these parameters were instantiated, the remaining constraint network would be a tree. Such a set of parameters is called a non-target cycle-cutset.

Consider, for example, the map-colouring network shown in Figure 3 (a), where Lilliput is the target node. One non-target cycle-cutset is {Gilgitia}.

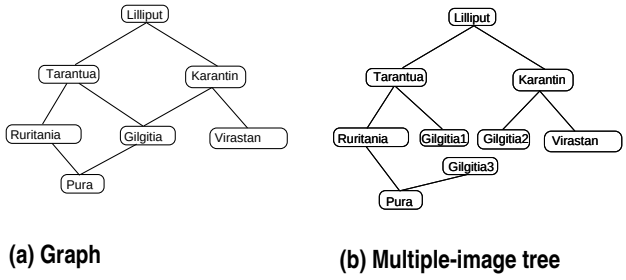


Fig. 3. Lilliput and its neighbours

Definition 8, Multiple-Image-Tree:
Together, a graph-structured network and a non-target cycle-cutset, O, for the network define a tree, called a multiple-image-tree. This tree is produced by making multiple distinct images of each parameter in O, one for each of its neighbours in the original graph-structured network.

The multiple-image-tree defined by the network in Figure 3 (a) and the non-target cycle-cutset {Gilgitia} is shown in Figure 3 (b); in this tree, there are three separate nodes which contain an image of the Gilgitia node in Figure 3 (a).

Definition 9, Multiple-Image Tree Constraint Network:
While a graph-structured network $\langle \mathbf{D}, \mathbf{X}, \mathbf{C} \rangle$ and a non-target cycle-cutset O define a unique multiple-image-tree, they define a set of constraint networks, called multiple-image-tree constraint networks. The set of such networks is defined as follows. Let the multiple-image-tree be M. Let $\mathbf{X}_o$ be that sub-tuple of $\mathbf{X}$ which contains the parameters that are in O. Let D_o be the Cartesian cross-product of the domains of the parameters in $\mathbf{X}_o$. Then there is one multiple-image-tree constraint network corresponding to each tuple t in D_o. A multiple-image-tree constraint network is a constraint network whose parameters and constraints are, respectively, the nodes and edges of M. Except for the multiple-image node parameters, the domain of each parameter is the same as its domain in the original given network. However, the domain of each multiple-image node parameter is a singleton set which contains the value in tuple t for the parameter in the original network of which the node is one of the multiple images.

For example, consider, again, the map-colouring constraint network in Figure 3 (a); suppose that the palette of colours contains the three colours $\{r, g, b\}$. Then, using the the non-target cycle-cutset {Gilgitia}, we get three multiple-image-tree constraint networks. The topology of each of these networks is as shown in Figure 3 (b). Except for parameters Gilgitia1, Gilgitia2 and Gilgitia3, the domain of each parameter is $\{r, g, b\}$. In one multiple-image-tree constraint

network, the three parameters, Gilgitia1, Gilgitia2 and Gilgitia3, all have the same singleton domain $\{r\}$; in another network they all have the same singleton domain $\{g\}$; in the third network they all have the same singleton domain $\{b\}$.

A little extra care must be taken in computing target shadows in multiple-image-tree constraint networks. Remember that all nodes in a group of image nodes that are related by virtue of being images of the same parameter in the original graph represent different facets of that parameter in the original graph. Specifically, propagating along the path between the target node to such an image node produces a set of values in the target node that are supported *via that path* by the single value in the domain of the image node. However, the target shadow must be supported along *all such paths*. Therefore, the target shadow of the single value in the domain of a group of related image nodes is the *intersection* of the sets of values that are supported along the different paths.

For example, consider one of the multiple-image-tree constraint networks based on the multiple-image-tree in Figure 3 (b) – consider the network where images nodes Gilgitia1, Gilgitia2 and Gilgitia3 all have the singleton domain $\{r\}$. The target shadow for the sole value r in this domain is the intersection of three subsets of the domain of the target parameter: the subset of the target domain that is supported by r along the path from Gilgitia1 to the target node; the subset of the target domain that is supported by r along the path from Gilgitia2 to the target node; the subset of the target domain that is supported by r along the path from Gilgitia3 to the target node.

With this infrastructure of ideas in place we can now prove that, to compute target shadows for graph-structured constraint networks, we need only 2-consistency.[8]

> *Theorem 2*: Only 2-consistency information is needed to compute the target shadows for all the values of all the non-target parameters in a graph-structured network.
>
> *Proof*: Given a graph-structured network, take any non-target cycle-cutset for the network. The network and the non-target cycle-cutset implicitly define a set of multiple-image-tree constraint networks, each of which represents the situation that would result from one possible instantiation of the parameters in the non-target cycle-cutset. The target shadow for a value in the domain of a non-target parameter in the graph-structured network represents the set of values in the domain of the target parameter that are supported by this value in the domain of the non-target parameter. Such a target shadow is the union of the target shadows that the value has in the various multiple-image-tree constraint networks that are produced by instantiating the parameters in a non-target cycle-cutset for the graph-structured network. Theorem 1 shows that we need only 2-consistency information to compute target shadows

[8] Note that, even though only 2-consistency is required, the amount of time taken by the algorithm can, in the general case, be exponential in the size of the cycle-cutset. However, needing only 2-consistency means that memory complexity is limited.

in tree-structured networks. Therefore we need only 2-consistency information to compute target shadows in the multiple-image-tree constraint networks that result from instantiating the parameters in the non-target cycle-cutset. Therefore we need only 2-consistency information to compute target shadows in graph-structured constraint networks.

Let see this approach at work in the map-colouring network shown in Figure 1 (a), remembering that the palette contains only three colours, $\{r, g, b\}$ and that Italy is the target parameter.

A non-target cycle-cutset is {Austria}, whose corresponding multiple-image-tree is as shown in Figure 1 (b). There are three images of Austria in this tree, one for each of Austria'a neighbours in the original, graph-structured, network.

This multiple-image-tree implicitly defines three multiple-image-tree constraint networks, because there are three values in the domain of Austria. Consider the first of these multiple-image-tree constraint networks, the network in which each of the multiple images of Austria is given the singleton domain $\{r\}$. Applying arc-consistency, the domains of the other three parameters are reduced to $\{g, b\}$. The value r in the domain of Austria1 is compatible with both values that remain in the domain of Italy; that is, r supports the set $\{g, b\}$ along the path from Austria1 to Italy. Each of the two values in the domain of Switzerland, g and b, is compatible with only one value in the domain of Italy; their target shadows are $\{b\}$ and $\{g\}$, respectively. The single value, r, in the domain of Austria2 is compatible with both values in the domain of Switzerland, so, along the path from Austria2 to Italy, the value r supports the union of the target shadows of the two values in the domain of Switzerland – it supports the set $\{g, b\}$. The value g in the domain of Germany is compatible with only only value in the domain of Switzerland, b, so it inherits its target shadow, namely $\{g\}$. Similarly, the value b in the domain of Germany inherits the target shadow of the value g in the domain of Switzerland, namely $\{b\}$. The single value, r, in the domain of Austria3 is compatible with both values in the domain of Germany, so, along the path from Austria3 to Italy, the value r supports the union of the target shadows for the two values in the domain of Germany – it supports the set $\{g, b\}$. The target shadow for the value r in the singleton domain that is shared by Austria1, Austria 2 and Austria3 is the intersection of the three sets that are supported along the paths from Austria1 to Italy, from Austria2 to Italy and from Austria3 to Italy; since the three sets are the same, the result is $\{g, b\}$.

The above reasoning is summarised in the left-most table below. The corresponding reasoning for the cases where Austria is given the singleton domain $\{g\}$ and the singleton domain $\{b\}$ are summarised in the next two tables.

Alternative 1:	Alternative 2:	Alternative 3:
domain(Austria1) $\leftarrow \{r\}$	domain(Austria1) $\leftarrow \{g\}$	domain(Austria1) $\leftarrow \{b\}$
domain(Austria2) $\leftarrow \{r\}$	domain(Austria2) $\leftarrow \{g\}$	domain(Austria2) $\leftarrow \{b\}$
domain(Austria3) $\leftarrow \{r\}$	domain(Austria3) $\leftarrow \{g\}$	domain(Austria3) $\leftarrow \{b\}$
domain(Italy) $\leftarrow \{g, b\}$	domain(Italy) $\leftarrow \{r, b\}$	domain(Italy) $\leftarrow \{r, g\}$
domain(Switzerland) $\leftarrow \{g, b\}$	domain(Switzerland) $\leftarrow \{r, b\}$	domain(Switzerland) $\leftarrow \{r, g\}$
domain(Germany) $\leftarrow \{g, b\}$	domain(Germany) $\leftarrow \{r, b\}$	domain(Germany) $\leftarrow \{r, g\}$
Austria1: $r \rightarrow \{g, b\}$	Austria1: $g \rightarrow \{r, b\}$	Austria1: $b \rightarrow \{r, g\}$
Switzerland: $g \rightarrow \{b\}$, $b \rightarrow \{g\}$	Switzerland: $r \rightarrow \{b\}$, $b \rightarrow \{r\}$	Switzerland: $r \rightarrow \{g\}$, $g \rightarrow \{r\}$
Austria2: $r \rightarrow \{g, b\}$	Austria2: $g \rightarrow \{r, b\}$	Austria2: $b \rightarrow \{r, g\}$
Germany: $g \rightarrow \{g\}$, $b \rightarrow \{b\}$	Germany: $r \rightarrow \{r\}$, $b \rightarrow \{b\}$	Germany: $r \rightarrow \{r\}$, $g \rightarrow \{g\}$
Austria3: $r \rightarrow \{g, b\}$	Austria3: $g \rightarrow \{r, b\}$	Austria3: $b \rightarrow \{r, g\}$
So, finally:	So, finally:	So, finally:
Switzerland: $g \rightarrow \{b\}$, $b \rightarrow \{g\}$	Switzerland: $r \rightarrow \{b\}$, $b \rightarrow \{r\}$	Switzerland: $r \rightarrow \{g\}$, $g \rightarrow \{r\}$
Austria: $r \rightarrow \{g, b\}$	Austria: $g \rightarrow \{r, b\}$	Austria: $b \rightarrow \{r, g\}$
Germany: $g \rightarrow \{g\}$, $b \rightarrow \{b\}$	Germany: $r \rightarrow \{r\}$, $b \rightarrow \{b\}$	Germany: $r \rightarrow \{r\}$, $g \rightarrow \{g\}$

Now unioning the corresponding target shadows from all three possible multiple-image-tree constraint networks, we get the target shadows for the original graph-structured constraint network, as follows:

Switzerland:	$r \rightarrow \{b\} \cup \{g\}, \quad g \rightarrow \{b\} \cup \{r\}, \quad b \rightarrow \{g\} \cup \{r\}$
Austria:	$r \rightarrow \{g, b\}, \quad g \rightarrow \{r, b\}, \quad b \rightarrow \{r, g\}$
Germany:	$r \rightarrow \{r\} \cup \{r\}, \quad g \rightarrow \{g\} \cup \{g\}, \quad b \rightarrow \{b\} \cup \{b\}$

In summary, therefore, the target shadow situation for the original graph-structured constraint network is as shown in the table below. Compare these results with the target shadows that were computed in Section 4, where knowledge of the network intent was assumed. The results are the same – as they should be, according to Theorem 2, the theorem that target shadows in graphs can be calculated without knowing the network intent.

Switzerland	$r \rightarrow \{g, b\}$, $g \rightarrow \{r, b\}$, $b \rightarrow \{r, g\}$
Austria	$r \rightarrow \{g, b\}$, $g \rightarrow \{r, b\}$, $b \rightarrow \{r, g\}$
Germany	$r \rightarrow \{r\}$, $g \rightarrow \{g\}$, $b \rightarrow \{b\}$

7 Experimental Results

We have implemented the target-shadow propagation approach to computing the best first question. We have applied it to randomly-generated networks with densities between 0.11 and 0.15, inclusive, and for constraint tightnesses ranging from 0.1 to 0.8[9] For each density-tightness combination, we generated 100 Single-Target OQS problems involving 20 parameters of domain size 10. We averaged the number of constraint checks required to compute the best first question for each of these problems. The results are shown in the table below.

Density	Tightness							
%	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
0.15	77,229,876	64,787,446	47,034,884	63,604,847	45,706,708	25,268,699	3,507,446	
0.14	13,256,210	28,504,676	18,801,745	23,994,163	11,540,742	13,799,755	2,256,582	
0.13	6,336,532	12,563,793	7,427,831	7,280,836	10,719,907	5,875,652	1,381,702	12,589
0.12	979,060	1,150,906	757,176	976,660	683,436	542,493	199,465	17,027
0.11	270,111	135,615	123,824	259,546	148,624	220,507	104,071	12,100

It can be seen that, as density increases, the amount of work required also increases. This is to be expected because, as network density increases, the size

[9] Networks with density 0.1 were only trees, so we excluded them as being too easy. Our random networks at 0.9 tightness were arc-inconsistent and those with tightness 0.8 were arc-inconsistent at density 0.14 and above.

of the cycle-cutset needed to produce a multiple-image tree increases. When several parameters are needed in a cycle-cutset and when the domains of these parameters are large, the amount of work needed to generate optimal questions can grow very quickly.

How practical is this approach? This depends on user-acceptability which, in turn, depends on the time taken to compute the best question. In networks of density 0.11 and 0.12, this was always less than 1 second, using a modern desk-top computer. For networks of density 0.13 and 0.14, the times were less than 10 seconds. The worst case, at density 0.15 and tightness 0.1, was 45 seconds.

It may be asked whether problems with 20 parameters of domain size 10, and densities of 15% and below, are meaningful surrogates for the real-world. We have found several real-world tasks which are smaller than this – for example, laptop computer selection.

Our results here are only preliminary. Nevertheless, we believe that they indicate that practical approaches can be found for generating optimal questions in some classes of real-world situation. The approach we have presented here is the first we have investigated in what we believe to be a new form of problem-solving. We hope to develop improvements which can function at acceptable speeds in larger problems.

8 Concluding Remarks

We have introduced the notion of question generation in constraint processing. We have considered three approaches to generating questions, based, respectively, on the prior computation of the network intent, on the prior computation of a sample from the network intent and on the propagation of target shadows without knowing any member of the network intent.

We have proven that target shadow propagation can be done on the basis of information that need be only 2-consistent – we have proven that this is the case in graph-structured constraint networks as well as in tree-structured networks. However, while target shadow propagation in tree-structured networks is relatively inexpensive, there are several factors which may make it an expensive tool when used in graph-structured networks – prior computation of a non-target cycle-cutset for the graph-structured network is required[10] and, more significantly, target shadow propagation must be performed for each of the multiple-image-tree constraint networks which result[11].

However, it must be pointed out that the situation, vis-a-vis graph-structured networks, is not as black as might be deemed from the above remark. Remember that an expert system for medical diagnosis or tele-marketing will be used repeatedly, on many occasions, by many users. It is, therefore, worthwhile spending a lot of effort processing the network before it is released to its user community.

[10] There is no known polynomial cost algorithm for computing the minimum non-target cycle-cutset. However, non-minimum sets can be computed at reasonable cost.

[11] This, of course, means that the best non-target cycle-cutset is one whose domains have the smallest cross-product of all non-target cycle-cutsets. While no polynomial cost algorithm for computing the optimal non-target cycle-cutset is known, heuristics exist which enable the optimal set to be computed at reasonable cost in many networks. These heuristics are beyond the scope of this paper.

Such pre-processing effort has two kinds of cost: the memory space needed and the time required. The fact that only 2-consistent information is required for target shadow propagation in graph-structured networks is good news on the memory front. If a complete questioning strategy (which, in most real-world applications, will require that multiple questions be asked of the user) can be computed before the expert system is released to the user community, the time cost of target shadow propagation may not be a problem either. (It is, of course, always possible to compute the best[12] candidate for the first question, before releasing an expert system to its users. However, there are some unexplored issues related to the complexity of using target shadow propagation for computing an optimal *sequence* of questions.)

There are several issues that we have not touched on in this paper, including the following. What is the appropriate probability distribution to use for the possible answers that a user could give when asked for the value of a parameter? Do there exist heuristics for reducing the extent of target shadow propagation needed in trees. Do heuristics exist for reducing the number of multiple-image-tree constraint networks that need be considered for graphs? What additional complexity is introduced by the fact that, in many real-world applications, some questions may be more expensive to answer than others? We have some results in each of these areas, but they are beyond the scope of this paper which has provided preliminary results in what we believe to be a new area of constraint processing.

References

1. Bowen J and Bahler D, 1992, "Frames, Quantification, Perspectives and Negotiation in Constraint Networks for Life-Cycle Engineering", *Intn'l Journal of AI in Engineering*, 7, 199-226.
2. Bowen J, 2001, "The (Minimal) Specialization CSP: A basis for Generalized Interactive Constraint Processing", CP-2001 Workshop on User-Interaction in Constraint Processing.
3. Dechter R, Kask K, Bin E and Emek R, "Generating Random Solutions for Constraint Satisfaction Problems", *Proceedings AAAI-02.*
4. Freuder E, Likitvivatanavong C and Wallace R, 2000, "A Case Study in Explanation and Implication", *Proc. CP-2000 Workshop on Analysis and Vizualization of Constraint Programs and Solvers.*
5. Junker U, 2001, "Quickxplain: Conflict detection for arbitrary constraint propagation algorithms", *IJCAI-2001 Workshop on Modeling and Solving Problems with Constraints.*
6. Jussien N and Barichard V, 2000, "The PaLM system: explanation-based constraint programming", *CP-2000 Workshop on Techniques for Implementing Constraint Programming Systems.*
7. Larkin D, "Generating Random Solutions from a Constraint Satisfaction problem with Controlled Probability", *Proceedings CP-2002.*
8. Mitchell M, 1997, *Machine Learning*, McGraw-Hill.
9. Shannon C and Weaver W, 1949, *The Mathematical Theory of Communication*, University of Ilinois Press.

[12] Modulo the remark, made earlier, about hill-climbing.

Challenging explanations for global constraints

Guillaume Rochart[1,2], Narendra Jussien[1], and François Laburthe[3]

[1] Département Informatique de l'École des Mines de Nantes
4, rue Alfred Kastler - B.P. 20722 – F-44307 Nantes Cedex 3
[2] Institut de Recherche en Informatique de Nantes – Université de Nantes
2, rue de la Houssinière - B.P. 92208 – F-44322 Nantes Cedex 3
[3] Bouygues e-lab
1 av. Eugène Freyssinet – F-78061 St Quentin en Yvelines Cedex
{grochart,jussien}@emn.fr,flaburthe@bouygues.com

Abstract. This article presents the challenge of implementing explanations within global constraints. After defining explanations, it introduces what explanations for global constraints could be through the example of the `all_different` constraint, then it presents the issues of their implementation and the interest of precise explanations. At last, it illustrates these principles with the `stretch` and `flow` constraints.

1 Introduction

Numerous industrial problems can be modelled as constraint satisfaction problems: scheduling, call centres, television spots, etc. Constraint programming offers high-level modelling and reusable techniques for solving such problems. In order to provide efficient solvers and to better meet user needs, global constraints are often used. They model complex constraints over numerous variables: `gcc` [17], `all_different` [16] or `stretch` [15].

Explanation-based algorithms like dynamic backtracking [8] or its extension mac-dbt [11] have now proven their efficiency. Explanations can also be used for debugging purposes (from problem modelling to constraint implementation itself) or to point out a part of the problem responsible for a contradiction. Hence, explained implementations of global constraints must be provided to solve real life problems. The aim of this work is to show what are the issues of adding explanations capabilities to global constraints and that investing in sophisticated algorithms to provide precise explanations is useful for both solving and debugging purposes.

First, explanations are introduced along with some explanation-based algorithms. Then, we show how explanations for global constraints could be computed thanks to the filtering algorithms. This is done through the case study of the `all_different` constraint [16]. We illustrate why such implementations are complex and above all why it can be really useful for debugging, documenting or solving purpose. Last, we present an application to some global constraints in order to illustrate these principles and to present some experimentations for the `stretch` constraint [15].

2 Explanations

Solving constraint satisfaction problems is often based upon chronological backtracking algorithms. The main disadvantages of these algorithms are well known: the *thrashing* phenomenon due to the impossibility to remember past failure conditions and to the poor relevance, in general, of getting back to the last choice point.

2.1 Definition

To compensate thrashing, explanation-based techniques were proposed [8, 10]. An explanation contains enough information to justify a decision (throwing a contradiction, reducing a domain...): it is composed of constraints and choices made during the search sufficient to justify such an inference.

Definition 1 (Explanation) *The explanation of an inference $\mathcal{X}$ (a filtering decision like value removal or bound modification for instance) is a subset of original user constraints ($\mathcal{C}' \subset \mathcal{C}$) and instantiation constraints (choices made during the search: $d_1, d_2, \ldots, d_k$) such that:*

$$\underbrace{\mathcal{C}' \wedge d_1 \wedge \ldots \wedge d_n}_{explanation} \Rightarrow \mathcal{X}$$

An explanation e_1 is said to be *more precise* than explanation e_2 if and only if $e_1 \subset e_2$. The more precise is an explanation, the more useful it is.

2.2 Explanation-based algorithms

Thanks to this information about propagation, algorithms such as dynamic backtracking (dbt [8]) know all the instantiations that imply a contradiction, and so can determine which instantiation should be undone (not necessarily the last one). The instantiation order is then modified to undo this instantiation and only this one (keeping non related inferences made in between).

A drawback of dynamic backtracking is that it does not take advantage of propagation techniques. mac-dbt is an algorithm which allows to maintain arc-consistency (mac [20]) within dbt. As illustrated in Figure 1, this algorithm extends the current partial solution (by instantiating variables) as long as no contradiction occurs. If a contradiction is raised, it is handled as described in Figure 2: a contradiction explanations is computed and a constraint is selected within. The incremental removal of this constraint will hopefully overcome the contradiction. In order to avoid unnecessary loops, the negation (obtained using the opposite function) of the retracted constraint is posted. Last, all the constraints are re-propagated if necessary to achieve a given local consistency.

This algorithm offers advantages from both filtering and repairing techniques (see Figure 1) but it requires that all filtering decisions are explained (contrarily

to dynamic backtracking that only needs explanations for contradictions). Moreover, since the cancelled decisions are not always the last choice, the implementation of an explained constraint must support incremental constraint removal that replaces backtracking (see Figure 2).

```
(1)  begin
(2)    while unassignedVars ≠ ∅ do
(3)      v ← problem.selectVarToAssign()
(4)      a ← problem.selectValToAssign(v)
(5)      try
(6)        problem.post(v == a)
(7)        problem.propagate()    % Filtering
(8)      catch
(9)        problem.handleContradiction()    % Decision repairing
(10)     endtry
(11)   endwhile
(12) end
```

Fig. 1. Generic explanation-based algorithm

```
(13) begin
(14)   e ← problem.getContradictionExplanation()
(15)   if e.isempty() then
(16)     problem.raiseProblemContradiction()    % Over-constrained problem
(17)   else
(18)     ct ← e.selectConstraintToRemove()
(19)     try
(20)       problem.remove(ct)    % Incremental constraint removal
(21)       e.delete(ct)
(22)       problem.post(opposite(ct), e)    % Contextual posting forbidding ct to be reintroduced
(23)       problem.propagate()    % Re-propagation
(24)     catch
(25)       problem.handleContradiction()    % Possible recursive contradiction
(26)     endtry
(27)   endif
(28) end
```

Fig. 2. Contradiction handling for mac-dbt algorithm

2.3 Computing explanations

The most interesting explanations are those which are minimal regarding inclusion. Those explanations allow highly focused information about dependency relations between constraints and variables. Junker [9] proposes the QUICKXPLAIN algorithm for computing minimal explanation with a non-intrusive conflict detection solution.

Unfortunately, computing such an explanation can be exponentially costly. We claim that a good compromise between precision and ease of computation is to use the solver embedded knowledge to provide interesting explanations[10].

Indeed, constraint solvers maintain information that makes possible the filtering decisions. By making that knowledge explicit, quite precise and interesting explanations can be computed more easily.

For example, let us consider two variables v_1 and v_2 whose domains are both $\{1, 2, 3\}$.

- Let c_1 be a first decision constraint: $c_1 : v_1 \geq 3$. Let us assume that the filtering algorithm in use is 2B-consistency. The constraint c_1 leads to the removal of $\{1, 2\}$ from the domain of v_1. An explanation for the new domain $\{3\}$ of v_1 is thus $\{c_1\}$.
- Let c_2 be a second constraint: $c_2 : v_2 \geq v_1$. Value 1 and value 2 of v_2 have no support in the domain of v_1, and thus c_2 leads to the removal of $\{1, 2\}$ from v_2. An explanation of the removal of $\{1, 2\}$ from v_2 will be: $c_1 \wedge c_2$ because c_2 leads to that removal only because previous removals occurred in v_1 due to c_1.

3 Explanations and global constraints

Global constraints considered as high-level algorithms designed for pruning large portions of the search space may not be, as is, well suited to provide precise explanations. Indeed, a theoretical analysis of the involved algorithm is often needed in order to provide precise explanations within a global constraint. Moreover, algorithms may need to be designed again in order to be able to both provide efficient propagation and efficient explanation computation. However, we show that such a hard work is well worth it when considering the different possible uses of explanations.

3.1 Explaining global constraint: a case study

Unlike simple constraints (such as inequalities presented in Section 2.3), explaining a global constraint may need the theoretical analysis of the algorithms used to perform the constraint propagation along with the used data structure. Although a simple explanation always exists (consider the explanations of the current domain of all variables of the considered constraint), it may not be useful (consider a problem with a single `all_different` posted on all the variables of the problem!). Some more precise information may be found. Consider for example the following situation where an `all_different` constraint [16] is considered over four variables with the domain $\{1, 2, 3, 4\}$, and suppose that after filtering, we obtain $v_1 = \{1, 2\}$ and $v_2 = \{1, 2\}$ as illustrated on the figure 3. We can deduce that v_3 and v_4 cannot have 1 or 2 in their domains since these values will be taken by the v_1 and v_2 variables. The explanation should only be composed of the explanations of v_1 and v_2 domains (that is the union of the explanations for each value removal and bound modifications). Indeed only these two domains imply these removals.

Understanding the way a global constraint is propagated is the first step to provide precise explanation. It may not be sufficient. Indeed, consider again

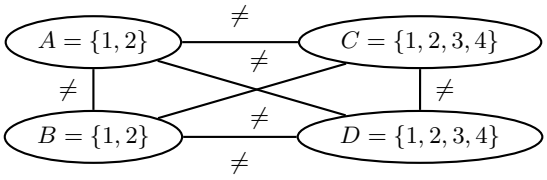


Fig. 3. Example of `all_different` constraint with four variables

the `all_different` constraint. The constraint can be seen as a flow problem as illustrated on the figure 3. Then, all the strongly connected components (SCC) are computed in the residual graph in order to know all the interchangeable affectations. Thus, if an edge is linking two different SCC, there is no way to make the flow go back to source SCC otherwise there would be a global SCC composed of at least these two ones. All the edges linking two SCC will be removed, since they cannot be used in any feasible solution to the `all_different` constraint [16].

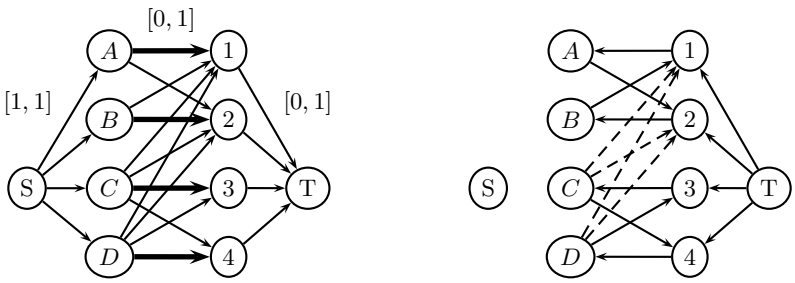


Fig. 4. The equivalent flow and the residual graph with its strongly connected components

The classical `all_different` constraint only needs to know that information about different SCC but if we want to justify value removal, we need to be able to explain why two nodes are in different SCC. We therefore need here an algorithm that does not only determine the different SCC but keeps track of the links between them. Indeed, the explanation is composed of all the edges that could link directly or indirectly the two strongly connected components that the edge to be removed links, as illustrated on the figure 5.

In the previous case (figure 4), an edge would be needed from the first component to the second one to avoid the removal. So the explanation will be composed of the explanation of the removal of 3 and 4 for the two variables A and B. That is the intuitive explanation of the removal we could make at the beginning of the section.

Formally, let $\leq$ be an order such that $SCC_k \leq SCC_l$ if and only if there is a path in the residual graph from SCC_l to SCC_k. Then the formal explanation

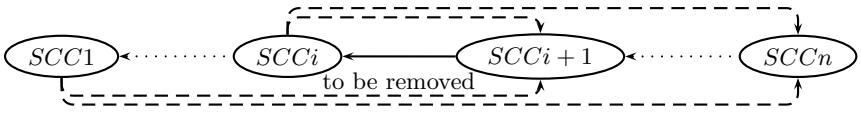


Fig. 5. Explanation for an edge removal: at least one of the dashed lines should be present to avoid the removal.

of a value removal will be (with i the number of the SCC containing the value u to be removed and $i+1$ the one containing the involved variable v):

$$expl(v \neq u) = \bigcup_{k, SCC_k \leq SCC_i} \bigcup_{x \in X \cap SCC_k} \left(\bigcup_{l, SCC_l \geq SCC_{i+1}} \bigcup_{t \in D_x \cap SCC_l} expl(x \neq t) \right)$$

where X contains all the variables and D_x is the domain of the variable x. Since all the SCC form a partition, the complexity is obviously $O(nc)$ where n is the number of variable and c is the maximal cardinality of the domains.

To conclude, explaining global constraint needs to understand the way this constraint is propagated and to *extend* this algorithm to provide some useful explanations.

3.2 The challenge of the implementation

Implementing an explained global constraint can be a challenge for two reasons: explanation-based constraint algorithms [10] need specific incremental algorithms and explained constraints need to remain efficient *w.r.t.* time complexity.

Algorithms like dynamic backtracking [8], mac-dbt [11] and decision-repair [12] have been proven quite efficient on structured problem. Their main feature is to replace chronological backtracking with a repair technique (see Figure 1) that reduces thrashing and allows an efficient conflict-driven exploration. In such an algorithm, the enumeration process is considered as a dynamic problem: enumeration constraints are dynamically added and removed (when repairing or backtracking) during search. Therefore, such algorithms do not rely on the backtracking process to maintain data structures and need explicit incremental algorithms for both handling value removals (constraint propagation) and value additions (repairing or backtracking) in the domain of the variables. This can be tricky for specific algorithms.

Moreover, the resulting propagation algorithm need to be efficient for both explanation computation and propagation itself. For example, in the `all_different` constraint the use of the underlying data structure keeping track of the SCC will be used for the explained version. Moreover, it can be easily proven that for `all_different` keeping also track of the relations between SCC can be done without modifying the overall complexity of the propagation algorithm leading to an efficient explained version of the constraint (even if computing the actual explanations has a cost as we have seen if the previous section).

However, this is not always the case. For example, when considering scheduling problems, one of the key techniques used is called *immediate selection*. It is a domain reduction technique for unary resource constraints [6]. The main idea of *immediate selection* is to identify a task t and a set S of tasks that share a common unary resource such that it can be proven that t cannot be scheduled before any task in set S. The lower bound for the starting time of task t can be modified to reflect that information. There are (at least) two implementations of *immediate selection*:

- one [6] does not explicitly compute set S but only the adjustment that can be made to the starting time of task t: it is not explanation-friendly. There is no way of precisely explaining the adjustments.
- another one [7] uses another point of view. Its idea is to maintain a set of tasks (prospective sets S) that fit in a given evolutive interval of time (namely *task-intervals*) and to check whether a candidate task t exists for adjustments. This is a technique equally efficient to the first one but it is explanation-friendly. Indeed, an explanation for any adjustment can be restricted to explanations of the current domains of the tasks in set S that is explicitly available. [12] shows the interest of such a technique for explanations.

3.3 Utility of explained global constraints

As we saw, designing explained global constraints can be a tedious task. However, we strongly believe that it is well worth it. Several usage of such efficient constraints exist:

- user-interaction requires precise explanations in order to point out parts of problems (set of constraints) responsible for value removal, current solutions, contradictions. For instance, computing timetables for a college can be dramatically simplified thanks to precise explanations of contradiction in order to modify only the involved constraints. Moreover, efficiently explained global constraint can be considered within recent debugging tools based on explanations [14].
- explanation-based constraint programming needs precise information to provide efficient solvers. The embedded conflict-driven techniques need precise explanations in order to directly perform the *good* repairs during search.
- explained propagation algorithms can act as self documentation for constraint solvers. Using and designing explained global constraints help understanding and diffusing the constraint technology by providing self documented tools. For example, consider scheduling problems that we mentioned earlier, the task-interval based implementation is much more understandable than an immediate-selection one.

4 Applications

We began to instrument various global constraints following the principles described above. We quickly present here what we have done for two global constraints after introducing the `palm` system.

4.1 The `palm` system

`palm` is an explanation-based constraint solver [10] provided as a free `choco` library. The main feature of `palm` is that it provides tools to explicitly compute, store and retrieve explanations for every domain modification in a given problem. But, it is also a constraint solver:

- `palm` is a **classic** constraint solver: it can be used to solve problems as if it were `choco`.
- `palm` is a **dynamic** constraint solver: it handles dynamic addition and retraction of constraints before, during or after resolution.
- `palm` is an **explanation-based** constraint solver: it provides specific search algorithms that make an active use of explanations (as illustrated on Figures 1, 2 in Section 2.2).

In the following, we will provide examples and experiments using the `palm` system.

4.2 The `stretch` constraint

This constraint was proposed by [15] for assigning shifts to employees (in call centres or for a nurse planning for instance). It allows to specify which minimal and maximal length, each stretch of identical sequencing values may have. It may be used for legal constraints (a nurse can not work on evening more than five sequencing days) or preference constraints (a nurse should have at least two days off between working stretches).

A *stretch* [15] is formally defined as a subsequence $x_i, \dots x_j$ of variables such that $x_i = \dots = x_j$ but $x_{(i-1) \bmod n} \neq x_i$ and $x_j \neq x_{(j+1) \bmod n}$. The *span* of the stretch containing x_k is then defined by $span(x_k) = 1 + (j - i) \bmod n$. Let $\check{\lambda}$ and $\hat{\lambda}$ be two vectors of length m (the domain size). The constraint `stretch` implies $\check{\lambda}_{x_k} \leq span(x_k) \leq \hat{\lambda}_{x_k}$.

For the filtering algorithm of this constraint, bounds are computed to know the minimal and the maximal span of the studied stretch. For instance, if the span is necessarily too great, a contradiction should be thrown.

Bound computation. To know the minimal span (that is, the variables that must be in the current stretch), the maximal beginning (β_{max}) and the minimal ending (ϵ_{min}) bounds should be computed and explained by the algorithm. In the case of these bounds, the computation is really easy: the variables are scanned until a non instantiated variables is reached. Then, the following explanation can be deduced: $expl(\beta_{max} \leq j) = \bigcup_{k \in [\![j,i]\!]} expl(\mathcal{D}_{x_j})$ where $\mathcal{D}_y$ is the domain of y and $expl(\mathcal{D}_y)$ the explanation of its state. The same explanation can be deduced for ϵ_{min}.

Filtering explanation. Let us take an example: one of the filtering rules checks that the span length is correct. If the stretch is too long, the following explanation can then be deduced: $expl(contradiction) = expl(\beta_{max}) \cup expl(\epsilon_{min})$. More generally, filtering decisions are explained using the explanation of the computed bounds [19].

Experimentations In order to check that using explanation does not decrease the performance, we evaluated the performance of an explanation-based algorithm: mac-dbt [11].

Several versions of the `stretch` constraint were implemented: a classical version (thanks to `choco`) and two explained versions: one with basic explanations (explanations of the domains of all the variables in the constraint) and one with the proposed explanations.

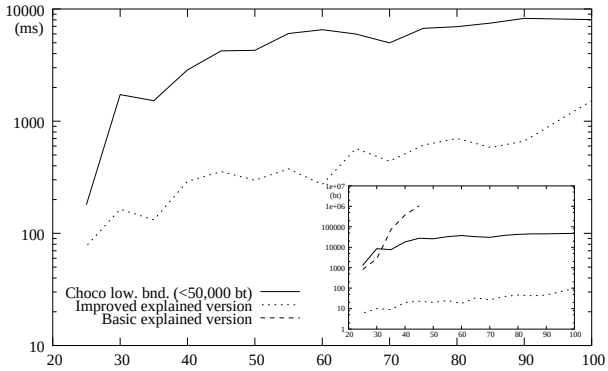


Fig. 6. Comparison between the different version (CPU time and BT number) according to the number of variables. The results for the `choco` version are lower bounds of the real values: the resolution is limited to 50,000 backtracks.

To estimate the contribution of explanations for the interaction of global constraints, we tried to make some tests with two overlapped constraints: half of all the variables are shared between the two instances of the constraint. Tests were made with 25 to 100 variables (each time with 20 instances to get significant mean values) with the three versions of the constraint: the `choco` version, the basic and improved explained versions. As in the original paper about `stretch` [15], instantiation and value choices are randomly made to simulate a complex problem.

The results presented in figure 6 clearly show that computation time and number of backtracks needed to solve the problems are lower in the case of the improved explained version than with the other versions. Indeed, in the `choco` version and the basic explained version, no information is shared between the two constraints. So the constraints work blindly and this implies a lot of unnecessary

backtracks. With the improved explained version, precise explanations allow to identify quickly and efficiently a subset of variables responsible of a contradiction, which permits a sensible improvement of the resolution.

4.3 Explaining a maximal flow

Following [2], we define a network as a directed graph $G(X, A, L, U, s, t)$ with: L lower bounds on the edges, U upper bounds, and s and t are the source and the sink of the network. A *flow* can then be defined as a function $\Phi : X \times X \rightarrow \mathbb{N}$ with:

- for each edge (i, j), $L_{ij} \leq \Phi_{ij} \leq U_{ij}$ (capacity constraint),
- for each node i, $\sum_{incoming} \Phi_{ji} = \sum_{outgoing} \Phi_{ij}$ (flow conservation constraint).

A `flow` *constraint* (see the second section of [4]) allows to check that there exists a flow Φ in G such that $\sum_i \Phi_{si}$ is in the domain of a variable F.

Explanation. A *cut of* G [2] is a subset S of X such that $s \in S$. This implies the existence of outgoing and incoming edges, linking S and $X \backslash S$. The *cut capacity* [2] can be defined by: $C(S,T) = \sum_{outgoing} U_{ij} - \sum_{incoming} L_{ij}$. Last, the min-cut/max-flow [2] property guarantees that "the value of any flow is less than or equal to the capacity of any cut in the network".

This implies the following explanation for $F \leq C(S,T)$ [18]:

$$expl(F \leq C(S,T)) = \bigcup_{outgoing} expl(bsup(V_{ij})) \cup \bigcup_{incoming} expl(binf(V_{ij}))$$

To get an explanation of the maximal flow F, the minimal cut should be chosen, since it can be proven that the maximal flow (if it exists) equals the minimal cut. This mechanism can be extended for other filtering inferences [18].

For instance, on figure 7, a maximal flow is provided. The cut is here basically $\{1\}$. So the explanation of the maximal flow is composed of the explanations of the maximal bounds on the two edges outgoing from 1. The following explanation would be used:

$$expl(F \leq 10) = expl(bsup(V_{12})) \cup expl(bsup(V_{16}))$$

Computing such explanations is quite easy. Indeed a really simple algorithm in $O(nm)$ (with n the number of nodes and m the number of edges) can be used to compute a minimal cut as soon as a maximal flow has been found :

- first marking all the accessible nodes from the source in the residual graph ($O(nm)$);
- then selecting all the edges linking a marked node to an unmarked one ($O(m)$).

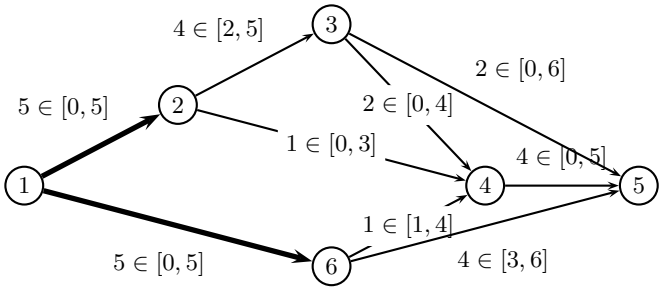


Fig. 7. Explaining a maximal flow

Application to the `all_different` constraint The `all_different` constraint can be explained with specialised explanations as proposed by [1] and as we explained in the section 3.1. But, `all_different` [16] and `gcc` (Global Cardinality Constraint) [17] constraints are specialisation of the flow constraint. Thus the explanations proposed for the flow constraint can be applied to these constraints too.

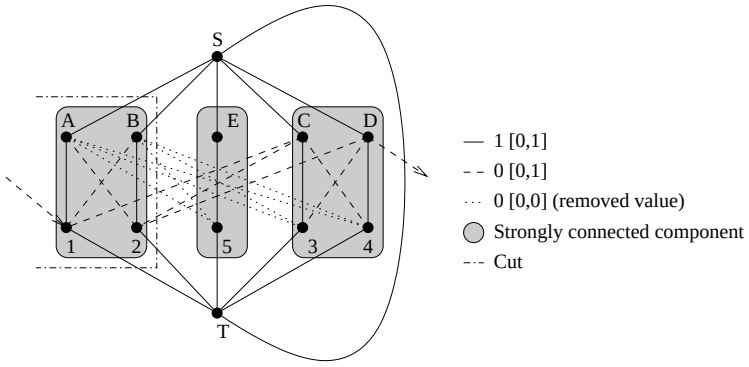


Fig. 8. A `flow` explanation for the `all_different` constraint

For instance, on the figure 8, we applied the `flow` explanation with the case study of `all_different` presented in the section 3.1 where we added a fifth variable with only 5 in the domain and we added the 5 value in the domain of A and B. As the figure shows, the cuts will include all the edges that link directly or indirectly the two components (like $(A, 3)$ for instance). But, it will contain some more edges. First, it contains some edges where no variable are attached (on the bottom and on the top of the graph): this is not a problem, the algorithm do not take care of such edges. But, it contains for instance the edge $(B, 5)$ which is useless, since there is no outgoing edges in the component $\{E, 5\}$.

In this precise case, the cut will provide some explanation which is actually not needed.

To conclude, similar explanations are found (that is edges linking two components) but the flow explanations are a bit less precise than the one presented in the section 3.1.

5 Discussion

We introduced in this paper how providing explanations for global constraints filtering algorithms. But we supposed that the user only need to know which assignments or which constraints are responsible for a contradiction. However one could need which part of a global constraint is responsible. For instance, with the `all_different` constraint, it may happen that only some of the inequalities are responsible for the contradiction and not all the global constraint[4].

Actually, this issue is only relevant for global constraints that may be decomposed into a conjunction of basic constraints. For *semantically global constraint* as defined in [5], the proposed explanations are totally relevant since they cannot be decomposed in sub-constraints. But for the other global constraints, it could be useful in future works to provide precise explanations with the exact sub-constraints responsible for each filtering decision, thanks to global constraints decomposition like the decompositions proposed by [3].

6 Conclusion

We introduced in this article how explanations for global constraints can be defined thanks to the use and the extension of the filtering algorithms. Then, we summarised the main issues of these implementations: the incremental aspect of these constraints, the need of explanation friendly algorithms, the lack of minimal explanations or the coarse grain aspect of explanations. Precise explanations are particularly useful for debugging, documenting and solving purpose since it provides quite precise information about each decision. Last we illustrated these principles through the implementation of explained versions of `stretch`, `flow` and `all_different` constraints.

Explanations for global constraints open new fields. First, explanations could be used for solver cooperation by providing precise information about a fail or each decision taken by a solver. They could be used for documentation, analysis or debugging since explanations provide quite precise information about the working of these constraints and the solving of the problem. Last, a very interesting feature would be to provide a way to explain a constraint without modifying the constraint code, or a generic framework for explaining global constraints.

[4] We would like to thank the anonymous referee who pointed this issue.

References

1. Magnus Ågren. Tracing and explaining the execution of clp(fd) programs in sicstus prolog. Master's thesis, Uppsala University, Sweden, July 2002.
2. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice Hall, New York, 1993.
3. Nicolas Beldiceanu. Global constraints as graph properties on structured network of elementary constaints of the same type. Technical report, SICS, 2000.
4. Thierry Benoist, Étienne Gaudin, and Benoît Rottembourg. Constraint Programming Contribution to Benders Decomposition: A Case Study. In *Principles and Practice of Constraint Programming (CP 2002)*, LNCS, pages 603–617, 2002.
5. Christian Bessière and Pascal Van Hentenryck. To be or not to be... a global constraint. In *Principles and Practice of Constraint Programming CP'03*, 2003.
6. Jacques Carlier and Éric Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161, 1994.
7. Yves Caseau and François Laburthe. Improving clp scheduling with task intervals. In P. Van Hentenryck, editor, *Proc. of the 11th International Conference on Logic Programming, ICLP'94*, pages 369–383. MIT Press, 1994.
8. Matthew Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.
9. Ulrich Junker. QuickXplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving Problems with Constraints*, 2001.
10. Narendra Jussien. e-constraints: explanation-based constraint programming. In *CP01 Workshop on User-Interaction in Constraint Satisfaction*, Cyprus, 2001.
11. Narendra Jussien, Romuald Debruyne, and Patrice Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Principles and Practice of Constraint Programming (CP 2000)*, LNCS, pages 249–261, Singapore, 2000.
12. Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139, July 2002.
13. François Laburthe. Choco: implementing a cp kernel. In *Proceedings of TRICS, a post-conference workshop of CP 2000*, Singapore, 2000.
14. Samir Ouis, Narendra Jussien, and Patrice Boizumault. k-relevant explanations for constraint programming. In *FLAIRS'03: Sixteenth international Florida Artificial Intelligence Research Society conference*, St. Augustine, USA, 2003.
15. Gilles Pesant. A filtering algorithm for the stretch constraint. In *Principles and Practice of CP (CP 2001)*, LNCS, pages 183–195, 2001.
16. Jean-Charles Régin. A filtering algorithm for constraints of difference in CSPs. In *AAAI 94, Twelth National Conference on AI*, pages 362–367, Seattle, 1994.
17. Jean-Charles Régin. Generalized arc consistency for global cardinality constraint. *AAAI*, 1:209–215, 1996.
18. Guillaume Rochart and Narendra Jussien. Explanations for a flow constraint (in French). Technical report, École des Mines de Nantes and Bouygues SA, 2002.
19. Guillaume Rochart and Narendra Jussien. Explanations for global constraints: instrumenting the stretch constraint. Technical report, École des Mines de Nantes, 03/1/INFO, 2003.
20. Daniel Sabin and Eugene C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Principles and Practice of CP*, LNCS, 1994. Second International Workshop, PPCP'94, USA.

Compiling CSPs into tree-driven automata for interactive solving

Hélène Fargier[1] and Marie-Catherine Vilarem[2]

[1] IRIT, Toulouse
[2] LIRMM, Montpellier

Abstract. *Constraint programming techniques are widely used to model and solve decision problems and many algorithms have been developed to solve automatically and efficiently families of CSPs; nevertheless, they do not help solve interactive decision support problems, like product configuration. In such problems, the user herself chooses the values of the variables, and the role of the system is not to solve the CSP, but to help the user in this task. Dynamic global consistency maintaining is one of the most useful functionalities that should be offered by such a CSP platform. Unfortunately, this task is intractable in the worst case. Since interactivity requires short response times, intractability must be circumvented some way. To this end, compilation methods have been proposed that transform the original problem into a data structure allowing a short response time. In this paper, we extend the work of [14, 1] by the use of a new structure, tree-driven automata, that take advantage of the structural characteristics of configuration problems (decomposition of the components into independent subcomponents). Tree-driven automata can be far more compact than classical automata while keeping their good properties, especially a tractable complexity for the maintenance of global consistency.*

1 Introduction

Constraint programming techniques are widely used to model and solve decision problems and many algorithms have been developed to solve automatically and efficiently some families of CSPs; nevertheless, they do not help solve decision support problems that are interactive in essence. For such problems, the user is in charge of the choice of values for the variables, and the role of the system is not to solve the CSP, but to help the user in this task. Product configuration [13][11] is a typical example of such problems: a configurable product is defined by a finite set of components, options, or more generally by a set of attributes, the values of which have to be chosen by the user. These values must satisfy a finite set of configuration constraints that encode the feasibility of the product, the compatibility between components, their availability, etc. A configurable product can thus be represented by means of a CSP, the solutions of which represents the catalog, i.e., all the feasible variants of the product.

When configuring a product, the user specifies her requirements by interactively giving values to variables or more generally by stating some unary constraints that restrict the possible values of the decision variables. Now each time a new choice is made, the domains of the variables must be pruned so as to ensure that the values available for the further variables can lead to a feasible product (i.e., a product satisfying all the initial configuration constraints). These dynamical constraints can also be removed during the configuration process because they lead to a solution that is judged unacceptable by the user. Dynamic global consistency maintaining is thus one of the most useful functionalities that should be offered by the CSP platform.

Unfortunately, this task is intractable in the worst case [1]. Since interactivity requires short response times, intractability must be circumvented in some way. To this end, compilation methods have been proposed [15, 14, 1], that transform the original problem into a data structure allowing a short response time. This principle was already be used in different domains of Artificial Intelligence, like planning under uncertainty [5] or automated reasoning [3] [6].

The approach of [14, 1] emphasizes the compilation of the original problem into a finite automaton, from which much better performance can be obtained. The construction of the automaton requires choosing carefully a total order on the variables: otherwise, the size of the automaton, exponential in the worst case, could be too large to be used in practice. This approach relies on the frequent interchangeability of values in real configurable products: it can thus be described concisely by the automaton. But, contrarily to other constraint-based approaches to product configuration ([7] [8][10]), it does not take advantage of the following structural characteristic: complex products are generally structured into subcomponents (more or less independent from each other) the existence of which depends on the values given to some of the variables of the upper component. The linear order on the variables required by the Vempaty's automaton is not suited at all for adding and removing sub-components.

In the present paper, we propose a new structure, tree-driven automaton, to represent the solution set of a CSP. Shortly, it can be understood as a tree of small linear automata, each of them corresponding to a sub-component (and recursively). Tree driven automata are obviously a generalization of classical, linear, automata. Importantly, they can be far more compact than classical automata while keeping their good properties, especially a tractable complexity for the maintenance of global consistency. In order to rigorously show the interest of our structure, we to proof that there are some CSP that necessarily lead to an exponentially-sized linear automaton, and whereas they yield a polynomialy sized tree-driven automaton.

In Section 2, we briefly recall how a linear automaton can represent the set of solutions of a CSP. Section 3 then presents tree-driven automata and Section 4 shows how a useful set of computational tasks (such as maintaining global consistency) can be efficiently achieved on this structure. In Section 5, we finally study how a tree-driven automaton can be more compact than a classical automaton.

2 Compiling a CSP into a finite automaton [14][1]

Automata
A finite-state automaton (*FA*) is a directed graph the edges of which (the *transitions* of the automaton) are labelled by elements of a set Σ (the *alphabet*). The nodes of the graph are called *states*; this graph has one initial state I and at least one final state. It is such that any transition and any state belong to at least one path from I to a final state.

A word $m = a_1 a_2 \dots a_n$ is recognized by the automaton if there exists a path from the initial state I to a final state F with the label m. The language recognized by the automaton is the set of words it recognizes.

Finally, a finite automaton is said to be deterministic if and only if all the transitions coming from a single state have different labels.

Associating FA with CSP
Let $\Pi = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP. Given a permutation $O = [X_1, X_2, \dots, X_n]$ of $\mathcal{X}$, any solution of $\Pi = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ defines a word of length n over the alphabet $\mathcal{D}$. Hence, the set of solutions of Π defines a language over $\mathcal{D}$. This language, called the solution language of Π w.r.t. O and denoted $\mathcal{S}_O$ can be represented by a FA. This automaton has only one initial state I and one final state F, and is such that the length of any path from I to F is n.

Computing the automaton associated with a CSP
The automaton can be generated by composition operators on (small) automata representing the constraints of $\mathcal{C}$. The complexity of the computation obviously depends on its size, which is mainly influenced by the order in which the variables of the CSP are taken into account [2].

3 Tree-driven automata

As said in the introduction the structure of configuration CSPs is close to an hypertree (each hyper-edge corresponding to a sub-component) Moreover, the sub-components induced by the user's choices of values for variables are loosely connected to the upper-components. So we wish to take better advantage of this nearly tree structure.

3.1 Definitions

Definition 1. *The quotient of a graph (Q, E) by coloring τ of its nodes, is the graph $\mathcal{T} = (Q_\mathcal{T}, E_\mathcal{T})$ defined by:*

$$Q_\mathcal{T} = \{\tau(q), q \in Q\} = \{0, 1, \dots, n\}$$
$$E_\mathcal{T} = \{\{i, j\} / \exists\ q, q' \in Q \ s.t.\ \tau(q) = i, \tau(q') = j, \{q, q'\} \in E\}$$

Definition 2. *A **tree-driven automaton** $\mathcal{A}$ of order n on the alphabet Σ is a graph (Q, E) such that:*

- *there is a coloring of the vertices Q with $n+1$ colors, denoted by τ : each vertex is labelled by an element of $\{0, 1, \ldots, n\}$ and adjacent vertices have different colors*
- *the edges are labelled by subsets of Σ ; this labelling is denoted by σ*
- *the quotient (Q, E) by τ, denoted by $\mathcal{T} = (Q_\mathcal{T}, E_\mathcal{T})$ is a tree ($\mathcal{T} = (Q_\mathcal{T}, E_\mathcal{T})$ is called the* support *of $\mathcal{A}$).*

Definition 3. *A **transition** of $\mathcal{A}$ is a pair $\langle a, e\rangle$ where e is an edge of E and a an element of Σ belonging to the labelling of e ($a \in \sigma(e)$)*

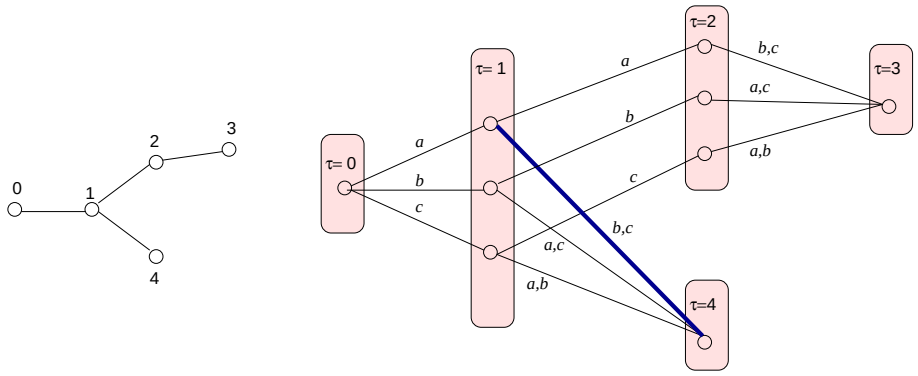


Fig. 1. An example of a tree-driven automaton and its support; the bold edge defines two transitions corresponding to the two labels b and c.

Since a chain is a tree, a classical, linear, automata is no more than a tree-driven automata the support of which is a chain (it is enough to build a τ that maps every vertex of rank X_i to X_i)

Definition 4. *A **word** on a support $\mathcal{T} = (Q_\mathcal{T}, E_\mathcal{T})$ is a labelling w of $E_\mathcal{T}$ which associates to each edge of $\mathcal{T}$ an element of Σ.*

*A **$\mathcal{T}$-tree** t is a labelled subgraph (Q_t, E_t, w_t) of $\mathcal{A}$ which is isomorphic to $\mathcal{T}$ (i.e. (Q_t, E_t) is a tree, τ induces a one-to-one correspondence between Q_t and $Q_\mathcal{T}$ and between E_t and $E_\mathcal{T}$) and, for each edge e of $E_t, w_t(e) \in \sigma(e)$.*

In fact, E_t and w_t define a set of transitions. In the following, for sake of simplicity, we consider a $\mathcal{T}$-tree as a set of transitions.

*A **word w is recognized** by a tree-driven automaton $\mathcal{A}$ with support $\mathcal{T}$ if it has an associated $\mathcal{T}$-tree in $\mathcal{A}$, i.e. if there exists a $\mathcal{T}$-tree t of $\mathcal{A}$ such that for each e in E_t, $w_t(e) = w(\tau(e))$.*

The language recognized by a tree-driven automaton $\mathcal{A}$ is the set $\mathcal{L}(\mathcal{A})$ of the words that it recognizes.

Equivalent tree-driven automata : *two tree-driven automata built on the same support are equivalent if and only if they recognize the same language.*

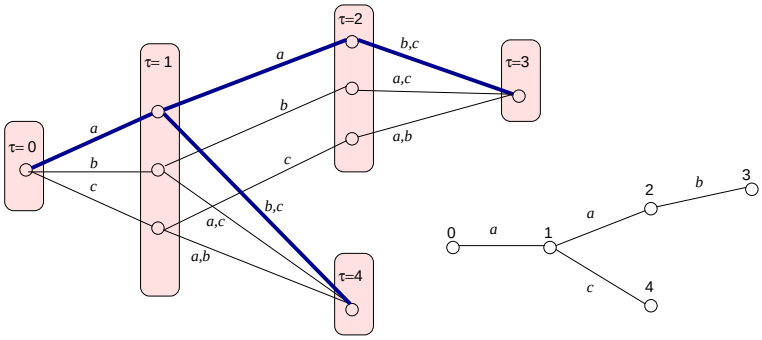


Fig. 2. An example of a word recognized by the tree-driven automaton on the left; the bold edges of the tree-driven automaton define four $\mathcal{T}$-tree , one of them corresponds to the word on the right

3.2 Special cases of tree-driven automata

- *Normalized tree-driven automata*
  Let $\mathcal{A} = (Q, E)$ be a tree-driven automaton with support $\mathcal{T}$ and let $Q_i = \{q \in Q \text{ such that } \tau(q) = i\}$. It is easy to see that collapsing all the vertices of Q_i when i is a leaf of $\mathcal{T}$ gives an automaton equivalent to $\mathcal{A}$ (this operation is called the normalization of the automaton). So, in the following, we consider only normalized automata i.e. automata such that for each leaf i of $\mathcal{T}$, $\#Q_i = 1$.
- *Trimmed tree-driven automata*
  A tree-driven automaton $\mathcal{A}$ may contain useless vertices : vertices which do not belong to any subtree isomorphic to the support. Formally, a vertex $q \in Q_i$ of $\mathcal{A} = (Q, E)$ is useless if $\exists j$ s.t. $\{i, j\} \in E_{\mathcal{T}}$, and there is no edge between q and a vertex of Q_j.
  The suppression of the useless vertices (and of their incident edges) yields an equivalent automaton and can be achieved in linear time in the size of (Q, E).
- *Deterministic tree-driven automata*
  It is also possible to extend the classical definition and properties of deterministic automata to tree-driven automata. The key point is that the support

is no longer an undirected tree, but must be a directed rooted tree (so, one consider a τ defines a directed rooted tree) : A tree driven automata is said to be deterministic if and only if it is normalized, it only has one initial state (of level 0), and for any couple of transitions (x_1, y_1) and (x_2, y_2) of the same level ($\tau(y_1) = \tau(y_2)$ and $\tau(y_1) = \tau(y_2)$), $x_1 = x_2$ and $y_1 \neq y_2 \Rightarrow \sigma((x_1, y_1)) \cap \sigma((x_2, y_2)) = \emptyset$. The definition is quasi-equivalent to the one provided for linear automata, and it can be proved that, like in the latter case, the deterministic representation of a language is unique (up to a given τ). Any tree driven automaton can be determinized with a procedure very close to the one used for classical automata – as in this latter case, it is polynomial in the size of the original automaton.

3.3 Associating a tree-driven automata with a CSP

So, in order to represent the solution set of a CSP, we can choose a support such that there is a one-to-one correspondence between the edges of the support and the variables of the CSP. A word can then be build for each solution, and we associate to the CSP the tree-driven automaton the langage of which is precisely this set of words.

3.4 Associating a tree-driven automaton with a tree-structured CSP

When the CSP is tree-structured, a tree-driven automaton can be built which is very close to its microstructure.

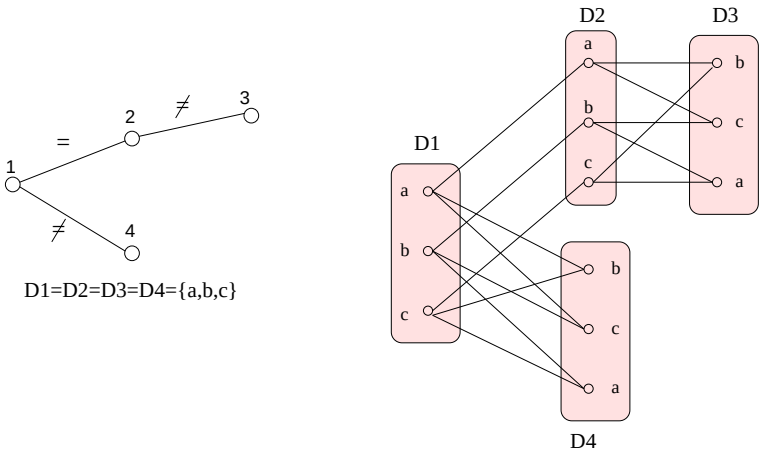


Fig. 3. A CSP Π and its microstructure

We have noticed previously that the edges of the support are in one-to-one correspondence with the variables of the CSP ; so we can slightly modify the constraint graph of Π in order to obtain the support : we add a new vertex (0) and a new edge , linking 0 and another vertex 1 for instance (this vertex is called the *handle* – it roots the tree). This new graph form the support of the forthcoming automata. Then we construct $\mathcal{A}$ by adding a new vertex q_0 to the micro-structure of Π which is linked to all the vertices corresponding to D_1. q_0 is the root of a tree. The labelling σ is then obtained by making the values (that originally pertain to the vertices of the micro structure) "slide" along the incident edges.

For instance, Figure 3 defines a CSP Π and shows its microstructure, while Figure 2 shows the tree-driven automaton $\mathcal{A}$ built by this procedure from Π.

3.5 Associating a tree driven automaton with a CSP

Finding a support
The first step is to build a tree, the vertices of which correspond to the variables $\{1, \ldots, n\}$. The second step is to add a "handle"(0) and a new edge that links it to the tree (say for example $(0, 1)$). We thus obtain a tree with vertices $\{0, \ldots, n\}$. There is a one-to-one correspondence between the edges of this tree and the variables of the CSP, and also between the set of vertices $\{1, \ldots, n\}$ and the variables of CSP - this defines the mapping τ and the support of the automaton.

Computing a tree-driven automaton
The approach is similar to the one used for classical automata. In theory, it is possible to compute the solution set of $\Pi = (\mathcal{X}, \mathcal{D}, \mathcal{C})$, and so the set of words which must be recognized by the automaton. This set forms a tree-driven automaton. The normalization (and other reductions operations which are beyond the scope of this paper) of this automaton yield an exploitable tree-driven automaton. Another method,that relies on tree-driven automata composition can also be used. We don't address here the problem of efficiently computing the tree-driven automaton.

The computational cost depends on the size of the tree-driven automaton and of course on the choice of the underlying tree. The details are beyond the scope of this paper. Nevertheless, since in configuration problems the constraint graph is close to a tree, the choice is rather natural.

4 Maintaining global consistency

Following [1], we consider that in configuration problems, a CSP $\Pi = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ defines the feasible products, and that a set of constraints $\mathcal{H}$ represents the current set of user's choices; we also assume that $\mathcal{H}$ is a set of unary constraints.

To be useful, a decision support system for such problems should at least achieve :

(1) detection of inconsistency of $\Pi' = (\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H})$: at any time, the system must tell whether there is a feasible product satisfying the user's requirements
(2) maintenance of global consistency : at any time, the system must discard from the domain of available values those that cannot lead to a feasible product, i.e. it must be able to compute the projection of the solution set of Π' on each variable X_i.

Besides, other functionalities are necessary (for instance, the computation of nogoods in case of inconsistency), but we address here only the ones described above.

4.1 Adding costs to the edges of $\mathcal{A}$

The approach used to efficiently perform these tasks on a tree-driven automaton is a straightforward extension of the approach presented in [1]. The principle is to associate a cost ϕ to each transition of the automaton : for the transition $\langle a, e \rangle$, the cost $\phi(\langle a, e \rangle)$ is 1 if a corresponds to an assignment to $var(e)$ [3] forbidden by a constraint of $\mathcal{H}$, and 0 otherwise[4].

Definition 5. *Let $\mathcal{A} = (Q, E)$ be a tree-driven automaton with support $\mathcal{T} = (Q_\mathcal{T}, E_\mathcal{T})$.*

- *cost(t) $= \Sigma_{\langle a,e \rangle \in t}\ \phi(e, a)$ is the **cost of a** $\mathcal{T}$-tree t.*
- *cost($\langle a, e \rangle$) is the cost of a better (i.e. minimal cost) $\mathcal{T}$-tree which contains $\langle a, e \rangle$*
- *cost(q) with $q \in Q$ is the cost of a best $\mathcal{T}$-tree containing q*

Property 1. The CSP $\Pi' = (\mathcal{X}, \mathcal{D}, \mathcal{C} \cup \mathcal{H})$ is consistent if and only if there is in $\mathcal{A}$ a $\mathcal{T}$-tree of cost 0.

Property 2. Let P_{X_i} denote the projection of the solution set $S(\Pi')$ on the variable X_i. Then, a belongs to P_{X_i} if and only if there is a transition $\langle a, e \rangle$ in $\mathcal{A}$ such that $var(e) = X_i$ and $\langle a, e \rangle$ belongs to a $\mathcal{T}$-tree of cost 0 .

4.2 Algorithms

The tasks (1) and (2) amount to the search of some minimal $\mathcal{T}$-tree in the tree-driven automaton. We associate to each q of Q two counters : $c_l(q)$ and $c_r(q)$. For any vertex i of $Q_\mathcal{T}$ which is not a leaf, the removing of i induces two connected components ; by convention we call the one containing the vertex 0 the *left* one and denote it by Q_l^i; the other one is called the *right* one and is denoted by Q_r^i.

[3] For an edge e of $E_\mathcal{T}$, **var**(**e**) denotes the associated variable in $\mathcal{X}$

[4] for sake of simplicity, we choose here to consider that all the violations have the same cost 1 ; it is easy to extend this by associating a cost to any constraint of $\mathcal{H}$.

Definition 6. ***counters associated to the vertices of*** $\mathcal{A} = (Q, E)$

- *for* $q \in Q_i$,
  - $c_l(q)$ *is the minimal cost of a* $\mathcal{T}$*-tree of the subautomaton induced by* $\{q' \in Q_j, j \in Q_l^i \cup \{i\}\}$
  - $c_r(q)$ *is the minimal cost of a* $\mathcal{T}$*-tree of the sub-automaton induced by* $\{q' \in Q_j, j \in Q_r^i \cup \{i\}\}$
- *for the unique* $q_0 \in Q_0$, $c_l(q) = 0$
- *for each* $i \in Q_{\mathcal{T}}$ *which is a leaf,* $\forall q \in Q_i, c_r(q) = 0$

When unary constraints are added or deleted, the costs of some transitions have to be updated; the counters defined previously can then be maintained by propagation algorithms which are very similar to those used for classical automata in [1]. This work can be done in time polynomial in the size of the tree-driven automaton.

5 Space complexity

The computational cost of these algorithms depends heavily on the automaton size – it is also the case with linear automata. Our claim is that tree-driven automata can be far more compact than linear automata. The first justification relies on the characteristic of our target application: configurable products are often structured into independent sub-components the existence of which depends on the values given to some of the variables of the upper component. Such products can be understood as composite CSPs [8]. The linear order on the variables required by Vempaty's automata does not suit the compilation of such problems: we will show in the following that even a very simple, tree structured, CSP can lead to an exponential data structure. On the contrary, tree-driven automata allow to compose a tree of small linear automata, each of them corresponding to a sub-component (and recursively). Tree driven automata are a form of "composite" automata, comparable to a compiled for of composite CSPs. The tree automaton can indeed be syntethised dynamically, each component being "mounted" only once its existence has been decided by a choice of the user. So, tree-driven automata can be far more compact than classical automata while keeping their good properties, especially a tractable complexity for the maintenance of global consistency.

The second justification is a theoretical proof of this claim: it can be shown that there are some structures that necessarily need to an exponential linear automaton, while they yields a polynomialy-sized tree-driven automaton. We prove it by showing that tree structured CSPs always admit a polynomialy-sized tree automata, while some of them necessarily lead to an exponential linear automata.

Formally, we consider that the size of a CSP is given by the size of its microstructure (it is true at least if the constraints are given by a list of tuples) and we denote it by $size(\Pi)$.

A polynomial size representation of a CSP Π by an automaton is an automaton the size of which *whose size* is a polynomial function of $size(\Pi)$.

Property 3. Let Π be a tree-structured CSP, and let $\mathcal{A}$ be the associated normalized tree-driven automaton constructed as in section 3.4. The size of $\mathcal{A}$ is linear in $size(\Pi)$

This is trivial when considering the construction depicted in Section 3.4.

Property 4. There exists tree-structured CSPs which don't have a polynomial size representation by a classical (linear) deterministic automaton.

The following class of CSPs supports this property:

Definition 7. *Let h, k, d be three positive integers, with $k \geq 2, k < d - 1$. The CSP $\Pi(h, k, d) = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is defined by:*

- *the constraint-graph is the complete k-ary tree with height h (see Fig. 4 for an example with $k = 3$ and $h = 2$)*
  Therefore, we have $n = |\mathcal{X}| = \Sigma_{i=1,\ldots,h}\ k^i = (k^{h+1} - 1)/(k - 1)$.
- *$D_1 = \ldots = D_n = D$ with $|D| = d$*
- *each constraint relation is the relation $\neq$*

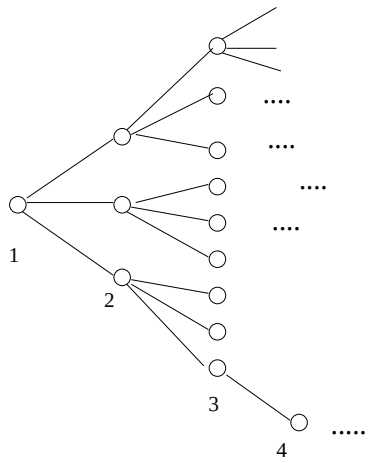


Fig. 4. The constraint graph of $\Pi(h, k, d)$ where $h = 2$ and $k = 3$

It can be shown that, whatever the order $\mathcal{O}$ chosen for building a deterministic linear automaton, it leads to an exponential data structure.

6 Conclusion

Solving decision support problems that are interactive in essence implies *achieving* efficiently some reasoning tasks (for example maintaining global consistency); as these tasks are highly combinatorial in the general case, a possible approach is to push the cost into an off-line pre-computation step ; [1] propose to compile the CSP under the form of a classical automaton. Under this form, the usual reasoning has a complexity polynomial in the size of the automaton. It suffers of two drawbacks: first, the size of the automaton can be very large, and finding a good order of the variables which minimizes it relies heavily on heuristics; secondly, the linear structure of the automaton does not really suit the decomposition to of products into components and sub-components.

In this paper, we have introduced tree-driven automata as a new compilation target. Tree-automata allow to take better advantage of the structure of the CSP, need less space (in the case of tree-structured CSPs, space is linear in the size of the microstructure), are more suited to the adjunction and removing of subcomponents, and keep polynomial complexity for usual reasoning tasks.

In the framework of automated reasoning, compilation methods have been proposed (see [6], which can be compared with ours (with the slight difference that they are restricted to boolean variables). Tree driven automata do not correspond exactly to any of these, but they take advantage of the same properties: decomposability, "Read once" property, order on the variables and determinism. This suggests to study other classes of representation, e.g. generalized tree driven automata (which should relax the partial order to a "read once" property) or multi-valued DNNF.

References

1. J. Amilhastre, H. Fargier, P. Marquis, Consistency restoration and explanations in dynamic CSPs - Application to configuration. Artificial Intelligence, 135, 199–234, 2002.
2. J. Amilhastre. *Représentation par automate de l'ensemble des solutions de problèmes de satisfaction de contraintes.* PhD thesis, Université du Languedoc, Montpellier, 1 1999.
3. M. Cadoli and F. M. Donini, A survey on knowledge compilation, *AI Communications*, **10**, 137–150, 1997.
4. E. Gelle and R. Weigel, 'Interactive configuration using constraint satisfaction techniques', in *Artificial Intelligence and Manufacturing Research Planning Workshop, AAAI Technical Report FS-96-03*, 37–44, (1996).
5. J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In Proceedings UAI'99, 279–288, 1999.
6. A. Darwiche, P. Marquis. A perspective on knowledge compilation. In IJCAI 2001,175–182.
7. Sanjay Mittal et Brian Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of AAAI-90*, pages 25–32, Boston, MA, 1990.
8. Daniel Sabin et Eugene C. Freuder. Composite constraint satisfaction. In *Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 153–161, 1996.

9. M. Sabin and E. C. Freuder, 'Detecting and resolving inconsistency in conditional constraint satisfaction problems', in *Proceedings of the AAAI'99 Workshop on configuration*, 90–94, Orlando, Florida, (1999).
10. M. Sabin and E. C. Freuder, 'Detecting and resolving inconsistency in conditional constraint satisfaction problems', in *Proceedings of the AAAI'99 Workshop on configuration*, 90–94, Orlando, Florida, (1999).
11. D. Sabin and R. Weigel, 'Product configuration frameworks - a survey', *IEEE Intelligent Systems and their applications*, 42–49, (1998).
12. T. Soininen and E. Gelle, 'Dynamic constraint satisfaction in configuration', in *Proceedings of the AAAI'99 Workshop on configuration*, 95–100, Orlando, Florida, (1999).
13. M. Stumptner, 'An overview of knowledge-based configuration', *AI Communications*, 111–125, (1997).
14. N. R. Vempaty. Solving constraint satisfaction problems using finite state automata, in *Proceedings of AAAI'92*, 453–458, 1992.
15. R. Weigel and Boi Faltings, 'Compiling constraint satisfaction problems', *Artificial Intelligence*, **115**, 257–287, (1999).

Channel Theory for User-Interactions in Constraint Satisfaction and Design

Makoto Kikuchi[1], Ichiro Nagasaka[2], and Mutsunori Banbara[3]

[1] Department or Computer and Systems Engineering,
Kobe University, Kobe 657-8501, Japan,
mkikuchi@kobe-u.ac.jp,
WWW home page: http://kurt.scitec.kobe-u.ac.jp/~kikuchi/
[2] Faculty of Letters, Kobe University, Kobe 657-8501, Japan,
nagasaka@kobe-u.ac.jp
[3] Information Science and Technology Center,
Kobe University, Kobe 657-8501, Japan,
banbara@kobe-u.ac.jp

Abstract. There are two sorts of constraint satisfaction problems. One is abstract, like solving mathematical equations, and the other is real, like designing an artifact satisfying requirements. We need to represents real constraints by abstract ones when we use CSP's in the real problems. This is not a easy task, and user-interactions are inevitable in order to adjust the constraints and to solve the real problem properly. In this paper, by applying a mathematical theory of information flow by Barwise and Seligman called Channel Theory, we propose a formal model of user-interactions in constraint satisfaction problems in order to analyze the phenomena about the interactions in an abstract way, and show an application of the model to a formal theory of design.

1 Introduction

There are many kinds of constraints satisfaction problems (CSP's). Solving equations in mathematics is a typical problem of constraints satisfaction, and most of famous combinatorial problems, like N-queens, are also considered as CSP's. They are *abstract* problems. Abstract CSP's have been studied in mathematics and computer science for a long time, and now we have deep theories and many useful algorithms about such abstract problems. There are also lots of *real* CSP's in our every day life, especially in engineering (cf. [16]). We design an artifact, like automobiles and houses, so as to satisfy given requirements. The requirements are regarded as constraints for the artifacts, and designing can be regarded as an activity of solving a CSP. In fact, there are many applications of CSP's to problems about design. Although abstract CSP's themselves are important and interesting topics, we can say that abstract CSP's are basically tools for solving real CSP's.

Unfortunately, applications of theory on abstract CSP's to real CSP's are not easy. There are two major problems. One is the *complexity* of the real cases.

Many combinatorial problems are known to NP-complete, and it may happen that methods for abstract CSP's make sense only when the size of the factors are restricted. So many factors are related to a real CSP, and it is hopeless in many cases to apply an algorithm which is designed for abstract CSP's. The other is the *representability* of the constraints. We need to formulate the original and real constraints in an abstract way in order to apply methods for abstract CSP's, but this formulation is not straight forward. The most important reason for the difficulty is that the original constraints are closely related our conscious or intention, which are hard to handle formally. The original and real constraints tend to be vague, inconsistent, and wrong, and this is a crucial problem when we consider design as real CSP's.

User-interaction is a key notion when we consider these problems. A user is assumed to have the *user-constraints*, and they are represented as *system-constraint* in an abstract system. In a process of solving the real CSP, system-constraints are to be modified and revised through user-interactions. Several models and algorithms for user-interactions have been proposed. For example, Freuder and O'Sullivan argued in [5] how to add new system-generated *tradeoff constraints* to the original *preference constraint*, and O'Sullivan, Freuder, and O'Connell discussed in [17], by using the list-then-eliminate algorithm, how to modify the system-constraints by proposing and asking to evaluate examples to the user. If we consider a CSP as a problem finding Roughly speaking, a CSP is a problem of calculating or estimating the image of a function whose inputs are constraints and whose outputs are solutions which satisfy the input constraint. The above type of user-interaction can be seen as a composition $f \circ g$ of two functions f and g: f is the function of an abstract CSP, and g stands for a translation of real constraints to abstract constraint.

In real problems, the original constraints are sometimes vague and implicit. Even user-constraints are modified in a process of design: a user does not know completely what she or he is desiring at the beginning. It is a new phase of user-interactions, and we cannot consider a translation function from real constraints to abstract constraints. This phase of user-interactions is hard to deal with, and it is related to the essence of the problem about creativity in design. We need new framework in order to discuss this phase of user-interactions.

The purpose of this paper is to propose a mathematical model of interactions between user- and system-constraints by using Channel Theory. Channel Theory is a mathematical theory of information flow proposed by Barwise and Seligman [2] in 1990's. based on philosophical discussions about information flow by Dretske [4] and Situation Semantics, a formal semantics of natural language introduced by Barwise and Perry [1] (cf. [3]). The two sorts of constraints inhabit in different worlds: user-constraints live in the real, and system-constraints belong to abstract. In our model, we represents these two worlds as mathematical structures which are called *classifications*, and user- and system-constraints are represented and analyzed in terms of these structures.

We can regard Channel Theory as an extension of formal logic and our argument is an extension of formulation of CSP in terms of formal logic (cf. [12])

to user-interactions. One of the main characteristics of Channel Theory is that we have two kinds of connection between two classifications (or two kinds of interpretations between two theories). One is called an *infomorphism*, and the other is called a *channel*. The difference of the two phases of user-interactions are represented by means of these two kinds of connections.

Our model has a correspondence to a scheme for a mathematical theory about design proposed by Yoshikawa [20] and Kakuda and Kikuchi [8]. We can say that these theory of design is an analysis of design from the viewpoint of regarding design activity as solving a constraint satisfaction problem. Although we have not reached mathematically interesting discussions in this paper, it is a step to new connections between analysis of user-interaction in constraint satisfaction, mathematical study on engineering design, and natural languages semantics.

2 Classical Models of Constraint Satisfaction Problems

We need to formulate CSP's mathematically in order to argue user-interaction in CSP's formally. There are various types of CSP's, but there is something common in the problems. Jeavons, Cohen and Person gave the following formal and uniform definition of CSP's in [6], which can be seen as a classical model of CSP's (cf. [19]).

Definition 1. *1. A* constraint satisfaction problem *is a triple $\langle V, D, \mathcal{C}\rangle$, where:*
   (a) V is a set of variables,
   (b) D is a domain of values,
   (c) $\mathcal{C}$ is a set of constraints $\{C_1, C_2, \ldots, C_q\}$. Each constraint $C_i \in \mathcal{C}$ is a pair $\langle s_i, R_i\rangle$, where
      i. s_i is a tuple of variables of length m_i, called the constraint scope*, and*
      ii. R_i is any m_i-ary relation over D, called the constraint relation*.*

2. A solution to a constraint satisfaction problem $\langle V, D, \mathcal{C}\rangle$ is a function f from the set of variables V to the domain of values D such that for each constraint $\langle s_i, R_i\rangle$, with $s_i = \langle v_{i_1}, v_{i_2}, \ldots, v_{i_m}\rangle$, the tuple $\langle f(v_{i_1}), f(v_{i_2}), \ldots, f(v_{i_m})\rangle$ is a member of R_i.

For any sets V and D, let D^V be the set of all functions from V to D. We remark that $f \in D^V$ can be denoted by $(f(v_1), f(v_2), \ldots, f(v_r)) \in D^r$ when $V = \{v_1, v_2, \ldots, v_r\}$.

Example 1. Let $V = \{v_1, v_2\}$ and $D = \mathbb{R}$ (the set of real numbers). Consider a CSP which consists of the following three constraints:

$$f_1 : v_1^2 + v_2^2 = 2, \tag{1}$$
$$f_2 : v_1 = v_2, \tag{2}$$
$$f_3 : v_1 > 0. \tag{3}$$

These constraints are represented by

$$C_1 = \langle \{v_1, v_2\}, \{(a, b) \in \mathbb{R}^2 : a^2 + b^2 = 2\} \rangle, \tag{4}$$
$$C_2 = \langle \{v_1, v_2\}, \{(a, b) \in \mathbb{R}^2 : a = b\} \rangle, \tag{5}$$
$$C_3 = \langle \{v_1\}, \{a \in \mathbb{R} : a > 0\} \rangle \tag{6}$$

Let $\mathcal{C} = \{C_1, C_2, C_3\}$. Then, the unique solution to a constraint satisfaction problem $\langle V, D, \mathcal{C} \rangle$ is $(1, 1) \in \mathbb{R}^2$.

Any constraint can be represented by a subset of D^V. Assume that $\langle V, D, \mathcal{C} \rangle$ is a CSP (in the sense of the above definition). For each constraint $C_i = \langle s_i, R_i \rangle$ with $s_i = \langle v_{i_1}, v_{i_2}, \ldots, v_{i_m} \rangle$, we define a subset B_i of D^V by $B_i = \{f \in D^V : \langle f(v_{i_1}), f(v_{i_2}), \ldots, f(v_{i_m}) \rangle \in R_i\}$. That is, B_i is the set of solutions to a CSP $\langle V, D, \{C_i\} \rangle$ with only one constraint C_i. Then, $f \in D^V$ is a solution to $\langle V, D, \mathcal{C} \rangle$ if and only if $f \in B_1 \cap B_2 \cap \cdots \cap B_q$. Conversely, assume that we have a triple $\langle V, D, \mathcal{B} \rangle$ such that V and D are sets, and $\mathcal{B} = \{B_1, B_2, \ldots, B_q\}$ where $B_i \subseteq D^V$ for all $i = 1, 2, \ldots, q$. For each $i = 1, 2, \ldots, q$, define C_i as a pair $C_i = \langle V, B_i \rangle$. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$. Then, $\langle V, D, \mathcal{C} \rangle$ is a CSP and $f \in D^V$ is a solution to $\langle V, D, \mathcal{C} \rangle$ if and only if $f \in B_1 \cap B_2 \cap \cdots \cap B_q$. That is to say, we can represent a constraint C_i by a subset B_i of D^V, and any subset B_i of D^V can be seen as a constraint. Therefore, the following definition of CSP's is essentially (or mathematically) equivalent to the above one. (It is also a variation of a formal definition of CSP's by Lassez and McAloon [12].)

Definition 2. *1. A* constraint satisfaction problem *is a triple $\langle V, D, \mathcal{B} \rangle$, where:*
   (a) V is a set of variables,
   (b) D is a domain of values,
   (c) $\mathcal{B}$ is a subset of D^V, whose elements are called constraints.
2. A solution to a constraint satisfaction problem $\langle V, D, \mathcal{B} \rangle$ is an element of $B_1 \cap B_2 \cap \cdots \cap B_q$.

Example 2. Let $\langle V, D, \mathcal{C} \rangle$ be a CSP in Example 1. The subsets B_1, B_2, B_3 of D^V which correspond to C_1, C_2, C_3 respectively are

$$B_1 = \{(a, b) \in \mathbb{R}^2 : a^2 + b^2 = 2\}, \tag{7}$$
$$B_2 = \{(a, b) \in \mathbb{R}^2 : a = b\}, \tag{8}$$
$$B_3 = \{(a, b) \in \mathbb{R}^2 : a > 0\} \tag{9}$$

Then, $\langle V, D, \mathcal{B} \rangle$ is a CSP in the sense of this definition for $\mathcal{B} = \{B_1, B_2, B_3\}$, and $B_1 \cap B_2 \cap B_3 = \{(1, 1)\}$.

This definition has two problems. One is that a constraint B_i does not have information to which variables the constraint is related. In the former definition, C_i is a pair of $\langle s_i, R_i \rangle$, and s_i denote the information to which variable the constraint is related. This information is important when we make an actual constraint solver. This problem can be resolved partially by defining the related variable from B_i. For any constraint B_i in the latter definition of CSP, we define a

subset s_i of $V = \{v_1, v_2, \ldots, v_r\}$ by $s_i = \{v_i \in V : \forall(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_r) \in D^{r-1} \forall(b, c) \in D^2((a_1, \ldots, b, \ldots, a_r) \in B_i \Leftrightarrow (a_1, \ldots, c, \ldots, a_r) \in B_i)\}$ This set s_i represents the information about D_i's related variable. (Strictly speaking, the information should be represented by a subset of D^V containing s_i) The other one is that, in our definition, a solution is obtained immediately after describing a constraint satisfaction problem and we do not need to *solve* the problem anytime. This is a problem in common with the former definition, and it is caused by defining a constraint by a subset of D^V (or a relation on D). But this problem does not matter in our case. We are not interested in how to solve a CSP, but we are interested in how to obtain a suitable set of constraints.

3 Constraint-Revisions and User-Interactions

In this section, we shall review some phenomena of revisions of constraints via user-interactions in CSP's. These phenomena will be formalized and analyzed mathematically in Section 6.

Let Sol($\mathcal{P}$) be the set of all solutions to a constraint satisfaction problem $\mathcal{P}$. In our definition, we can classify three kinds of problems on Sol($\mathcal{P}$): under constrained, over constrained, and inappropriateness. (There is a similar analysis of problems on design specifications in a framework of a general theory of design by Yoshikawa in [20].)

1. *Under-constrained* – When we do not have constraints sufficiently, Sol($\mathcal{P}$) becomes to big. This phenomena itself is not a problematic, but Sol($\mathcal{P}$) tends to be inappropriate.
2. *Over-constrained* – When we have too many constraints, Sol($\mathcal{P}$) becomes to small, or empty in the worst case. This case may have a side effects of increasing of computational complexity in solving the problem.
3. *Inappropriateness* – This is the problem that an element of Sol($\mathcal{P}$) is different from what a user desired really. This may happen in engineering design problems in the real world in many case.

In any case, we have to add or eliminate constraints, or we have to replace a part of the constraints. We do not have any criteria for this operation in the problem $\mathcal{P}$, so user-interactions become inevitable.

Pu and Faltings [14] classified modifications of constraints through user-interactions into three types: *Preferences* (specifying explicit preference of value or tuples within an individual constraints), *Hidden Constraints* (specifying constraints unknown), and *Tradeoffs* (specifying the relative importance of the constraints). Preferences and Hidden Constraints corresponds to Under Constrained in our classification, and Tradeoffs corresponds to Over-constrained or Inappropriateness. These modifications through user-interactions can be seen as revisions of the set $\mathcal{B}$ in a constraint satisfaction problem $\mathcal{P} = \langle V, D, \mathcal{B}\rangle$: adding, eliminating, and replacing elements of $\mathcal{B}$. Preferences is replacing a constraint $B \in \mathcal{B}$ by a stronger one $B' \subseteq B$ which consists of preferred elements. Hidden Constraints

is adding a new constraints to $\mathcal{B}$, and Tradeoffs is replacing a constraint $B \in \mathcal{B}$ by a new one $B' \subseteq D^V$.

We remark that the constraints in $\mathcal{B}$ in a constraint satisfaction problem $\langle V, D, \mathcal{B} \rangle$ is system-constraints, in the sense of the previous section. We need to know what is the user-constraint in the real world when we try to revise the set of system-constraints. We assume that user-constraints can be represented in terms of another constraint satisfaction problem $\langle V', D', \mathcal{B}' \rangle$. We can consider four kinds of conditions about these two sorts of CSP's by standing on a viewpoint which is slightly different from the above classification.

1. *Self-consistency* – User- and system-constraints must be small enough to have a solution.
2. *Self-completeness* – User- and system-constraints must be big enough to specify a solution.
3. *Inter-consistency* – System-constraints must be small enough to consistent with user-constraints.
4. *Inter-completeness* – System-constraints must be big enough to represent sufficiently user-constraints.

Under-constrained situations contravene self- and/or inter-completeness. Being Over-constrained violates self- and/or inter-consistency. Inappropriateness is related to inter-consistency and inter-completeness. self-consistency and completeness can be analyzed naturally by using the classical definitions of CSP's. In order to elicit the phenomena of inter-consistency and completeness, we shall reformulate CSP's within a framework of Channel Theory, and discuss the relationships between user- and system-constraints in terms of information flow between the real and the abstract worlds.

4 Channels and Information Flow

Channel Theory is a mathematical theory of information flow developed by Barwise and Seligman [2], which is based on philosophical discussions about information flow by Dretske [4] and Situation Semantics, a formal semantics of natural language proposed by Barwise and Perry [1]. The notions of in this section including classification, theory, local logic, infomorphism, channel are mostly due to Barwise and Seligman [2], but some of them are different.

Definition 3. *A* classification *is a triple* $\boldsymbol{A} = \langle \mathrm{tok}(\boldsymbol{A}), \mathrm{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}} \rangle$ *where* $\models_{\boldsymbol{A}}$ *is a binary relation between two sets* $\mathrm{tok}(\boldsymbol{A})$ *and* $\mathrm{typ}(\boldsymbol{A})$. *Elements of* $\mathrm{tok}(\boldsymbol{A})$ *and* $\mathrm{typ}(\boldsymbol{A})$ *are called* tokens *and* types *of* $\boldsymbol{A}$.

$$\begin{array}{c} \mathrm{typ}(\boldsymbol{A}) \\ \Big| \models_{\boldsymbol{A}} \\ \mathrm{tok}(\boldsymbol{A}) \end{array} \tag{10}$$

Definition 4. *A* state space *is a triple* $\boldsymbol{S} = \langle \mathrm{tok}(\boldsymbol{S}), \mathrm{typ}(\boldsymbol{S}), \mathrm{state}_{\boldsymbol{S}} \rangle$ *where* $\mathrm{state} : \mathrm{tok}(\boldsymbol{S}) \to \mathrm{typ}(\boldsymbol{S})$ *is a map. The* event classification *of a state space* $\boldsymbol{S}$ *is a classification* $\mathrm{Evt}(\boldsymbol{S}) = \langle \mathrm{tok}(\boldsymbol{S}), \mathcal{P}(\mathrm{typ}(\boldsymbol{S})), \models_{\mathrm{Evt}(\boldsymbol{S})} \rangle$ *where* $a \models_{\mathrm{Evt}(\boldsymbol{S})} \alpha$ *iff* $\alpha \in \mathrm{state}_{\boldsymbol{S}}(a)$ *and* $\mathcal{P}(\mathrm{typ}(\boldsymbol{S})) = \{A : A \subseteq \mathrm{typ}(\boldsymbol{S})\}$ *is the power set of* $\mathrm{typ}(\boldsymbol{S})$.

A *sequent* on a set Σ is a pair $\langle \Gamma, \Delta \rangle$ of subsets of Σ. $\langle \Gamma, \Delta \rangle$ is a *partition* of Σ if $\Gamma \cup \Delta = \Sigma$ and $\Gamma \cap \Delta = \varnothing$. We say $\langle \Gamma', \Delta' \rangle$ is an *extension* of $\langle \Gamma, \Delta \rangle$ and write $\langle \Gamma, \Delta \rangle \leq \langle \Gamma', \Delta' \rangle$ when $\Gamma \subseteq \Gamma'$ and $\Delta \subseteq \Delta'$. $\langle \mathrm{typ}_{\boldsymbol{A}}(a), \mathrm{typ}(\boldsymbol{A}) \setminus \mathrm{typ}_{\boldsymbol{A}}(a) \rangle$ is called the *state description* of a and denoted by $\mathrm{state}_{\boldsymbol{A}}(a)$ for a classification $\boldsymbol{A}$ and $a \in \mathrm{tok}(\boldsymbol{A})$.

We say $a \in \mathrm{tok}(\boldsymbol{A})$ *realizes* a sequent $\langle \Gamma, \Delta \rangle$ on $\mathrm{typ}(\boldsymbol{A})$ if $\langle \Gamma, \Delta \rangle \leq \mathrm{state}_{\boldsymbol{A}}(a)$, and $\langle \Gamma, \Delta \rangle$ is *realizable* in $\boldsymbol{A}$ if it is realized by a token of $\boldsymbol{A}$. We say a *satisfies* $\langle \Gamma, \Delta \rangle$ if a does not realize $\langle \Gamma, \Delta \rangle$. It is obvious that a satisfies $\langle \varnothing, \{\alpha\} \rangle$ if and only if $a \models_{\boldsymbol{A}} \alpha$ for $\alpha \in \mathrm{typ}(\boldsymbol{A})$.

Definition 5. *A* theory *is a pair* $T = \langle \mathrm{typ}(T), \vdash_T \rangle$ *where* $\vdash_T$ *is a set of sequents on a set* $\mathrm{typ}(T)$. *We denote* $\Gamma \vdash_T \Delta$ *when* $\langle \Gamma, \Delta \rangle$ *is an element of* $\vdash_T$. *We say* $\langle \Gamma, \Delta \rangle$ *is a* constraint *of* T *if* $\Gamma \vdash_T \Delta$. *We say a theory* T *is* regular *if and only if* T *satisfies the following conditions:* **(Weakening)** *if* $\Gamma \vdash_T \Delta$ *and* $\langle \Gamma, \Delta \rangle \leq \langle \Gamma', \Delta' \rangle$ *then* $\Gamma' \vdash_T \Delta'$, **(Partition)** *if* $\Gamma \nvdash_T \Delta$ *then there is a partition* $\langle \Gamma', \Delta' \rangle$ *such that* $\Gamma' \nvdash_T \Delta'$.

$\Gamma \vdash_T \Delta$ means Γ *implies* Δ. More precisely, it means that if all the types in Γ holds, then at least one of the types in Δ holds. By the conditions of Weakening and Partition, a theory is a generalization of classical logic.

Let T and T' be theories such that $\mathrm{typ}(T) = \mathrm{typ}(T')$. We say T' is *stronger* than T (or T is *weaker* than T') and write $T \leq T'$ when every constraint of T is also a constraint of T'. We define a theory $T \cap T'$ and $T \cup T'$ on $\mathrm{typ}(T) = \mathrm{typ}(T')$ by defining $(\vdash_{T \cap T'}) = (\vdash_T) \cap (\vdash_{T'})$ and $(\vdash_{T \cup T'}) = (\vdash_T) \cup (\vdash_{T'})$. It is easy to show that $T \leq T'$ if and only if $T \cap T' = T$, or $T \cup T' = T'$. For any theory T, there exists the least regular theory T' such that $T \leq T'$. This T' is called the *regular closure* of T and denoted by $\mathrm{Reg}(T)$.

Let $\boldsymbol{A}$ be a classification. For $a \in \mathrm{tok}(\boldsymbol{A})$, define a regular theory $\mathrm{Th}(a) = \langle \mathrm{typ}(\boldsymbol{A}), \vdash_a \rangle$ by $\vdash_a = \{\langle \Gamma, \Delta \rangle : \langle \Gamma, \Delta \rangle \not\leq \mathrm{state}_{\boldsymbol{A}}(a)\}$. Then, the regular theory $\mathrm{Th}(a) = \langle \mathrm{typ}(\boldsymbol{A}), \vdash_{\boldsymbol{A}} \rangle$ *generated by* $\boldsymbol{A}$ is defined by $\vdash_{\boldsymbol{A}} = \bigcap_{a \in \mathrm{tok}(\boldsymbol{A})} (\vdash_a)$.

Definition 6. *Let* T *and* T' *be theories. A* theory interpretation $f : T \to T'$ *is a map* $f : \mathrm{typ}(T) \to \mathrm{typ}(T')$ *such that* $\Gamma \vdash_T \Delta$ *implies* $f[\Gamma] \vdash_{T'} f[\Delta]$ *for every sequent* $\langle \Gamma, \Delta \rangle$ *on* $\mathrm{typ}(T)$.

Let Σ and Σ' be two sets, and $f : \Sigma \to \Sigma'$ be a map. For a theory $T' = \langle \Sigma, \vdash_{T'} \rangle$ on Σ', we define a theory $T = \langle \Sigma, \vdash_T \rangle$ by $\Gamma \vdash_T \Delta$ if and only if $f[\Gamma] \vdash_{T'} f[\Delta]$, for any sequent $\langle \Gamma, \Delta \rangle$ on Σ. This T is denoted by $f^{-1}(T')$. Then, $f^{-1}(T')$ is regular if T' is regular, and f is an theory interpretation from $f^{-1}(T')$ to T'. Conversely, for a theory $T = \langle \Sigma, \vdash_T \rangle$, we define a theory $T' = \langle \Sigma, \vdash_{T'} \rangle$ on Σ' by $\Gamma' \vdash_{T'} \Delta'$ if and only if $f^{-1}[\Gamma'] \vdash_T f^{-1}[\Delta']$, for any sequent $\langle \Gamma', \Delta' \rangle$ on Σ'. This T' is denoted by $f(T)$. We remark that there is no guarantee that $f(T)$

is regular even if T is regular, nor f is a theory interpretation. (There is another definition of $f(T)$ in [2] which is complex but has good features.) It is easy to show that $f(f^{-1}(T')) \leq T'$ and $T \leq f^{-1}(f(T))$ if T' and T satisfy Weakening, but equalities does not hold in general.

Definition 7. *An* infomorpshim $f : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$ *is a pair* $f = \langle f^{\wedge}, f^{\vee} \rangle$ *of maps* $f^{\wedge} : \mathrm{typ}(\boldsymbol{A}) \to \mathrm{typ}(\boldsymbol{B})$ *and* $f^{\vee} : \mathrm{tok}(\boldsymbol{B}) \to \mathrm{tok}(\boldsymbol{A})$ *such that* $f^{\vee}(b) \models_{\boldsymbol{A}} \alpha$ *if and only if* $b \models_{\boldsymbol{B}} f^{\wedge}(\alpha)$ *for every* $\alpha \in \mathrm{typ}(\boldsymbol{A})$ *and* $b \in \mathrm{tok}(\boldsymbol{B})$.

$$\begin{array}{ccc} \mathrm{typ}(\boldsymbol{A}) & \xrightarrow{f^{\wedge}} & \mathrm{typ}(\boldsymbol{B}) \\ \Big| \models_{\boldsymbol{A}} & & \Big| \models_{\boldsymbol{B}} \\ \mathrm{tok}(\boldsymbol{A}) & \xleftarrow[f^{\vee}]{} & \mathrm{tok}(\boldsymbol{B}) \end{array} \tag{11}$$

For any infomorphism $f : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$, $f^{\wedge} : \mathrm{typ}(\boldsymbol{A}) \to \mathrm{typ}(\boldsymbol{B})$ is a theory interpretation from $\mathrm{Th}(\boldsymbol{A})$ to $\mathrm{Th}(\boldsymbol{B})$. Theories can be transmitted by infomorphisms. For theories T on $\mathrm{typ}(\boldsymbol{A})$ and T' on $\mathrm{typ}(\boldsymbol{B})$, $f^{\wedge}(T)$ and $f^{\wedge -1}(T')$ is abbreviated by $f(T)$ and $f^{-1}(T')$.

A part-whole relation is defined by an infomorphism: for any infomorphism $f : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$ and $b \in \mathrm{tok}(\boldsymbol{B})$, $f^{\vee}(b)$ is a part of b in the sense of f.

Definition 8. *A (binary)* channel *is a set* $\mathcal{C} = \{f : \boldsymbol{A} \rightleftarrows \boldsymbol{C}, g : \boldsymbol{B} \rightleftarrows \boldsymbol{C}\}$ *of infomorphisms.* $\boldsymbol{C}$ *is called the core of the channel* $\mathcal{C}$.

$$\begin{array}{ccccc} & & \boldsymbol{C} & & \\ & \overset{f}{\nearrow} & & \overset{g}{\nwarrow} & \\ \boldsymbol{A} & & & & \boldsymbol{B} \end{array} \tag{12}$$

We can define n-ary channel as a set of infomorphisms $\{f_1 : \boldsymbol{A}_1 \rightleftarrows \boldsymbol{C}, \ldots, f_n : \boldsymbol{A}_n \rightleftarrows \boldsymbol{C}\}$.

An infomorpshim $f : \boldsymbol{A} \rightleftarrows \boldsymbol{B}$ can be regarded as a channel by considering $\boldsymbol{B}$ as a core of the channel. That is, there is a trivial channel $\{f : \boldsymbol{A} \rightleftarrows \boldsymbol{B}, \mathrm{id}_{\boldsymbol{B}} : \boldsymbol{B} \rightleftarrows \boldsymbol{B}\}$ where $\mathrm{id}_{\boldsymbol{B}}$ is the identity infomorphism (an infomophism consists of the two identity maps) on $\boldsymbol{B}$. Similarly, an infomorpshim $g : \boldsymbol{B} \rightleftarrows \boldsymbol{A}$ can be regarded as a channel $\{\mathrm{id}_{\boldsymbol{A}} : \boldsymbol{A} \rightleftarrows \boldsymbol{A}, g : \boldsymbol{B} \rightleftarrows \boldsymbol{A}\}$ where $\mathrm{id}_{\boldsymbol{A}}$ is the identity infomorphism on $\boldsymbol{A}$. A channel is a generalization of an infomorphism.

In situation semantics, the meaning $[D]$ of an expression D is defined as a pair of situations u and s, where u is the situation of utterance of D and s is the described situation by D. Channel is a structure for the relationships between situations for natural language semantics.

Definition 9. *A local logic is a triple* $\mathcal{L} = \langle \boldsymbol{C}, T, N \rangle$ *of a classification* $\boldsymbol{C}$*, a theory* T*, and a set of normal tokens* N *such that* $\mathrm{typ}(T) = \mathrm{typ}(\boldsymbol{C})$*,* $N \subseteq \mathrm{tok}(\boldsymbol{C})$*, and every token in* N *satisfies every constraint in* T.

Let $\mathcal{C} = \{f : \boldsymbol{A} \rightleftarrows \boldsymbol{C}, g : \boldsymbol{B} \rightleftarrows \boldsymbol{C}\}$ be a channel and $\mathcal{L} = \langle \boldsymbol{C}, T, N\rangle$ be a local logic, and $a \in \mathrm{tok}(\boldsymbol{A}), \alpha \in \mathrm{typ}(\boldsymbol{A}), b \in \mathrm{tok}(\boldsymbol{B})$ and $\beta \in \mathrm{typ}(\boldsymbol{B})$. An information $a \models_{\boldsymbol{A}} \alpha$ on $\boldsymbol{A}$ flows to another information $b \models_{\boldsymbol{B}} \beta$ on $\boldsymbol{B}$ if and only if there is a token $c \in N$ such that $f^{\check{}}(c) = a$, $g^{\check{}}(c) = b$, and $f^{\hat{}}(\alpha) \vdash_T g^{\hat{}}(\beta)$. In many cases, $\mathcal{L}$ is chosen as $\mathcal{L} = \langle \boldsymbol{C}, \mathrm{Th}(\boldsymbol{C}), \mathrm{tok}(\boldsymbol{C})\rangle$ which is called the local logic generated by $\mathcal{L}$.

5 Constraint Satisfaction Problems in Channel Theory

Now we shall formulate CSP's and discuss the relationship between user- and system-constraints within Channel Theory. These two constraints are represented on two different classifications: one is about a user, and the other is about a system. In this section, we firstly give a new definition of CSP's by using a notion of classification in Channel Theory, and we formulate mathematically the four category of self- and inter-consistency, and self- and inter-completeness, and discuss their characteristics.

Although our new definition of CSP is rather simple, we can translate the previous and the new definitions each other.

Definition 10. *Let* $\boldsymbol{A} = \langle \mathrm{tok}(\boldsymbol{A}), \mathrm{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}}\rangle$ *be a classification.*

1. *A* constraint satisfaction problem *on* $\boldsymbol{A}$ *is a theory* T *on* $\mathrm{typ}(\boldsymbol{A})$.
2. *A* solution *to a constraint satisfaction problem* T *is a token* $a \in \mathrm{tok}(\boldsymbol{A})$ *such that* a *satisfies all the constraints in* T.

We note that a constraint satisfaction problem is not assumed to be a regular theory.

We shall show that the previous definition and this new definition are interpretable. Let $\mathcal{P} = \langle V, D, \mathcal{B}\rangle$ be a CSP in the sense of the second definition such that $\mathcal{B} = \{B_1, B_2, \ldots, B_q\}$. We define a classification $\boldsymbol{A} = \langle \mathrm{tok}(\boldsymbol{A}), \mathrm{typ}(\boldsymbol{A}), \models_{\boldsymbol{A}}\rangle$ by $\mathrm{tok}(\boldsymbol{A}) = D^V$, $\mathrm{typ}(\boldsymbol{A}) = \{B : B \subseteq D^V\}$ (the power set of D^V), and $a \models_{\boldsymbol{A}} B$ if and only if $a \in B$ for all $a \in \mathrm{tok}(\boldsymbol{A})$ and $B \in \mathrm{typ}(\boldsymbol{A})$. We remark that $\mathrm{tok}_{\boldsymbol{A}}(B) = B$ for $B \in \mathrm{typ}(\boldsymbol{A})$, and $\mathcal{B}$ is a subset of $\mathrm{typ}(\boldsymbol{A})$. Define a theory $T = \langle \mathrm{typ}(\boldsymbol{A}), \vdash_T\rangle$ by $(\vdash_T) = \{\langle \varnothing, \{B_i\}\rangle : i = 1, 2, \ldots, q\}$ Then, T is the corresponding CSP on $\boldsymbol{A}$. Conversely, let $T = \langle \mathrm{typ}(\boldsymbol{A}), \vdash_T\rangle$ be a CSP on $\boldsymbol{A}$ such that $(\vdash_T) = \{\langle \Gamma_i, \Delta_i\rangle : i = 1, 2, \ldots, q\}$ Define $V = \{v_1\}$, $D = \mathrm{tok}(\boldsymbol{A})$, and $B_i = \{a \in \mathrm{tok}(\boldsymbol{A}) : \langle \Gamma_i, \Delta_i\rangle \not\leq \mathrm{state}_{\boldsymbol{A}}(a)\}$ for $i = 1, 2, \ldots, q$. Then, B_i can be regarded as a subset of D^V since D is isomorphic to D^V, and $\langle V, D, \mathcal{B}\rangle$ is the corresponding CSP in the sense of Definition 2, where $\mathcal{B} = \{B_1, B_2, \ldots, B_q\}$.

The following example corresponds to a CSP in Example 1.

Example 3. Let $\boldsymbol{A}$ be a classification such that $\mathrm{tok}(\boldsymbol{A}) = \mathbb{R}^2$, $\mathrm{typ}(\boldsymbol{A})$ is the set of equalities and inequalities in v_1 and v_2, and, for every $(a_1, a_2, a_3) \in \mathbb{R}^3$ and $f \in \mathrm{typ}(\boldsymbol{A})$, $(a_1, a_2, a_3) \models_{\boldsymbol{A}} f(v_1, v_2, v_3)$ if and only if $f(a_1, a_2, a_3)$ is true. Then, for equalities f_1, f_2 and an inequality f_3 in Example 1, a theory $T = \langle \mathrm{typ}(\boldsymbol{A}), \vdash_T\rangle$ where $(\vdash_T) = \{\langle \varnothing, \{f_i\}\rangle : i = 1, 2, 3\}$ is the corresponding CSP on $\boldsymbol{A}$.

The difference between this example and previous examples is that equalities and inequalities are used for denoting constraints, although subsets of D^V (or some other similar sets) are used for this purpose in the previous examples. Hence the second problem that solutions of a CSP is trivial in the previous examples has been dissolved in our definition.

6 User-Interactions in Channel Theory

In this section, we shall formulate and discuss phenomena of user-interactions in CSP's which are argued in Section 3.

It is straight forward to define what self-consistency and self-completeness are. Let T be a CSP on $\boldsymbol{A}$. Self-consistency means that all the constraints of T can be realized by a single token in $\boldsymbol{A}$, and self-completeness says that T is specifying a token in $\boldsymbol{A}$.

Definition 11. *1. We say T is* self-consistent *on $\boldsymbol{A}$ if there is $a \in \mathrm{tok}(\boldsymbol{A})$ such that $T \leq \mathrm{state}_{\boldsymbol{A}}(a)$.*

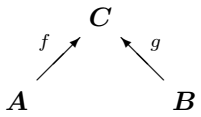
2. We say T is self-complete *on $\boldsymbol{A}$ if there is $a \in \mathrm{tok}(\boldsymbol{A})$ such that $T = \mathrm{state}_{\boldsymbol{A}}(a)$.*

The relation between user- and system-constraints are formulated as a relation between two classifications. let us assume we have two classifications $\boldsymbol{A}$ and $\boldsymbol{B}$ on which user- and system-constraints are described as theories T and T' on $\mathrm{typ}(\boldsymbol{A})$ and $\mathrm{typ}(\boldsymbol{B})$ respectively.

We have mentioned in Section 1 that there are two phases in user-interactions in CSP's: one is that system-constraints are images user-constraints, and the other is that user-constraints are also revised through the interactions. The first phase of user-interactions, interaction along a map from user-constraints to system-constraints can be represented by an infomorphism h from $\boldsymbol{A}$ to $\boldsymbol{B}$.

$$\boldsymbol{A} \xrightarrow{h} \boldsymbol{B} \tag{13}$$

That is, user-interaction is a process of estimation of the map $h\hat{}$. On the contrary, a channel $\mathcal{C} = \{f : \boldsymbol{A} \rightleftarrows \boldsymbol{C}, g : \boldsymbol{B} \rightleftarrows \boldsymbol{C}\}$ between $\boldsymbol{A}$ and $\boldsymbol{B}$ corresponds to the second phase of user-interactions.



(14)

By introducing the notion of channel, we can argue inter-consistency and inter-completeness in terms of theories on the type of the core $\boldsymbol{C}$ of the channel.

Definition 12. *1. We say T' is* inter-consistent *with T along $\mathcal{C}$ if $f(T) \cup g(T')$ is self-consistent on $\boldsymbol{C}$.*

2. We say T' is inter-complete *with T along $\mathcal{C}$ if $f(T) \leq g(T')$.*

Let us consider simple cases of channels, in which we have T in advance and T' is defined from T. Assume that $\boldsymbol{C} = \boldsymbol{B}$ and $g = \mathrm{id}_{\boldsymbol{B}}$, and define $T' = f(T)$. This is a case that we know how to represents user-constraints abstractly, and $g(T') = f(T)$. Hence inter-completeness of T' is satisfied automatically, and T' is inter-consistent with T if and only if $f(T)$ is self-consistent. Assume that $\boldsymbol{C} = \boldsymbol{A}$ and $f = \mathrm{id}_{\boldsymbol{A}}$, and define $T' = g^{-1}(T)$. This is a more common case that we can understand system-constraints. In this case, $g(T') = g(g^{-1}(T))$ and $f(T) = T$. Hence inter-completeness of T' does not hold in general since $g(g^{-1}(T))$ may weaker than T, but T' is inter-consistent with T if and only if T is self-consistent. In these two simple cases, T is completely responsible for the choice of T'.

However, we hardly have such simple cases. There are two problems. One is that we do not have neither f nor g usually. That is, we cannot connect simply the two classifications $\boldsymbol{A}$ and $\boldsymbol{B}$ for the two worlds of a user and a system. These two worlds are worlds of the reals and our consciousness. An infomorphism between $\boldsymbol{A}$ and $\boldsymbol{B}$ represents the relationship between these two worlds, and it is a very strong assumption that we have such an infomorphism. The other is that we hardly have T explicitly in many cases. As mentioned earlier, we usually do not know what we want. T' is a representation of T. It means that we can approach to T only through T', and we cannot assume we have T in advance.

In general cases, it is important to consider the core $\mathcal{C}$ of a channel $\mathcal{C} = \{f : \boldsymbol{A} \rightleftarrows \boldsymbol{C}, g : \boldsymbol{B} \rightleftarrows \boldsymbol{C}\}$, which is different from $\boldsymbol{A}$ and $\boldsymbol{B}$, and a local logic $\mathcal{L} = \langle \boldsymbol{C}, T_{\mathcal{L}}, N_{\mathcal{L}} \rangle$ on the core $\boldsymbol{C}$. T and T' are unified on $\mathcal{C}$; they are symmetric. When we have $\mathcal{L}$, we can define T and T' by $f^{-1}(T_{\mathcal{L}})$ and $g^{-1}(T_{\mathcal{L}})$. In this case, inter-consistency can be considered as self-consistency of $T_{\mathcal{L}}$ on $\boldsymbol{C}$, and inter-completeness becomes trivial; what is more important is self-completeness of $T_{\mathcal{L}}$. And then, the revision of T and T' can be argued as revision of $T_{\mathcal{L}}$. T and T' are synchronized, and neither of them has a priority. Solutions in $\boldsymbol{A}$ and $\boldsymbol{B}$ for T and T' are connected by $N_{\mathcal{L}}$. That is, the translation of solutions in two classification is given by $N_{\mathcal{L}}$. $\mathcal{L}$ is the structure discussing the relation between user- and system-constraints.

From technical point of view, we should consider $\mathrm{Reg}(T)$ and $\mathrm{Reg}(T')$ instead of T and T' themselves. It is similar to the fact that in algebraic geometry we use the ideal $I(f_1, f_2, \ldots, f_r)$ generated by a set of polynomials $\{f_1, f_2, \ldots, f_r\}$ instead of using the set $\{f_1, f_2, \ldots, f_r\}$: They have the same set of solutions, but an ideal has mathematically good properties. A regular theory has similar properties also. Getting $\mathrm{Reg}(T)$ from T is a kind of revision of T. When we consider T as a set of knowledge about our constraints, this revision is a form of inference. (See [9] for some arguments on regular closure from the viewpoint of formulation of non-deductive inferences.)

One of the most important consequence of user-interaction in constraint satisfaction is not changing user-constraints T on $\boldsymbol{A}$ and system-constraints T' on $\boldsymbol{B}$, but revisions of $\boldsymbol{A}$ and $\boldsymbol{B}$. These revisions are acquisitions of new concepts and objects, and which are essential for analysis of creativity of reasoning and design.

7 User-Interactions in Design

In this section, we argue some theories of design in terms of user-interactions of constraint satisfaction problems.

Design is one of the essential activities of human beings, and it is a main theme in engineering. There are many arguments on design in each area of engineering, but it had not been discussed design in a general form until quite recently. Around 1970's, several theories on design was proposed. Paul and Beitz [13] discussed a general methodologies of design, in which how a overall function of an artifacts are designed and realized by reducing it to sub-functions. Simon [15] advocated a new research area of the science of artifacts and defined an artifact as an interface between its inner and outer environment.

The common basic idea of these theories on design is that design is an activity of realizing our desire or needs, and the notion of function plays an important role for describing our desire. Here, the existence of our desire is pre-assumed, and design is regarded as an activity how to find or create an artifact satisfying the desire. The desire can be seen as a kind of constraints, hence we can say that design is (or an aspect of design is) a sort of constraint satisfaction problems. There are lots of important and interesting studies on design by menas of CSP's in 1990's (cf. [16]). We remark that to define the notion of function is a difficult philosophical problem, and it is one of the difficulties about understanding design. (cf. [11])

Yoshikawa's General Design Theory (GDT) [20] is one of domain independent design theory. In GDT, design is investigated axiomatically in terms of the relationships between the notions of entities (artifacts) and their functions and attributes within a mathematical framework of topology. In GDT, it is assumed that we have $\mathcal{S}$ be the set of all entities, and functions and attributes of entities are represented by subsets of $\mathcal{S}$. A specification for design is a set of functions $\mathcal{F}$, a solution to $\mathcal{F}$ is an entity $a \in \mathcal{S}$ such that $\bigcap_{X \in \mathcal{F}} X$. It is clear that this formulation of a specification and its solution is an instance of the classical formal definitions of constraint satisfaction problems.

It is important that a design solution is assumed to be specified by its attributes in GDT. That is, a design solution $a \in \mathcal{F}$ is indicated by setting a set $\mathcal{A}$ of attributes. Then, design is formulated as an activity of finding to a set of $\mathcal{A}$ of attributes to a given set $\mathcal{F}$ of functions such that $\bigcap_{Y \in \mathcal{A}} Y \subseteq \bigcap_{X \in \mathcal{F}} X$ We can regard $\mathcal{F}$ and $\mathcal{A}$ as user- and system-constraints respectively. Hence, we can see some arguments on user-interactions (in the sense of our arguments) within GDT although it is not explicitly mentioned in GDT. This viewpoint make the framework of GDT clear: GDT consists of two parts. One is solving a constraint satisfaction problem (the relationship between $\mathcal{F}$ (or $\mathcal{A}$) and elements of $\mathcal{S}$), and the other is user-interactions (the relationship between $\mathcal{F}$ and $\mathcal{A}$). (We remark that the former relationship is trivial in GDT by the same reason to the classical definition of CSP's: functions and attributes are defined as a set of entities.)

Basically, design is regarded as an activity of evaluation of a function from specification to a solution in GDT. In this sense, GDT obeys the classical view of user-interaction such that an interaction is a process of calculation or estimation

of a function. A notion of a *metamodel* was introduced in GDT (cf. [18]), which is a structure for connecting functions and attributes. A metamodel was a way-stop between two spaces in the early stage of GDT, but it becomes a notion which corresponds to a channel in Channel Theory in the later stage. Introducing the notion of metamodel can be considered as a step from the classical view of user-interactions to a new one.

Kakuda reformulated the philosophy of GDT by using Channel Theory (cf. [8]), and named his theory Abstract Design Theory (ADT). The basic idea of ADT is that design is an activity which makes information flow between the world of our conscious and the real world. The basic scheme of ADT is similar to GDT, although the existence of the set $\mathcal{S}$ is not assumed (the assumption may be too strong). The relationship between *functions* and *attributes* of artifacts has been replaced by the relationship between *desire* and *behavior* of artifacts (cf. [7]). The difference is not big mathematically, but the new locution in ADT seems to be more familiar to the arguments on user-interactions in CSP's.

Interaction is now widely believed as an important notion in engineering. Kikuchi et al. [10] gave a mathematical model of interactions between human and environments through artifacts by using Channel Theory. Modeling interactions is a key step to understanding interactions.

8 Conclusions

We firstly analyze problems and conditions about user-interactions in constraint satisfaction, and we gave a mathematical definition of constraint satisfaction problems within the framework of Channel Theory, a mathematical theory of information flow. Then, user-interaction in constraint satisfaction is formulated as a relationship (or information flow) between two classifications, and we gave also a formulation of the conditions about user-interactions. At last, we discuss the relationship between our mathematical model of user-interactions and several mathematical theory of design.

Acknowledgments

This research is partly supported by Grant-in-Aid for Scientific Research of Japan Society for the Promotion of Science 14350210, 15700038, 15700132.

References

1. Barwise, J. and Perry, J., "Situation and Attitude", CSLI Publications, Stanford, 1999. (Originally published by MIT Press, Cambridge, 1983.)
2. Barwise, J. and Seligman, J., "Information Flow: The Logic of Distributed Systems", Cambridge University Press, Cambridge, 1997.
3. Devlin, K., "Logic and Information", Cambridge University Press, Cambridge, 1991.

4. Dretske, F.I., "Knowledge and the Flow of Information", MIT Press, Cambridge,1981; CLSI Publications, Stanford,1999.
5. Freuder, E.C. and O'Sullivan, B., "Generating Tradeoffs for Constraint-Based Configuration", Proceedings of 7th International Conference on Principles and Practice of Constraint Programming, 2001, 590–594.
6. Jeavons, P., Cohen, D. and Person, J., "Constraints and Universal Algebra", Annals of Mathematics and Artificial Intelligence, 24, 1998, 51–67.
7. Kakuda, Y., "On the Direction of Flow of Information –Information Paths, Agents, Information Channels–", Proceedings of International Workshop of Emergent Synthesis '02, 2002, pp.83–88.
8. Kakuda, Y. and Kikuchi, M., "Abstract Design Theory", The Annals of the Japan Association for Philosophy of Science, Vol. 10, No. 3, 2001, 109–125.
9. Kikuchi, M. and Nagasaka, I., "On the Three Forms of Non-Deductive Inferences: Induction, Abduction, and Design", Proceedings of the 21st IASTED International Conference on Applied Informatics, 2003, 357–362.
10. Kikuchi, M. Nagasaka, I., Toyoda, S. and Kitamura, S., "A Mathematical Model of Interactions in Artifact Environments", to appear in Proceedings of SICE Annual Conference '03, 2003.
11. Kikuchi, M. and Nagasaka, I, "Situation Theoretic Analysis of Functions for a Formal Theory of Design", to appear in Proceedings of International Conference of Engineering Design '03, 2003.
12. Lassez, J.-L. and McAloon, K., "A Constraint Sequent Calculus", Constraint Logic Programming: Selected Research, F. Benhamou and A. Colmerauer (eds.), MIT Press, 1993, 33–43.
13. Pahl, G. and Beitz, W., "Engineering Design - A Systematic Approach (2nd ed.)", Springer-Verlag, London, 1996.
14. Pu, P. and Faltings, B., "Effective Interaction Principles for User-Involved Constraint Problem Solving", 2nd International Workshop on User-Interaction in Constraint Satisfaction, the 8th International Conference on Principles and Practice of Constraint Programming, 2002.
15. Simon, H.A., "The Sciences of Artificial (3rd ed.)", MIT Press, Cambridge, 1996.
16. O'Sullivan, B., "Constraint Aided Conceptual Design", Engineering Research Series 9, Professional Engineering Publishing, London, 2001.
17. O'Sullivan, B., Freuder, E.C. and O'Connell, S., "Interactive Constraint Acquisition", 1st Workshop on User-Interaction in Constraint Satisfaction, the 7th International Conference on Principles and Practice of Constraint Programming, 2001.
18. Tomiyama, T., Kiriyama, T., Takeda, H., Xue, D. and Yoshikawa, H., "Metamodel: a key to intelligent CAD systems", Research in Engineering Design, 1, 1981, 19–34.
19. Tsang, E., "Foundations of Constraint Satisfaction", Academic Press, London, 1993.
20. Yoshikawa, H., "General Design Theory and a CAD system", Man-Machine Communication in CAD/CAM, Sata T. and Warman E. (eds.), North-Holland, Amsterdam, 1981, pp.35–58.

CSPs at Work — Relevance of Interaction Modules to Deploy Applications *

Amedeo Cesta, Gabriella Cortellessa, Angelo Oddi and Nicola Policella

Planning & Scheduling Team
Institute for Cognitive Science and Technology
Viale K. Marx 15, I-00137 Rome, Italy
{cesta|corte|oddi|policella}@ip.rm.cnr.it
http://pst.ip.rm.cnr.it

Abstract. This paper describes an innovative application of AI technology in the area of space mission planning. A system called MEXAR has been developed to synthesize spacecraft operational commands for the memory dumping problem of the ESA mission called MARS EXPRESS. The approach implemented in MEXAR is centered on constraint satisfaction techniques enhanced with flexible user interaction modalities. This paper describes the effort in developing a complete application that models and solves a problem, and also offers functionalities to help users in interacting with different aspects of the problem. An elaborate interactive environment has been designed and integrated into the system, that helps mission planners to analyze the current problem and take scheduling decisions as result of an interactive process enhanced by different and sophisticated facilities. The paper surveys the design principles underlying the whole project and shows the leading role of the interactive environment in the development of end-to-end applications.

1 Introduction

MARS EXPRESS is a space probe launched by the European Space Agency (ESA) on June 2, 2003 that will be orbiting around Mars starting from the beginning of 2004 for two years. Like all space missions, this program generates challenging problems for the AI planning and scheduling community. Mission planning is a term that defines a complex set of activities aimed at deciding the "day by day" tasks on a spacecraft and at figuring out if spacecraft safety is maintained and mission goals are met on a continuous base. Supporting a complete mission planning problem is a quite challenging goal involving several sub-activities. One of such sub-activities, the dumping of the on-board memories to the ground station is the topic of a study which the authors have conducted for MARS EXPRESS.

A space system continuously produces a large amount of data which derives from the activities of its payloads (e.g. on-board scientific programs) and from

* This work describes results obtained in the framework of a research study conducted for the European Space Agency (ESA-ESOC) under contract No.14709/00/D/IM.

on-board device monitoring and verification tasks (the so called *housekeeping data*). All these data, usually referred to as *telemetry*, are to be transferred to Earth during downlink connections. MARS EXPRESS is endowed with a single pointing system, thus during regular operations, it will either point to Mars and perform payload operations or point to Earth and transmit data through the downlink channel. As a consequence on-board data are first stored on the Solid State Mass Memory (SSMM) then transferred to Earth during *temporal visibility windows*. In Fig. 1 a sketchy view of the components on MARS EXPRESS that are relevant for this problem is presented. An effective management of on-board memory and a good policy for downlinking its data are very important for a successful operation of the spacecraft. The authors' study has addressed the problem of automatically generating downlink commands for on-board memory dumping. They have formalized the problem as the MEX-MDP (the MARS EXPRESS Memory Dumping Problem), defined a set of algorithms solving this problem, and implemented an interactive system, called MEXAR, which allows human planners to continuously model new MEX-MDP instances, solve them, and inspect a number of the solution's features.

The approach described herein is centered on the formalization of the problem as a CSP (Constraint Satisfaction Problem) and on integrating a basic CSP representation with a module endowed with multi-strategy solvers and a second module devoted to interaction with human users. This paper gives an overview of this experience and in particular aims at demonstrating the leading role of an effective interactive environment in the development of complete applications containing AI technology. The module responsible for the interaction with the user should guarantee a friendly and comprehensible representation of the problem, the problem solving process and the solutions provided, especially in those cases in which the automated system is devoted to support a user in solving complex problem. It should allow a user to verify the correctness of the results, the possibility to express her own preferences and give her a supporting environment with different problem solving abilities, while preserving her decisional authority.

The reminder of the paper is organized as follows. Section 2 describes how the MEX-MDP has been formalized, Section 3 introduces a general view of the CSP software architecture, Sections 4-5 describe the components of the architecture responsible for the modeling and the solution of the problem, whereas Section 6 shows in detail the interactive facilities designed with the aim of inserting the human planner in the loop. Some comments end the paper.

2 The Mars Express Memory Dumping Problem

The basic ontology to describe the MEX-MDP domain focuses on two classes of objects: *resources* and *activities*. Resources represent subsystems able to give services, and activities model tasks to be executed on such resources. In addition, a set of *constraints* refines the relationships between the two types of objects. Three types of resources are modeled:

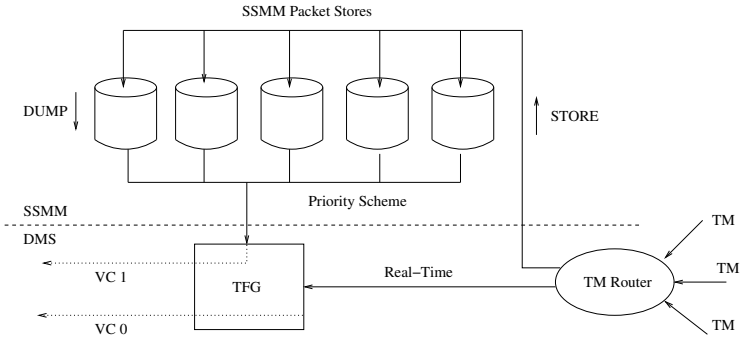


Fig. 1. On-board telemetry flow. The different telemetry (TM) data produced on board are stored on the on-board memory (SSMM) subdivided into packet stores. Memory stores are then downloaded in different dumps that transfer data to the ground.

- *Packet Stores.* The on-board memory is subdivided into a set of separated packet stores pk_i which cannot exchange data among each other. Each one has a fixed capacity c_i and can be assigned a priority value to model different relevance of their data content. Each packet store, which can be seen as a fixed size file, is managed cyclically: when it is full the older data are overwritten. Within a packet store, data are segmented in data packets.
- *On-Board Payloads.* An on-board payload can be considered as a *finite state machine* in which each state has a different behavior in generating observation data (i.e., in each possible state the payload has a different generation data rate).
- *Communication Channels.* These resources are characterized by a set of separated communication windows identifying intervals of time for downlink. Each temporal window has a constant data rate.

Activities describe *how* resources are used. Each activity a_i has an associated execution time interval, which is identified by its start-time $s(a_i)$ and end-time $e(a_i)$. Each activity is characterized by a particular set of resource requirements and constraints. MEX-MDP includes three types of activities:

- *Payload Operations.* A payload operation por_i corresponds to a scientific observation. Each por_i generates a certain amount of data which is decomposed into different *store* operations according to the MARS EXPRESS operational modalities, and distributed over the set of available packet stores.
- *Continuous Data Streams.* The particular case of the continuous data stream operations cds_i is such that $s(cds_i) = 0$ and $e(cds_i) = +\infty$ (where $+\infty$ is internally represented as a finite temporal horizon). This activity represents a continuous generation of data with a fixed average data rate (it is used to model housekeeping). Indeed we represent a *cds* as a periodic sequence of store operations. In particular, given *cds* with a flat rate r, we define a

period T_{cds}, such that, for each instant of time $t_j = j \cdot T_{cds}$ $(j = 1, 2, \dots)$ an activity st_{ij} stores an amount of data equal to $r \cdot T_{cds}$.
- *Memory Dumps.* A memory dump operation md_i transfers a set of data from a packet store to a transfer device (Transfer Frame Generator, the TFG of Fig. 1). Those activities represent the transmission of the data through the communication channel.

Given a set of memory store operations from both the scientific observations $POR = \{por_1, por_2, \dots, por_n\}$ and the housekeeping $CDS = \{cds_1, cds_2, \dots, cds_m\}$ a *solution* is a set of dumping operations $S = \{md_1, md_2, \dots, md_s\}$ such that the following constraints are satisfied:

- The whole set of on board data are "available" on ground within a temporal horizon $\mathcal{H} = [0, H]$.
- Each dump operation starts after the generation of the corresponding data. For each packet store, the data are moved through the communication channel according to a FIFO policy.
- Each dump md_i has an assigned time window $w_j = \langle r_j, s_j, e_j \rangle$, such that the dumping rate is r_j and the constraint $s_j \leq s(md_i) \leq e(md_i) \leq e_j$ holds. Dump operations cannot reciprocally overlap.
- For each packet store pk_i and for each instant t within the considered temporal horizon, the amount of stored data has to be below or equal to its capacity c_i (no overwriting is allowed).

The solutions should satisfy a quality measure. According to requirements from ESA personnel a high quality plan delivers all the stored data as soon as possible according to a definite policy or objective function. To build an effective objective function for this problem the key factor is:

- the turnover time of a payload operation por_i: $tt(por_i) = del(por_i) - e(por_i)$, where $del(por_i)$ is the delivery time of por_i and $e(por_i)$ is the end time of the payload operation on board;

An objective function which considers this item is the *mean α-weighted turnover time* MTT_α of a solution S:

$$MTT_\alpha(S) = \frac{1}{n} \sum_{i=1}^{n} \alpha_i \; tt(por_i) \tag{1}$$

Given an instance of a Mex-Mdp, an *optimal solution* with respect to a weight α is a solution S which *minimizes* the objective function $MTT_\alpha(S)$. The weight α can be used to take into account two additional factors: the data priority and the generated data volume. In this paper, we use the *Mean Turnover Time* (MTT) with $\alpha_i = 1$, $i = 1..n$. [1].

[1] For a more detailed description of the problem see [6] where it is also shown how the optimization problem for Mex-Mdp is *NP-hard*

3 A Software Architecture for a CSP Representation

In our previous work we have developed a software framework for constraint-based scheduling called O-OSCAR [3]. This framework has spawned a preliminary prototype of MEXAR which has been used to deepen our comprehension of different aspects of the problem. A new CSP approach to the problem was then synthesized and an optimized version of MEXAR was developed in Java and delivered to the ESA. It is worth noting that MEXAR was developed using O-OSCAR's same architectural approach, even if specific subcomponents were re-implemented. The general architecture we are referring to is composed of four modules which, as shown in Fig. 2, implement a complete CSP approach to problem solving:

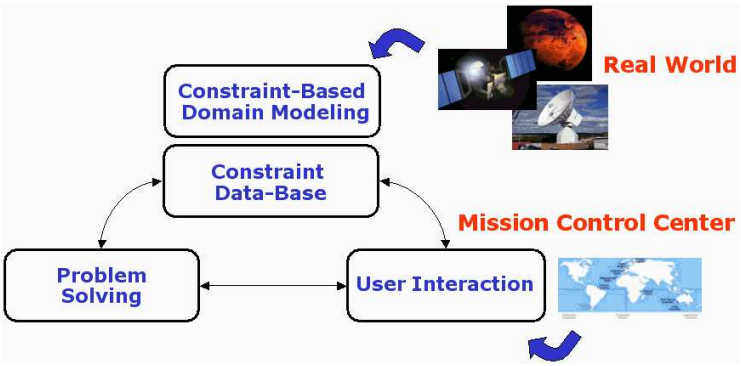


Fig. 2. Components of a CSP architecture

Constraint-Based Domain Modeling. This module models the real domain, captures the dynamic rules according to which the domain evolves and establishes a representation of the problem.

Constraint Data Base. The key part of a CSP architecture is the Constraint Data Base (CDB), a module all the others rely on. It provides data structures for representing the domain, the problem, the solution and its management as a set of constraints.

Problem Solving. This module is responsible for implementing the solution algorithms. In a planning and scheduling domain, we are not necessarily concerned with optimization problems, and approximations of the optimum solution are also acceptable. A solutions is often the result of an iterative process: an initial schedule is found, problems are identified, constraints are relaxed and changes are made on the solution. The Problem Solver captures this iterative process endowing the user with multiple algorithms he or she can choose among.

User Interaction. This module directly interacts with the user, and allows him or her to take part in the process of finding a solution by providing advanced problem solving functionalities. Along with the information which is fed back to the user, the interactive components deliver variable levels of control which range from strategy selection to iterative solution optimization.

This is a general and abstract picture of the approach to the problem. Indeed this structure underscores the different aspects involved in a complete approach and significantly impacted the subdivision of work during system development. It is worth noting that the first two modules are strictly interconnected, and will be described together in the following Section. They are separated in the figure to remark the existence of a basic "core" data structure, the constraint-data base, which is responsible for the integration of the other three functionalities.

4 A Constraint-Based Model for MEX-MDP

A CSP instance involves a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ in which each element has its own domain D_i, and their possible combinations are defined by a set of constraints $C = \{C_1, C_2, \ldots, C_m\}$ s.t. $C_i \subseteq D_1 \times D_2 \times \cdots \times D_n$. A solution consists of assigning to each variable one of its possible values s.t. all the constraints are satisfied. A CSP representation of a problem should focus on its important features. In the case of MEX-MDP, we selected the following characteristics: (1) the temporal horizon $\mathcal{H} = [0, H]$, (2) the store operations that are characterized by their start time t and their amount of data d, (3) the temporal windows in which no communication may occur, (4) the finite capacity c of each memory bank, (5) the FIFO behavior of the memory banks.

Considering the first three items we split the temporal horizon $\mathcal{H}$ in different contiguous temporal windows according to *significant events*: store operations and change of transmission rate. The idea is to create a new window for each significant event on the timeline. This partitioning allows us to consider a temporal interval w_j in which store operations do not happen (except for its upper bound) and the data rate is constant. Furthermore, the packet stores' behavior allows us to perform an important simplification. In fact, it is possible to consider both the data in input and those in output to/from the memory as flows of data, neglecting the information about which operations those data refer to (such information can be rebuilt with a straightforward post-processing step). Thus, the decision variables are defined according to the set of windows w_j and to the different packet stores. In particular we consider as *decision variables* δ_{ij}, the amount of data dumped from the packet store pk_i within the window w_j. According to the partition in separate windows we introduce also: (a) d_{ij}, the amount of data stored in pk_j at t_i, (b) l_{ij}, the available capacity of pk_j at t_i, (c) b_j, the maximal dumping capacity within the window w_j. All these items represent the input of the problem.

A fundamental constraint captures the fact that for each window w_j the difference between the amount of generated data and the amount of dumped

data cannot exceed l_{ij}, the maximal imposed level in the window (*overwriting*). Additionally, the dumped data cannot exceed the generated data (*overdumping*). We define the following inequalities as *conservative constraints*.

$$\begin{array}{l} \sum_{k=0}^{j} d_{ik} - \sum_{k=1}^{j} \delta_{ik} \leq l_{ij} \\ \sum_{k=0}^{j-1} d_{ik} - \sum_{k=1}^{j} \delta_{ik} \geq 0 \end{array} \qquad i = 1 \ldots n,\ j = 0 \ldots m \tag{2}$$

A second class of constraints considers the dumping capacity imposed by the communication channel. The following inequalities, called *downlink constraints*, state that for each window w_j it is not possible to dump more data than the available capacity b_j.

$$0 \leq \sum_{i=1}^{n} \delta_{ij} \leq b_j \quad j = 1 \ldots m \tag{3}$$

The formalization of the problem as a CSP allows for the synthesis of *propagation rules* which examine the existing search state to find implied commitments. Though, in general, it is not possible to remove all inconsistent values through propagation rules, they considerably speed up the solving procedures. In MEX-MDP two basic results allows the generation of propagation rules on the basis of the conservative and downlink constraints. In [6] a detailed description of these propagation rules is shown.

5 Problem Solving

A detailed description of the problem solving techniques is given in [6] where also experimental evaluations are given, here we summarize the main results. To solve a MEX-MDP represented as described in the previous section we have developed a two-stage approach:

Data Dump Level. We first figure out an assignment for the set of decision variables δ_{ij} such that the constraints from MEX-MDP, see (2) and (3), are satisfied.

Packetization Level. The second stage is a constructive step: starting from the solution of the CSP derived at the first stage, the single data dumps within each of the windows w_j are synthesized (that is, each δ_{ij} of the previous phase is translated into a set of dump activities).

MEXAR is endowed with a multi-strategy solver implemented on the basic blackboard represented by the Constraint Data Base. In particular, we have two algorithms, a greedy and a randomized algorithm, which compute new solutions for a MEX-MDP. A third, tabu search algorithm is used to look for local optimizations

on a current solution obtained by the first two approaches. The *Greedy Algorithm* simply consists in assigning a value to each decision variable according to a heuristic. The variables are selected considering the windows in increasing temporal order. Two different solving priority rules are implemented: (a) CFF (Closest to Fill First) selects the packet store with the highest percentage of data volume. (b) HPF (Highest Priority First) selects the packet store with the highest priority. In case a subset of packet stores has the same priority, the packet store with the smallest store as outcome data is chosen. The *Randomized Algorithm* implements a basic random search [5] that turns out to be quite effective in this domain. This method iteratively performs a random sampling of the search space until some termination criteria is met. In our approach we select the variable in a random way, then the maximal possible value is assigned, considering: (1) the data contained in the packet store, (2) the amount of data already planned for dumping and (3) the dump capacity of the window. Both greedy and randomized algorithms are aided by the propagation rules. As usual in CSP, they allow to avoid inconsistent allocation and to speed up the search. The *Tabu Search* implements an instance of this well-known local search procedure for this domain. A specialized move tries to improve the objective function $MTT_1(S)$ performing exchanges on data quantities between pairs of windows w_j. The current tabu is very basic, and further studies are under way to refine its effectiveness.

6 Designing a User Interaction for MEXAR

The idea we have pursued in the development of a support tool for the Mex-Mdp is to endow a human planner with an advanced and helpful tool able to support and enhance his/her solving capabilities. Such a support tool should enable the user to more easily inspect a problem, analyze its features and find satisfactory solutions through step by step procedures, while guaranteeing continuous control over the problem solving process. It should indeed guarantee an easy interaction protocol, providing friendly representations of the problem, the solutions and all the entities of the domain, while hiding the underlying complexity so that the user can concentrate on higher level decision tasks. Figure 3 shows how these ideas have been actually implemented in Mexar. A user is part of the real world and Mexar endows him or her with an additional "lens" to analyze the world through the tool. To obtain this the CSP solving system —represented by the right box— is coupled with an Interaction Module —left box.

Different levels of interaction. The MEXAR Interaction Module has been designed subdividing the interaction capabilities in two layers. This is to provide different levels of participation in the solving process. A first layer allows the user to get acquainted with the problem and its features, optionally compiling an initial solution. Once the human planner has a deeper knowledge of the problem, he/she can access the second interaction layer trying to contribute with his/her expertise and judgment to the problem solving. This mechanism makes

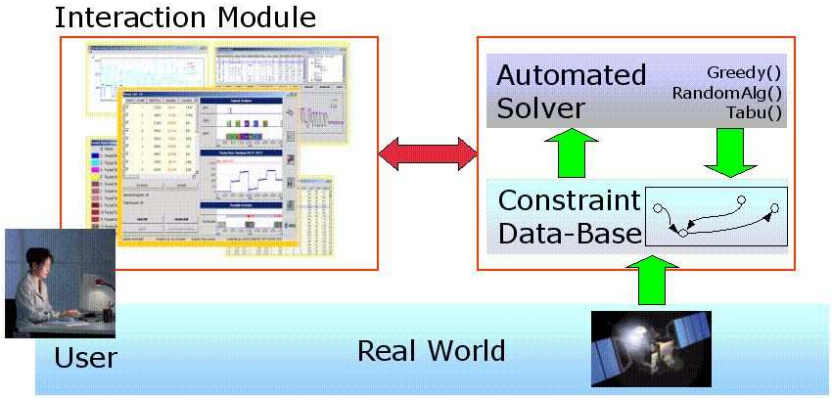


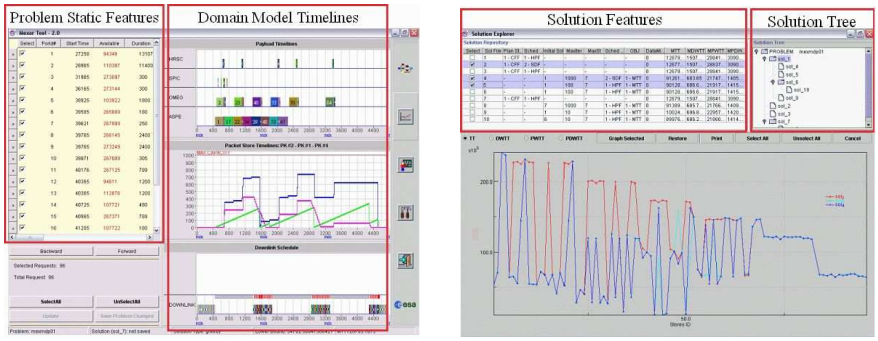
Fig. 3. The "human in the loop" schema

it possible to choose between two modes of operation: to completely entrust the system with the task of finding a solution or to participate more interactively in the problem solving process.

The different facilities are grouped together in two different graphic environments, the Problem Analyzer and the Solution Explorer shown in Fig. 4, where additional squares and text have been inserted to underscore the meaning of subparts of the layouts. The following subsections give further details on the interaction functionalities.

6.1 The Problem Analyzer

The Problem Analyzer contains two groups of functionalities, one for problem editing and refining and a second for problem solving. The interaction layout is based on the idea of "transparency" of all the different components of the representation of an MEX-MDP. This transparency follows what we call the "glass box principle": it enables the user to visually control the temporal behavior of the domain variables that are modeled. Figure 4(a) shows the basic idea used for visualizing a MEX-MDP problem and its solution. MEXAR provides different representations of the problem features allowing a user to choose which one focusing on. In particular a textual representation of the problem (left box in Figure 4(a)) provides a description very close to the one the human mission planner is used to deal with. It provides the PORs list in textual form, specifying all the related information. This representation is nevertheless linked and synchronized to an alternative one based on a graphic representation of the input activities (PORs) of the problem and representing their distribution on the payloads timelines (right box). For each payload a different timeline is shown where each POR is represented by a colored rectangle labeled with a natural number and starting from its start time; the used capacity of the packet-stores is also



(a) Problem Analyzer (b) Solution Explorer

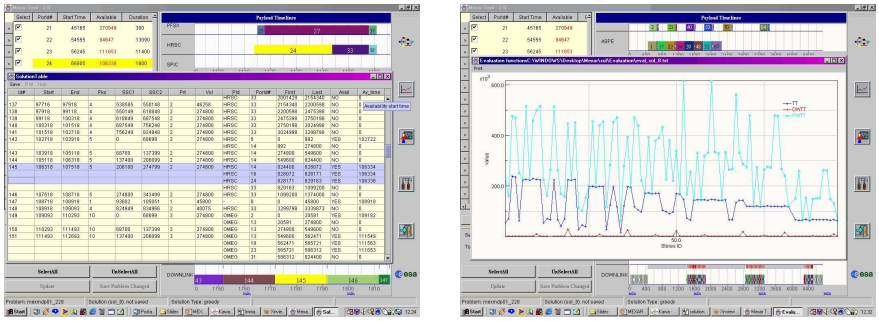
Fig. 4. User interaction in MEXAR

provided, that contains a graphic view of the temporal function representing the volume of stored data and the packet store capacity. The first representation of the problem presents information in more details and is very close to the files an tables the mission planner is used to manage, while the graphic representation is more compact and provide a more intuitive and high level vision. In designing these two different representations we tried to provide the user with the possibility to personalize the interaction by choosing her own interaction modality and express her preferences.

Analyzing the solution MEXAR is endowed with a number of interactive functionalities to inspect single aspects of the solution. Such services have been build to help understanding better an amount of information that is compacted to guarantee intuitiveness of the representation. We are talking about ability of zooming, temporal synchronization, etc. There are two different way of representing a solution. The first one is the graphic representation over the downlink channel which provides a general and qualitative vision of the dump activities, the second one is a more detailed representation that is a solution table.

The solution table is a data structure that reconstructs all the details concerning the solution of the current problem. It presents information in more details and is very close to the files an tables the mission planner is used to manage, while the graphic representation is more compact and provide a more intuitive and high level vision. An example is shown in Figure 5(a). Using the table is possible to check for example (a) how the data from a single POR are segmented in different dump operations, (b) how the data return time has been generated, etc. In general it could be also possible to directly generate the dump commands from the lines of the table. In fact the whole table can be saved as a separate file and manipulated by different programs. We have additionally used

the table to validate the results produced by the problem solver using different algorithms. It is also possible to go further in the integration of this window with the visual features of the PA layout. Another feature of the Problem Analyzer is the possibility to evaluate a solution according to some metrics. This possibility allows the user to estimate easily the quality of a solution with rispect to some choosen parameters. In fact, a graphic evaluation has been added to obtain an immediate level of evaluation on the current solution. When calling the command from the menu a dialogue window allows to choose one or more evaluation functions (e.g. Turnover Time, Data Weighted Turnover Time, etc).



(a) showing tabular data on the solution

(b) showing evaluation parameters

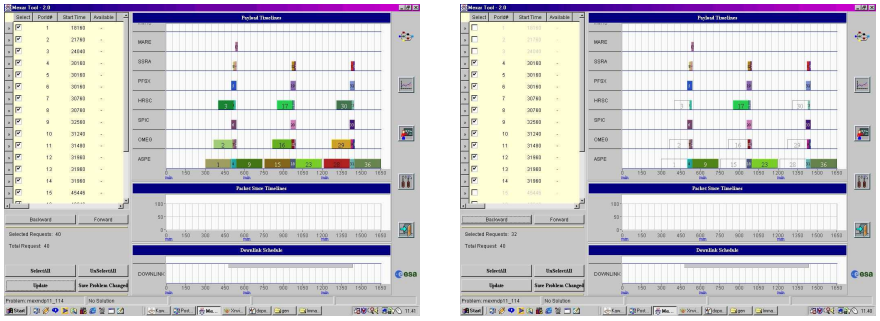
Fig. 5. Macro tools for analyzing the solution

Figure 5(b) shows an example. The x axis represents the PORs according to their increasing ID number, while the y axis represent the turn-over time .

Refining a problem incrementally. The current release of MEXAR integrates functionalities of problem refinement. The broad idea is to suggest a modality of work in which the tool is inserted in a continuous loop with the user that uses it to test different alternatives. Figure 3 suggests an hypothetical "Solve-Inspect-Revise" cycle in which the MEXAR tool and the user are integrated. After the solution of the current problem, the user reasons on the obtained result and can modify the problem behind, save the changes and start a new solution cycle.

The pursued idea allows the user to decide changes to the current problem by, for example, deleting some activities (e.g., if it is not necessary or has a lower priority and the problem is over constrained). The automated tool, built on top of a CSP representation, dynamically updates the changes. In the real situations, in fact, a certain degree of negotiation must be guaranteed. Both the CSP representation and the interactive approach to the problem solving promotes a

certain level of negotiation, often necessary in space missions. Before showing examples of use of this service, it is worth noting that we have currently implemented such a functionality starting from a basic problem and working on it by deleting activities. This functionality can be easily extended allowing incremental modifications in any direction (allowing also addition of activities) because this is fully supported by the background CSP representation. An example of problem refinement is shown in Figure 6.



(a) before (b) after

Fig. 6. Exploring different problem formulations

The human observes the available scenario in 6(a) for a given problem. He goes through the POR table and, according to his expertise, follows a criteria to remove some of the activities from the problem deselecting them. Then these choices are transferred on the graphical representation of the problem and the user can see if the changes have been effective on the packet stores, and can also directly ask for a solution of the same problem. Two different modalities for saving such a new problem are provided: (a) it is possible to save the problem including the deleted activities, so that later on they can be reinserted; (b) or it is possible to define a completely new problem composed only by the selected activities. The idea is one of allowing a trial-and-error refinement of a problem that can be useful when deciding what POR activities actually plan for executed plan.

The choice of the user in very important but the tool is acting as an intelligent blackboard that shows updated information. This schema can be furtherly enriched and represents a potential line for future developments.

6.2 The Solution Explorer

Once the human planner has a deeper knowledge of the problem and all the aspects involved in, she can start a deeper level of interaction with the system

trying to contribute with her expertise and judgment to the problem solving. In this way it is possible to choose either if completely entrust the system with the task of finding a solution or to participate more interactively in the problem solving process. As we said before the Problem Solver allows a user to apply different solving methods to the same problem (greedy solver, randomized algorithm, local search, each with different possible tuning parameters). In addition specific functionalities allow the user to save different solutions for the same problem and to guide a search for improvements of the current best result applying different optimization algorithms. The idea behind the Solution Explorer (see Fig. 4(b)) is that an expert user could try to participate more deeply in the problem solving process. A user might generate an initial solution, save it, try to improve it by local search, save the results, try to improve it by local search with different tuning parameters and so on. This procedure can be repeated for different starting points, resulting in the generation of different paths in the search space.

The Solution Explorer is essentially built on top of a repository of solutions. It contains three panes: (a) a table containing a distinct saved solution on each line and the information on how it has been generated (with greedy, randomized algorithm or tabu search) and the average evaluation parameters for that solution (left box in Fig. 4(b)); (b) a representation of the set of solution trees that are generated for that problem. This is to show the relation among solutions and to keep the user under control of the names of the generated solutions and their starting points (right box in Fig. 4(b)); a graphic window where it is possible to evaluate and compare multiple solutions among them according to different parameters (in the bottom of Fig. 4(b)). Looking at the solutions tree and using both the evaluation capability and his or her own experience the user can visit the different saved solution series both to create new ones and, to choose the one which is most fit for execution.

The current interface still does not completely abstract away the technicality of the algorithms from users. Further work is needed to fill the gap between the reasoning abilities of the user and the automated solver. Nevertheless, a step has been made to fill this gap by giving the user a supporting environment with different problem solving abilities while preserving the user's decisional authority.

7 The System at Work

MEXAR has been delivered to ESA-ESOC in May 2002 and is currently available to mission planners. Users' reactions have been quite positive. We highlight in particular a real interest in the idea of using an automated tool that performs boring and repetitive task on their behalf which preserves the user's control on the flow of actions and the possibility to choose the final solution supported by the potentialities of the automated tool. In addition, the users' interest in the use of intelligent techniques for shortening the time needed to solve complex mission tasks is quite interesting. MEXAR's functionalities for solution revision suggest a modality of work with a loop $\langle$*user, automated support*$\rangle$, which is currently under

further investigation. In particular, the Solution Explorer provides a concept of human guided search (see also different approaches like [2]), which can be potentially very useful in all those applications where the combinatorics of the problems at hand are very high and the integration of competencies could be of great help in the search for a solution. Additionally, we are considering the addition of explanation functionalities to the system in order to increase users awareness of the problem solving cycle.

8 Conclusions

This paper has introduced MEXAR, a complete software system grounded on AI techniques that solves a quite relevant sub-problem in space mission planning. The paper is focused on describing the general principles that are underlying the architectural development of MEXAR, on technical work we have done on the CSP representation of the basic problem and on the interactive services that allow to develop an end-to-end solution for the human mission planners.

The flexibility of the CSP representation has turned out to be very useful. The declarative problem representation has allowed to realize different solvers but also to integrate effective interaction modalities. It is worth emphasizing the key role of the interactive environment in the development of end-to-end application. In fact, in the system development process, an equal subdivision of the working effort between problem solving and user interaction has been needed.

This second module turned out to be of great importance in space applications dedicated to mission ground segments where the experience of the users should be taken constantly into account and integrated in the solving process. In this area the synthesis of mixed-initiative problem solvers is playing an increasingly important role in deployed applications (see [1] for another example).

Acknowledgements

This work could not have been possible without the support at ESA-ESOC of both the project officer Fabienne Delhaise and the MARS EXPRESS mission planners Michel Denis, Pattam Jayaraman, Alan Moorhouse and Erhard Rabenau.

References

[1] M. Ai-Chang, J. Bresina, L. Charest, A. Jonsson, J. Hsu, B. Kanefsky, P. Maldague, P. Morris, K. Rajan, and J. Yglesias. MAPGEN: Mixed Initiative Planning and Scheduling for the Mars 03 MER Mission. In *Proceedings Seventh Int. Symposium on Artificial Intelligence, Robotics and Automation in Space (i-Sairas-03), Nara, Japan, May*, 2001.

[2] D. Anderson, E. Anderson, N.B. Lesh, J.W. Marks, B. Mirtich, D. Ratajczack, and K. Ryall. Human-Guided Simple Search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI 2000)*, 2000.

[3] A. Cesta, G. Cortellessa, A. Oddi, N. Policella, and A. Susi. A Constraint-Based Architecture for Flexible Support to Activity Scheduling. In *Proceedings of Italian Conference of Artificial Intelligence, AI*IA 01*, 2001.

[4] A. Cesta, A. Oddi, G. Cortellessa, and N. Policella. Automating the Generation of Spacecraft Downlink Operations in MARS EXPRESS: Analysis, Algorithms and an Interactive Solution Aid. Technical Report MEXAR-TR-02-10 (Project Final Report), ISTC-CNR [PST], Italian National Research Council, July 2002.

[5] S. Motwani and P. Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[6] A. Oddi, N. Policella, A. Cesta, and G. Cortellessa. Generating High Quality Schedules for Spacecraft Memory Downlink Problem. In *Principles and Practice of Constraint Programming,* 9^{th} *International Conference, CP 2003*, Lecture Notes in Computer Science. Springer, 2003.

User-Involved Tradeoff Analysis in Configuration Tasks

Pearl Pu, Boi Faltings, and Pratyush Kumar

Swiss Federal Institute of Technology
[1]Human Computer Interaction Group
[2]Artificial Intelligence Laboratory
CH-1015 Lausanne, Switzerland
{pearl.pu, pratyush.kumar, boi.faltings}@epfl.ch

Abstract. We describe configuration systems using constraint problem solving formalisms where feasible products are computed by constraint problem solvers. A feasible product is a configuration of constituent components that violates none of the configuration constraints and meets users' preferences as much as possible. To help users find desirable products, a system must possess and understand users' preference model as well as value functions. A constraint-based multi-attribute optimization problem (MOP) assigns utility functions to the set of feasible configurable products so that optimal ones stand out. Due to the incompleteness and uncertainties of user's preference models, optimal solutions are difficult to compute in practical settings if systems do not constantly interact with users and refine user's preference models. While building four user-involved MOPs, we have accumulated a set of interaction principles that optimize user and system collaboration while computing optimal solutions. In particular, we will concentrate here on our approaches and solutions to address tradeoff tasks in interactive MOPS.

1 Introduction

To perform complex tasks, such as searching the web for suitable products or services, planning a trip, or scheduling resources, people increasingly rely on computerized configuration systems to find outcomes that best satisfy their needs and preferences. However, preferences and desires get into conflicts and choosing the best product is a decision problem, or more specifically a tradeoff problem. Many automated decision support systems cannot satisfactorily help users make tradeoffs without a fully specified value function. On the other hand, a fully specified value function for all decision outcomes is difficult to establish for the following reasons:

- Users' preference models are incomplete. Therefore, it is hard to state value functions for preferences that do not exist.
- Users' beliefs about desirability are ephemeral and uncertain. As we studied our users in choosing vacation packages, we observed that they often start with a high value function on low-cost deals. However, as they discovered features that they found more important to them than price, they would change the initial value function.

Therefore, non-classical cases of tradeoff analysis are a fundamental problem of growing importance, especially if humans are becoming increasingly involved in the decision process. We investigate how to give maximum information support to users and enable them to make a happy choice. We are therefore concerned with the following questions:

- What are the types of non-classical tradeoffs tasks (task analysis and taxonomy)?
- What information do users need when they perform these tradeoff tasks (need assessments)?
- How to design interfaces that optimally present and explain this information to users?
- When there is too much information to present, how to structure the information?

We have built four user-involved configuration systems. During this process, we have found some answers to the above-mentioned questions. We have identified and accumulated a set of interaction principles that aim at augmenting and optimizing user and system collaboration while computing desirable outcomes and making tradeoffs. We have tested them with formative user studies throughout the implementation cycles.

2 Related Works

Previous work in the area of user preference elicitation and tradeoff analysis in a configuration tasks have generally followed two main approaches, based on the classical and modern decision theory. The classical theory deals with the idea of formulating a perfect model of users' preference utility function. Given a perfect utility function, the classical theory can accurately predict final configuration of a configurable item. Classical decision theory [Keeney] treats tradeoff problems under the assumption that a machine is able to help a human to externalize a value structure and use it to evaluate decision outcomes. A popular method to elicit such value function is to ask users to choose a set of outcomes and infer the model from their choices. This process can be lengthy and cognitively demanding.

Behavioral decision theory (Payne et al 1993, Carenini and Poole 2002), on the other hand, is very concerned with decision makers' behavior. Many years of studies have pointed out the adaptive and constructive nature of human decision making. Although individuals clearly aim at maximizing the accuracy of their decisions; they are often willing to tradeoff accuracy to reduce cognitive effort. The following key properties of a decision maker's behavior are fundamental validations to our hypotheses: 1) stating preferences is a process rather than a one-time enumeration of preferences that do not change over time; 2) user involved preference construction is likely to be more effective than using default or implicit models if a user is to understand and accept the solution outcomes (Carenini and Poole 2002); and finally 3) decision makers, when facing an unfamiliar task, are quite opportunistic in choosing their strategies by reassessing metagoals, and are likely to benefit from a

flexibility to choose between fundamental and means objectives (see also Keeney 1992).

Current research in the field has widely established the fact that consumer decision behavior is largely adaptive in nature. A consumer, in most scenarios, does not have a perfect formulation of his utility function, and it is during the decision process that hidden preferences come up to the fore front. These hidden preferences affect the decision process, as a result the decision maker in a sense adapts himself to the decision making process. The decision task and decision environment have a considerable effect on the adaptive decision making process.

Several decision support systems have taken a similar approach, such as FindMe (Burke et al 1997), ATA (Linden et al 1997), and Apt Decision (Shearin and Lieberman 2001). All of them are concerned with decision support for the search of multi-attribute products. FindMe promotes assisted browsing, relying domain knowledge in the process, and aims at reducing complexities in the multitude of product dimensions and data sources for users. Tradeoff analyses and tweaking (vs. example critiquing) are two main components from their system that are similar to ours. However, we investigate a precise concept for the minimal critiquing context in which tradeoff can effectively happen, while their tradeoff component is mainly used for explanation, and tweaking for adjusting values. Apt Decision uses learning techniques to synthesize a user's preference model by observing their critique of apartment features. Its main objective is profiling and predicting what a user wants in apartment searches. Adaptive decision making in AptDecision uses examples to elicit hidden preferences of users regarding apartments. User profiling is done through the process of example critiquing. Users decide features important to them by browsing through the examples to discover new features of interest and revising their preference model subsequently.

Linden et al (1997) described a preference elicitation method using travel planning as its example domain. Initially only few user preferences need to be expressed. The ATA system (automated travel assistant) uses a constraint solver to obtain several optimal solutions. Five of them are shown to the user, three optimal ones in addition to two extreme solutions (least expensive and shortest flying time). User preferences are modeled as soft constraints in the CSP formalism. A candidate critiquing agent (CCA), similar to our example critiquing, constantly observes user's modification to the expressed preference, and refines the elicited model in order to improve solution accuracy.

We have developed similar techniques for a range of applications rather than the travel domain alone, such as conceptual design, COMIND (Pu and Lalanne 1996, 2000, 2002), resource scheduling, ICARUS (Pu and Melissargos 1997), travel planning, SmartClient Travel (Pu & Faltings 2000, Torrens et al 2002), and vacation package finder, VacationPlanner (Jurca and Pu 2001). ATA displays only five solutions, which is not diverse enough to guarantee the inclusion of the most optimal solutions, especially when the current user deviates from the standard one (see Faltings, Torrens and Pu, the same issue). Furthermore, by not showing near optimal solutions, users are discouraged from opportunistic and creative discovery of outcomes. Using compact visualization methods, we are able to effectively display up to thirty solutions in SmartClient Travel, and more in other systems. Finally, we are more concerned with interaction design principles that can guide users to state hidden

preferences, revise ill-defined ones, and focus on fundamental objectives. This is only possible by investigating user issues in a number of application domains.

Several recent works use constraint satisfaction techniques to automate the process of decision making. They also employ an interactive style to elicit preferences. Freuder et al (2000) introduced Exemplification CSPs by using an interactive version of the MAC algorithm to elicit user's value assignments to network parameters. O'Sullivan et al (2001) described 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. Freuder and O'Sullivan proposed to model tradeoffs as additional constraints in CSP-based configuration systems. The main goal of their work is to propose appropriate tradeoffs automatically. The conclusion is that arc-consistency based tradeoff proposals are sufficient for finding optimal tradeoffs. The assumption is, however, that users have strong preferences. That is, the issues of discovering hidden preferences and treating uncertainties in users' beliefs of their desires are ignored for the moment. The idea of system assisted tradeoffs can be made even more powerful with concept of hidden preference elicitation. Users often are not aware of all the attributes of the configuration problem that they are solving, and when these attributes are revealed to them in a step by step process, they formulate new preferences and change their existing preferences to reflect the new information. In ways, our work and theirs can be considered as two parts of the same system. The work described here allows users to perform tradeoff while discovering their hidden preferences and value functions. According to business negotiation theories [Pruitt 1981, Unit 1999 and Luo et al 2003], this process must be done extensively before the actual negotiation itself. Likewise in tradeoff, exploration and discovery are important before tradeoff itself. On the other hand, if a user repeatedly performs similar tradeoff tasks, we can use the techniques suggested by Freuder and O'Sullivan to automate this process. 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. Thus, we hypothesize that even thought it is easier to assist a user already having strong preferences [Freuder and O'Sullivan], the process of hidden preference elicitation should make the task as simple for naïve users, who have relatively less formulated preferences.

Bowen (2001) provides a formal notion of interactive CSP, motivated by the introduction of additional constraints so that only a single solution is generated. A specialization CSP is a set of these constraints. We can see adding hidden constraints as a form of specialization CSP. We intend to incorporate these approaches and extend our category of principles. However, we must first validate them with user studies as we did with our own approaches.

3 Defining Configuration Tasks Using CSP

We use constraint problem solving (CSP formulation) techniques as an underlying reasoning engine for our decision support systems. This is largely due to the natural modeling of multiple attribute products and solution search within the CSP framework. A constraint satisfaction problem (CSP), which underlies the decision

generation process, is characterized by a set of n variables $X_1,...,X_n$ that can take values in associated discrete domains $D_1,..,D_n$, and a set of m constraints C1,..,Cm, which we assume to be either unary (one variable) or binary (two variables), and which define the tuple that are allowed for the variable or pair of variables. Besides integrity 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 integrity constraints are satisfied. When soft constraints are present, it also involves finding optimally preferred assignments.

We are concerned with a definition of attributes, criteria, preferences, and constraints from the user/system interaction point of view. A feature or component attribute is the aspect of a product (or solution) for which users would like to express preferences. A component attribute is a physical aspect of a product (e.g., the screen size of a PC), while a feature attribute is an abstract concept for a product such as the assimilability of a product (easy, hard). Optimization criteria are those attributes used for the evaluation of decision alternatives, and can be a function. For example, the total flying time of a trip is the sum of all of the flying segments plus the transfer time at intermediate airports. We can then state a criterion for the trip. For example, it should not take longer than x number of hours.

A preference is a statement about a desired condition on an attribute or a criterion. For example, “I would like to leave Geneva after 10am” is a user preference. Most user preferences are stated, although some can be inferred, such as those used in collaborative filtering. User’s preferences can be also inferred from demographical analysis, default logic, and case-based reasoning (Ha and Haddawy 1998). For example, knowing that a traveler is booking a business trip, we can infer that his most important criterion is to be at destination on time. In this paper, we focus on user stated preferences only.

A constraint is a statement about a condition whose violation will lead to user’s task failure. Being in Hamburg at 2pm for a business meeting is a constraint if that meeting cannot be pushed back. If the user fails to find a flight to satisfy that constraint, he will not be able to achieve his main objective. In most of our systems, user constraints are modeled as hard preference to achieve a uniform treatment of constraints and preferences. Besides user specified constraints, our system also deals with integrity constraints, those conditions that must be satisfied in the configuration process. For example, integrity constraints would state that there must be at least 30 minutes for changing flights within Europe, and 60 minutes for international ones. Integrity constraints are part of domain knowledge, and are elicited from domain experts during the construction of the system.

All of these variables are uniformly known as decision parameters, since they effect how users make decisions. We distinguish incomplete (or hidden) decision parameters from ill-specified ones. Ill-specified parameters give rise to conflicting solutions, assuming that attributes are not independent. An ill-specified parameter can emerge when users are too concerned with means objectives rather than fundamental objectives (Keeney 1992). For example, a user immediately restricting himself to the choice of minivans will not be able to explore station wagon possibilities. If his

fundamental objective is to have enough baggage space for his family, then the minivan preference is an ill-specified one.

4 Taxonomy of Tradeoff Tasks

Before presenting the taxonomy for tradeoff tasks (TOT), we describe some results from interviewing users and studying how they perform tradeoff tasks in conceptual design, select vacation packages and plan a trip. This process, called task analysis, must precede the design and implementation of any interactive system. Task analysis also includes needs assessment, which analyze users' information needs and functional requirements. The result of task analysis is a task model describing the structure and component breakdowns of the tasks being studied.
We participated in a product design course on conceptual design and design tradeoffs. We interviewed and worked with professors who taught conceptual design, as well as practicing designs of mechanical systems. A number of students from that course then became involved in the interface design of a conceptual design system, COMIND. Such participatory design is a way to bring functional and information requirements directly to the interaction design.

For the travel planning domain, we interviewed students, staff, and secretaries at various times in order to establish a task model. In addition, we surveyed 10 commercial on-line flight reservation systems (Jurca 2000). The task model only differs slightly from the one obtained in the conceptual design domain. We will, however, combine them into one reference model.

4.1 Tradeoff Task taxonomy

From the task analysis, we have identified five types of tradeoff tasks:

1. Tradeoff when desires are in conflict (value tradeoff) – users have specified a set of preference values which cannot be satisfied at the same time. Tradeoff, that is the revision of certain preference values, is necessary to obtain feasible configurations.
2. Tradeoff when users are uncertain about value functions (utility tradeoff) – while doing tradeoff and especially when users can visualize the set of decision outcomes, they change the importance attached to a certain criteria. In vacation planning especially, user will often relax the price preference in order to keep vacation options in exotic places. This is a form of the first type of tradeoff except users are manipulating the value function rather than the individual assigned values. That is, if a user is repeatedly relaxing the price preference, we may be able to infer that his preference for destination is stronger than that for price. Modeling value functions is very close to modeling users. We must do this with extreme care in order to keep users in control.
3. Tradeoff when a decision maker is undecided (outcome tradeoff) – users are ambivalent about outcomes of a decision process. In this case, we observed that users are likely to elicit additional preferences so that some outcomes stand out more.

4. Tradeoff at the constraint level (constraint tradeoff) – when one or a set of constraints give rise to zero solutions or very few solutions, diagnosing the problem at the value assignment level is inefficient. Rather, relaxing the constraints or introduce additional values will help. Which constraints to relax, or whether to relax constraints or introduce values, are some of the tradeoff tasks a user faces.
5. Tradeoff between model accuracy and decision effort (accuracy vs. effort tradeoff) – users often prefer accepting a decision outcome and deal with the consequences, rather than improving the preference model with great effort (see Luce 1998). While we do not deal with emotional aspect of decision making and decision justifiability, we must respect users' cognitive ceiling and design our interactive tradeoff systems respectively.

Accuracy and effort tradeoff depicts how much patience a user has to be engaged in doing the task, and serves as cognitive limit to all the other tradeoff tasks. Value tradeoffs are local adjustment of values. When the number of tradeoff value exceeds a threshold (unknown parameter), users can no longer keep track of the value dependencies. They will adopt the strategies of doing tradeoffs at the solution level. Revising value functions while users are searching products requires learning users' preferences from his behavior. This can be seen as a super class of tradeoff activities from the value tradeoff. Constraint tradeoff is a special case and will not be treated here.

5 Preference Revision – Value Tradeoff

Not only preferences change, preferences should change if we believe that we are adaptive decision makers. Value tradeoff requires revising the original stated preferences for one or more tradeoff variables in order to find feasible compromises. Consider someone looking for a vacation package in Mexico in December. He prefers to explore wildlife in Mexico rather than sun bathing on the beaches. His budget is around $1000. He has not considered kayaking so far. Table 1 shows several packages in the catalog, which match the destination Mexico. Taking into account the preferences on budget and vacation features, there is definitely a tradeoff to make: pay almost 10 times more and spend twice as much time to explore wildlife for one vacation package, or pay less and stay at the beach. Alternatively, the user can choose the kayaking package which matches neither the budget nor the feature. But in order to find these deals, the user cannot prematurely state the budget preference.

Table 1. Different vacation packages for Mexico in December

Vacation in Mexico in December, prefer wildlife	3 days	6 days	7 days
	$142 - $332	$2590	$1290
	beach	wildlife	kayaking

5.1 Exploring Search Spaces by Value Tradeoff

Suppose a configuration system elicits users' preferences in a particular order. If these preferences will eventually lead to conflicts, the discovery of this conflict depends on the order. In the vacation example, if the price is set to $2000 in the beginning, the wildlife deal will not be shown. In other words, the user is never informed of the possibility in which by paying a bit more, he is able to get his dream vacation. To overcome this problem, we have used the dynamic query technique (Shneiderman) and implemented an exploratory tool that enables users to discover his needs and preferences by informing him of all relevant decisions.

Fig. 1 is an interface of VacationPlanner (Jurca and Pu 2001). All attributes of a vacation package (destination, date, max price, max duration, youngest accepted age, and etc.) are displayed on the left-side panel. Users express preferences on these attributes by clicking on the desired check boxes or moving sliders into the right positions. The system will display the packages matching selected preferences. When a user states $2000 preference on price, for example, he sees immediately that the wildlife deal is no longer available because that feature is no longer selectable (grayed out). At this point, he can decide whether wildlife is a more fundamental objective, or his budget.

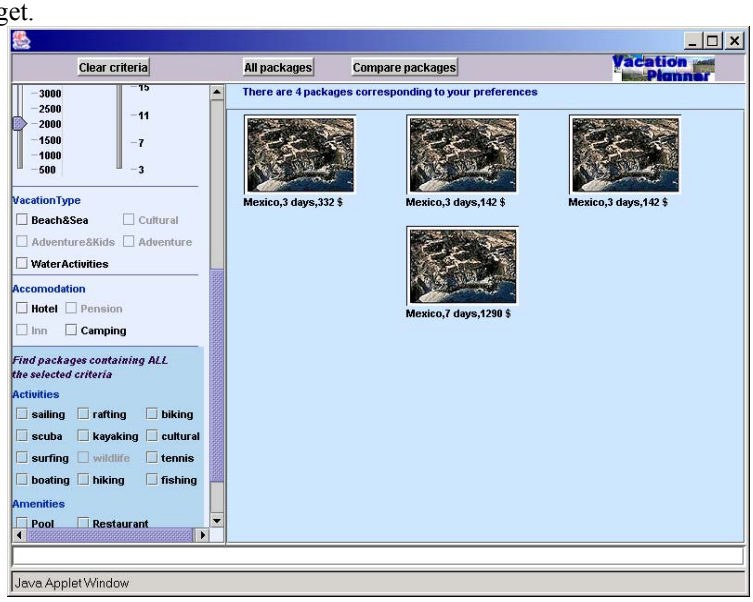


Fig. 1. Putting a low price for a vacation in Mexico will exclude the possibility of exploring wildlife (change the screen shot)

5.2 Partial Satisfaction and Example Critiquing

Displaying the whole set of flight combinations between two is often impossible, and is also not desirable. Making tradeoffs in this case is intertwined with the search

process. That is, rather than making a tradeoff with current solutions, the user would post a preference in order to see near-by solutions for tradeoff purposes. Suppose I only see the beach and kayaking deals and I would like to find a vacation to explore wildlife. I click on beach to choose another alternative: wildlife. Even though the solution with wildlife violates my earlier budget preference, it is still desirable for the system to show that package. Again, by doing so, the system informs the user of an important decision. Highlighting the fact that the price preference has been violated is even more informative for our decision makers as shown in Table 2.

Table 2. Combining search with tradeoffs

destination	duration	price	feature
Mexico	3 days	$332	beach
Mexico	6 days	$2590	Wildlife

We have implemented this tradeoff feature in SmartClient- and Isy-Travel, our configuration system for travel planning. Users can freely pose preferences, some of them being inconsistent. A partial satisfaction CSP algorithm based on soft constraints [Torrens et al] is used to compute near-best solutions where preferences over certain attributes have been violated. An explanation mechanism is used to highlight dependent attributes and their respective assigned values. Users can choose to keep the proposed solution, or retract inconsistent preferences.

In Figure 2, a user has stated a preference on both departure and arrival time for segment 1. He usually prefers to fly out of Geneva after 8:00 am, but for this trip he has to be at Hamburg before 11 am. The flight shown in red (the highlighted lines) violates the first preference, but satisfies the second. The violated attributed has been highlighted. Another solution (not highlighted) shows that leaving after 8:00 am will get him to Hamburg after 12 am. At this point, he must decide whether he will leave earlier, or he pushes back the meeting time.

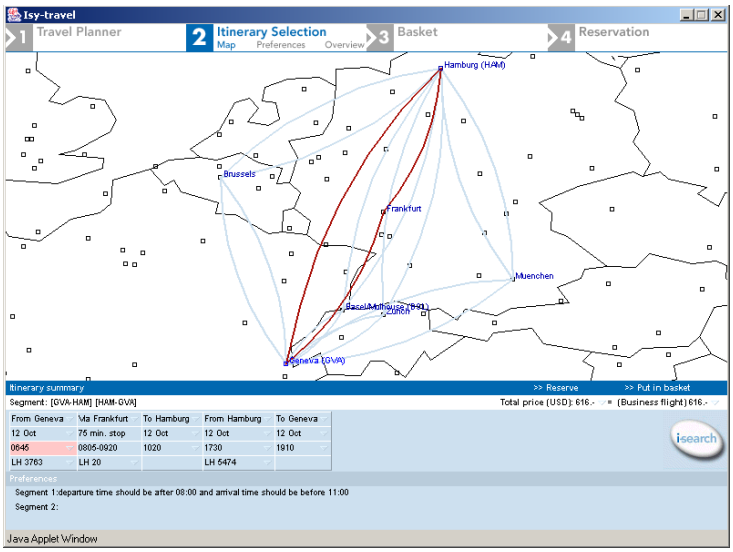


Fig. 2. Conflict values can be revised via tradeoff analysis show decision context

5.3 Quantitative and Non-linear Functions

Some tradeoff functions are highly quantitative. If the aspect ratio is almost constant, then the underlying tradeoff function is predictable, such as the case of using one dollar to buy 3 donuts, and 3 dollars to buy 9 donuts. When one dollar buys 3 donuts, but three dollars buy 12 donuts, it is harder to predict how many dollars it would take to buy 15 donuts, or how many donuts one would get by spending 5 dollars. Such scenarios require means to forward and backward propagate value changes for the dependent variables in tradeoff scenarios, so as to facilitate decision task.

In the case of travel planning, the departure time and arrival time of a segment (a trip consists of multi-segments if it involves one or more intermediate airports) are almost linear. In this case, the tradeoff analysis can be supported via a visual interface called the parallel coordinates, as shown in Figure 3. If a user desires a late departure, a late arrival is expected. If a user wants to arrive before a certain time, the parallel coordinates show the feasible departure times (back propagation). By setting ranges on the range sliders (vertical bars), a user can learn about tradeoff value dependencies and then make informed decisions.

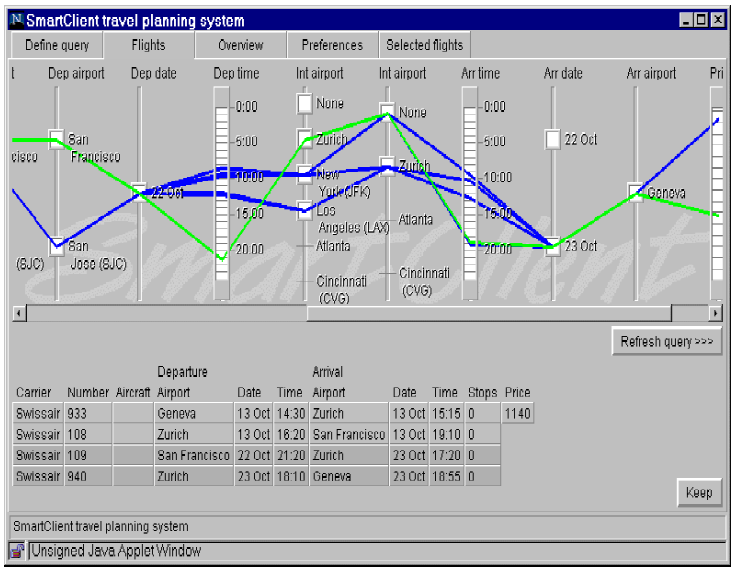


Fig. 3. Value tradeoff in parallel coordinates

For value tradeoff, we propose the following interaction principles:

1. *Use exploratory tools to help users discover their preferences.* Following studies from decision behavior theories, users want to discover their preferences, rather than being modeled as a system. Exploratory tool is the key to building configuration systems that box the products, not the user.
2. *Combine search with tradeoff.* While searching for similar solutions of a configuration, users like to post preferences in the solutions themselves. When preferences are in conflict, partial satisfaction can effectively inform users of meaningful tradeoff problems.
3. *Visual feedback:* Since some preference values do not have an obvious relationship with decision outcomes, an immediate feedback of results is important and allows users to correctly access how to proceed with the incremental process of decision making.

5.4 Tradeoff Context and Minimal Context

In most cases, it is effective to use an entire example solution as a critiquing context, because it accommodates all attributes and it is easy to implement. There are two situations for which showing an entire solution is not ideal: 1) when a solution contains too much detail to show, and 2) attributes influencing decision objectives are not explicit in the example solution. For the first situation, business trips for example may well contain more than the usual outbound and inbound segments. Flying from Geneva to San Francisco with a visit to Denver for a couple of days is a three-

segment trip. In the second situation, choosing a PC feature, such as advanced rather than (rudimentary, basic, high volume), entails to a certain configuration of PC components. When displaying a solution to users, the feature attributes do not become explicit in the example solutions.

We define the dependent attribute cluster to be a set of dependent attributes whose values influence each other in the decision process. Such a cluster is the minimal critiquing context. If a decision problem entails a number of such clusters, we should allow users to critique them one at a time for an effective management of display complexity and cognitive load. As an example, we can have several critiquing contexts in the travel domain:

- cabin class, price
- departure time, arrival time, airline, intermediate airport for the outbound
- departure time, arrival time, airline, intermediate airport for the inbound

This is because the choice of cabin class influences price and vice-versa. Departure time, on the other hand, influences the arrival time, the choice of airlines, and the intermediate airport (if they are available). Given a CSP based decision system, it is possible to detect such clusters either by examining the network structure, inferring these cluster structure from solutions (not trivial), or performing domain knowledge engineering.

In practice however, attributes tend to be related in a complex dependency structure. In the above example, airline can be an attribute in the first group as well if we assume that airlines offer very different fares. If they are strictly modeled as dependent, then the three clusters will join and the minimal critiquing context becomes again the entire solution, thus possibly too complex for the multi-segment trips. A possible solution is to perform conditional preference elicitation i.e. after a user has stated a preference on airline, other attributes become independent and form clusters (see conditional preferences in Boutilier et al 1997). However, this forces users to state preferences in a particular order.

1. *Show minimal context:* If attributes form smaller dependent clusters, it is more optimal to use each cluster as the minimal critiquing context. This reduces the cognitive and display complexities for users.
2. *Show feature attributes in critiquing context:* If preferences were elicited on feature attributes, then the critiquing context should be enlarged to accommodate them as well.

6 Solution Tradeoffs – Eliciting Additional Preferences

Many decision problems involve selecting a single outcome (a vacation package, a trip, a spouse), although other decision problems can afford several choices (a range of values, a set of values). When several criteria compete to argue for the desirability of a single solution, a decision maker is unable to choose. This ambivalence causes many decision makers to seek additional preferences to justify their final choices,

although another equally known strategy is to discover additional constraints to eliminate choices.

Consider the case of selecting apartments whose current features include their size and their price. Figure 4 shows the available apartments and how they rank in the tradeoff space. Suppose users are unable to decide among the apartments lying on the Pareto curve (those nodes labeled black). Upon examining the apartments, a user discovered that one apartment is especially close to public transportation, which saves 20-30 minutes of commute for him every day. Even though it is rather small, he takes this apartment because it is cheaper than the biggest apartment and saves time. This example shows that discovering additional preferences will help a user choose good candidates in tradeoffs.

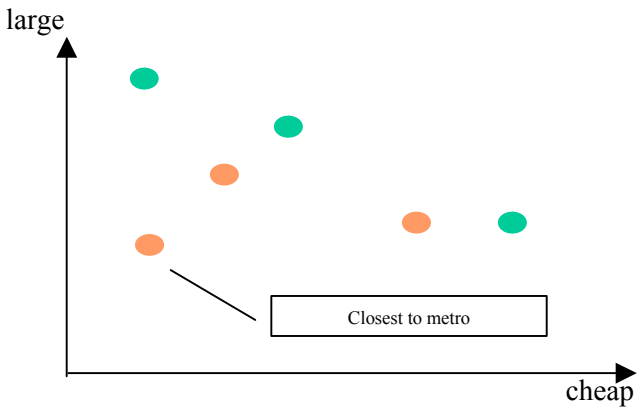


Fig. 4. Apartments in the tradeoff space on price and size, where three of them are Pareto optimal solutions.

In traditional decision theories, the notion of Pareto optimality is very important. A Pareto optimal solution is superior to all other solutions in at least in one criteria dimension. The solution set is divided into Pareto optimal solutions and dominated ones. Most researchers advocate displaying only Pareto optimal solutions. However, the above example shows that via exploration and discovery, a new criterion can push dominated solutions to the Pareto surface. The key for interface design is therefore to show solution details as well as how solutions present themselves in the tradeoff space. This maximizes the exploration and discovery needed to help the undecided decision makers.

We now show several implementations of tradeoff spaces in 2D, 3D, multidimensional and discrete cases. We will then discuss their effectiveness.

Figure 5 is a 2D tradeoff space for selecting flights where the x-axis is the total flying time and y-axis the overall preference satisfaction score (the higher the score the better a solution). These axes can be changed to other values, such as the total cost, and the departure and arrival times of the flights. Consider a user who wants to leave at 8am from Geneva airport. The only flights available from search leave either at 6:20am or 12am. Even though 6:20am is closer to 8am, getting there would require

driving since trains do not operate yet that early. Since parking a car is really costly for a two-week vacation, this user may choose the 12am flight.

Fig. 5. 2D Tradeoff space for selecting flights; x-axis is the total flying time and y-axis the general preference satisfaction score.

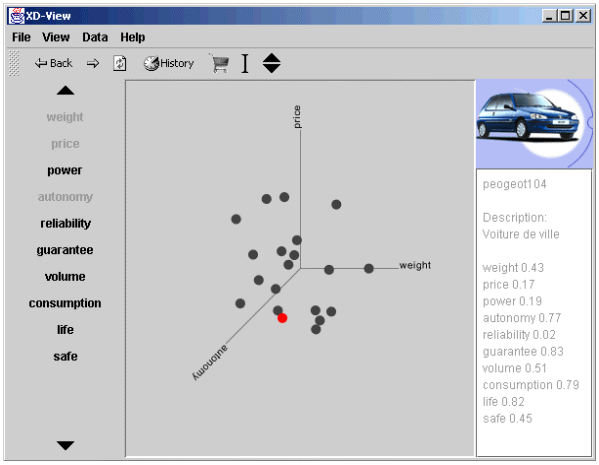


Fig. 6. 3D tradeoff space for a car configurator

Figure 6 shows a three dimensional tradeoff space. All of the criteria for tradeoff are displayed on the left panel. A user can select any three and view the results in the display. This 3D visualization supports rotation, navigation, and zooming. Although it is fun to use the 3D viewer, the advanced features such as navigation may frustrate a novice user. It is also more difficult to compare solutions when they are too close to together.

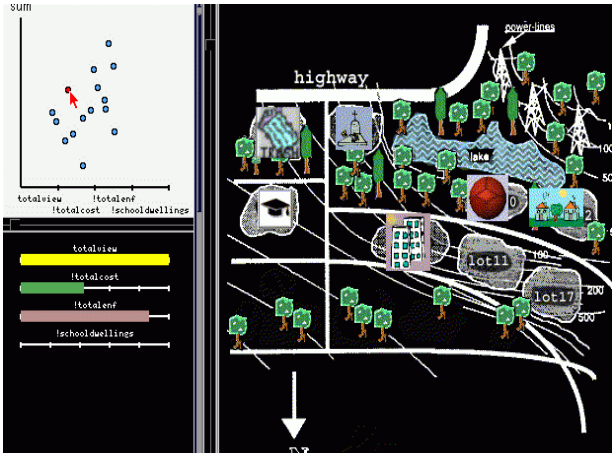


Fig. 7. Multidimensional tradeoff spaces

When the number of simultaneous tradeoff variables exceeds two, we implemented an interface as shown in Figure 7. The user task is to select a solution so that pieces of land are assigned for building different structures. During evaluation, four criteria are being considered: the total view score, the total cost, the noise levels, and the score for measuring how close residential buildings are to schools. We reformulated the optimization problem so that we are always maximizing. That explains the ! sign before some variables. It denotes the negation of that variable. So maximizing !totalcost is minimizing the total cost.

The y-axis represents the total scalar value a solution obtains. On the x-axis, we lay down all the individual criteria. These bottom bars act as "weights" and they attract solutions that score very high in the features that they represent. It is obvious that rearranging the bottom bars will give different displays of the solutions and the ambiguities that exist. However, a user can always consult the slider bars below the tradeoff space for detail. Conversely, changing the values of these bars, a user will see matching solutions in the tradeoff space. This interface overcomes the limitation of not displaying all tradeoff variables and their values in a single overview. It gives a notion of how solutions perform on individual accounts, as well as their details if ambiguities exist. It is fun to explore. Even though we still have to carry out more refined user studies to demonstrate its discovery capabilities, our informal user studies show that users found this interface very effective and have allowed them to discover many solutions that were evaluated as inferior by the initial standards that they had set. In the figure, the selected node is actually chosen by our test user as the best solution, even though it does not perform well in the absolute score (y-axis), on noise, or school distance features.

The interaction principles in this section on solution tradeoffs can be summarized as follows:

- *Show examples to provoke interaction:* Most users are not aware of or cannot articulate preferences. Solving ambivalent tradeoff problems requires discovering

and weighing these newly discovered features of certain solutions. An exploratory tool must provide examples as well as the tradeoff spaces.

- *Do not systematically eliminate dominated solutions:* allow ambivalent users to explore solutions nearby the "optimal" ones. Very often, surprises can be found to explain a tradeoff decision.

8 User studies: Combining Decision Behavior Research with User Requirement Analysis and Formative Usability Studies

Some existing findings from consumer behavior research and behavioral decision theory allowed us to avoid costly experiments to understand users' behaviors when they face decision tasks. However, we still analyzed usability issues by surveying existing commercial systems, such as the survey on travel planning (Jurca 2000). We also used online decision systems such as Personalogic in our user requirement studies. The characteristics, both in terms of shortcomings and strengths, were integrated into the design of Isy-travel and VacationPlanner. Earlier decision support systems were built from analysis done on commercial resource allocation systems employed in the airline industry. COMIND was based on extensive interviews with conceptual designers. The design principles reported here are therefore a result of consumer behavior literature, user requirement analyses, and most importantly testing of these principles in systems that we have built.

Formative user studies were also performed for all of the systems described here. It's a process involving writing scripts for each usage scenario, identifying potential users, conducting usability testing, analyzing results, and proposing design changes. The systems all went through several phases of change as a result of formative user studies.

For the future, we intend to conduct several comparative user studies on rather focused claims. For example, we believe that users select different outcomes under two different conditions: under the condition that they will view only solutions and under the condition that they tradeoff on values without ever seeing an example until the end. We can then further establish interaction principles that address these two conditions.

9 Conclusion

We have pointed out the importance of tradeoff tasks in CSP-based configuration systems. Via tasks analysis, we established taxonomy of various tradeoff tasks. We then focused our discussion on designing, implementing and testing a set of interaction principles that maximizes user and machine's collaboration in finding "optimal" solutions. A recurring theme in these principles is to provide tools to help users model themselves, rather than devising algorithms to model them. Through exploration, we turn the preference model incompleteness and user belief uncertainty to advantages, which is decision discovery. Through exploration, users discover knowledge of the configuration space and make better and more informed decisions.

References

1. J. F. Allen, L. K. Schubert, G. M. Ferguson, P. A. Heeman, C. H. Hwang, T. Kato, M. N. Light, N. G. Martin, B. W. Miller, M. Poesio, and D. R. Traum. *The TRAINS project: A case study in building a conversational planning agent*. TRAINS Technical Note 94-3, University of Rochester, Department of Computer Science, September 1994.
2. C. Boutilier, R. Brafman, C. Geib, and D. Poole. *A Constraint-Based Approach to Preference Elicitation and Decision Making*. In AAAI Spring Symposium on Qualitative Decision Theory, Stanford, 1997.
3. 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.
4. R. Burke, K. Hammond, and B. Young. *The findme approach to assisted browsing*. In IEEE Expert, volume 12(4), pages 32--40, 1997.
5. Carenini G. and Poole D. *Constructed Preferences and Value-focused Thinking: Implications for AI research on Preference Elicitation*. AAAI-02 Workshop on Preferences in AI and CP: symbolic approaches - Edmonton, Canada, 2002.
6. J. Doyle and R. Thomason. *Background to qualitative decision theory*. AI magazine, 20(2), summer 1999.
7. Freuder E, Likitvivatanovong C and Wallace R. *A case Study in Explanation and Implication*, in Proceedings of CP-2000 Workshop on Analysis and Visualization of Constraint Programs and Solvers, 2000.
8. E.C. Freuder and B. O'Sullivan. Generating Tradeoffs for Interactive Constraint-Based Configuration.
9. D. Pruitt. *Negotiation Behavior*. Academic Press, 1981.
10. I. Unt. Negotiation Without a loser. Copenhagen Business School, 1999.
11. Jurca, A. and Pu, P. *Algorithms for Online Vacation Planning*. Technical Report, Swiss Federal Institute of Technology, Lausanne. p. 1-12, 2001.
12. Jurca, A. *Survey of Online Travel Planning Systems.* Technical Report, Swiss Federal Institute of Technology, Lausanne, 2000.
13. Keeney, R. and Raiffa, H., *Decision with multiple objectives: Preferences and value tradeoffs*. 1976: Cambridge University Press.
14. Keeney, R., *Value-Focused Thinking: A Path to Creative Decision Making*. 1992: Harvard University Press.
15. V. Ha and P. Haddawy. *Problem-focused incremental elicitation of multiattribute utility models*, in Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, pages 215--222, August 1997.
16. Ha, V. and Haddawy, P. *Toward Case-Based Preference Elicitation: Similarity Measures on Preference Structures*, in Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, 1998.
17. Horvitz, E., *Principles of mixed-initiative user interfaces*, in Proceedings of The ACM SIGCHI Conference on Human Factors in Computing Systems, ACM Press, p. 159-166, 1999.
18. Inselberg, A. and Dimsdale, B. *Parallel Coordinates: a Tool for Visualizing Multidimensional Geometry*, in Proceedings of IEEE Visualization '90, IEEE Computer Society, p. 361-378, 1990.
19. Lalanne, D. *Computer Aided Creativity and Multi-criteria optimization in Design*. Ph.D. thesis, the Swiss Institute of Technology (EPFL), Lausanne, 1998.

20. G. Linden, S. Hanks, and N. Lesh. *Interactive assessment of user preference models: The automated travel assistant*. In Proceedings of User Modeling '97, 1997.
21. Mackay, W.E., Holm, E., and Horn, S.B., *Who's in Control? Exploring human-agent interaction in the McPie Interactive Theater project*, in Proceedings of The ACM SIGCHI Conference on Human Factors in Computing Systems. 2001.
22. Payne, J.W., Bettman, J.R., and Johnson, E.J., *The Adaptive Decision Maker*. Cambridge University Press, 1993.
23. Pu, P. and Faltings, B. *Personalized Navigation of Heterogeneous Product Spaces using SmartClient*, in Proceedings of the International Conference on Intelligent User Interfaces, ACM Press, 2002.
24. Pu, P. and Faltings, B. *Enriching Buyers' experiences: the SmartClient Approach*, in Proceedings of The ACM SIGCHI Conference on Human Factors in Computing Systems, 2000.
25. Pu, P. and Lalanne, D. *Human and Machine Collaboration in Creative Design*, in Proceedings of the European Conference of Artificial Intelligence, 1996.
26. Pu, P. and Lalanne, D. *Interactive Problem Solving via Algorithm Visualization*, in Proceedings of the IEEE Information Visualization Symposium, IEEE Press, 2000.
27. Pu, P. and Lalanne, D. *Design Visual Thinking Tools for Mixed Initiative Systems*, in Proceedings of the International Conference on Intelligent User Interfaces, ACM Press, 2002.
28. Pu, P. and Melissargos, G. *Visualizing Resource Allocation Tasks*. IEEE Computer Graphics and Applications, 1997. **17**(4).
29. 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.
30. Shearin, S. and Lieberman, H. *Intelligent Profiling by Example*. in Proceedings of the Conference on Intelligent User Interfaces, ACM Press, 2001.
31. Johnson, B. and Shneiderman, B. *Tree-Maps: A Space-Filling Approach to the Visualization of Hierarchical Information Structures*, in *Visualization '91*, p. 284-291, 1991.
32. 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.