

Peeking in Solver Strategies Using Explanations

Visualization of Dynamic Graphs for Constraint Programming

Mohammad Ghoniem*
École des Mines de Nantes

Hadrien Cambazard†
École des Mines de Nantes

Jean-Daniel Fekete‡
INRIA Futurs/LRI

Narendra Jussien**
École des Mines de Nantes

ABSTRACT

In this paper, we describe the use of visualization tools in the context of constraint programming. Specifically, we focus on the visualization of dynamic graphs using adjacency matrices, and show that visualization tools can provide valuable insights into the behavior of constraint-based solvers. This contributes to better understanding solver dynamics for teaching purposes, and can enhance the analysis and the optimization of constraint-based programs in connection with known and difficult research problems.

Categories and Subject Descriptors

I.3.3 Picture/Image Generation; Display algorithms, I.2.8 Problem Solving, Control Methods, and Search; Heuristic methods, D.2.6 Programming Environments, Graphical environments.

General Terms

Algorithms, Performance, Experimentation, Human Factors, Languages.

Keywords

Visualization of algorithms, Graph drawing algorithms for software visualization, Visual debugging, Constraint Programming, Finite Domain Solver.

Introduction

Solving a problem within the constraint-based programming (CP) paradigm consists in providing a user-defined model of the problem at hand through the declaration of its constitutive variables, and the constraints that relate them. Once the problem is modeled, a specialized solver – a sort of black box – takes care of the resolution process. CP languages have been successfully used in various application fields in both academic and industrial contexts. These fruitful applications include machine scheduling, sports scheduling, and routing problems - to name but a few.

However, CP programmers and end-users have very often been left without effective tools allowing them to reach a good

* email: Mohammad.Ghoniem@emn.fr

† email: Hadrien.Cambazard@emn.fr

‡ email: Jean-Daniel.Fekete@inria.fr

** email: Narendra.Jussien@emn.fr

understanding of the dynamic behavior of solvers, and further analyze, debug, and tune the programs and algorithms they implement.

In this paper, we start with a description of CP needs in terms of visualization along with a quick overview of works related to the visual analysis of constraint-based programs. This is followed by a description of the visualization tools we implemented to help understand, analyze and tune constraint-based programs. We elaborate on the use of advanced visualization techniques such as matrix-based visualizations of graphs and time filtering in order to handle large dynamic graphs. Further, we provide several use cases that describe and discuss the use of our tools. Finally, we highlight future works that can extend the present effort.

1 Visualization Needs of Constraint-Programming

A finite domain constraint problem is specified by the following elements:

- a finite set V of finite domain variables;
- a finite set D that contains all possible values for variables in V ; in finite domain solvers, D is a set of non negative integers ranging over 0 to $maxint$;
- a function which associates to each variable v its current domain (a subset of D), denoted by D_v ;
- a finite set of constraints C . Each constraint c in C defines a relation between the set of variables $var(c)$, the variables of c – i.e. a subset of V .

A solution of the constraint problem is an assignment of the variables of V to values in D (a *tuple*), such that all constraints in C are satisfied. All the constraints are thus solved, and the problem is said to have a solution.

When no such tuple can be found, the problem is said to be “over-constrained”, and the user needs to relax some constraints to achieve feasibility. Very often, a problem may admit several solutions of various quality or cost. Explicitly, the solver always starts by propagating the immediate effects of the constraint set; this often results in the reduction of the variable domains, through the withdrawal of inconsistent values.

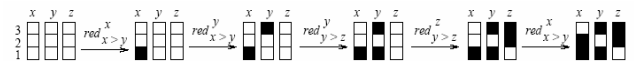


Figure 1: Applications of reductions to the system $\{x > y; y > z\} (x,y,z) \in [1..3]^3$

Figure 1 shows the reduction of three variables x , y and z and two constraints $x > y$ and $y > z$. At the beginning, the initial variable domains, $D_x=D_y=D_z=\{1,2,3\}$, are represented by three columns of white squares. Considering the first constraint, it appears that x cannot take the value 1 because otherwise there would be no value for y such that $x>y$; $red_{x>y}^x$ eliminates this inconsistent value from D_x . The deleted value is marked with a black square.

In a similar fashion, $\text{red}_{x>y}^y$ filters the value 3 from the domain of D_y . Then, considering the constraint $y>z$, $\text{red}_{y>z}^y$ and $\text{red}_{y>z}^z$ withdraw respectively the sets $\{1\}$ and $\{2,3\}$ from D_y and D_z . Finally, a second application of $\text{red}_{x>y}^x$ reduces D_x to the singleton $\{3\}$. The final solution is $\{x=3, y=2, z=1\}$.

If, after this stage, some variable domains are not reduced to singletons, the solver takes one of these variables and tries to assign it each of the possible values in turn. This enumeration stage triggers more reductions, which possibly leads to solutions, or failures, or additional enumerations on non-reduced variable domains. Obviously, the search space can be represented by a tree with an exponentially growing width where each level corresponds to a variable of the problem. For instance, in a problem having N binary variables, the search tree can contain a total of $2^{N+1} - 1$ nodes with 2^k nodes at level k of the hierarchy where k varies from 0 to N if no reduction is done, i.e., if all tuples are enumerated. Industrial problems usually bear thousands of variables with large domains and hundreds or thousands of constraints. Therefore, monitoring such huge hierarchies is definitely a challenging task.

Earlier works have mainly focused on representing such trees in view of supporting the analysis of constraint-based programs, and visualizing solver strategies. These works started with Spy [Brayshaw and Eisenstadt 1991], the Prolog Trace Package (PTP) [Eisenstadt 1984], the Transparent Prolog Machine (TPM) [Brayshaw and Eisenstadt 1990; 1991], and the Textual Tree Tracer (TTT) [Taylor et al. 1991]. Spy provides a linear textual trace of prolog execution using the Byrd Box model. PTP is also a linear textual trace with an extended range of symbols compared to Spy. TPM provides pretty much the same information in the form of an AND/OR tree with a detailed view of individual nodes of the tree. Graphical node-link visualizations were also made available in support of TPM as can be seen in [Brayshaw and Eisenstadt 1990]. TTT uses a non-linear textual notation providing a tree representation of Prolog close to the source code. An evaluation of these four methods is available in [Mulholland 1995].

Recent research projects such as *Discipl* [Deransart et al. 2000] produced a collection of tools aimed at the monitoring of constraint programs and, especially, the visualization of search trees such as the SearchTree visualizer packaged in Chip++ [Deransart et al. 2000] development suite. An important contribution of the *Discipl* project consisted in reaching a better understanding of the needs of CP users. That is, besides search trees, CP users would also be interested in the evolution of variable domains, and the evolution of activity within the graphs of variables, the graphs of constraints, and also within mixed graphs that involve constraints and variables interacting with each other.

In more recent works, authors applied radial space-filling visualization techniques similar to sunburst trees [Stasko et al. 1998] or interRings [Yang et al. 2002] to represent search trees in constraint-based programming applications such as “COMIND” [Lalanne and Pu 2000]. In CLPGUI [Fages 2002], search trees were represented as node-link diagrams in 3D space, and posting queries at certain nodes of the search tree was supported in order to guide the solver on runtime.

All visualization tools are confronted with the major challenge of representing exponentially growing data structures on a limited

screen real-estate with an equally limited memory real-estate. [Fekete and Plaisant 2002; Abello and Ham 2004].

Building on the recommendations of the *Discipl* project, we tackle the problem from a different standpoint. Instead of focusing on the search tree per se, we decided to consider the network of variables and constraints that collaborate together to get the problem solved. For example, in Figure 1, the network connects the variables x and y to the first constraint ($x<y$), and the variables y and z to the second constraint ($y<z$). Section 4 describes various uses of such networks, mainly the visualization of the activity of the variables and constraints during the resolution process.

We have also created activity networks based on “explanations” that are maintained dynamically by a newer generation of constraint solvers such as PaLM [Jussien and Barichard 2000] or Choco [Laburthe 2000]. We describe explanations and their visualization in section 5.

2 Visualization Tools for Activity Graphs

The activity in a network is simply the number of times a variable or a constraint is used over an interval of time. Activity can be represented by a set attribute (a set of timestamps where activity occurred) associated with either the vertices of the network or its edges, or with both. When at time t_0 a variable v is modified, its activity attribute will contain t_0 . This modification will then trigger at time t_1 a constraint c that uses v , and add t_1 to the activity attribute of the edge linking v to c . It will also add t_1 to the activity attribute of c . In turn, c will reduce other variables at time t_i and hence, the attributed activity network builds up progressively.

Activity networks are potentially very large and dense, and are characterized by a varying connectivity or edge weight over time. Therefore, the user’s vision becomes quickly clouded with traditional node-link representations of graphs, and it becomes difficult to follow how they evolve during the solving process, and to identify which parts of these networks are active. As a side note, we may recall that the size of the network refers to the number of variables and constraints in the problem.

2.1 Matrix-Based Visualization of Graphs

So far, visualization of graphs 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 unstable, and they become quickly unreadable when the size of the graph and the link density increase. We use 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, the darker a link. Unlike node-link diagrams, matrix-based representations of graphs do not suffer from link and node overlapping. Virtually, every link (out of the N^2 links) in the graph can be seen separately (Figure 2). When dealing with directed graphs, columns represent the origin of edges and the

rows represent their endpoint vertices, although conventions may vary. 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 visualization techniques such as dynamic queries [Card et al. 1999], fisheye lenses [Carpendale and Montagnese 2001] and Excentric labels [Fekete and Plaisant 1999] enhance the exploration of large graphs, and push the matrix-based visualization one step further in coping with large networks.

2.2 Readability of Matrix-Based Representations

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. In a controlled user experiment [Ghoniem et al. 2004] carried out on seven exploration tasks and nine graphs of different sizes and densities, we have shown that matrix-based representations perform significantly better than node-links for medium to large graphs (50 to 100 vertices) and for dense graphs with all tasks involved in the experiment, except path finding.

2.3 Related Works

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. Bertin [1983] shows that it is possible to reveal the underlying structure of a network represented by a matrix through successive permutations of its rows and columns. 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 [Becker et al. 1995], the authors visualize the load distribution of a telecommunication network using a matrix but most of their effort aims at improving the display of a node-link representation such as displaying half-links or postponing the display of important links to minimize occlusion problems. More recently, in [Ham 2003], the authors implemented a multi-scale matrix-based visualization representing the call graph between software components in a big medical imagery application. In [Ghoniem, Jussien and Fekete 2004], we have shown that a matrix-based representation can be used to effectively grasp the structure of a co-activity graph and assess the activity taking place across time whereas the equivalent node-link representation was unusable.

2.4 Time Filtering

As already mentioned, all links in those graphs are weighted with the number of times the relation is actually established throughout computation. This helps enhance the static structure with dynamic information pointing out active relations. More precisely, we keep a full history of activity within the graph. Thus, 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 entire 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.

We also offer 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 as well.

3 Visualization of Constraint-Variable Networks

Figure 2 shows the activity graph obtained by sorting one hundred variables $x_{i \in]1,100]}$ in domain $[1..100]$ using 99 constraints: $\forall i \in]1,99], x_i < x_{i+1}$

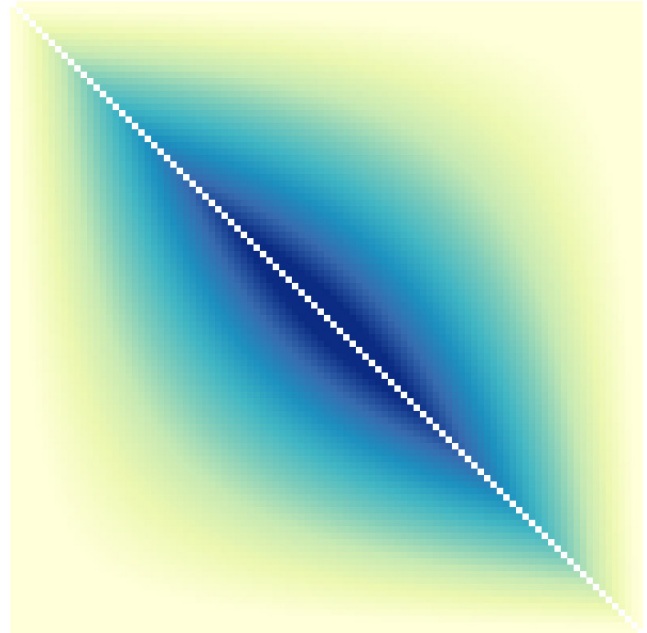


Figure 2: The Activity Graph of Constraints of the Sort Problem.

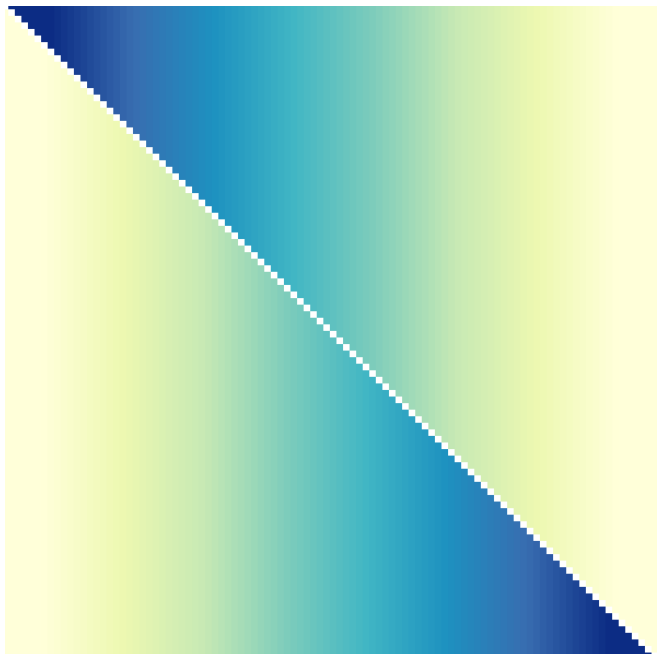


Figure 3: The Activity Graph of Variables of the Sort Problem.

Reductions are sufficient to lead to a solution. Sorting the variables is, therefore, a sheer (and long) propagation problem. On Figure 2, we display the graph of one hundred constraints involved in the resolution of the problem without variables. The edge activity is computed by omitting the intermediate variable when a constraint triggers another constraint. On Figure 3, we represent the graph of variables of the same problem, obtained conversely by omitting the constraints. These graphs provide much information about what happened during propagation. In fact, a strong interaction can be observed between close constraints (in the posting sequence), but we also see that all constraints are related to each other. Moreover, we can see that the farther the constraints from one another in the posting sequence, the less they interact (but they still have some level of interaction). Of course, all that has been said about constraints can be transposed about variables. Moreover, the middle constraints seem to interact more often than peripheral constraints: see the darker part in the center of the constraint graph. What do we learn? Those observations on the constraint graph help us understand why the propagation takes so long in such a problem. All constraints need to be propagated several times as they all have both a short-term and a long-term impact during the resolution process. Moreover, we illustrate here the interest of the visualization of the explanation network with regard to classical analysis of the constraint graph which is not able to provide any information. However, one question still needs to be addressed: why do middle constraints seem to be so active? For that, we need to observe the dynamics of propagation.

One of the key features of our system is that it makes it possible to visualize the dynamics of search and propagation. Animations are not easy to render on a printed paper. However, Figure 4 shows seven time slices from the constraint graph of our sorting problem. The visualization of the dynamics within the propagation phase of this particular problem reveals the existence of two clearly identified activity waves. Actually, this illustrates the way our solver works. The first constraint to be propagated is the last one in the posting sequence (constraints are not propagated upon posting but all at once) inducing a chain of modifications of the upper bounds of the variables through the awakening of all other constraints. Then, the other constraints are propagated, each modifying the lower bound of one variable. The two phases displayed in Figure 4 illustrate those two modification waves. Our animation is invaluable to illustrate such an intimate behavior of the solver as it provides information about the way propagation works. Moreover, this information explains why middle constraints appear to have more activity than other constraints. This is due to the fact that the propagation of those constraints impacts both the upper and lower bounds of the variables whereas the propagation of peripheral constraints impacts only one bound of the variables.

The added value of the visualization tools in this case can be summed up as follows:

1. Using matrix-based representations is essential when handling very dense graphs.
2. The stability of the layout of matrices and the lack of occlusions makes it possible to monitor dynamic graphs, and grasp the changes that occur therein in a pre-attentive fashion.
3. Visualization tools can help illustrate notions such as propagation more pedagogically, as in this example, thanks to proper animation. From this standpoint, visualization tools may be useful to some extent in teaching environments.

4 Visualizing Explanations

We worked with an “explained solver” instrumented to produce a trace. Thus, we could collect a fine grained history of the solver execution for postmortem analysis.

4.1 Explanations about Explanations

Explanations (specializing (A)TMS [Doyle 1979]) have been initially introduced to improve backtracking-based algorithms [Ginsberg 1993]. However, they have been recently used for many other purposes [Jussien 2003] including debugging and analysis of the behavior of constraint solvers.

4.1.1 Definition

An explanation contains enough information to justify a decision (throwing a contradiction, reducing a domain, etc.), it is composed of the constraints and the choices made during the search which are sufficient to justify such an inference.

An explanation of an inference (X) consists of a subset of original constraints ($C' \subset C$) and a set of instantiation constraints (choices made during the search: d_1, d_2, \dots, d_k) such that:

$$C' \wedge d_1 \wedge \dots \wedge d_n \Rightarrow X$$

$C' \wedge d_1 \wedge \dots \wedge d_n$ is called an explanation. Roughly speaking, an explanation is a set of constraints that explain decisions taken by the solver, such as the withdrawal of a value from the domain of a variable.

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 an explanation, the more useful it is.

4.1.2 The explanation network

Usually, explanations may be used for user interaction [Freuder et al. 2001; Ouis et al. 2003], dynamic constraint handling [Debruyne et al. 2003], or even improvement of search algorithms [Ginsberg 1993; Jussien et al. 2000, Jussien and Lhomme 2003]. However, explanations, as we defined them, provide insightful information about the constraint solver dynamics.

Considering explanations or, more precisely, the pair of networks they induce between constraints and between variables, proves to be fruitful for understanding the behavior of the constraint solver on a given problem. Indeed, constraints appearing in an

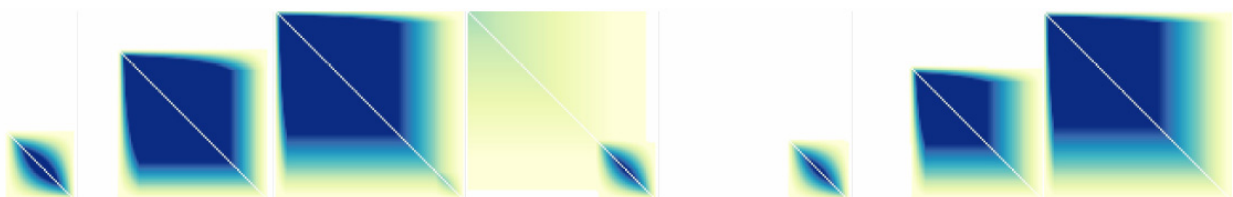


Figure 4: Visualizing the dynamics of propagation in our sorted problem. From left to right: a first slice, in the middle of the first phase, at the end of the first phase, changing phases, at the beginning of the first phase, in the middle of the second phase, and the last slice.

explanation are cooperating (event if they do not share any variable) to remove some values and are, therefore, related. Similarly, variables related to constraints that appear in an explanation are related during the resolution.

In this paper, we show that it is interesting to visualize, and analyze the relationships (and their dynamics) between constraints and between variables. Note that we are interested in the use of the explanation network as a representation of the solver dynamics. Therefore, a priori [Amilhastre et al. 2002] or a posteriori [Junker 2001] explanation computation techniques would be of no use here.

Recently, resolution algorithms evolve towards adaptive schemas that try and integrate the particularities of a given instance in their behavior. In all cases, the detection of the underlying structure of problems may significantly accelerate the search. In the present work, visualization tools help in classifying the information susceptible of being integrated in such an adaptive behavior, and in revealing the underlying structure of problems as well. In the following cases of study, we are able to exhibit various structures. In the first case, the search activity creates dynamic structures that occlude the intrinsic structure of the problem, and drives the solver into numerous costly enumerations. In the second case, very specific static structures are exhibited with our visualization tools; these structures may help define an ad hoc resolution strategy. In the last case, we show how the visualization can uncover activity patterns during constraint propagation in very dense graphs, and help “feel” the behavior of the solver.

4.2 Investigations on Problem Structure

Due to the combinatorial nature of the problems tackled by CP, the efficiency of any given heuristic is very much related to its ability to discard those areas of the search tree containing no solutions. The quicker the sterile branches are pruned, the faster the solver can concentrate on branches worthy of investigation. In this context, understanding the structure of the problem at hand and identifying its most influential variables may dramatically reduce the resolution time as we shall see in the sequel.

4.2.1 Hypotheses

Recent works [Refalo 2004] proved that the propagation phase provides valuable information that may contribute towards crafting generic search heuristics. One such information is the average reduction of the search space due to a given variable. In the present experiment, we are interested in understanding:

- the direct impact of a variable on the reduction of the search space,
- the indirect impact of a variable, i.e., the reduction of the search space caused by the variable even a long time after its instantiation occurs,
- the areas of the problem which are influenced by a variable,
- the relationships between variables, if any.

Such information relies on the explanations computed by the solver during the resolution process, as they account for the logic underlying the chain of consecutive inferences made by the solver during the propagation. The influence of a variable v_i on another variable v_j may be seen as the number of times where the instantiation of v_i justifies the reduction of v_j . The influence of a

variable v_i on the entire problem can then be seen as the sum of influences of v_i on every variable of the problem.

In the following examples, we shall see how visualization tools help understand the structure of the problem at hand, and determine the role played by its influential variables. Such analyses may inspire some improvements on the search strategy of the solver.

4.2.2 Experiment with random structured problems

4.2.2.1 Study of a hard/peculiar instance

In this first experiment, we crafted a binary random problem where we induced a structure by increasing the hardness of some constraints. This results in the emergence of three sets of 10 variables with strong internal relations – i.e. relations between the variables within each set – and loose external relations – i.e. relation with variables in other sets. In this perspective, the variables of a given set are strongly interrelated, and are expected to be involved in the same inferences made by the solver.

This problem is particularly interesting because it proved to be harder than we could initially expect on classical heuristics such as MinDom (the variable with the current domain of minimum size is selected to branch) as well as random instantiation heuristics. This problem raised the following questions.

1. Is it possible to detect the structure induced on the problem, i.e., the aforementioned sets of variables?
2. Can we account for the poor performance of the MinDom heuristic? Is it related to this peculiar instance or is it due to the heuristic per se?
3. How can the information collected from the propagation phase help answer these questions?

4.2.2.2 Results

In Figure 5, we display the graph of 30 variables involved in the problem using a matrix-based representation. The variables of the problem are represented by the rows and columns of the matrix. The cell at the intersection of column i and row j corresponds to the impact of the variable v_i on the variable v_j . The stronger the impact, the heavier the edge, and the darker the cell. The matrix is ordered according to the order of declaration of the variables by the solver, which happens to be a relevant order with regard to CP graphs. A singleton consistency algorithm is applied (every value of every variable is propagated) to ensure that the impact of variables is initialized homogeneously. Although the graph is almost entirely connected, the matrix-based visualization makes it possible to see very clearly the structure of the problem, i.e., the three sets of variables having strong internal links.

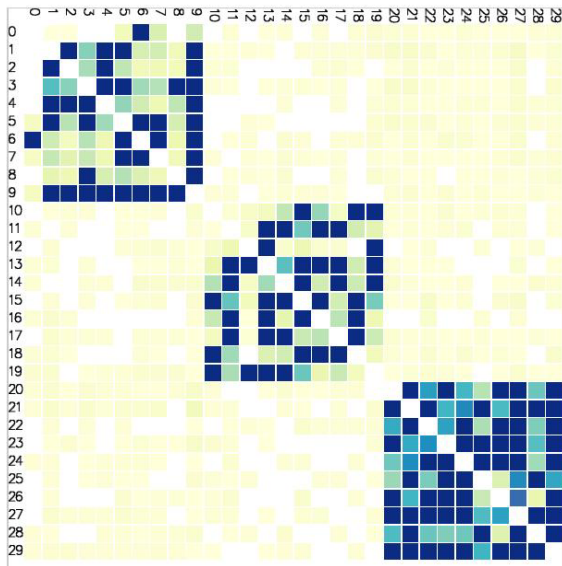


Figure 5: The representation of the graph of variables at the end of the initialization phase.

Figure 6 shows the graph of variables after running the MinDom heuristic. We interrupted the execution after two minutes. Compared to Figure 5, we note that the structure of the problem is no longer visible. The blue area at the bottom left corner means that the variables in the first two sets have a strong influence on the variables belonging to the third set. This can be accounted for by the fact that poor decisions taken early concerning the variables of the first sets lead the solver into numerous try-and-fail steps on the variables of the third set.

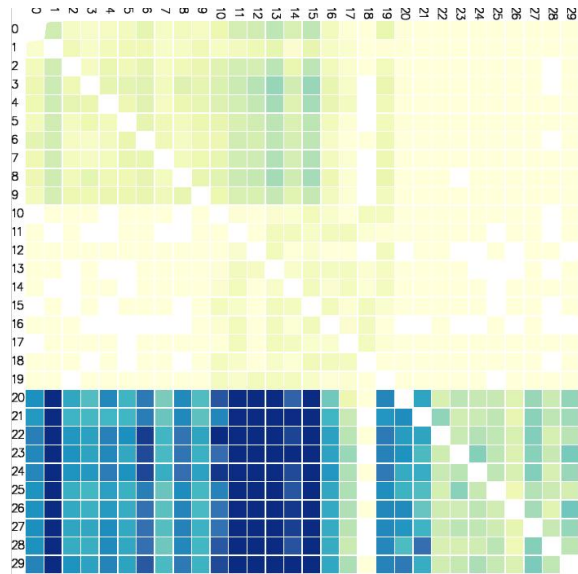


Figure 6: The graph of variables after two minutes of computation using the MinDom heuristic.

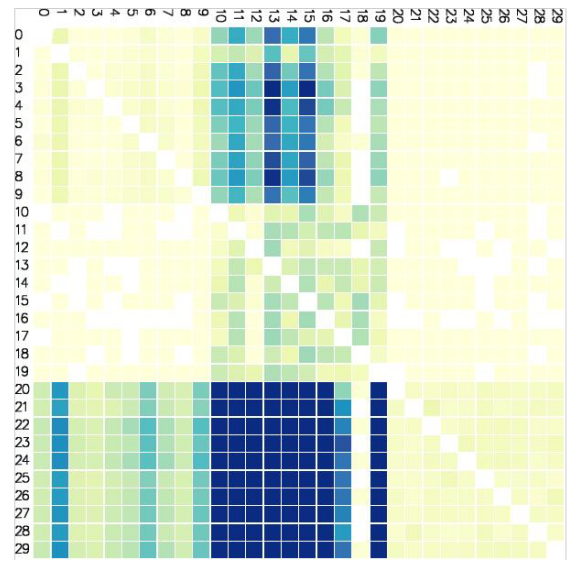


Figure 7: A normalized representation of the graph of variables using the MinDom heuristic.

On Figure 7, we display a normalized representation of the same graph where the influence of a decision taken by the solver is divided by the number of times this decision occurred during the resolution. By doing so, we aim at refining the previous analysis by distinguishing two types of decisions: those having a great influence and are, therefore, repeated frequently, and those having a great influence because the solver is stuck in some inconsistent branch of the search tree. In this way, we can isolate early poor decisions that seem to involve the second set of variables by taking into account the age of decisions as seen on Figure 8.

Figure 8 reflects the activity within the graph of variables after two minutes of computation using the MinDom heuristic. In the assessment of activity, the effect of old decisions is gradually discarded. As expected, it appears that the solver keeps going back and forth between the first and the third set of variables, with very negligible involvement of the second set. This must be related to early poor decisions taken on the variables of the second set. It is also worthwhile to note that the structure of the problem is made visible once more.

In order to further confirm this interpretation, we adapted the search heuristic so that it takes into account a measure of influence of variables during the resolution, and revokes immediately decisions whose influence increase outstandingly. As a consequence, the problem was solved almost instantaneously, and the structure of the problem is exhibited as in Figure 9.

4.2.3 Conclusion

The gist of the previous results is that poor decisions taken early on the variables may trap the solver in a combinatorial, expensive enumeration of values in the variable domains of the problem. An early detection of the critical variables – those having a great influence on the problem – allows the constraint programmer to adapt and direct his heuristic so that it avoids falling into time-consuming enumerations.

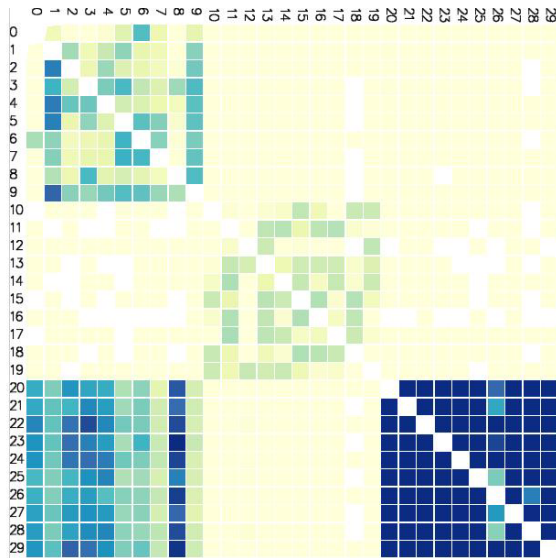


Figure 8: The graph of variables taking into account the age of decisions.

The visualization of the graph of variables plays an essential role in helping the programmer see the underlying structure of the problem, detect the critical variables, and ultimately adapt his heuristics to avoid poor decisions. To the best of our knowledge, the matrix-based representation of graph is definitely the only metaphor that makes the display of a very dense graph possible without overlapping. Moreover, color-coding the edges of such graphs reflects their activity, and makes it possible to grasp the distribution of activity throughout the graph in a pre-attentive fashion. In contrast, the visual clutter induced in node-link layouts makes them of no practical use for monitoring purposes.

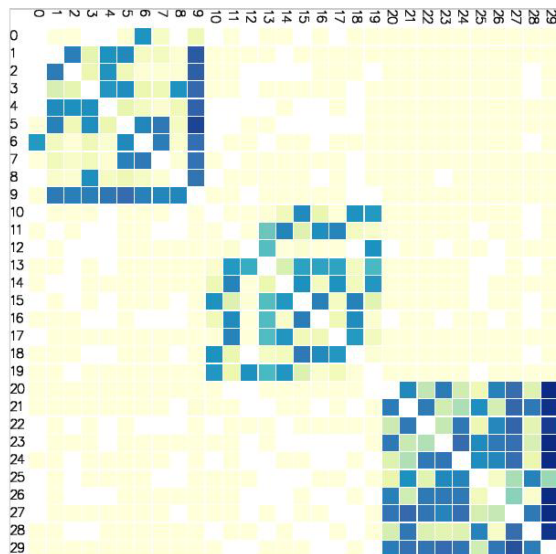


Figure 9: The graph of variables using an adaptive search heuristic.

4.3 Investigations on Incremental Resolution of Problems

In some problems, the inherent structure of a given instance may provide valuable clues in view of accelerating the search strategy. In the previous case, the structure of the problem was occulted, and could only be revealed through the propagation phase. In the

present case, we are interested in the use of visualization tools as a means of discovering the underlying structure of hard academic problems. The gained insights can be used to enhance the search strategy.

4.3.1 Graph Coloring Problems

We have examined hard graph coloring problems included in the DIMACS 2002 Challenge¹. Graph coloring problems consist in using the minimal number of colors so that a color can be attributed to every vertex in the graph, and every two adjacent vertices should have different colors. The presence of cliques in the graph provides an obvious lower bound of the number of colors, and helps accelerate the search. Hence, the detection of maximal cliques is integrated in the best coloring algorithms. Some hard graphs are called “triangle-free” because their maximal cliques contain no more than two vertices. We have studied two categories of triangle-free graphs: Mycielski graphs (graphs based on Mycielski transformation) and k -insertion graphs.

4.3.1.1 Mycielski graphs

Figure 10 and Figure 11 reveal the static structure of Mycielski 4 and Mycielski 5 instances obtained as described in the previous section, after the propagation phase in the root node and after the instantiation of the most connected variables (which reduced the symmetries in the problem). The use of a matrix-based representation reveals its peculiar structure, and suggests the following directions for its resolution.

- Examining the relations between the variables of Mycielski 4, we realize that variables 11 to 21 are not related (the bottom right quadrant is blank). Consequently, the coloring of variables 1 to 10 can be immediately extended to (or proved inconsistent with) variables 11 to 21. Such a configuration suggests the decomposition of the problem into a master problem comprising variables 1 to 10 with a combinatory role, and a secondary problem that includes the remaining variables.
- Examining Mycielski 5, we can see a recurrent structure common to all instances of the problem such that the instance of size n can be defined from the instance of size $n - 1$. This structure suggests an incremental resolution strategy (kind of Russian dolls search) where the resolution of a subset of the problem may help in setting a new bound that would accelerate the resolution of the following problem. Thus, solving a series of imbricated problems may prove far more effective than tackling one big problem from scratch.

¹ <http://mat.gsia.cmu.edu/COLORING02/>

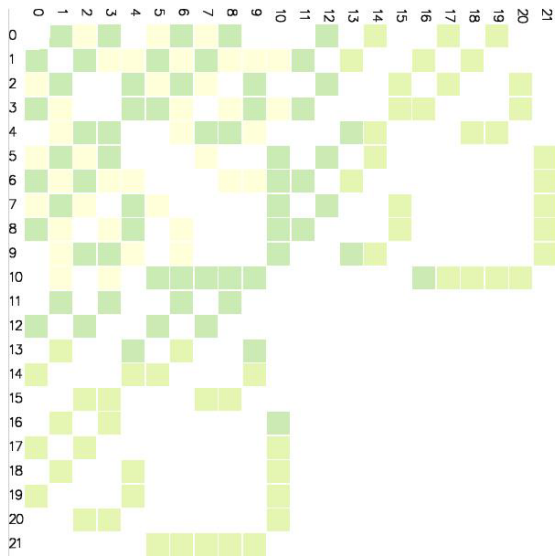


Figure 10: The graph of variables of Mycielski 4.

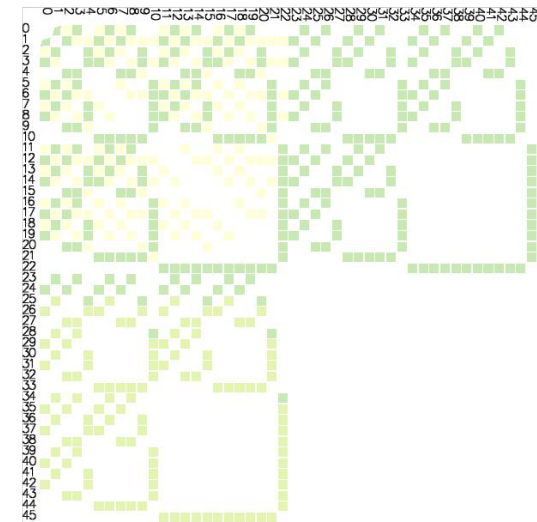


Figure 11: The graph of variables of Mycielski 5.

4.3.1.2 The Contribution of Visualization Tools

In the present experiment, the matrix-based representation provides a clear overview of the graphs at hand. Also, it effectively displays the symmetries present in the problem. Of course, this can be achieved by advanced node-link layout packages when dealing with sparse graphs. An interesting advantage of the matrix-based metaphor lies in its “duality”, that is, it displays the “missing links” as well as the established ones. One glance allows the user to see that variables 11 to 21 are not connected at all, whereas this may be very difficult to detect with a node-link representation.

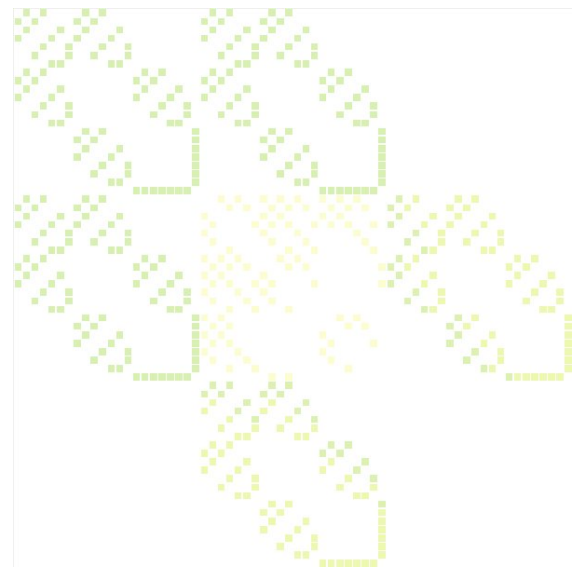


Figure 12: The graph of variables of the 1-insertion problem, instance of size 4.

4.3.1.3 Future developments

In further developments of this case, we may craft a special strategy that benefits from the previous observations, and monitor the way the activity spreads across the graph. We can also tackle other “triangle-free” graphs having a similar incremental nature as the k -insertion problem represented in Figure 12. This is a coloring problem consisting in the insertion of k vertices into a Mycielski graph without changing its density. As we said earlier, the user is tempted to begin with the resolution of the elementary pattern, and subsequently use the information collected during this stage to accelerate the resolution of the rest of the problem.

5 Conclusion

In this article, we have described a novel use of matrix-based graph representations to visualize dynamic activity graphs generated by constraint-based programs. We have shown how this representation can be used to understand the dynamic behavior of constraint programs. We have applied the visualization to several kinds of graphs such as constraint graphs (constraint-variable, variable-variable, constraint-constraint) and explanation graphs.

These visualization tools have been very effective at opening the black box of constraint-based programs. We have already gained novel insights into the actual behavior of complex solver strategies on hard problems, and we are currently exploring the full potential of our tools. The variety of the use cases discussed in this work suggests strongly that our tools are extendible to many tasks that occur in the life-cycle of constraint-based programs. A controlled user experiment is yet to be designed and carried out in order to measure the effectiveness of such tools against different user populations e.g. students, confirmed constraint programmers, and expert users, in connection with various tasks or requirements. By all means, such an evaluation is a very challenging endeavor and must be carried out on a limited set of tasks and a well identified population for it to be conclusive.

We are currently applying the visualizations for pedagogical purposes, and are investigating their use to further improve

heuristics conceived for challenging combinatorial optimization problems.

Acknowledgments

This work has partly been funded by the French RNTL project OADymPPaC.

More material is available at <http://tra4cp.sourceforge.net>.

References

- Abello, J. and Ham, F. v. *Matrix Zoom: A Visual Interface to Semi-External Graphs*. In Proceedings of the 10th IEEE Symposium on Information Visualization (InfoVis'04), Austin, TX, Oct 2004. IEEE Press. pp. 183-190.
- Amilhastre, J., Fargier, H., and Marquis, P. 2002. *Consistency restoration and explanations in dynamic cps - application to configuration*. Artificial Intelligence 135(2002):199-234.
- Becker, R.A., Eick, S.G. and Wills, G.J. (1995) *Visualizing network data*. IEEE Transaction on Visualizations and Graphics, 1 (1). 16-28.
- Bertin, J. 1983. *Semiology of Graphics: Diagrams, Networks, Maps*. University of Wisconsin Press.
- Brayshaw, M. & Eisenstadt, M. (1991). *A Practical Tracer for Prolog*. International Journal of Man-Machine Studies, 42, 597-631.
- Byrd, L. 1980. *Understanding the control flow of Prolog programs*. Proceedings of the Logic Programming Workshop. Debrecen, Hungary.
- Card, S.; Mackinlay, J.; and Shneiderman, B. 1999. *Readings in Information Visualization: Using Vision to Think*. San Francisco, USA: Morgan Kaufmann. chapter Dynamic Queries, 235-261.
- Carpendale, M. S. T., and Montagnese, C. 2001. *A framework for unifying presentation space*. In Proceedings of the 14th annual ACM symposium on User interface software and technology (UIST'01), 61-70.
- Debruyne, R.; Ferrand, G.; Jussien, N.; Lesaint, W.; Ouis, S., and Tessier, A. 2003. *Correctness of constraint retraction algorithms*. In FLAIRS'03: Sixteenth international Florida Artificial Intelligence Research Society conference, 172-176. St. Augustine, Florida, USA: AAAI press.
- Doyle, J. 1979. *A truth maintenance system*. Artificial Intelligence 12:231-272.
- Eisenstadt, M. (1984). *A Powerful Prolog Trace Package*. Sixth European Conference on Artificial Intelligence, Pisa, Italy.
- Eisenstadt, M. & Brayshaw, M. (1990). *A fine grained account of Prolog execution for teaching and debugging*. Instructional Science, 19(4/5), 407-436.
- Eisenstadt, M., and Brayshaw, M. *The truth about Prolog execution*. In J.Stasko, J.Domingue, M.H.Brown, and B.A.Price (Eds.) *Software visualization: programming as a multimedia experience*. Cambridge, MA: MIT Press, 1998.
- Fages, F. 2002. *CLPGUI: a generic graphical user interface for constraint logic programming over finite domains*. In ICLP'02 12th Workshop on Logic Programming Environments (WLPE'02).
- Fekete, J.-D., and Plaisant, C. 1999. *Excentric labeling: Dynamic neighborhood labeling for data visualization*. In Proceedings of the International Conference on Human Factors in Computing Systems (CHI 99), 512-519. ACM.
- Fekete, J.-D., and Plaisant, C. 2002. *Interactive Information Visualization of a Million Items*. In Proceedings of IEEE Symposium on Information Visualization 2002 (InfoVis 2002), Boston, USA, Octobre 2002. IEEE Press, pp. 117-124.
- Freuder, E. C., Likitvivanavong, C., and Wallace, R. J. 2001. *Deriving explanations and implications for constraint satisfaction problems*. In Principles and Practice of Constraint Programming (CP 2001), LNCS, 585-589.
- Ghoniem, M., Jussien, N. and Fekete, J.-D., VISEXP: Visualizing Constraint Solver Dynamics Using Explanations. In FLAIRS'04: Seventeenth international Florida Artificial Intelligence Research Society conference, (Miami Beach, FL, May 2004), AAAI press.
- Ghoniem, M., Fekete, J.-D. and Castagliola, P. *A Comparison of the Readability of Graphs Using Node-Link and Matrix-Based Representations*. In Proceedings of the 10th IEEE Symposium on Information Visualization (InfoVis'04), Austin, TX, Oct 2004. IEEE Press. pp. 17-24.
- Ginsberg, M. 1993. *Dynamic backtracking*. Journal of Artificial Intelligence Research 1:25-46.
- Ham, F.v., *Using Multilevel Call Matrices in Large Software Projects*. In Proc. IEEE Symp. Information Visualization 2003, (Seattle, WA, USA, 2003), IEEE Press, 227-232.
- Junker, U. 2001. *QUICKXPLAIN: Conflict detection for arbitrary constraint propagation algorithms*. In IJCAI'01 Workshop on Modelling and Solving problems with constraints.
- Jussien, N., Debruyne, R., and Boizumault, P. 2000. *Maintaining arc-consistency within dynamic backtracking*. In Principles and Practice of Constraint Programming (CP 2000), LNCS 1894, 249-261. Singapore: Springer-Verlag.
- Jussien, N., and Lhomme, O. 2002. *Local search with constraint propagation and conflict-based heuristics*. Artificial Intelligence 139(1):21-45.
- Jussien, N. 2003. *The versatility of using explanations within constraint programming*. Research Report 03-04-INFO, Ecole des Mines de Nantes, Nantes, France.
- Lalanne, D. and Pu, P. *Interactive Problem Solving Via Algorithm Visualization*, IEEE Symposium on Information Visualization 2000 (InfoVis 2000), October 9-10, 2000, Salt Lake City, Utah.
- Mulholland, P. (1995). *Prolog without tears: An evaluation of the effectiveness of a non Byrd Box model for students*. 7th Annual Workshop 4-6 January 1995. Collected Papers of the Psychology of Programming Interest Group, University of Edinburgh.
- Ouis, S.; Jussien, N., and Boizumault, P. 2003. *k-relevant explanations for constraint programming*. In FLAIRS'03: Sixteenth international Florida Artificial Intelligence Research Society conference, 192-196. St. Augustine, Florida, USA: AAAI press.
- Stasko, J., Catrambone, R., Guzdial, M. and McDonald, K.. *An evaluation of space-filling information visualizations for depicting hierarchical structures*. International Journal of Human-Computer Studies, 53(5): 663-694, November 1998.
- Taylor, C., du Boulay, B., & Patel, M. (1991). *Outline proposal for a Prolog 'Textual Tree Tracer' (TTT)*. CSRP No. 177, University of Sussex.

Yang, J., Ward, M. O. and Rundensteiner, E. A., *InterRing: An Interactive Tool for Visually Navigating and Manipulating Hierarchical Structures*. In *Proceedings of IEEE Symposium on Information Visualization (Infovis 2002)*, IEEE Computer Soc. Press, Boston, Massachusetts, 28-29 October 2002, pp. 77-84.

Deransart, P., Hermenegildo, M. and J. Małuszyński, editors. *Analysis and Visualization Tools for Constraint Programming*. Number 1870 in LNCS. Springer Verlag, 2000. European Project (1997-2000) <http://discipl.inria.fr>.

Jussien, N. and Barichard, V.. 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.

Laburthe, F.. Choco: implementing a cp kernel. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, Singapore, September 2000.

Refalo, P. Impact-Based Search Strategies for Constraint Programming, *Proceedings of the 10th intern. Conference on the Principles and Practice of Constraint Programming (CP 2004)*, LNCS, Springer-Verlag, 2004, pp. 557-571.