

# Visualizing explanations to exhibit dynamic structure in constraint problems<sup>\*</sup>

Mohammad Ghoniem<sup>1</sup>, Narendra Jussien<sup>1</sup>, and Jean-Daniel Fekete<sup>2</sup>

<sup>1</sup> École des Mines de Nantes  
4, rue Alfred Kastler – BP 20722  
F-44307 Nantes Cedex 3 – France  
{Mohammad.Ghoniem,Narendra.Jussien}@emn.fr

<sup>2</sup> 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.

---

<sup>\*</sup> 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:  $d_1, d_2, \dots, d_k$ ) such that:*

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

$\mathcal{C}' \wedge d_1 \wedge \dots \wedge d_n$  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.

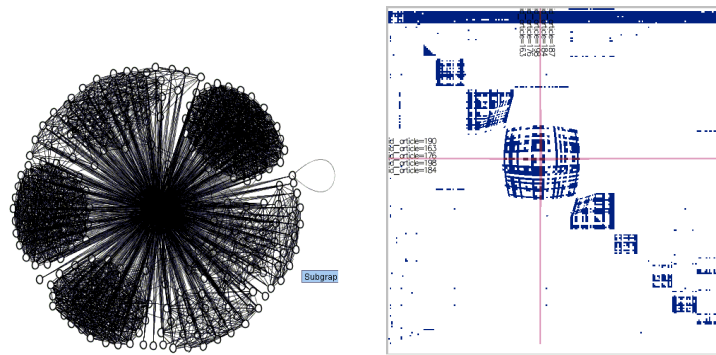
### 3.1 An alternate representation of graphs

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  $1600 \times 1200$  display. Moreover, advanced information visualisation techniques such as dynamic queries [2], fish-eye 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**

Constraints  $c_i$  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<sup>3</sup> 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  $c_{00006}$  (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  $c_{00005}$ ).

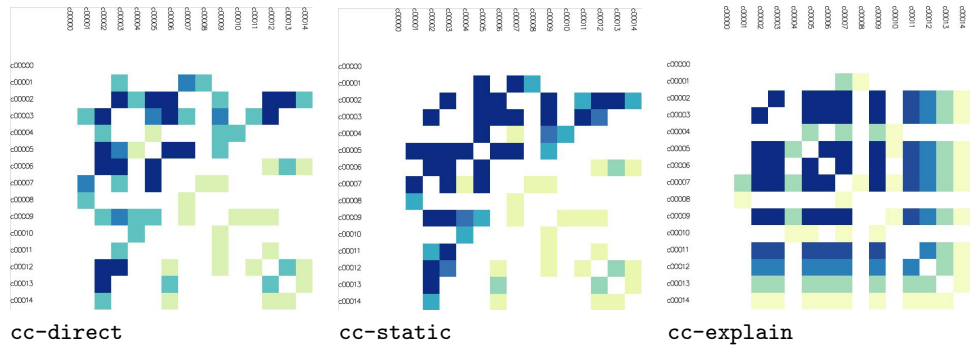
The **cc-static** graph gives some more information. Indeed, in this graph, constraints  $c_{00004}$  and  $c_{00001}$  are not related to the other **allDifferent** constraints (except  $c_{00005}$ ) as expected. However, the dynamic structure that we would like to appear between enumeration constraints 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.

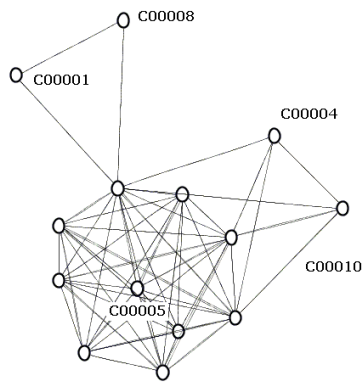
**Variable-variable graphs** The **vv-direct** graph of figure 4 confirms that  $a_2$ ,  $a_3$ ,  $b_2$ , and  $b_3$  have a limited impact on other variable during the search of the first solution to this toy problem. Notice that this graph also suggests that variable  $a_1$  has no impact on  $a_2$  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<sup>4</sup>

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

<sup>4</sup> In the sense that it confirms intuitions about the problem.

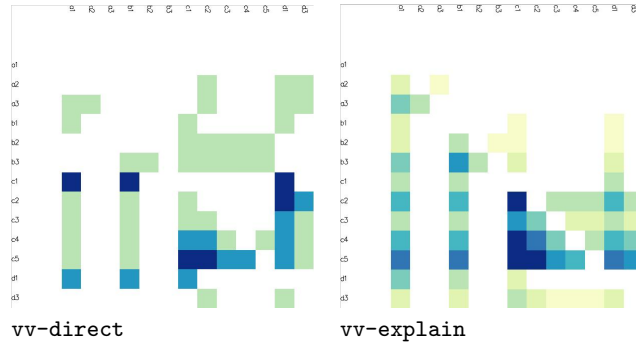


**Fig. 2.** A toy problem (first solution): constraint-constraint graphs. The darker the dot, the more often the constraints are related.



**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

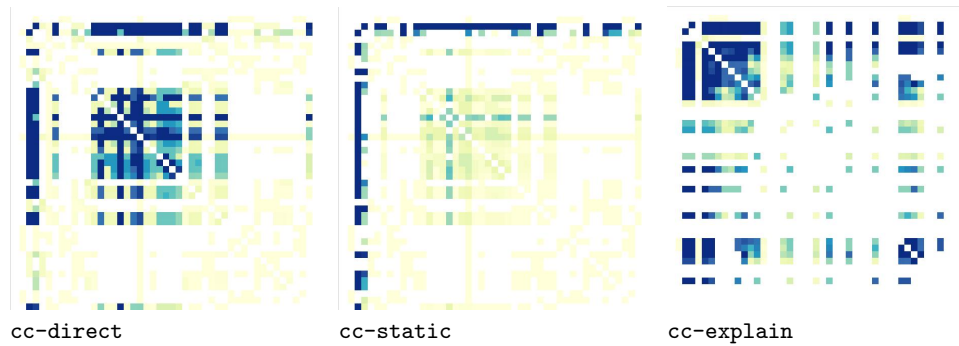
### 4.3 A toy problem: new information

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.



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

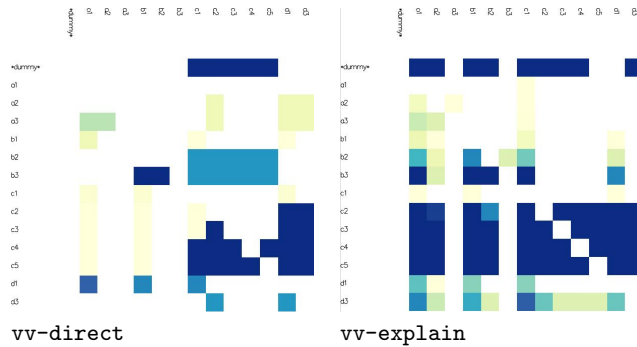


**Fig. 6.** A toy problem (all solutions): a clustered matrix representation of a constraint-constraint graph

#### 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  $c_i$ .

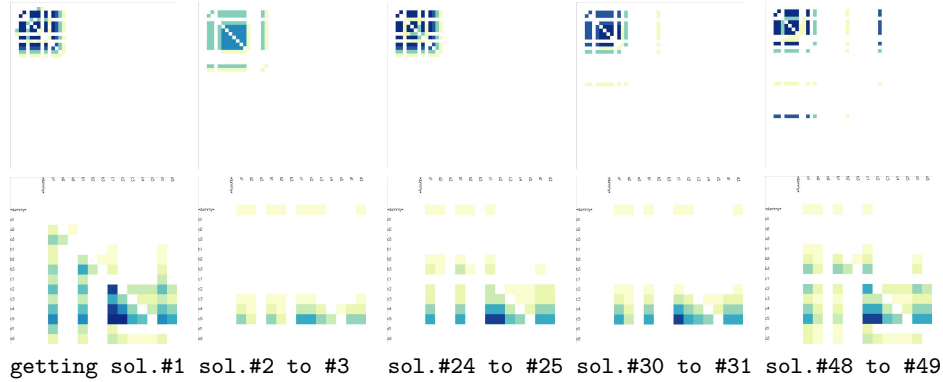


**Fig. 7.** A toy problem (first solution): variable-variable graphs. The `*dummy*` variable is used to generate new solutions and should not be considered as part of the problem.

### 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<sup>5</sup>. 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<sup>6</sup>.

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  $6 \times 6$  (*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  $4 \times 4$  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<sup>7</sup> used by the relevant tasks). During search, 102 enumeration

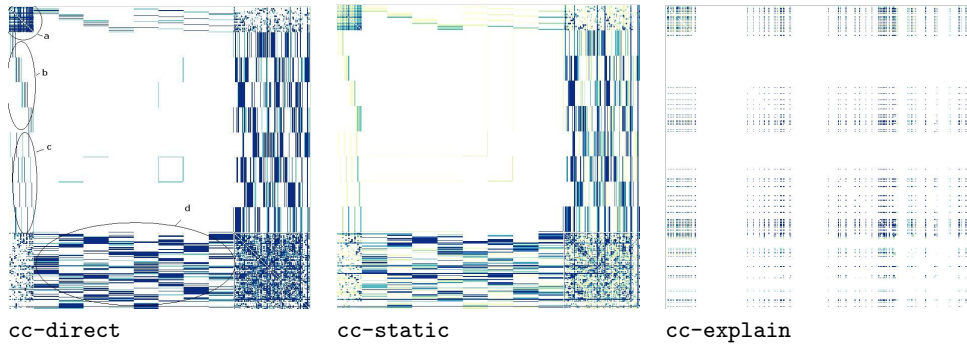
<sup>5</sup> The variables of the CSP are the starting dates of the tasks. Bounds thus represent the least feasible starting time and the least feasible ending time of the associated tasks.

<sup>6</sup> Ending time of the last task.

<sup>7</sup> 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).

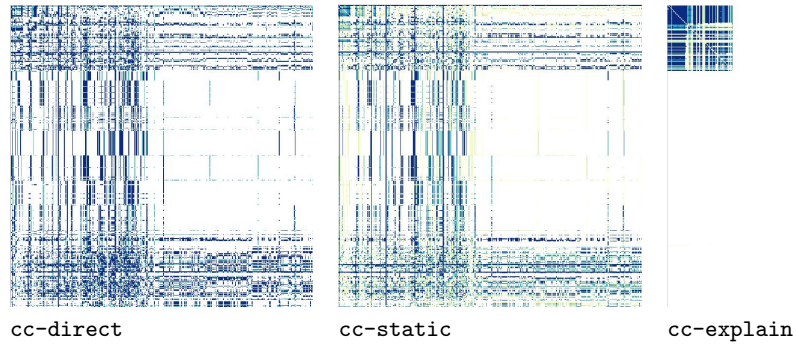


**Fig. 9.** Open-shop scheduling: constraint-constraint graphs

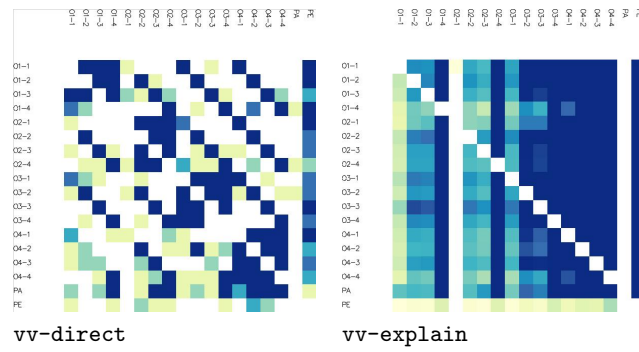
## 5 Conclusion

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 interface 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).