What do Constraint Programming Users Want to See? Exploring the role of Visualisation in Profiling of Models and Search

Sarah Goodwin, Christopher Mears, Tim Dwyer, Maria Garcia de la Banda, Guido Tack and Mark Wallace

Abstract— Constraint programming allows difficult combinatorial problems to be modelled declaratively and solved automatically. Advances in solver technologies over recent years have allowed the successful use of constraint programming in many application areas. However, when a particular solver's search for a solution takes too long, the complexity of the constraint program execution hinders the programmer's ability to *profile* that search and understand how it relates to their model. Therefore, effective tools to support such profiling and allow users of constraint programming technologies to refine their model or experiment with different search parameters are essential. This paper details the first user-centred design process for visual profiling tools in this domain. We report on: our insights and opportunities identified through an on-line questionnaire and a creativity workshop with domain experts carried out to elicit requirements for analytical and visual profiling techniques; our designs and functional prototypes realising such techniques; and case studies demonstrating how these techniques shed light on the behaviour of the solvers in practice.

Index Terms—visual analytics, user-centred design, profiling, constraint programming, tree visualisations

1 Introduction

Combinatorial problems require finding a combination of choices (a *solution*) that satisfies a set of constraints and (optionally) is optimal with respect to some objective function. This type of problem occurs in all aspects of our lives. For example, a familiar case occurs when a flight is delayed and the airline must find a new time, gate and runway slot for that flight and new schedules for subsequent legs of passenger journeys. An optimal solution will also minimise traveller inconvenience and cost to the airline, thus improving the quality and efficiency of their services.

The dream of Constraint Programming (CP) has long been to separate problem definition (*modelling*) from the search for its solution (*solving*) in such a way that programmers only need to model difficult combinatorial problems and have clever, automatic solving systems readily available to then solve the problem unaided [14]. As a result, solving systems (composed of a constraint solver and a search strategy) have now reached a high degree of sophistication. But in doing this, their operation has become exceedingly opaque to humans, who often see them as 'black-boxes' producing unexpected or unpredictable results. This is unfortunate because in practice human involvement is still frequently required; for example: to choose the best solver for the type of problem; modify the model to reduce the search space the solver must explore; or to select the correct search strategy.

Finding the best combination of model, solver and search is therefore a very challenging and iterative process. This is particularly true for real-world problems with large-scale input data. This process is a classic instance of *profiling*, where the workflow typically involves three steps that are iterated:

1. **Observe** the behaviour of the program;
2. **Hypothesise** about why certain unwanted or unsatisfactory behaviour occurs;
3. **Modify** the program to test the hypothesis (by observing a *change in behaviour*).

- *Sarah Goodwin and Tim Dwyer are with the Adaptive Visualisation Lab, Monash University. E-mail: {sarah.goodwin, tim.dwyer}@monash.edu.*
- *Christopher Mears, Maria Garcia de la Banda, Guido Tack and Mark Wallace are optimisation researchers with the Faculty of Information Technology, Monash University. E-mail: {chris.mears, mariagarciadelabanda, guido.tack, mark.wallace}@monash.edu.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx/

In this paper we explore the use of visualisation for profiling to better understand model and search performance. We focus on the needs and priorities of users of CP technology through a user-centred visualisation design approach. We gather requirements within the CP community through an on-line questionnaire and a full-day creativity workshop (see Sec. 3). Our co-located team combines experts in visualisation with experts in CP (see authors). We collaborate to build upon our recent research into search statistics and visual profiling [35]. The results of this study identify that knowledge discovery and insight through visual analytics [20] can significantly benefit the CP community. Our contributions include:

1. The first user-centred visualisation design project in the field;
2. Insights and opportunities identified through our requirements gathering process, which is applicable to other domains;
3. Implementations of prototype designs based on the user requirements elicited;
4. Exploration of three case studies demonstrating insights, knowledge discovery and improvements to the models through the use of our prototype visual profiling techniques;
5. Informed visual design ideas and plans for development to allow for more efficient CP profiling in the future.

2 Context and Related Work

Finding quality solutions to combinatorial problems is remarkably difficult. Modern approaches focus on developing a *model* that describes the problem in terms of parameters, variables, constraints and an objective function. The parameters can later be instantiated with input data describing a particular *instance* of the problem. The aim is then to find an optimal solution for the instance, i.e. an assignment of variables to values that satisfies all constraints in the model and optimises the objective function. To find such a solution, the programmer must select a *solver* to satisfy the constraints in the model and a *search strategy* to explore the search space. The combination of model, input data, solver and search strategy is referred to as the *constraint program*.

Existing profilers (e.g. [4, 6, 7, 10, 11, 28, 32, 36]) allow programmers to observe the behaviour of the program execution. In general, these profilers use standard tree-drawing techniques, which are commonly known by CP users. Some incorporate alternative views. For example, CPViz [36] is an open-source generic visualisation library aiming to enable users to better understand the search and solver process. The visualisations are demonstrated for a number of specific problems; however, on investigation the library of visualisations is limited and, importantly, they do not scale to large instances.

In general, traditional tree representations are often found to be inadequate for visualising large problem instances, which have hun-

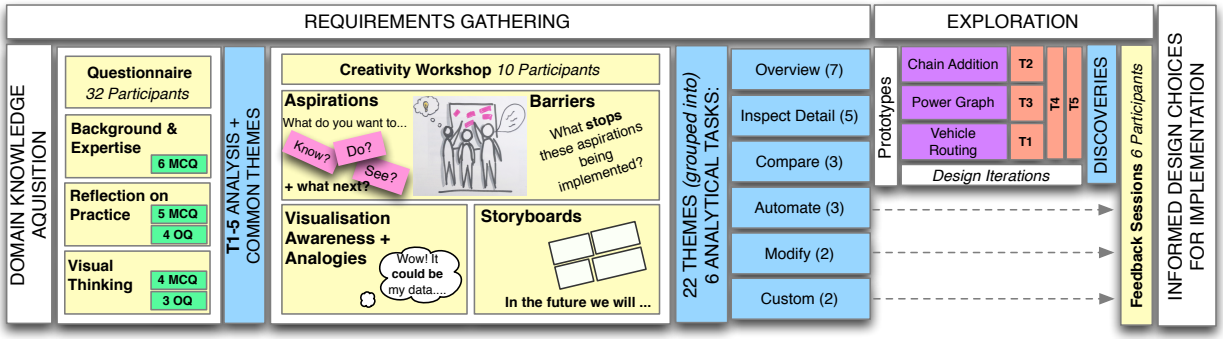


Fig. 1: The requirements gathering and exploration process. Stages of the process in white. Participant activities in yellow with question types: multiple choice (MCQ) or open-ended (OQ). Key findings in blue. Exploration case studies in purple with relevant profiling tasks T1-5 (orange).

dreds or even thousands of variables, values and constraints. The general problem of tree visualisation has received significant attention from visualisation researchers [33], including user experiments with tree visualisation systems [22]. Some of these alternative tree representations—often more applicable to larger instances—have appeared in the CP-related literature. For example, treemaps are mentioned in [36], pixel trees [5] are briefly explored in our previous work [35], and sunbursts are explored in [29] but only for small problems. There has not been however, to the best of our knowledge, a thorough user-centred exploration of tree or alternative visualisation for CP.

Relevant research on analytical and visual tools for monitoring constraint programs includes DiSCiPl [9]. This research identified that beside the search tree, users are also interested in the evolution of variable domains, the activity of variables and constraints, and the interaction between them. Following these recommendations, visualisation research by Ghoniem et al. [15, 16] delves into the details of the dynamics of the solver by exploring the relations between constraints and variables that arise through propagation (the reduction of variable domains according to the constraints, see Sec. 4). Based on the generic trace format GenTra4CP [1], the research uses adjacency matrices for displaying variable co-activity and constraint relationships. Matrices allow for many more variables and constraints than network diagrams but, as with many visualisations, there are scalability issues for larger instances. User-defined time slices allow the user to filter and navigate. Whilst the research demonstrates their potential for insight into the static and dynamic structure of propagation between variables or constraints, these visual options are not available in current software or known to the wider CP community. The visual representation is also relatively unintuitive at first glance, although the use of interaction and animation do aid comprehension.

Despite all the research in the area, there are still only limited visual resources for profiling CP models in practice. The notion of visual profiling is hindered by a small diverse user group across a wide application area using a mixture of software, solvers and outputs. Visualisation options seem to be either so specific as to be useful only for certain problems, or so broad as to be too simple for real-world problems. Visual representations are also confronted with the challenge of presenting exponentially growing data, together with technical limitations relating to screen space and memory usage. The current visual representations do not allow programmers to develop effective hypotheses regarding the program's search behaviour and, to the best of our knowledge, none of the profiling methods proposed involve a user-centred visualisation design approach.

Our research into opening the complex world of constraint programming execution to users follows growing interest in the use of visualisation for aiding the understanding of complex computational algorithms and models (e.g. [27, 34]). Mühlbacher *et al.* [27] identify two different *types of user involvement* for visualising algorithms and models - *Direction of the Information* or the *Entity of Interest*, both are broken down into: 'feedback' or 'control' of the 'execution' or the 'result'. These *types of user involvement* are all highlighted as important for CP profiling during our requirements gathering process. Sedlmair *et al.* [34] present an extensive literature review and framework relating to the role of visualisation in parameter space analysis for simulation modelling. Some of the analytical tasks and navigation strategies described in the framework are relevant, in particular the task of optimising the model parameters for the best output.

In more general programming domains, there has been significant work on understanding how people think and communicate visually about the complex systems that they develop and maintain [8]. This willingness to turn to visual explanation suggests there is also room for improvement with more visual digital tools and techniques to complement the programming experience. For more efficient modelling, it is therefore important to identify what CP users actually need, and to explore which visualisations enable exploration of the program's behaviour and allow programmers to develop useful hypotheses with which to modify their program.

3 Gathering Requirements

In order to gain a broad overview of the visual and non-visual methods currently used to profile constraint programs we ran an on-line questionnaire targetting the global CP community[1]. We analysed the responses and followed this with a one-day workshop with selected participants, who were challenged to think creatively about future visual profiling opportunities. An overview of the process is shown in Fig. 1. We summarise the findings of both the questionnaire and workshop in this section. More context and definitions are described in Sec. 4, prior to the description of our exploration process, prototype designs and case study exploration in Sec. 5.

3.1 Questionnaire

The questionnaire was designed to achieve two main aims.

1. To understand the different tasks currently performed by CP programmers when trying to improve the execution of their programs, and the associated difficulty and impact of these tasks.
2. To discover the different ways in which programmers visualise problems internally or externally, and whether these differ depending on background and expertise.

Following internal piloting, we recruited 32 participants of the 52 invited from the CP community. The questionnaire took approximately 25 minutes to complete and consisted of three sections including a balance of multiple-choice (MCQ) and open questions (OQ), see Fig. 1. MCQ were chosen to provide quantitative values on expertise and current practice, whilst the OQ give us greater insight into the actual information and visual representations currently being used and/or would like to be available. From the OQ responses we defined a number of codes and two researchers—one from CP and another from visualisation—coded all questionnaire responses to group quotes into common topics.

Expertise and Knowledge of Participants Participants were chosen by the authors as experts in the field. Nearly all participants iden-

[1] Affiliates of the Association of Constraint Programming `www.a4cp.org`

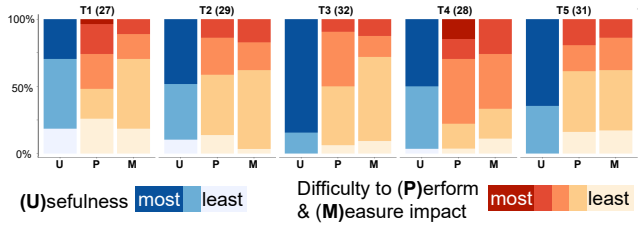


Fig. 2: Ranking of profiling tasks T1–5 (with response numbers).

tified themselves as researchers, whether industrial or academic. In addition, the participants identified themselves as CP teachers (14 responses), practitioners (13), solver/modelling language developers (10), or students (5). The overall majority of participants had a background in computer science, mathematics or engineering. In terms of self-grading expertise, nearly all participants identified themselves as having advanced knowledge of programming and optimisation, and advanced or expert knowledge of modelling languages and CP. Participants had novice-intermediate expertise with visualisation and design.

Reflection on Current Practice Participants were asked 'when the program is executing, how well do you understand what it is doing?' 20 of 32 responded *well* or *very well*. This was higher than expected. Yet when participants were asked to identify 'which part of the process is particularly unclear', responses were varied. 10 identified the propagation and solver phase of the programming experience with one stating *"The actual workings of the global constraints (propagators) are often a blackbox"* and another commenting *"There is no way (that I know) to tap into the execution of the solver and see what's happening."* The relationship between the constraints, the variables and the behaviour of the search strategy were also identified as being unclear.

Prior to the questionnaire, five options were identified as common tasks to undertake when a program is not performing satisfactorily:

T1 Use Alternative Solver
T2 Use Alternative Search Strategy
T3 Alter Model Variables/Domains/Constraints
T4 Observe/Trace the Execution
T5 Simplify the Instance

Participants were asked which tasks they use, how useful they seem, how difficult they are to perform, and how hard it is to measure their impact. The results are shown in Fig. 2. All tasks were used and most were found to be useful. Most notably, **T1** was the least used with 27 responses with mixed ratings on difficult to perform. **T3** was used by all and rated by the majority as *very useful*, and **T4** was on average rated as the most difficult to perform and to measure its impact.

Seven participants also identified other types of task, mainly related to writing specialised code to debug the program, including recording additional statistics, visualising the data and trying to discover areas where a customised constraint is needed. These participants had a higher than average response to understanding what the program was doing when executing, and a higher than average experience of CP. Unsurprisingly, some were solver or modelling language developers.

When asked what extra information they wish they could see during or after the execution process, nearly all respondents suggested *"more statistics"* and many stated that visualising information would be particularly useful, for instance: *"[to] visualize the searching progress will help me to understand what the solver is doing and if it is doing what I expect it to do."*

As CP is used in many different application areas, it is unsurprising that a high level of quotes relate to problem-specific visualisations. We identified 31 quotes from 19 unique participants relating to inherent representations, for example, for a routing problem being able to see the routes change on a map as solutions are found. Other common themes from the OQ were being able to visualise the search space, see the interaction between the constraints and the propagation, trace stages of execution and visualise the solutions.

Visual Thinking For the visual thinking section we divided the programming process into four stages:

1. Formalising the Problem (e.g. representing variables, constraints and data)
2. Choosing the Technique (e.g. type of solver and search)
3. Writing the Program
4. Post-Execution (e.g. solving, debugging or improving performance)

When asked at what point in the process respondents think about the problem visually, all but two chose Stage 1 (with one stating they did not think visually at any stage). Stage 4 was the next most common with 21 of 32 participants, while only 13 and 12 chose Stage 2 and 3, respectively. These findings follow our expectations of (a) programmers often using visualisation to break down a problem (Stage 1) visually via pen and paper or sketching on whiteboards, and of (b) visualisation often being used to present solutions or to debug/improve program performance via statistics (Stage 4).

26 of 32 respondents said they have drawn diagrams before to better understand or to help solve optimisation problems. We further investigated some of these visual representations (by communicating with the respondents) and found that all these related to Stage 1 and Stage 4. The fact that a third of participants do not think about the problem visually at the final stage of the process is interesting and indicates there is scope for better visual aids to allow users to profile their models visually. However, there is not necessarily a consensus regarding the usefulness of this. One participant stated: *"I value the idea of separation of model and search, so that I usually try to not monitor closely the execution process"* and another: *"it might be that the specific execution of the solver is unclear but that is not a problem to me."*

We were also interested in the differences between the internal and the external 'mental models' [23] built for any given problem. We described a mental model to the participants as a small scale abstract internal representation of a real-world phenomenon and emphasised that mental models are different for everyone, whether novices or experts, and that models can adapt as knowledge is acquired.

Two thirds of the participants indicated they were aware of their mental model when tackling a problem. When asked to think about how these could be visualised using common visual representations, matrices or tables were the most popular methods, as were node-link diagrams / graphs, tree diagrams, Gantt charts and timelines.

Summary of Results Many responses confirm the complexity of the profiling process and the limited number of tools available: as a participant stated, the process *"can be so complicated, the problem as well as the solution, and to be sure that it is doing what you intend can be a real nightmare!"*

While few visualisation tools are available to users, they often turn to visual representations for additional information. One user states that *"[for] almost every problem I have solved I produce a visual model of the constraint model"*; another, *"Sometimes thinking about these aspects visually inspires me to develop a new relaxation scheme"*; and a third that *"often I will 'draw' the constraint model, maybe thinking of it as a machine, mechanical, or something that you can 'prod' and it produces a new state"*. One even had reflected on whether visualisation actually helps or not: *"I think often about whether visualisations help me or keep me from building better models, I am not sure what the answer is."*

One participant also reflected on the pros and cons of the visualisation library discussed in the previous section [36]: *"I really liked the features of CPViz that would visualise the variable assignments through search. This way I was able to check if the search strategy did what I expected. However, it's cumbersome to hook up to these visualisation tools, which creates quite a barrier for using them."*

Overall there seems to be a lot of potential for improved feedback, whether statistical or visual. Currently even *"Getting a global feeling for the process without having to study the very detailed output"* is difficult. The users are aware that their variables are related and often ordered or multi-dimensional: *"Variables are often organized in matrices, I often visualize the matrices in my head"*, and that there are often patterns to be found in the search: *"more detailed visualisation of search tree, e.g. highlighting similar patterns in the search tree to find out where the search is duplicating effort."*

Whilst responses show that participants would like improved analytical and visual CP profiling, suggestions were diverse and rather limited in their novelty. From these results it was therefore difficult to determine an area on which to focus our attention. In order to push the boundaries beyond the current thinking and determine the real aspirations for visual profiling, we decided to follow the questionnaire with a *creativity workshop*. These have been shown to be particularly beneficial in provoking creative ideas and identifying new opportunities during requirements gathering [25, 26].

3.2 Creativity Workshop

A full day workshop was designed to think about the profiling problem from a new angle. To select potential workshop candidates, we analysed the survey responses and highlighted particularly interesting, unusual or creative ones. Candidate participants were discussed based on their responses and, to get a mixture of individuals, on the diversity of their application area. All 10 candidates selected accepted our call, many being particularly eager for a chance to look at the problem from a different perspective. Due to travel costs, selection was skewed to local or already visiting participants, though one did fly interstate for the workshop.

Workshop Design The structure of the workshop follows one specifically designed for the gathering of visualisation requirements with experts in the use of *creativity techniques* (see [17]). The workshop has been successful in eliciting requirements for visualisation from energy analysts [17] and neuroscientists [24]. The structure was slightly amended due both to the different area of expertise and upon reflection on the previous workshops: keyword prompts were changed, the visualisation awareness examples updated (and participants given a prompt sheet for notations), and the storyboard activity was given extra time and made optionally individual or paired to reflect the diverse application area. Many factors were considered to provide a creative atmosphere [18], including: room choice, timed breaks between exercises, the engagement of all participants, group discussion, group lunch activity and the mixture of diverging and converging activities. Creative thinking and techniques were described, ground rules identified, laptops put away and the goal of the workshop was outlined as: *To identify data analysis and visualisation opportunities for more effective profiling techniques for (complex) constraint programs*. Finally, as way of introduction, the same *what animal would you be today?* exercise [17] was used to ensure participants began to think creatively.

Aspirations and Barriers To explore aspirations for more effective CP profiling techniques without being hindered by implementation, technology or resources, participants were asked: *Your program does not execute as expected...: What would you like to be able to **do**?, What would you like to **know**?, What do you wish you could **see**?*

122 post-its (some repeated) were produced during the initial individual exercise. A further 36 were produced during part two, where appealing aspirations were discussed by small groups and new ideas were identified, presuming the initial aspiration had been implemented. Post-its were arranged into common themes.

The barriers to implementing these aspirations were subsequently identified in the second activity. In total, 27 barriers were identified. Some examples are: "being focused on the problem not the visual representation", "conflict of interest between academia and industry", "complexity of data"[2] as well as a diverse application area and small user base. Specifically, a lack of purpose built visualisation tools was highlighted and adopting general visualisation tools was seen as being too difficult. For the latter, it was felt that the visualisations these tools produce are inappropriate to the problem or that seemingly promising visuals are always "a mess" for real problems.

Part two of the barrier activity involved removing the barriers [19] and identifying what would be possible to *see*, *know* and *do* if the barrier no longer existed. A further 31 post-its were produced. The removal of the particularly relevant barrier 'visualisation is always a mess' sparked participants' interest; they realised that it was indeed possible to interact with the search by filtering and zooming, and to discover the impact of constraints.

Analogies and Storyboards The visualisation awareness with analogical reasoning activity saw 10 diverse visualisations presented and described by the facilitators. Having already been prepared for analogical thinking with a lunchtime activity (see [17]), participants thought of, and wrote down analogies to their own work. The *Small MultiPiles* system [2] for dynamic brain network analysis was seen as applicable to presenting variable co-activity, comparing model executions or inspecting similar sub-trees. A visualisation of rental-bicycle flow [38] was seen as analogous to local search activity involving a network of constraints, brightness indicating important bottleneck areas or *nogood* (see Sec. 4) activity. The *Time Curves* [3] technique—for visualising high-dimensional data evolving over time—was linked to the interaction of constraint propagation, seeing how solutions change over time, or the progress of the search. *TimeNotes* [37] were identified as good to show repetitions in the search over time. *GapMinder* [30] prompted discussion regarding the replaying and animating of the search process to identify and highlight key aspects of the search and meaningful features. Finally, an application to profile energy consumption by appliance—*HorizonGraphs* [17]—inspired discussion on ways to interact with model parameters and constraints and see how these changes affect the overall search.

The final activity involved participants producing storyboards to link and loosely prioritise the ideas from the day. Coloured pens and A3 storyboards were provided with visual stimuli all around the room relating to the day's activities. Everyone then presented their story to the group prior to wrapping up the workshop. Common ideas included: being able to interact, explore and see what happens during the search; see a progress bar; debug easily; see the effect of constraints; see links between constraints and variables; identify conflicting constraints; see the propagation graph; identify bottlenecks in the search; compare alternative algorithms and models; extract meaning from solutions; and be able to use visualisation to improve the explain of models and how they work.

Post-Workshop Analysis Post-workshop analysis involved grouping aspirations and ideas into topics, themes and tasks (as in [17]). 22 core themes were identified and grouped into six common analytical tasks (see Fig. 3). Notably, the importance of these six groups differed with our requirements gathering methods. We compared the workshop outcomes to the quotes from the questionnaire (OQ) (*What extra information do you wish you could see during or after the execution and how would this help you?*), recoded using these 22 Themes / 6 Groups. The groups found as most important during the questionnaire —*inspect detail* and *overview*—became less dominant during the workshop. While these were still important in the workshop, two new topics emerged: *Automate* (**A16-18**: i.e. automatic detection of bottlenecks, anomalies, redundant constraints; expected errors in the model; or automated suggestions for change) was identified as being desirable to improve the profiling process, as was the ability to *Modify* the model parameters through the visual profiler.

3.3 Summary and Prioritisation

The outcomes of the requirements gathering phase allowed us to gain a thorough and broad understanding of the opportunities and considerations for visual profiling. Referring back to the *types of user involvement* for visualising algorithms and models [27], we observe that all four types (the feedback and control of both the execution and result) emerged during our requirements gathering phase.

The 22 themes (Fig. 3) were subsequently prioritised by the team based on their ease of development (D)—with respect to acquiring the data from the system as well as developing the visualisations—and their impact value (I) to the user. We also assessed our group's previous (P) visualisation work—collapsible trees, pixel trees, similarities of subtrees and merging trees [35]—for how well they met these user requirements. Through this exercise we were able to identify core priorities for exploration in terms of high impact plus low–medium development cost, as well as determine which areas have not yet been explored by our work. Whilst controlling the model parameters (**A19-20**)

[2] A lack of standard outputs from solvers was identified as a key issue.

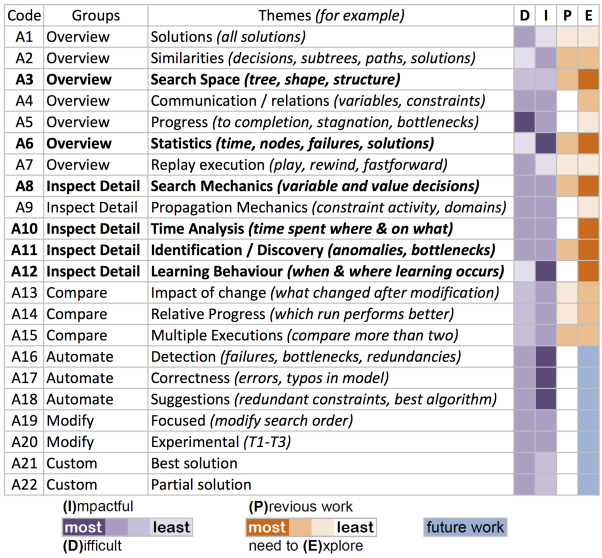


Fig. 3: Identification codes for our 22 themes as discovered through post-workshop analysis. Colours indicate priority by: potential **I**mpact, **D**ifficulty to develop, whether covered by our **P**revious work [35], and priority for current **E**xploration.

is a goal for our continued development, we must first extract statistics from the solvers and explore useful and usable visual representations. Such explorations will potentially allow us to identify search patterns or structure that we can use for automatic detection (**A16**) or suggestions (**A18**). These were therefore left for future work (blue in Fig. 3). We particularly focus on the key requirements (bold in Fig. 3) for our visualisation exploration phase (see Fig. 1), as described in the following sections.

4 Detailed Context

This section outlines the solvers and statistical information used in Sec. 5 to explore the visual profiling of three case studies. We focus on models defined over *finite domain variables*, which can only take a finite number of values, e.g., any value in between 1 and 100, or any value in the set $\{4,5,8,100\}$. We also focus on *propagation based solvers*, which work by propagating changes in the domains of variables, i.e., by eliminating from the domains of all variables any value now known not to be part of any solution. Finally, we focus on tree-based search methods, which proceed by making a series of decisions. Each decision is a constraint (often of the form $x = v$, where x is a variable and v a value in its domain) that corresponds to a node in the tree, and triggers a propagation step where all constraints affected by the changes to variable x start propagating. The search proceeds making decisions until either (1) all variables are assigned and the problem is solved, or (2) the associated solver can detect a failure (at least one variable has no values left in its domain), or else (3) a restart event has occurred. In case (2), the search will usually backtrack to a previous point where a different decision can be made. In case (3) the search will start a new search tree, possibly incorporating new constraints learnt during the previous search.

Importantly, the same problem can be solved using different constraint programs, i.e.: different models, solvers and search strategies. A change in model might arise from using different types of variables (e.g., integer, booleans, or sets), different types of constraints (e.g., in-built, user-defined, or global), or by adding redundant constraints to reduce the search space by increasing propagation. Programmers might also experiment with different solvers, such as SAT solvers (specialised on boolean variables), traditional CP solvers (e.g., *Gecode*[3]), nogood-learning solvers (able to infer reasons or *nogoods* for every failure node and use them to reduce search space), or linear integer solvers (e.g., CPLEX). Finally, programmers can experiment with different search strategies when selecting the particular variable and value being used for each decision by, for example, providing an initial fixed order for variable selection, or instructing the search to select values in a particular order (e.g., from minimum to maximum or vice-versa). The efficiency of the resulting program, measured in terms of how quickly an optimal solution can be found, depends crucially on the combination of model, solver and search strategy used.

4.1 Solvers

Our exploration has focused on the use of two open-source CP solvers—*Gecode* and *Chuffed*[4]—that can be easily instrumented to export new statistics for our analysis. Gecode is a mature traditional propagation solver that has been in development for over ten years. Chuffed is a newer solver that combines the strengths of CP (propagation) with those of SAT (learning). We consider Chuffed in particular, as one of our priorities is to investigate learning behaviour (**A12**).

The two solvers share a set of core features, including support for integer and boolean variables, tree-search, and the use of propagation algorithms for implementing constraints. As a learning solver, Chuffed differs from Gecode's traditional style of CP. In particular, it uses a hybrid of CP and SAT technology as follows. Whenever a CP propagation algorithm reduces the domain of a variable, it must explain the logical causes of the reduction in a way similar to that used by SAT solvers. For example, if the fact that variable x is now less than 3 causes variable y become greater than 10, this knowledge is stored by the solver in a *clause* ($x < 3 \rightarrow y > 10$). In so doing, the reasoning behind the constraint propagation is made explicit and can be used later in a way similar to that of SAT solvers: upon encountering a failure, the chains of reasoning (i.e., the clauses) that led to the failure are examined by the solver, and a new clause (a *nogood*) is derived. This nogood is added as a constraint and will not only prevent the failure from occurring again, but can also cause further propagation during the rest of the search. In addition, the clause analysis may allow the solver to identify past search decisions that are not involved in the failure and can, therefore, be safely backtracked over. In other words, they allow the solver to identify the latest decision (parent tree node) which, if changed, might lead to a solution. Thus, this search-tree node can be used as the backtracking point from which to continue the search exploration. Without conflict analysis, all past search decisions might have caused the failure and, thus, need to be explored. This is the case for traditional CP solvers.

While learning solvers have exhibited dramatically better performance than CP solvers for certain problems (e.g., see results of the MiniZinc Challenge[5]), there are also cases where the opposite holds. Unfortunately, learning solvers are even more complex than traditional CP solvers and, as a result, they are not well understood in practice. Thus, it is not yet clear to the research community under what circumstances learning may be better, or even how to identify when or why a learning solver is performing poorly. Our exploration aims to find visual ways to shed light on this (**A12**).

4.2 Statistics

We use the search tree as a primary object of examination. It describes how the solver traverses the problem's search space (**A3**) by adding search decisions (e.g., constraints of the form $x = v$) and later backtracking over these decisions to try others. If displayed appropriately, it can be used as a record of the program's behaviour (see Sec. 5).

Our system [35] records simple aggregate statistics of the search tree, such as the number of each *type* of node (e.g., failure, branch, solution) and the time taken to complete the search. These are useful as a statistical overview of the execution (**A6**). The system also records detailed information about every node in the tree (relates to **A8**), including: the search decision that led to it; its depth; the size and depth of its descendant subtree; the type of node; information about the learnt nogoods (**A12**); and the timestamp (**A10**).

[3] http://www.gecode.org

[4] https://github.com/geoffchu/chuffed

[5] http://www.minizinc.org/challenge.html

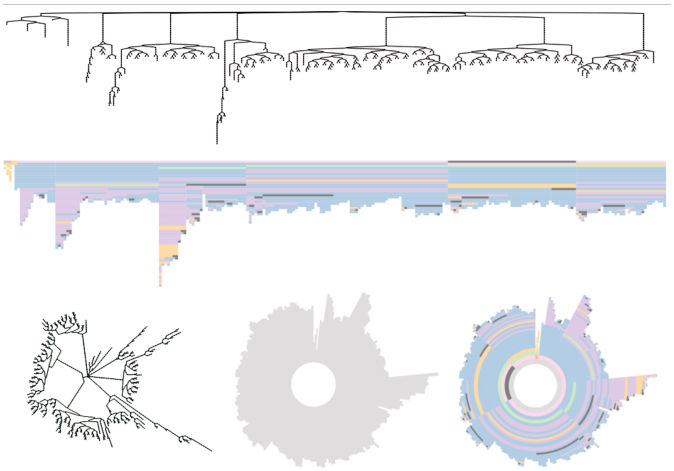


Fig. 4: Alternative tree representations. Colours relate to variables searched on (Case Study 2: Medium Instance), legend shown in Fig. 5.

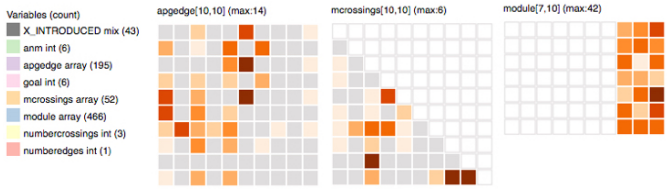


Fig. 5: Distinguishing variables by colour. The number of times variables are searched on is shown in brackets. Arrays of variables are presented as matrices and the colour used to represent the number. Maximum is scaled to the local variable group in this example.

We distinguish a node's depth—the length of the path from the node to the root of the tree—from its *decision level*. Certain search decisions are not truly decisions but are in fact forced consequences of a previous failure. A node's decision level is the number of true decisions made from the root to that node, and is equal to its depth minus the number of nodes whose "search decision" was actually a forced consequence.

For Chuffed, we also record details of every nogood learnt: its length; the variables appearing in it; the number of decisions it allows the search to backtrack over; and the failure nodes for which the nogood helps detect a failure.

5 Case Studies

With a CP programmer we explore three case studies, using the statistics gathered from running the two solvers (as described in Sec. 4) and follow the 3 step iterative profiling process: **observe [OBS]**, **hypothesise [HYP]** and **modify [MOD]** (see Sec. 1). For each case study, we provide an overview of the problem, the model, the reason for the investigation, our observations, hypotheses, visual insights, design iterations and outline any improvements made to the model.

As a starting point, we considered alternative tree representations, including radial trees, icicle plots and sunburst diagrams as shown in Fig. 4 and supplementary material. Depictions of search trees are familiar to the CP community and often used to see and comprehend the solver's exploration of the search space (**A3**). Unlike the traditional tree representation, icicle plots and sunbursts are space filling, which allows for additional information, such as details of the search mechanics (**A8**), to be presented. For example, we can use their space to encode the variable searched on at that node (see Fig. 4).

For the prototype, all colour schemes are sourced from ColorBrewer[6]. In the case studies, all variables defined in the model are allocated a colour from the quantitative *Pastel1* scheme to make them distinguishable but not overpowering. If the variable is in an array, a single colour is given to the group of variables, as shown in Fig. 5. Each array is allocated a separate representation where the variables of the array are presented as a line of squares (1D array) or a matrix of squares (2D array). Higher-dimension arrays were not investigated for the prototype. The colour (sequential *Oranges*) of each variable square represents a statistic about the variable, such as the number of times it is searched on. The maximum across all matrices is either scaled to present the maximum of all variables, which allows for global comparison, or scaled to be relative to the variable group allowing for greater comparison within individual variable groups (as some groups are searched on far more than others), as shown in Fig. 5. White squares indicate variables that are forced not to be searched on, as defined by the model. Grey squares show variables that have not been searched on (i.e. count = 0), but could have been.

All of the visuals created for this exploration were implemented with the D3 visualisation library[7], and they can be toggled for each investigation. All of the views are connected through highlighting and selecting visual elements, with data values shown at mouse-hover, and additional information displayed at the top of the screen.

5.1 Case Study 1: Addition Chain

A minimal addition chain for integer N is the shortest possible sequence of integers, starting with 1 and ending with N, such that each integer in the sequence is the sum of two integers occurring previously in the sequence. For example, for $N = 53$, a possible optimal solution is 1, 2, 4, 8, 16, 32, 48, 52, 53, which has length 9. Note that the shortest chain is known to be not greater than $M = \lceil 2\log_2 N \rceil$ [31]. Thus, M is an upper bound on the length of the shortest chain for a given N. Finding minimal chains is a classical problem in computer science known to be NP-complete with potentially significant applications (e.g. in cryptography [21, pp. 444–446]).

We model the problem using an array x of integer variables representing the sequence, with $x_1 = N$, $x_M = 1$, and an objective function to minimise the position i of the first "2" in the array (so that all variables x_j for j between i and $M-1$ will be set to 2 in the solution, and $i+1$ will be the length of the chain). In addition, the model has two variable arrays, a and b, where a_i and b_i are the indices in x of the values whose sum is x_i; that is, there is a constraint $x_i = x_{a_i} + x_{b_i}$.

The motivation for exploring this problem and model is that while the problem seems well suited to CP, the model performed much worse than a purpose-written logic program. The aim was then to determine why this was the case and improve the performance.

Visual Profiling [OBS] Initial visualisation with a traditional tree, using Chuffed as the solver, reveals a pattern we call a "weeping tree" (see Fig. 6). This *weeping* effect reveals deep lopsided subtrees where the search tries the possible values of a variable in turn, failing immediately after assigning each value. Simply visualising the tree with the colour-encoded icicle plot (x, a, b and solver-*introduced* variables) in combination with brushing shows that the same variable is being searched on repeatedly (see Fig. 6). The alternative trees can be augmented with another view that shows the nodes in a horizontal line, ordered by time (see top of Fig. 6). The multiple views allow us to see where this pattern occurs in time, and spatially in the search tree hierarchy. The interaction and linked views help us quickly identify that the model wastes time trying values one-by-one for each x variable. This behaviour accords with the search strategy in the model, which dictates that the variables in x should be searched in order, trying each value starting from the smallest. **[HYP]** Thus, from these clues we hypothesise that this search strategy is inefficient. We explore different search strategies (**T2**) and instances (**T5**), for growing values of N (see supplementary material).

[MOD] First, we alter the search strategy to split the domains of the x variables rather than assigning each value one-by-one, in the hope that a single failure may be able to eliminate several values at once. **[OBS]** We observe that this improves the search and continue to try other strategies. **[MOD]** In particular, we try to search on a and b together instead of x; that is, rather than deciding a value for

[6] http://colorbrewer2.org

[7] http://d3js.org

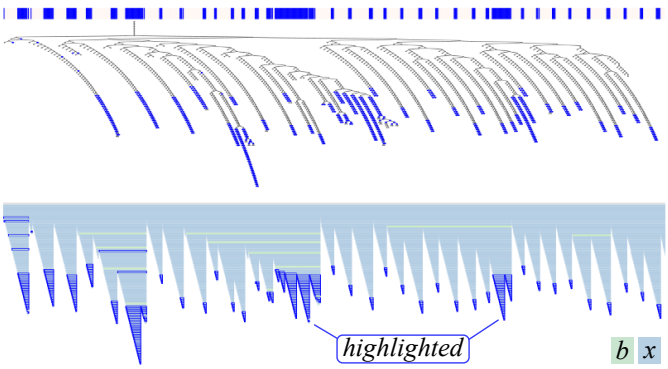


Fig. 6: Highlighting shows one variable from the x array repeatedly searched on, across all views—timeline, tree, coloured icicle plot. The tree shows the "weeping willow" effect.

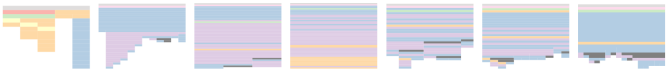
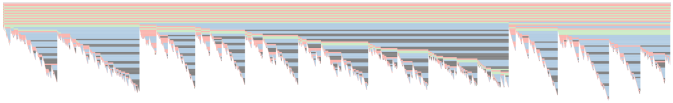
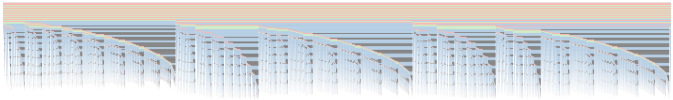


Fig. 7: Subset icicle plots for the first 50 nodes of each restart of the execution in Fig. 4.

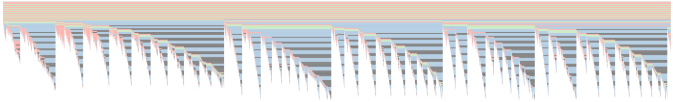
a variable in x, instead decide which previous values are its addends. **[MOD]** We try value-selection orders of minimum-domain-value and domain-splitting. In this case we also test maximum value, since this should lead the search towards shorter sequences (better solutions) earlier. **[OBS]** We see clear differences in the search pattern and observe that searching on a and b, using the maximum value first, is the best-performing strategy:



[MOD] We then increase N to 3979 to see if the patterns remain. **[OBS]** However, the search appears very different from that of the smaller instances, with extremely deep and narrow subtrees whose frequent repetition indicates poor propagation:



Upon investigation, we found that as the variable domains in the problem become larger, Chuffed switches its representation of the variables from using eager literals, where all the literals of the variables are created at the beginning, to lazy literals, where the literals are instead created on-demand. This change has an effect on constraint propagation, as the lazy-literal variables cannot be propagated as strongly in Chuffed. **[MOD,OBS]** By forcing the solver to always use eager literals we see that the search pattern of the earlier instances is preserved:



Design Iterations During our profiling, we found it necessary to highlight a node and show the path of decisions made to reach it (**A8**). We therefore added highlighting and selection to show this path together with textual information of relevant search decisions. While observing the hypothesis generating process, it became apparent that the first few nodes of the search, and subsequent restarts, were useful for hypothesis building, as these can help to indicate what the search does first and whether it is learning well (**A8**). Therefore, we added small multiple icicle plots (e.g., see Fig. 7) showing the first N nodes in each restart.

During the exploration we observed interesting tree structures. The clear *weeping* tree shown in Fig. 6 corresponds to a very spiky icicle plot or sunburst diagram. As we improved the efficiency of the model, the depth of the tree reduced and we saw far fewer spikes on the tree. These findings are potentially useful for our continued investigation of detecting search patterns and automating user feedback (**A16-18**).

5.2 Case Study 2: Power Graph Decomposition

A power graph decomposition is a type of graph compression shown to be useful for reducing clutter when visualising dense graphs, making them much more readable for certain types of path-following analysis tasks [12]. The decomposition aims to reduce the number of edges in a graph by grouping the vertices hierarchically, and introducing "power edges", each compressing as many edges in the original graph as possible. Specifically, a *module* is a set of vertices, a power edge is an edge where one or both endpoints is a module, a power edge from module M to module N represents the edges $\{(u,v) \mid u \in M, v \in N\}$, and a vertex endpoint is treated as a singleton module. The goal is to minimise a weighted combination of the number of edges, the number of modules, and the number of times an edge crosses a module boundary. This turns out to be a difficult optimisation problem [13].

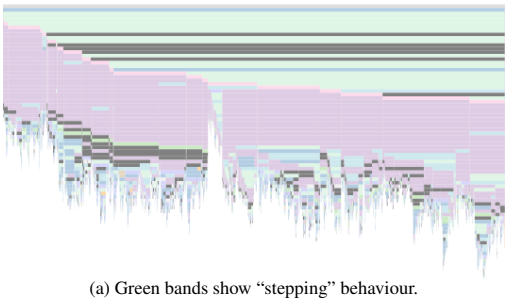
A CP model for this problem was previously presented at IEEE VIS [12] and used to create optimal decompositions for graph visualisations used in a readability study. This model produces optimal solutions for small instances, yet its execution time grows rapidly as the instance size increases and is too slow to solve large instances. The complexity of the model makes it difficult to understand the behaviour of the solver, and previous experimentation with the model using standard profiling techniques—including simple changes that were expected to improve performance—did not yield improvements [13].

Visual Profiling We initially explore three data input instances (**T5**): easy (7 nodes), medium (10 nodes) and difficult (15 nodes). **[OBS]** The prototype designs, particularly the coloured icicle plots and variable counts, reveal that many introduced variables (by the solver, not originally in the model) are being searched on in all three instances. **[HYP]** We expect that restricting the search to branch only on non-introduced variables will offer easier comprehension of the search. **[MOD]** We alter the search strategy to do this (illustrated in Fig. 4, 5 and 7). **[OBS]** Relating our plots to the model is easier; however, restricting the search in this way makes the execution time longer, and gives us no further insight into why the model is performing badly.

[OBS] Since we can see in the visualisations and the execution time that searching on introduced variables is useful, we question what these introduced variables represent. **[MOD]** We alter the model (**T3**) to explicitly name these variables, promoting them from introduced to non-introduced, so that we can visualise where the solver is focusing the search. **[HYP]** We expected the change to be essentially cosmetic, and not affect the behaviour of the solver. **[OBS]** However, we observe that the solving time for the modified model is reduced dramatically; e.g., on the medium instance, the search time reduces from $\sim$ 350 seconds to $\sim$ 50 seconds, and with the further addition of symmetry breaking constraints—which previously had been thought ineffective—reduces to $\sim$ 7 seconds.

By inspecting the early stages of the search—using the small multiple plots—we see that the slight changes to the model have caused the search to make different decisions even at the very beginning. As they affect the clauses that are learnt (**A8,A12**), and which variables are chosen later in the search, these small differences "snowball" such that the search becomes completely different. **[HYP]** This case exposes the fragility of automatic search, and also suggests that simple changes, such as reordering the variables or constraints in the model, might be a useful approach to improve performance.

Design Iterations In all of the instances—and in particular in the difficult instance—we see that the solver is able to learn "facts", i.e., nogoods with a single element like $x \leq 3$. These represent reductions in the domain of a variable which are globally true and, thus, are powerful inferences about the problem. We identified this pattern by seeing that the *goal* variable (the objective) gradually descends in the icicle tree, beginning as the first variable then stepping down whenever a new fact is learnt (see Fig. 8a). This observation identified a problem with search trees presented in this way. The learnt fact is not a search decision and therefore the tree actually looks deeper than it is. We explored altering the representation to draw the alternative trees



(a) Green bands show "stepping" behaviour.

(b) Nodes with same decision level presented as siblings.

Fig. 8: Extract of the power graph decomposition difficult instance.

by decision level rather than depth—that is, to present nodes with the same decision level as siblings, by making their parent node the nearest node on the path to the root with a different decision level. The result offers a more faithful tree representation (see Fig. 8b) where the stair-stepping pattern is removed. This view is more representative of the problem, but did not lead to further insights.

As encountered in the previous case study, larger search trees are hard to navigate. In this case study we required an overview of the full execution, which the small multiple subsets do not provide. We created a simple overview of the search statistics over time (see Fig. 9A) (**A6**). The search nodes are aggregated into blocks representing a fixed percentage of all the search nodes, and drawn as a sequence of rectangles. These are shaded using the sequential *Greys* scheme. Since in the prototype each row represents a different statistic, the colour scale is local to each row.

To give an overview of the progress of the objective during the search (**A5,A14**), we coloured the timeline (shown in Fig. 6) to show how the domain of the objective variable changes over time (see Fig. 9B2). We use the sequential *Reds* scheme, where the darker the colour, the larger the value. While this helps see the effect of propagation on the bounds of the objective throughout the search, for very large searches such detail at the node level is difficult to present and an aggregation of the nodes might be more applicable. For example, we show each restart as a block of nodes in the timeline, coloured by the value of the best solution found previously (see Fig. 9B1). The final rectangle, coloured cream, shows how much of the search was devoted to prove that the last solution is optimal.

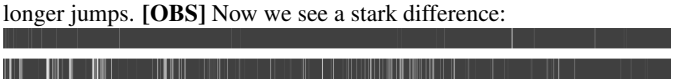
5.3 Case Study 3: Vehicle Routing

Vehicle routing problems aim to find an optimal set of routes for a fleet of vehicles, so that they visit particular client locations. Optimality might be defined, for example, as the shortest total distance travelled by the fleet. Many variations to this family of problems exist, such as using time-windows to restrict when locations may be visited, adding capacities to vehicles and distinguishing between classes of vehicles or locations. Our model is for an industrial application, with capacities on vehicles and multiple visits to each location over several days.

In this case study, we explore the task of *altering the solver* (**T1**). The model performs much better using Gecode than with Chuffed, but it is not clear what causes the difference. **[HYP]** One hypothesis is that the nogoods learnt by Chuffed are simply not useful, and the effort spent computing them is wasteful. To visually explore the learning behaviour (**A12**), we use the information provided by the learnt clauses to relate future and past nodes. When a failed node is activated, we can identify the past clauses used to infer the failure and where the current nogood is used in the future (through highlighting). We also list the most important clauses for quick reference.

Visual Profiling [OBS] For this problem, the search trees for the two solvers are visually similar. Both exhibit a "full search tree" pattern, wherein all possibilities in a subtree seem to be explored. **[OBS]** By examining the most active nogoods (those used more often in detecting failure), and highlighting the nodes where nogoods become active, we can see that most nogoods are not useful. That is, most clauses are not used again later in the search and they do not cause the search to backjump significantly.

[HYP] We hypothesise that this pattern is indicative of poor learning. We compared this problem with the open stacks problem [39], where learning solvers are known to perform well. **[OBS]** One clear difference is that the number of backjumped nodes—those which are not explored because conflict analysis has proved them to be irrelevant—is much higher in the open stacks search (approximately 1.3% of all nodes) than in the vehicle routing search (0.2%). This raw number alone does not necessarily correspond to the amount of search saved, but it prompts us to look at the backjump distance, which is a related measure. **[HYP]** If learning is performing well, we expect to see the search depth jumping back high in the tree throughout the search's lifetime. With this in mind, we iterated the design of the timeline plot to colour the nodes by backjump distance, using lighter shading for longer jumps. **[OBS]** Now we see a stark difference:



the vehicle routing plot (top) has almost no long backjumps, while the open stacks tree (bottom) shows longer backjumps consistently throughout the search. These are comparable with their equivalent tree representations.

Our exploration does not explain *why* the nogoods in the vehicle routing problem are ineffective. **[HYP]** The likely explanation is that the solver is learning the "wrong" explanations for failure: the solver is limited to learning about what is explicitly expressed in the model and cannot make higher leaps of reasoning. For example, the model has a variable to represent the successor of each location – the location that is visited immediately afterwards – but there is no explicit representation of the fact that, say, location A is visited at some point (but not necessarily immediately) before location B. It is possible that changing the fundamental nature of the model, or the learning process of the solver, would allow more useful nogoods to be learnt, but this is beyond the scope of this study.

Whilst we did not improve the model, we did discover properties of the learning behaviour (**A12**). The visualisations enabled us to better understand the effect of nogoods, which may enable the automatic detection of poor learning behaviour during the search (**A16**).

5.4 Design Recommendations and Participant Feedback

We conducted feedback sessions with 6 participants involved in the workshop. In the session we explained the findings from the requirements gathering phase and our prioritisation process. Via annotated videos, we gave a brief explanation of how to interpret the visualisations and presented Case Studies 1 and 2 (see supplementary material). Participants were asked to reflect on their current profiling experience, the themes identified in the workshop, the case studies and the visualisations themselves. The sessions were individual and face-to-face (2 were conducted virtually due to distance) because the participants work in diverse application areas, have different expertise and each have their own workflows. In this section, we reflect on participant feedback and our experience gained while collaborating on the visual profiling tasks.

We discovered that the icicle plot is relatively easy to comprehend in the context of the search, as the length of each bar represents the length of time when that node is active in the search. This allows for more emphasis on, and easier interaction with, the decisions of the search that are higher in the tree. Icicle plots present the notions of tree depth (height) and search pattern (shape) similarly to the traditional tree, which should make them easily comprehended by the CP

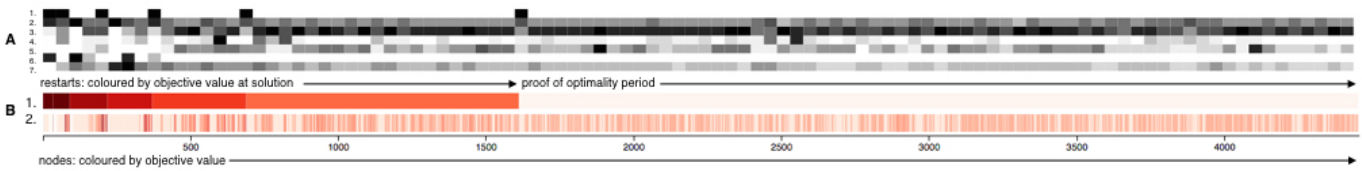


Fig. 9: A. Search statistics aggregated for every 1% of search nodes – 1) number of solutions, 2) % of search nodes that are left branches, 3) % failed nodes, 4) frequency of reuse of learnt clauses, 5) median learnt clause length, 6) mean backjump distance, 7) median tree depth. B. 1) restarts and 2) all nodes – shaded to indicate objective (if applicable to the model) function reduction; the darker the shade, the larger the value.

community. This was confirmed by our feedback sessions. All participants liked the icicle plots and thought they were easy to learn and very intuitive. Structure and patterns in the search were easily identifiable, particularly due to the colour encoding. Their downside is that, like traditional trees, they need a lot of space.

Sunburst plots and radial plots offer a more compact overview. However, these representations make comparison of different regions of the plot more difficult as the crucial notions of execution time and search depth are distorted. We found the icicle plot to be the most useful of the alternative tree representations we examined (see Fig. 4). We rarely used the traditional tree in our exploration because the icicle plot superseded it, but on occasions we referred to the sunbursts for a quick overview of search shape. Some participants really liked the sunburst diagrams, others noted that they had to refer back to the traditional tree for context, as this was the most well known for them.

A negative aspect of icicle plots is that as the search becomes larger, the deepest nodes become impossible to see and interact with. We alleviate this drawback to a degree by using small multiple subsets. These allow detail to be investigated at the cost of removing the overview. They were useful for us to understand the early search decisions but are confusing in the current prototype because, despite the coordinated views, it is difficult to see the relative size of each subset in comparison to the full tree. This will need improvement in future iterations. The small multiples did inspire one participant to think of these as a useful way to display clusters of similar subtree structure within large trees to the user, similar to the *Small MultiPiles* system [2] shown in the workshop. Alternative options for inspecting the detail within the tree include allowing the user to zoom into the detail or inspect detail through the use of a fish-eye lens.

All participants liked the ability to interact with the variables and variable arrays and see where and how often each variable is searched on (**A8**). This was seen as *"really interesting"* and a *"very useful"* feature that is not possible with current profiling tools. The combination of timeline and search tree was useful as it offers both spatial and temporal perspectives. The coordination of these views through interaction was crucial for understanding their relationship. The timeline view also enabled various properties to be shown, including objective function reduction (**A5,A14**), backjump distance and the locations where learnt clauses were used (**A12**).

The aggregations (Fig. 9A) were useful as a quick overview (**A6**) for large searches, but the current subdivision of the nodes does not reflect the structure of the search tree and thus was sometimes frustrating or misleading. A more sophisticated subdivision may make this view more useful in future. Unlike these aggregations, all the alternative tree visualisations have an issue with scalability. In our exploration we occasionally used filtering to reduce the size and better see the search decisions. Nonetheless, we were able to study larger searches than in related work (e.g. [29, 36]), partially due to improvements in technologies and visualisation libraries. One possibility for addressing scalability and maintaining the overview of the full search space (**A3**) is to provide a skeleton of the search tree shape without drawing all the nodes.

When asked how the case studies differ from their current model profiling workflow, participants replied *"very different"*, *"much more detail"* and *"more informative"*, adding that the visual profiling case studies were *"really interesting"*. One participant stated that *"the tool allows for a better understanding of the interaction between the problem and algorithm."* Another realised that visualising the search in this way reveals a new perspective, remarking: *"It is not just a search strategy; it is a different shape. You can't see this in the statistics."*

6 Conclusions and Future Work

This study has highlighted the need for improved visualisation tools to aid the complex CP profiling process. The user-centred method has identified many opportunities where visualisation can have a positive impact. The prototype solutions that combine alternative views, interactions and navigation, show that visualisation can aid in ways that other forms of analysis cannot.

Our extensive requirements gathering process was invaluable for understanding existing profiling practice, and discovering users' aspirations. This gave us motivation and a focus for the initial prototype designs. Our user-centred design process (Fig. 1) is applicable to the wider VAST community as it can be adopted for other domains. Indeed, as discussed in Sec. 3, this is the third domain for which this creativity workshop structure has been used to successfully inform visualisation design.

The questionnaire helped us understand when users mostly use visual thinking: for formalising the problem; representing the solutions; and, occasionally, for debugging and improving the model. It seems many CP users do think visually about their problems and are aware of their mental model, which is promising for the adoption of future visual profiling tools. Whilst a few participants were unsure about the value of visual profiling of solvers to their modelling practices (Sec. 3.1), we are encouraged by our final feedback sessions, which provoked very enthusiastic responses (Sec. 5.4).

Both the workshop participants and the authors envisage that embedding visualisation in CP practice can not only aid users, but also improve the explanation of solutions and models to others, even those outside the domain. This could lead to CP becoming more accessible and therefore widely used, as the frustrations of the current profiling process are reduced.

The positive outcomes of our process have encouraged us to increase our prototype visualisation library, where we continue to test alternative visuals with case studies and users. Such visualisations will include linking the constraint activity into the visualisations (e.g. [15]). We will continue to investigate the themes that emerged from the workshop; for instance, the ability to control/modify the model through the visual interface (**A19-A20**) and automated suggestions (**A18**) or detection of anomalies (**A16**). Through the exploration of the three case studies we identified a number of useful insights, including good and poor learning characteristics and visual patterns of search structure. This will aid our continued work and may lead to incorporating automatic detection or the ability to educate users on search structure characteristics. The process will conclude with improved visual profiling implemented into CP software [35] and tested within the wider community.

Acknowledgments

This research was sponsored by the Australian Research Council grant DP140100058. We thank Maxim Shishmarev for his profiling software architecture, the questionnaire and workshop participants, Bart Demoen for models of the addition chain problem, as well as Sara Jones, Ethan Kerzner, Jason Dykes, Miriah Meyer and Graham Dove for their reflection and feedback on the workshop structure.

REFERENCES

[1] OADymPPaC: Tools for dynamic analysis and debugging of constraint programs. `http://contraintes.inria.fr/OADymPPaC`, 2001-2004.

[2] B. Bach, N. Henry Riche, T. Dwyer, T. Madhyastha, J.-D. Fekete, and T. Grabowski. Small MultiPiles: Piling time to explore temporal patterns in dynamic networks. *Computer Graphics Forum*, 34(3):31–40, 2015.

[3] B. Bach, C. Shi, N. Heulot, T. Madhyastha, T. Grabowski, and P. Dragicevic. Time curves: Folding time to visualize patterns of temporal evolution in data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):559–568, Jan 2016.

[4] A. Bauer, V. Botea, M. Brown, M. Gray, D. Harabor, and J. Slaney. An integrated modelling, debugging, and visualisation environment for G12. In D. Cohen, editor, *Principles and Practice of Constraint Programming – CP 2010: 16th International Conference, CP 2010, St. Andrews, Scotland, September 6-10, 2010. Proceedings*, pages 522–536. Springer Berlin Heidelberg, 2010.

[5] M. Burch, M. Raschke, and D. Weiskopf. Indented pixel tree plots. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, R. Chung, R. Hammoud, M. Hussain, T. Kar-Han, R. Crawfis, D. Thalmann, D. Kao, and L. Avila, editors, *Advances in Visual Computing*, number 6453 in Lecture Notes in Computer Science, pages 338–349. Springer Berlin Heidelberg, 2010.

[6] M. Carro and M. Hermenegildo. Tools for constraint visualisation: The VIFID/TRIFID tool. In Deransart et al. [9], pages 253–272.

[7] M. Carro and M. Hermenegildo. Tools for search-tree visualisation: The APT tool. In P. Deransart, M. V. Hermenegildo, and J. Małuszyński, editors, *Analysis and Visualization Tools for Constraint Programming*, number 1870 in Lecture Notes in Computer Science, pages 237–252. Springer Berlin Heidelberg, 2000.

[8] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko. Let's go to the whiteboard: How and why software developers use drawings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 557–566, New York, NY, USA, 2007. ACM.

[9] P. Deransart, M. V. Hermenegildo, and J. Maluszynski, editors. *Analysis and Visualization Tools for Constraint Programming, Constrain Debugging (DiSCiPl Project)*, London, UK, 2000. Springer-Verlag.

[10] M. Dincbas, P. V. Hentenryck, H. Simonis, A. Aggoun, and A. Herold. The chip system: Constraint handling in prolog. In *Proceedings of the 9th International Conference on Automated Deduction*, pages 774–775, London, UK, 1988. Springer-Verlag.

[11] G. Dooms, P. Van Hentenryck, and L. Michel. Model-driven visualizations of constraint-based local search. In *CP 2007*, volume 4741 of *LNCS*, pages 271–285. Springer, 2007.

[12] T. Dwyer, N. Henry Riche, K. Marriott, and C. Mears. Edge compression techniques for visualization of dense directed graphs. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2596–2605, 2013.

[13] T. Dwyer, C. Mears, K. Morgan, T. Niven, K. Marriott, and M. Wallace. Improved optimal and approximate power graph compression for clearer visualisation of dense graphs. In *2014 IEEE Pacific Visualization Symposium (PacificVis)*, pages 105–112. IEEE, 2014.

[14] E. C. Freuder. In pursuit of the holy grail. *Constraints*, 2(1):57–61, Apr. 1997.

[15] M. Ghoniem, H. Cambazard, J.-D. Fekete, and N. Jussien. Peeking in solver strategies using explanations visualization of dynamic graphs for constraint programming. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, SoftVis '05, pages 27–36. ACM, 2005.

[16] M. Ghoniem, N. Jussien, and J.-D. Fekete. Visexp: visualizing constraint solver dynamics using explanations. In *FLAIRS'04: 17th intl Florida Artificial Intelligence Research Society conf.* AAAI press, 2004.

[17] S. Goodwin, J. Dykes, S. Jones, I. Dillingham, G. Dove, A. Duffy, A. Kachkaev, A. Slingsby, and J. Wood. Creative user-centered visualization design for energy analysts and modelers. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2516–2525, 2013.

[18] S. G. Isaksen, K. J. Lauer, and G. Ekvall. Situational outlook questionnaire: A measure of the climate for creativity and change. *Psychological Reports*, 85(2):665–674, 1999.

[19] S. Jones, P. Lynch, N. Maiden, and S. Lindstaedt. Use and influence of creative ideas and requirements for a work-integrated learning system. In *16th IEEE International Conference on Reqirements Engineering*, pages 289 – 294, Sep 2008.

[20] D. A. Keim, F. Mansmann, J. Schneidewind, and H. Ziegler. Challenges in visual data analysis. In *Tenth International Conference on Information Visualization, 2006. IV 2006*, pages 9–16, Jul 2006.

[21] D. Knuth. *The Art of Computer Programming: Volume 2 - Seminumerical Algorithms*. Addison-Wesley, 1981.

[22] A. Kobsa. User experiments with tree visualization systems. In *IEEE Symposium on Information Visualization, 2004. Infovis*, pages 9–16, 2004.

[23] L. Lambert and D. Walker. Learning and leading theory: A century in the making. In L. Lambert, D. Walker, D. Zimmerman, M. Gardner, and P. Slack, editors, *The constructivist leader*, pages 1–27, New York: USA, 1995. Teachers College Press.

[24] J. S. Lauritzen, C. Sigulinsky, J. R. Anderson, M. Kalloniatis, N. Nelson, D. P. Emrich, C. Rapp, N. McCarthy, E. Kerzner, M. Meyer, B. Jones, and R. E. Marc. The Rod-Cone Crossover Connectome of Mammalian Bipolar Cells. *Journal of Comparative Neurology (Under Review)*, 2016.

[25] N. Maiden, A. Gizikis, and S. Robertson. Provoking creativity: Imagine what your requirements could be like. *IEEE Software*, 21(5):68–75, 2004.

[26] N. Maiden, C. Ncube, and S. Robertson. Can requirements be creative? experiences with an enhanced air space management system. In *29th IEEE International Conference on ICSE*, pages 632–641, 2007.

[27] T. Mühlbacher, H. Piringer, S. Gratzl, M. Sedlmair, and M. Streit. Opening the black box: Strategies for increased user involvement in existing algorithm implementations. *IEEE transactions on visualization and computer graphics*, 20(12):1643–1652, 2014.

[28] M. Paltrinieri. A visual constraint-programming environment. In U. Montanari and F. Rossi, editors, *Principles and Practice of Constraint Programming CP '95*, number 976 in Lecture Notes in Computer Science, pages 499–514. Springer Berlin Heidelberg, Sep 1995.

[29] P. Pu and D. Lalanne. Interactive problem solving via algorithm visualization. In *IEEE Symposium on Information Visualization, InfoVis*, pages 145–153, 2000.

[30] H. Rosling. Hans Rosling's 200 countries, 200 years, 4 minutes - the joy of stats, 2010. [Accessed on: 2016-03-30].

[31] A. Schönhage. A lower bound on the length of addition chains. *Theoretical Computer Science*, 1:1–12, 1975.

[32] C. Schulte. Oz explorer: A visual constraint programming tool. In H. Kuchen and S. D. Swierstra, editors, *Programming Languages: Implementations, Logics, and Programs*, number 1140 in Lecture Notes in Computer Science, pages 477–478. Springer Berlin Heidelberg, 1996.

[33] H. J. Schulz. Treevis.net: A tree visualization reference. *IEEE Computer Graphics and Applications*, 31(6):11–15, Nov 2011.

[34] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, and T. Möller. Visual parameter space analysis: A conceptual framework. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2161–2170, Dec. 2014.

[35] M. Shishmarev, C. Mears, G. Tack, and M. Garcia de la Banda. Visual search tree profiling. *Constraints*, 21(1):77–94, Aug 2015.

[36] H. Simonis, P. Davern, J. Feldman, D. Mehta, L. Quesada, and M. Carlsson. A generic visualization platform for CP. In D. Cohen, editor, *Principles and Practice of Constraint Programming – CP 2010: 16th International Conference, CP 2010, St. Andrews, Scotland, September 6-10, 2010. Proceedings*, pages 460–474. Springer Berlin Heidelberg, 2010.

[37] J. Walker, R. Borgo, and M. W. Jones. TimeNotes: A study on effective chart visualization techniques for time-series data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):549–558, Jan. 2016.

[38] J. Wood. Experiments in bicycle flow animation, 2012. [Accessed on: 2016-03-30].

[39] B. J. Yuen and K. V. Richardson. Establishing the optimality of sequencing heuristics for cutting stock problems. *European Journal of Operational Research*, 84:590–598, 1995.