Understanding How People Approach Constraint Modelling and Solving

Ruth Hoffmann
School of Computer Science, University of St Andrews, UK

Xu Zhu
School of Computer Science, University of St Andrews, UK

Özgür Akgün
School of Computer Science, University of St Andrews, UK

Miguel A. Nacenta
Department of Computer Science, University of Victoria, Canada

Abstract

Research in constraint programming typically focuses on problem solving efficiency. However, the way users conceptualise problems and communicate with constraint programming tools is often sidelined. How humans think about constraint problems can be important for the development of efficient tools that are useful to a broader audience. For example, a system incorporating knowledge on how people think about constraint problems can provide explanations to users and improve the communication between the human and the solver.

We present an initial step towards a better understanding of the human side of the constraint solving process. To our knowledge, this is the first human-centred study addressing how people approach constraint modelling and solving. We observed three sets of ten users each (constraint programmers, computer scientists and non-computer scientists) and analysed how they find solutions for well-known constraint problems. We found regularities offering clues about how to design systems that are more intelligible to humans.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming; Human-centered computing → Empirical studies in interaction design

Keywords and phrases Constraint Modelling, HCI, User Study, Grounded Theory

Digital Object Identifier 10.4230/LIPIcs.CP.2022.13

Supplementary Material *Artefacts, Raw Observation Data, Intercoder Data, Code Explanations*: `https://doi.org/10.5281/zenodo.6519841`

Funding This work is partially funded by NSERC Discovery Grant 2020-04401 (Canada).
Xu Zhu: University of St Andrews and EPSRC grant DTG1796157

1 Introduction

Research in constraint programming (CP) techniques and algorithms during the last few decades have resulted in significant advances in how a large number of problems of practical significance can be addressed. For example, companies routinely use CP to schedule delivery routes [34], educational authorities leverage CP to match medical students to training positions [5] and administrators of high-performance computing clusters use CP for scheduling jobs [15]. CP has been used in commercial settings for several decades [13].

Despite the demonstrated value of CP in these and many other applications, CP remains the domain of a relatively small set of specialists and, arguably, an underappreciated area of computer science. In his seminal paper from 1996 [11], Freuder identifies the potential of CP technology for widespread adoption. In his recent paper from 2018 [12] he recognises the significant progress made by the field and identifies the next challenge as *ease of use*.

We believe that an increase in awareness of the hard-won knowledge in CP and increased accessibility of its techniques and tools (e.g., solvers, constraint modelling languages and modelling tools) can deliver positive improvements for individuals and communities.

For example, consider how a programmer might be able to generate a more efficient solution to a problem if they recognize a subproblem as a constraint satisfaction problem (CSP) for which they can include a CP solver library. Similarly, a person without specialist programming knowledge who identifies a problem as a CSP is more likely to learn how to access existing knowledge and software to find a solution.

One of the barriers to this wider awareness and better understanding of CP by a wider population is that CP tends to be approached from a mathematical, logical or computational point of view and this can be conceptually challenging. To make progress towards this goal we instead approach this problem from a human-centered point of view. In this work we study how people with different levels of expertise with respect to computer programming and CP think about CP and the process of solving constraint problems. Our assumption is that we can leverage a better understanding of how people think about constraint problems and their solutions to create tools such as CP languages or CP applications that are easier to access and use by non-specialists. For example, a programming language that uses core concepts closer to people's default conceptual approach to their problem will, presumably, require less effort to learn and co-opt as a personal tool [16]. Although we do not explicitly focus on improving the teaching of CP technology, this same information can help improve the teaching of constraint programming to novices because it offers a model of how people are likely to think about constraint problems.

We present a qualitative study of 30 participants with different degrees of familiarity with computer science and CP attempting to solve constraint problems. Our analysis shows that non-experts are not aware of the implications of the problem representation and will gravitate to simpler visualisations. Similarly, the visualisation/representation of solving strategies is something that has an impact on the understanding of a solving step. Finally, we found that there is a general belief that constraint problems are solved strategically (almost mathematical), rather than through search. The results from our analysis represent an initial step towards a better understanding of people's conceptual models of constraint problems and solvers. We also discuss ways in which this analysis is relevant for the design of CP languages and tools.

2 Overarching Goal

The overarching goal of this research is to gain knowledge into how people think about constraint problems and their solution. We work under the assumption that gaining this knowledge will help researchers and practitioners in CP in several meaningful ways: 1) A better understanding of how people think about CP can support designers of the languages and interfaces that users of CP come in contact with. For example, an end-user CP language could use terms or constructs that are more likely to be correctly understood by the user. In turn, this could result in more usable, easier to learn or faster to write languages. 2) Communication between the solver software and the end user is likely to benefit from a better understanding of the end-user's expectations. For example, an end-user could benefit from explanations provided by the solver software when performance is likely to mismatch the end-user's expectations (e.g., if an additional constraint results in computation times orders of magnitude larger). In the future, solvers might be able to provide explanations of performance that support the user's modeling activity, but to do this it is important to also

understand how the user thinks about the problem. 3) Learners and future practitioners of CP, including future researchers in the area, can benefit from courses and instructors that are aware of which concepts, techniques and procedures come more naturally to learners of different backgrounds.

3 Methodology

To make progress towards the goal stated in the previous section, we designed a controlled observation experiment in which people attempt to visually model, solve, and program or constraint model constraint problems as described in a previous paper [40].

3.1 Participants

We recruited 30 participants from local universities, 10 belonging to each of the three expertise groups: non-computer scientists (identified as non-CS), computer scientists (CS), and constraint programmers (CP). Non-CS participants (7 female & 3 male, between 19 and 28 years in age), were non-computer scientists with negligible or no programming experience. Of the non-CS participants, 1 was studying towards a degree in a science faculty, 2 in a medicine faculty, and 7 in an art faculty. CS participants (4 female & 6 male, between 19 and 42 years in age) were students in a computer science degree with little or no experience in constraint programming but with experience of computer programming. CP participants (1 female & 9 male, between 21 and 64 years in age) were a mixture of students and staff who have either taken a constraint programming module, taught one or conduct research in that area. Participants received gift vouchers as compensation for their time. The three distinct groups were chosen because the way in which people solve problems is likely to be influenced by their experience and formal education, and they represent a range of levels of familiarity with formal problem specifications.

3.2 Procedure, Tasks and Problem Selection

In the experiment, participants were asked to visually model, solve, and program or model specific constraint problem instances by hand. Each participant provided written consent and was then assigned two problems. Problems were selected from a pool of constraint problems collected from CSPLib [21]. The selected problems are: Word Crypto, Subset Sum, Sudoku, Scheduling, Magic Square, and Knapsack. The exact formulations are in the supplementary materials. For the exact problem selection criteria, see the previous paper [40].

Out of all the participants, Sudoku and Subset Sum were attempted by two participants from each of the expertise groups while the rest were each attempted by 4 participants in each of the groups. For each of the two problems assigned to them, participants had to carry out a visual modelling task as well as a problem solving task, in sequence. Programmers in the CS and CP groups had to perform an additional programming/constraint programming task after. Participants always completed the tasks in order, which precludes bias due to the additional tasks performed by the CS and CP groups. Participants had 14 minutes to complete each task.

The visual modelling task was analysed in the previous paper [40]. For the solving task, participants were given the same problem they had during the modelling task and asked to try to solve the problem by hand, without computer aid. In this paper we focus exclusively on the solving task, although parts of the analysis is influenced by the visual modelling task from the previous paper. In this task, participants were encouraged to talk aloud their

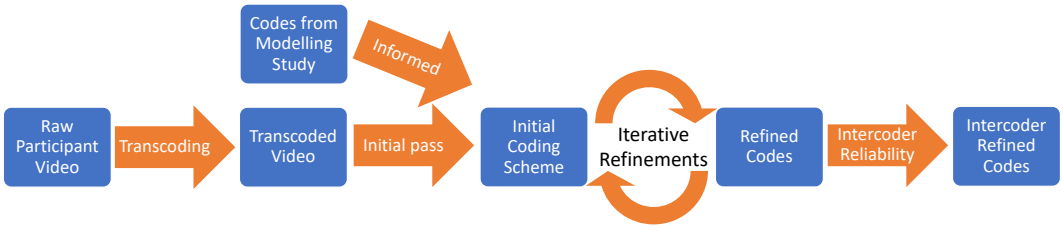

Figure 1 Flowchart of the different steps of the analysis methodology

thought processes and decisions to allow the experimenter to more accurately understand what they were doing [39]. However, participants did not always do this and often needed prompting to describe their thoughts as they were trying to solve the problems. Occasionally, the experimenter would ask for clarifications or offered short reminders of the task. As part of the analysis, we try to infer thought processes from their actions.

We had trouble persuading one of the participants to solve the problems by hand. We have included their data as the participant was persuaded to describe how they would attempt to solve the problem.

3.3 Analysis Methodology

The analysed data consists of two streams of video for each of the participants (resulting in a total of around 32 hours of video per stream), and the paper output from their specifications (scans available in the Supplementary Materials). The two video streams captured two different views, one directly overhead and one pointing diagonally onto the workspace. Snippets from these materials in the remainder of this paper appear marked with the expertise group (non-CS, CS, CP), the number of the participant within that group (from 1 to 10) and whether this was their first or second problem (e.g., CS 7.2).

We followed a bottom-up open coding approach inspired by grounded theory [17]. We analysed both the artefacts from each participant (written notes and solutions) and the video recording of the solving process. An overview of the analysis workflow can be seen in Figure 1. As a preliminary step, we transcoded the two video streams to allow simultaneous viewing of the different camera angles. In a first analysis step, we analysed the artefacts and video produced by creating an affinity diagram of common occurrences and general themes. We then iteratively coded the features that appear within the artefacts and video using Microsoft Excel, refining the code list on each pass. A subset of the codes were initially adopted from the previous analysis from the visual modelling task [40], but were then refined for this analysis. However, most of the final codes (briefly summarised in Subsection 3.4) ended up being specific to this task. The authors met several times during this period to clarify any ambiguities in the codes and refine them.

3.4 A summary of the codes used in the analysis

The codes for annotating the participant behaviour are split into 3 categories. These categories are Visual Elements (VE), Memory related visual elements (MEM), and Solving Approaches (SA). In Table 1 we provide a small selection of codes to preserve space, the full list of codes can be found in the Supplementary Materials.

The VE codes describe what the participants wrote or drew on the paper, for example if they used the representation they were given in the problem statement (VE1.9) or if they crossed out/scribbled out any of the work they had written down at this point (VE1.11).

MEM codes indicate concepts written on the paper as well but they are used to keep track of information, such as keeping track of temporary solutions (MEM3) or having a representation of globally available values (MEM4.1).

Finally, the SA codes are a combination of behavioural observations and what is being written down or said by the participant in relation to their solving steps. The SA codes were split into problem specific strategies and universal strategies which can be found or used in any of the problems. Two examples in the universal strategy category are codes that identify whether a participant restarted solving the problem from scratch without remembering anything from their previous attempt (SA1.2), or if the participants followed a trial and error approach through random partial assignments of values to the variables without any rhyme or reason (SA1.3). For the problem specific strategies, we coded not only the strategies that made sense for the problem (such as SA4.2 where in the Subset Sum problem participants did brute force matching by creating all possible set sizes and then evaluated them), but also strategies that the participants used but which would not lead to a solution (such as SA2.5, where the participants packed the knapsack by the weight or value of the product, rather than the more traditional ratio).

3.5 Coding Validation Analysis

The bulk of the coding was performed by one of the authors. In order to ensure the robustness of the process, the remaining three authors performed an independent coding pass of a subset of 3 of the 60 videos. The inter-coder reliability ratio (the number of agreements divided by the total number of codes from the results of the independent coding pass) and the Cohen Kappa statistic [25] (calculated using the `scikit-learn` python library [32]) measured an inter-coder reliability ratio of 79% and a Cohen Kappa statistic of 0.50.

In an iterative second stage, all authors discussed any ambiguities and discrepancies between codings and resolved some of their differences. Some of these differences resulted from either lack of clarity regarding the scope of a particular code, or lack of familiarity with the video and missing a briefly appearing code. The authors then prepared another code pass based on this discussion. The post-discussion codes have a 92% agreement. The Cohen Kappa statistic post-discussion is 0.80, which is a significant improvement from pre-discussion result. The Supplementary Materials contain the CSV files and python code used to carry out the Kappa calculations.

4 Findings

We describe the regularities that we found in how people approach the constraint problem solving process roughly in chronological solving order. We split the process into three stages.

In the first stage, participants needed to decide how to represent the problem. When presenting the problem statement to them, we have a problem representation as part of the statement. Participants might reuse this representation or they may choose to re-represent the problem in a way that is more intuitive to them before they start operating on it. We discuss this process in the Representation section (Subsection 4.1).

Solving the problem usually involves proposing candidate (partial or complete) solutions and noting this solution on the chosen representation. Participants use several different approaches at this stage (Process and Strategies, Subsection 4.2).

Finally, delivering a solution might involve verifying that the candidate solution is correct and, in the case of solution enumeration or optimisation problems, whether there are other valid (or better) solutions (Solution Verification and Multiple Solutions, Subsection 4.3).

Group	Code	Name	Description
Visual elements	VE1.9	Recreating Instance	Use of the visual element given in the problem description or recreation of the representation used in the previous part of the experiment.
	VE1.11	Crossing out	Line(s) through a single or more characters, or scribbles over a larger area.
Memory elements	MEM3	Temporary solution	Noting and pointing out a temporary solution. This solution could be incorrect or partially valid. A temporary solution is one where there is a majority of variable and value assignments, and is written out as such.
	MEM4.1	Visual representation of globally possible values	Note (or a mark) of possible values that can be assigned to a variable. The assessment of the possibilities was made with all variables or values in consideration.
Solving approaches	SA1.2.1	Restart remembering (no)goods	The participant has decided to give up on their current partial (or full) solution. It is irrelevant whether that solution is correct. They then start the solving from the beginning and remember something they about the variables and values.
	SA1.3	Random Partial Assignment	A strategy of randomly assigning a few variable value pairs.
Problem specific SA	SA2.5	By weight value product	The set of object will be sorted into a highest to lowest weight by value product. A subset of the highest products will be chosen.
	SA4.2	Brute force by set size	The participant creates all sets of a given set size and checks which have the correct sum. This set enumeration can be done explicitly or implicitly.

Table 1 Representative sample codes for the four groups of codes used in the analysis. We have a total of 77 codes in the four groups listed above. The breakdown is 23 in the VE group, 6 in the MEM group, 11 in the SA1 group and 48 in the SA2 group. SA2 is the largest since it contains problem specific codes for the 6 problems. Full list of codes can be found in supplementary materials.

At each step we found similarities and differences between the different groups of participants; we introduce these as appropriate. Note that, due to the design of the study, participants were exposed to different subsets of problems and some regularities belong to specific problems or problem types (see Section 3 for details). To facilitate the interpretation of the results in Section 6 we mark the notable observations of this Section with labels (e.g., **O1**, **O2**, etc.).

In some cases we provide aggregate measures about problems across different participant groups. Unless otherwise stated, aggregate numbers (e.g., 5 in the CS group and 21 in the CP group) refer to *problems*, not participants. Since participants did two problems each, the total counts are out of 20 for each of the groups or out of 60 for the whole data set.

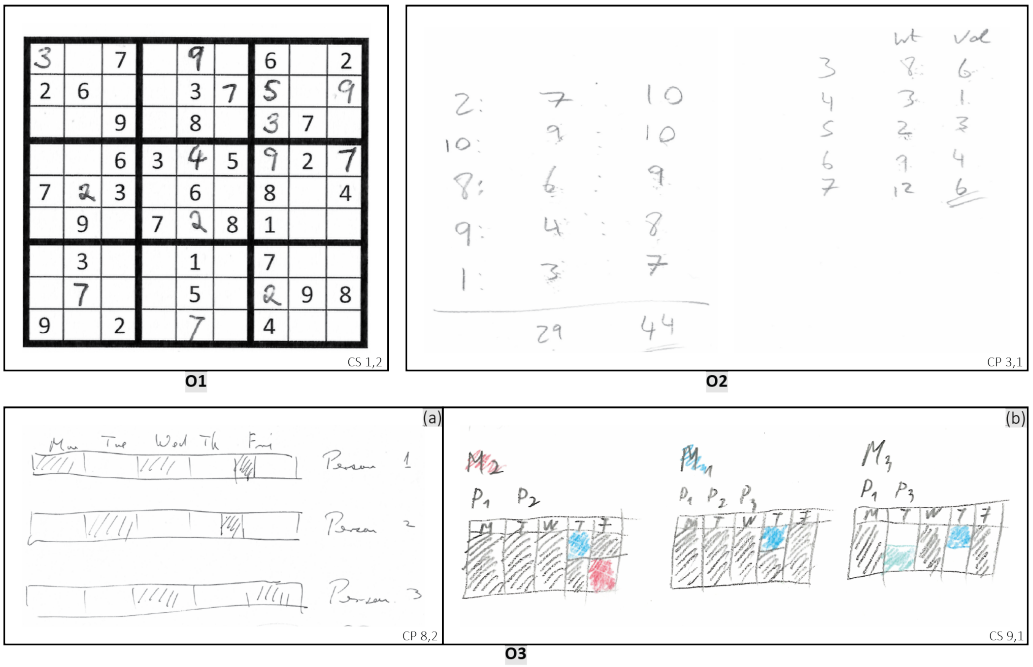

Figure 2 Artefacts from our user study that support the observations regarding representations. These are explained in Subsection 4.1. Each sub-figure is labelled with the name of the observation. Each image is labelled with the anonymised identifier of the relevant participant.

4.1 Representation

Participants were given an initial written problem statement (the exact problem statements are included the Supplementary Materials). In itself, the problem statement can be seen as a representation of the structure of the problem. Sometimes, these representations are adequate to directly operate on (Sudoku and Magic Square can be solved by filling the numbers on the grid provided and Word Crypto allows the reuse of the same shape used in the problem statement for the summation), whereas in other problems the participants need to come up with a representation that is not given in the problem statement itself (e.g., to solve the Knapsack problem one has to create some kind of list or sequence of objects).

(**O1**) When solving a problem, participants attempted to solve most problems (70%) using a representation they already had. That is, participants used a verbatim copy of the representation from the problem statement or, when this was not available, the representation that they themselves created in the modelling part of the study. See Figure 2.O1 for an example of reusing a given representation when solving a Sudoku instance. It has been long known that how a problem is first framed or presented to a human solver can have an effect on how it gets solved (e.g., [19, 35]), due the differing representation of the problem by people [37]. We refer to this issue as *framing* or *representational inertia*.

(**O2**) It is still key to highlight that framing has a sizeable effect on how the problem is understood, at least initially. Participants in the CS and CP groups were more likely to adopt alternative and more sophisticated representations of the problems such as using a multi-column table for the Knapsack problem (See Figure 2.O2), which is akin to using more sophisticated data structures to address the problem. This suggests that one of the key differences between experts and non-experts is the awareness of representational alternatives.

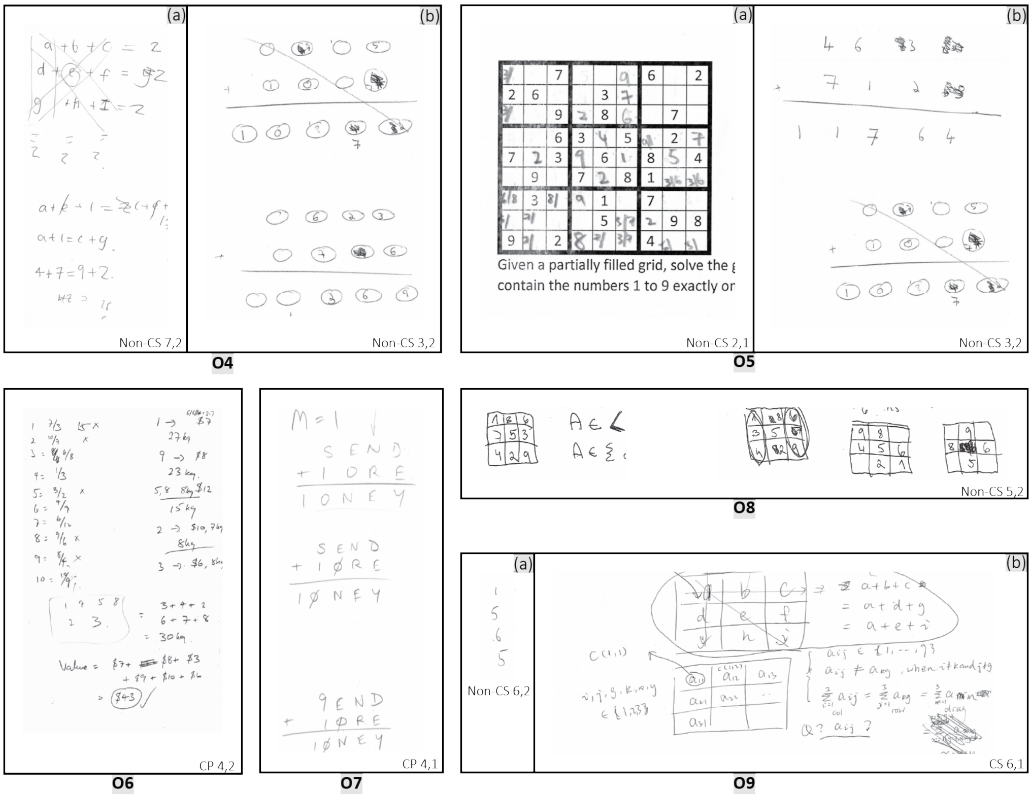

Figure 3 Artefacts from our user study that support the observations regarding process and strategies. These are explained in Subsection 4.2. Each sub-figure is labelled with the name of the observation. Each image is labelled with the anonymised identifier of the relevant participant.

(**O3**) Beyond the structure of the representation itself, some of the problems allowed for alternative choices in the focus of the representation. For example, the scheduling problem could be solved either by modelling the problem from the perspective of the people or focusing on a representation of a timetable. Figure 2.O3 gives an example of these two perspectives ((a) focusing on people, (b) using a timetable). This choice is likely to have an impact on the ability of the participant to solve the problem and correctly understand it. Additionally, the choice is likely to be affected by the semantics of the problem. That is, depending on the nature of the problem, participants might be inclined to model in a way that is not necessarily the most efficient. We suspect that in a problem which involves people a model that puts the person at the center might be preferred at least initially to a more "abstract" way to model the situation. This is consistent with our observation of the scheduling problem, where we saw all participants using a people-centric view of the problem.

4.2 Process and Strategies

Once participants have settled on a representation (or re-representation) they need to get into the process of finding solutions. This is where we find that the assumptions and attitudes towards the problem solving process can have the most dramatic effects.

(**O4**) We observed, for example, that many participants in the non-CS and CS groups thought that there is an ideal way of finding the solution that does not include too much

guesswork. Their assumption is that there is some kind of mathematical formula, algorithm, hidden insight, or an optimal sequence of steps through the solution space that will lead them to a solution without much effort and, presumably, with some level of satisfaction. We find evidence of this attitude in some of the approaches of the participants where they attempt to use a mathematical approach instead of trying to make progress with the puzzle with more intuitive steps of inference. In one case a participant says "there must be a mathematical approach to this". In another case, when solving the Magic Square problem, we observed a participant who turned the grid into a system of equations, which were then meant to be solved to find the common sum of the rows, columns and diagonals (see Figure 3.O4).

(**O5**) Some people simply apply a trial and error approach, in a similar fashion to a random partial (or full) assignment. This seems to be more prevalent among non-CS participants, 7 apply trial and error, in comparison to 5 and 3 in CS and CP categories respectively. For example, one of the non-CS participants, who has never solved a Sudoku puzzle before, started by filling the squares with seemingly random numbers (see Figure 3.O5.a). They then started noticing the implications of the filled in numbers (almost in the style of propagation) and picked up on more common solving strategies such as block/row/column elimination. Another participant from the non-CS group started by assigning a subset of the numbers of the Word Crypto problem randomly and trying to fill in the rest greedily (see Figure 3.O5.b).

(**O6**) As seen above trial and error is a natural first approach when one does not have a good understanding of the structure of the problem or lacks experience in how to address it. However, we found that participants can have quite different reactions to the almost inevitable discovery that the first trial does not comply with the constraints. In some cases participants use a backtracking process in which they undo a certain number of steps (one or more) that they have found to be incorrect or undesirable (e.g., because it might be, or be perceived as, a dead end), to then try a different alternative. This is a fairly sophisticated approach that generally requires keeping a more sophisticated tally of the options tried and the order they occurred in. In general we found that CP participants were less likely to backtrack as they might have a better grasp of the inference of a solution step, we observed 6 CP backtracking whereas 9 CS and 10 non-CS backtracked. But when the CP participants were to backtrack they were generally more likely to employ more sophisticated backtracking. Figure 3.O6 shows an example of this, in which the participant greedily populates the knapsack to find an intermediate solution. Then, they undo some of their decisions and replace the items with higher value items to improve the total value of the solution.

(**O7**) The alternative to backtracking is a restart of the solving problem process. This seemed to be the preferred approach for many, especially in the non-CS group where we observe restarts 12 times, in comparison to 7 and 2 in the CS and CP groups. In Figure 3.O7 we can see how the participant restarted the Word Crypto problem twice, while still remembering some of the information from their last approach. Similarly Figure 3.O4.b shows a participant who attempted the problem a few times from scratch with different strategies, even though they did not make any mistakes. Participants from the CP group seem to have a more realistic understanding of the solving process, and we observed many of them systematically exploring the solution space instead of prematurely restarting from scratch.

(**O8**) Amongst participants who applied restart approaches we observed two main subsets. A majority (17 vs 8 spanning all groups) restart the process without any evident sign of having learned anything from the previous approach. In Figure 3.O8 we have observed the participant attempting to solve the Magic Square problem, by doing four consecutive trials without making any substantial changes to their solving approach.

(**O9**) In contrast, other participants seem to learn about the structure of the problem after

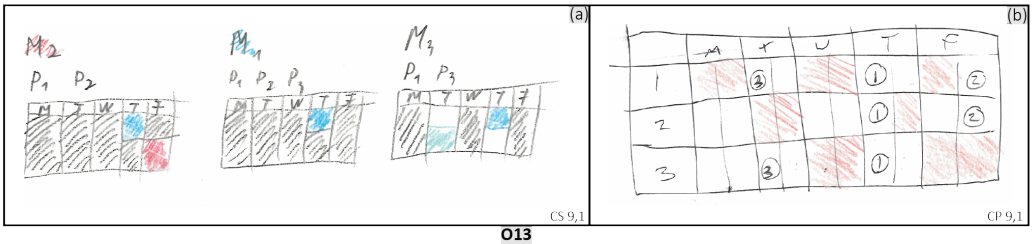

Figure 4 Artefacts from our user study that support the observations regarding how participants check their candidate solutions. These are explained in Subsection 4.3. Each sub-figure is labelled with the name of the observation. Each image is labelled with the anonymised identifier of the relevant participant.

the first attempt and seamlessly transition into a more structured approach. For example, in Figure 3.O9.a we see a participant who started solving the Subset Sum problem by choosing a seemingly random set of numbers. They then realised that they can split the numbers into two groups (positive and negative numbers) and attempt to balance their selection across the two groups. Note that this is more likely to happen for certain kinds of problems due to their more obvious solution space structure. As seen above this is applicable in the Subset Sum problem, but also in the Magic Sum problem, where participants filled in the grid partially to see that in fact some numbers cannot be (or vice versa have to be) in certain cells. In some cases participants transitioned to approaches that are not necessarily useful; for example, in Figure 3.O9.b a participant attempted to create a formal system of equations for solving the Magic Square problem as opposed to trying to develop an intuition for the puzzle – this made the problem harder to solve for them.

4.3 Solution Verification and Multiple Solutions

Once the participants finish solving the problem, either by declaring that they have finished or by giving up, we found regularities in whether they attempted to verify their solutions and whether they attempted to find multiple solutions (where appropriate).

(**O10**) When participants arrive to a potential solution there is a strong tendency to declare the problem solved. We assumed that participants would naturally seek to validate the results (e.g., check the constraints), yet not everybody did this, only 3 non-CS, 8 CS and 6 CP validated their solutions, in total that is only 28% of problem instances.

(**O11**) Despite **O10**, only in 2 instances participants announced that what they have found is the solution, did not validate it and in fact the solution was not correct.

(**O12**) Our scheduling problem happened to have two correct solutions (while it only asked for one). Here 9 out of the 11 participants who finished solving this problem found both solutions. How the participants decided to arrange the results seemed to have an effect on their ability to find more solutions.

(**O13**) In Figure 4.O13.a we can see an example of how the participant's representation of the solution failed to make it possible for them to see the multiple solutions, as the week tables are separate for each meeting. Whereas, when the representation combines all information bundled into a single table, the compact overview allows a precise insight into the options, as shown in Figure 4.O13.b.

(**O14**) Amongst our problems there is an additional solution type category. The Knapsack problem is an optimisation problem, which requires a confirmation as to whether one has

found the best solution. 11 participants claimed to have finished solving the problem, while only 8 of these actually found the correct solution, and 1 of them was lucky as they did not validate their solution yet they did find the correct one.

(**O15**) In the Subset Sum problem we asked the participants to find as many solutions as possible. This prompted a majority of the participants to find multiple solutions, but not everyone managed to find all solutions. In general it is difficult for a participant to know when they found all solutions. This is comparable to knowing whether a solution to an optimisation problem is indeed optimal. None of the participants validated their solutions, yet 3 announced that they were finished with the solving process. All participants who attempted to solve the problem found some of the correct solutions, while 4 of the 6 found all solutions.

5 Limitations

Before we move on to interpret our observations, it is important to highlight the limitations implied by the methodological and study design choices. First, the study uses methodologies appropriate for an initial exploratory assessment of the research questions (i.e., a grounded theory approach). This choice, justified in Section 3, prioritises identifying phenomena over reliably assessing their prevalence. Although we provide proportions and counts which serve as an initial estimation of how reliable or broadly represented an observation is likely to be in the general populations under study, the numbers of participants and problems that we sampled are not enough to make strong claims about the frequency of their occurrence, let alone run statistical analysis. The much higher participant numbers required for this would be prohibitive timewise, or would have led to a coarser analysis, which would not have exposed the most interesting phenomena. Instead, our findings should be more generally interpreted as a "prove of existence" of these effects. This is in the same spirit as Nielsen's observation that a relatively small number of participants in a user study (5 in their case) can surface a majority of usability issues in an interface design [29, 20].

Another limitation of our study is also inherent to the limits of observation methodologies. Although we can be reasonably sure of the actions that participants carry out on paper or express verbally, it is not possible to assess with certainty the internal motivations of participants. We endeavor to keep observations factual; nevertheless, readers must be aware that some interpretation and error are unavoidable.

Practitioners should also be aware that the problems that our participants faced, although representative of a reasonable variety of existing constraint programming types, only partially match the variety and complexity of problems that actual users will face in their real lives. We believe that it is useful and productive to start at the lower end of complexity for an initial study of the topic. Nevertheless, further study of the challenges and barriers encountered by users when facing more complex constraint problems is granted.

When interpreting or applying our findings, practitioners should be aware that, although we aimed at a representative sample of participants with different levels of computing expertise, the majority of the participants had a high level of educational achievement. This means that populations with average levels of formal education are likely underrepresented. Despite this, we do not believe that this invalidates our results or makes them less valuable; if constraint modelling and programming are to become more widely used, it is likely that the broadening will take place first for non-CP computer scientists and then for professionals with high levels of education, before it can be further democratised. This is akin to how spreadsheet tools spread in the 1980's (see e.g., [27]).

Similarly, we agree that the presence of people with mathematical or scientific modeling expertise in the non-CS cohort could be a potential source of bias for the results. However, in our experience we cannot appreciate a difference between people with science and non-science backgrounds in the non-CS group. In fact, we suspect that it is the computational background of participants in the CS and CP groups is what makes the biggest difference. Whether this kind of modelling is more natural for scientists than non-scientists is an interesting question in itself which is, to the best of our knowledge, still unresolved in the scientific literature.

Finally, the interpretations, suggestions and lessons for practitioners that we offer in the next section, although evidence based, will still need to be validated after being put into practice through real modelling languages and tools. We see this study as an initial step to inform the design of a new generation of CP tools that are more aware of the human factor, not as a full characterisation of human behavior in relationship with constraint problems.

6 Discussion

In this section we interpret the observations from Section 4 and indicate how these might affect the design of user interfaces that solve constraint problems or communicate with users about constraint problems. There are two main categories of design decisions in which we imagine the findings being applied. The first category involves design of human-facing parts of existing constraint solving technologies such as constraint modelling languages, editors or graphical interfaces. The second is the design of system-provided automatic explanations or guidance that exposes the working of solvers to make it more intelligible to their users, perhaps to address issues of human trust and accountability in artificial intelligence systems. This is in line with the broader current push for intelligible, explainable or interpretable artificial intelligence (e.g., [1, 9]).

We group our discussion into three main topics, one about the importance of representations, one about people's mental models of the operations involved in the solving of constraint problems, and one about the larger context in which constraint solving takes place.

6.1 Representation, Representational Competence and Visualisation

Solving constraint problems can be characterised, at least in part, as a representational problem. When people recognise a world state that could benefit from calculating a solution based on constraints, the challenge is often for them to achieve a sufficiently accurate representation of the structure of the problem. Observations **O1**, **O2** and **O3** directly highlight some of the challenges of an efficient human-machine interface for constraint solving. The initial representation in which a problem appears to a person might lock their thinking to representations that might be suboptimal or even pernicious to the overall objective. We use a fictional (but plausible) example here to illustrate the different challenges. Consider a university administrator trying to schedule a series of rooms for exams of a group of students; the same student should not be required to be in two rooms at the same time, but different students take different combination of courses, whose associated exams should take place simultaneously and preferably in the same room. Our observations suggest that people will tend to adopt the most salient representation of the problem. In this case, it is likely that the administrator would naturally use a room-centric representation of the problem, since booking of rooms is their most visible required action. But a room-based representation can make it difficult for the administrator to express some of the student-centric constraints, and might also not be the best way to express the problem in a modelling language to obtain a solution efficiently. The ability of the problem holder to consider

different representations might be a key element in the success of the modelling activity, yet we observed that non-CS people seem to have very little awareness of the importance of representations. Representational awareness and the related concepts of representational and meta-representational competence [23, 31, 8, 35] (i.e., the idea that people's ability to create different representations of a problem and translate between different representations is a fundamental skill for its understanding and solution) have been identified in other areas such as chemistry and physics education as key elements of expertise and solving skills [22].

The issue of representation is also likely to have an influence in the other direction as well. Once people get used to specific ways of representing problems (e.g., because they get used to a particular language or interface), the representation could become their default way of expressing this type of problems. If the representation is powerful and efficient for computation, this can be useful, but if the representation is unduly constrained or inappropriate it can prevent people from even recognizing a problem as such.

Our observations also bring a reminder that how solutions are represented is also important (**O13**). Ideally, constraint solving software should be able to present results in ways that are readily accessible to the problem holder. In fact, how the solutions are represented might have an influence on how they understand the solution space (see also Subsection 6.3) and whether they will try to iterate their modelling and choose solutions when multiple are available (this is discussed further in Subsection 6.2).

Overall, we believe that the design of representation specification tools and solution visualization are both areas of inquiry that can leverage significant benefit towards the goals stated in Section 2.

6.2 The Larger Context of Constraint Solving

Current tools for modelling and solving constraint problems (such as Conjure [2], SavileRow [30] and MiniZinc [28]) assume a programming workflow where the focus is on the sequence of a) model writing; b) model running/execution; c) output. However, some of our observations above suggest that there is benefit in considering the other surrounding activities in which the problem holder is involved. We have already indirectly mentioned two: users might need to switch representations (based on **O2** and **O3**), and users will need to interpret the solutions to see if they fit their needs (**O13**).

Our findings indirectly point to other parts of the process that might be also important. One is the need for validation of solutions (**O10**, **O11**); even if we can safely assume that the software will only provide solutions that are correct, it is possible that a human error occurred elsewhere (e.g., in the modelling, or when specifying to the solver whether the problem should be treated as a satisfaction problem — any solution will do — vs. an optimisation problem). An effective user interface can facilitate user validation that the solutions actually address the intended problem (not only the problem that the user managed to model). The need for validation is also recognised in previous work on using animation and model checking to allow users to gain confidence in their specifications [24].

In general, our observations of the process lead us to an understanding of the process of constraint problem solving that is less compartmentalised than what current interfaces assume. The iterative process of trying to solve a problem often drives people to improve their solution finding strategies (**O9**), but also to a better general understanding of the structure of the problem itself. For people, it might not be until they try to solve a problem that they start understanding whether the problem can have a single, more than one or even no solutions. Similarly, insights acquired by trying to manually solve the problem might lead to re-modeling (re-representations) of the problem, which could lead to improved efficiency.

This follows findings in the domain of data analysis that shows that manual (and often tedious) specification of visualizations can enhance understanding of data [26]. Additionally, we have very little doubt that the general process of constraint problem solving with human intervention most often requires multiple iterations of modelling, execution and verification (note, however, that our study did not offer sufficient context to expose this larger loop).

6.3 Human Thinking about Constraint Solving

At first sight it might seem gratuitous to study how people solve constraint problems manually. After all, modern computers running modern solvers are more accurate and orders of magnitude faster than humans at this task. However, we think that there are several reasons why our observations can be beneficial for the design of future systems.

One such reason is that the user's expectations matter. For example, we observed that many participants in the non-CS and CS groups expected problems to have a "happy", straightforward, or analytical solution (**O4**). Users which assume that every problem has this kind of solution might expect fewer solving steps, even when the state space is large and even an efficient solver can take seconds or even hours to find a solution, let alone all solutions or demonstrate that there are none. It is not clear where this assumption comes from, but it might be from how people often encounter constraint problems as puzzles or exam questions, which are usually designed precisely to avoid the trial-error-backtracking process that can be inevitable in many problems.

Through the study of how people attempt to solve these problems we have also observed that many non-CP experts have very little understanding of the size of the state space and the task of solving some of these problems, even when they have some background in CS. This presents a problem for systems integrating constraint solvers because interfaces will often not be instantaneous, which is the current expectation for most applications. We see two possible ways to ameliorate this problem. The system could explain and communicate with users why a solution can not be arrived at very fast. Beyond this, systems could educate the user to recognise the type of modeling structures that cause the delay in the first place, and perhaps suggest alternatives that preserve the meaning of the model but are faster to solve. A very sophisticated system that enables this kind of human-CP solver dialogue likely requires maintaining in the solver some kind of user model that incorporates knowledge about regularities of the kind that we found in our study (e.g., preference for modelling from a certain point of view — **O3**).

7 Related Work

Previous work considers the interaction of constraint programming and its users in the context of teaching, visualisation and explainable AI.

A typical way of teaching constraint programming is beginning with its theoretical foundations, which often takes a mathematical approach. For example the 'Essentials of Constraint Programming' book [14] contains chapters on logic, Boolean algebra, linear polynomial equations and non-linear equations. Similarly, the 'Principles of Constraint Programming' book [3] defines constraint programming as a linear equation solver and a unification algorithm. On the other hand, in his invited talk at CP 2014, Prosser explains his experiences with teaching constraint programming [33]. According to this talk[1], getting

[1] Available online: `http://www.dcs.gla.ac.uk/~pat/presentations/CP2014.pptx`

students to solve problems as soon as possible increases their engagement as opposed to starting with a more theoretical background. Chan et al. [7] presents a unique approach to teaching CP in a MOOC (Massive Online Open Course) setting. They use a Fable-Based learning approach and present the topics in the course using a coherent story plot and through problem solving. Both Prosser and Chan et al. describe benefits gained from listening to the learners and explain how they improved their teaching style upon feedback.

There are several examples of visualisation tools for the *solving* process, we describe a small selection here. Bauer et al. [4] presented an integrated development environment (IDE) for constraint programming. The IDE provides a visual debugger which displays the search tree that is explored by the constraint solver. The debugger is solver-independent, with minor modifications it can support any solver. However, their system only focuses on visualising the solving process and not modelling. Recently Goodwin et al. [18] described a user-centred design process for tools that visualise the solving process, building on earlier work by Shishmarev et al. [36]. From an Information Visualisation perspective, Goodwin et al. [18] looked at how different visualisations could be useful in the process of profiling constraint models. This allows users of constraint programming to refine their models or test different parameters.

Constraint programming lends itself to automatically creating explanations for unsatisfiable problems, as well as explanations for how certain inferences or decisions are made. This is mainly due to the model-based nature of CP: typically users write a declarative model when applying CP technology and this explicitly captures the requirements of the task at hand. Sqalli et al. [38] produce explanations for inference-based CSPs, Bogaerts et al. [6] extend this work to multiple steps in the context of logic puzzles, and Espasa et al. [10] show human-like solving steps in a variety of pen and paper puzzles.

8 Conclusion

This paper presents a user-centric qualitative study, aiming to understand how people approach constraint modelling and solving. The study is done on three groups of participants (30 in total), and hence we are able to explore regularities within and across distinct groups of people. To the best of our knowledge this paper presents the first user-study of its kind: recording participants model and solve constraint problems with maximum freedom in their approach, analysing 32+ hours of video recordings, coding each recording to characterise it, and using grounded theory to offer an analysis of the codes we produce.

Some directions for future work are related to the limitations in our study. Further experiments are necessary to statistically quantify the preponderance of the different phenomena that we observed (our study did not have sufficient numbers for it, since it was designed for identifying the phenomena instead), to ascertain possible sub-populations of interest in the non-CS cohort (e.g., see whether people with and without science backgrounds show differences in how they think about constraint problem solving—this could have biased our results, since three participants in our non-CS group had science backgrounds) and to generalize to people without formal tertiary education (including children). More advanced CP-specific aspects during modelling (like the use of global constraints, the effect of implied constraints and symmetry/dominance breaking constraints) and during the solving process (like search heuristics and clause learning) that affect the efficiency of solving need to be studied further as well. Another direction of future work is the development of prototype user interface systems that build on top of our findings and a thorough evaluation of such user interfaces.

References

1 Amina Adadi and Mohammed Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*, 6:52138–52160, 2018. doi:10.1109/ACCESS.2018.2870052.

2 Özgür Akgün, Ian Miguel, Christopher Jefferson, Alan M. Frisch, and Brahim Hnich. Extensible automated constraint modelling. In *AAAI 2011*. AAAI Press, 2011. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3687.

3 Krzysztof R. Apt. *Principles of constraint programming*. Cambridge University Press, 2003.

4 Andreas Bauer, Viorica Botea, Mark Brown, Matt Gray, Daniel Harabor, and John K. Slaney. An integrated modelling, debugging, and visualisation environment for G12. In *CP 2010*. Springer, 2010. doi:10.1007/978-3-642-15396-9_42.

5 Amine Benamrane, Imade Benelallam, and El-Houssine Bouyakhf. Constraint programming based techniques for medical resources optimization: medical internships planning. *J. Ambient Intell. Humaniz. Comput.*, 11(9):3801–3810, 2020. doi:10.1007/s12652-019-01587-6.

6 Bart Bogaerts, Emilio Gamba, Jens Claes, and Tias Guns. Step-wise explanations of constraint satisfaction problems. In *ECAI 2020*. IOS Press, 2020. doi:10.3233/FAIA200149.

7 Mavis Chan, Cecilia Chun, Holly Fung, Jimmy H. M. Lee, and Peter J. Stuckey. Teaching constraint programming using fable-based learning. In *AAAI*. AAAI Press, 2020. URL: https://aaai.org/ojs/index.php/AAAI/article/view/7059.

8 Andrea A. diSessa. Metarepresentation: Native Competence and Targets for Instruction. *Cognition and Instruction*, 22(3):293–331, 2004. doi:10.1207/s1532690xci2203_2.

9 Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Commun. ACM*, 63(1):68–77, 2020. doi:10.1145/3359786.

10 Joan Espasa, Ian P. Gent, Ruth Hoffmann, Christopher Jefferson, and Alice M. Lynch. Using small muses to explain how to solve pen and paper puzzles. *CoRR*, abs/2104.15040, 2021. URL: https://arxiv.org/abs/2104.15040, arXiv:2104.15040.

11 Eugene C. Freuder. In pursuit of the holy grail. *Constraints An Int. J.*, 2(1):57–61, 1997. doi:10.1023/A:1009749006768.

12 Eugene C. Freuder. Progress towards the holy grail. *Constraints An Int. J.*, 23(2):158–171, 2018. doi:10.1007/s10601-017-9275-0.

13 Eugene C. Freuder and Mark Wallace. Constraint technology and the commercial world (interview). *IEEE Intell. Syst.*, 15(1):20–23, 2000. doi:10.1109/MIS.2000.820324.

14 Thom W. Frühwirth and Slim Abdennadher. *Essentials of constraint programming*. COGTECH. Springer, 2003. URL: http://www.springer.com/computer/swe/book/978-3-540-67623-2.

15 Cristian Galleguillos, Zeynep Kiziltan, and Ricardo Soto. A job dispatcher for large and heterogeneous HPC systems running modern applications. In *CP 2021*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CP.2021.26.

16 James H. Gerlach and Feng-Yang Kuo. Understanding human-computer interaction for information systems design. *MIS Q.*, 15(4):527–549, 1991. URL: http://misq.org/understanding-human-computer-interaction-for-information-systems-design.html.

17 Barney G Glaser and Anselm L Strauss. *The discovery of grounded theory: Strategies for qualitative research*. Routledge, 2017.

18 Sarah Goodwin, Christopher Mears, Tim Dwyer, Maria Garcia de la Banda, Guido Tack, and Mark Wallace. What do constraint programming users want to see? exploring the role of visualisation in profiling of models and search. *IEEE Trans. Vis. Comput. Graph.*, 23(1):281–290, 2017. doi:10.1109/TVCG.2016.2598545.

19 Francis Heylighen. Formulating the Problem of Problem-Formulation. In *Cybernetics and Systems '88*, pages 949–957. Kluwer Academic Publishers, Dordrecht, 1988.

20 Wonil Hwang and Gavriel Salvendy. Number of people required for usability evaluation: the 10+/-2 rule. *Commun. ACM*, 53(5):130–133, 2010. doi:10.1145/1735223.1735255.

21 Christopher Jefferson, Ian Miguel, Brahim Hnich, Toby Walsh, and Ian P Gent. CSPLib: A problem library for constraints, 1999. URL: http://www.csplib.org.

22 Antje Kohnle and Gina Passante. Characterizing representational learning: A combined simulation and tutorial on perturbation theory. *Physical Review Physics Education Research*, 13(2):020131, 2017. doi:10.1103/PhysRevPhysEducRes.13.020131.

23 Robert Kozma and Joel Russell. Students Becoming Chemists: Developing Representational Competence. In *Visualization in Science Education*, Models and Modeling in Science Education, pages 121–145. Springer Netherlands, Dordrecht, 2005. doi:10.1007/1-4020-3613-2_8.

24 Michael Leuschel and Michael Butler. Prob: A model checker for b. In *International symposium of formal methods europe*, pages 855–874. Springer, 2003.

25 Mary L McHugh. Interrater reliability: the kappa statistic. *Biochemia Medica*, 22(3):276–282, 2012. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3900052/.

26 Gonzalo Gabriel Méndez, Uta Hinrichs, and Miguel A. Nacenta. Bottom-up vs. top-down: Trade-offs in efficiency, understanding, freedom and creativity with infovis tools. In *CHI 2017*. ACM, 2017. doi:10.1145/3025453.3025942.

27 Bonnie A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT press, 1993. URL: https://books.google.co.uk/books?hl=en&lr=&id=0drDRT370eoC&oi=fnd&pg=PR11&dq=nardi+small+matter+of+programming&ots=eFmV2hPujA&sig=ddWW4jgU1jebzDZpk7plDRCtcoU.

28 Nicholas Nethercote, Peter J. Stuckey, Ralph Becket, Sebastian Brand, Gregory J. Duck, and Guido Tack. Minizinc: Towards a standard CP modelling language. In *CP 2007*. Springer, 2007. doi:10.1007/978-3-540-74970-7_38.

29 Jakob Nielsen. Estimating the number of subjects needed for a thinking aloud test. *Int. J. Hum. Comput. Stud.*, 41(3):385–397, 1994. doi:10.1006/ijhc.1994.1065.

30 Peter Nightingale, Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Patrick Spracklen. Automatically improving constraint models in savile row. *Artif. Intell.*, 251:35–61, 2017. doi:10.1016/j.artint.2017.07.001.

31 Orit Parnafes and Andrea A. diSessa. Relations between types of reasoning and computational representations. *Int. J. Comput. Math. Learn.*, 9(3):251–280, 2004. doi:10.1007/s10758-004-3794-7.

32 Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, 2011. URL: http://dl.acm.org/citation.cfm?id=2078195.

33 Patrick Prosser. Teaching constraint programming. In *CP 2014*. Springer, 2014. doi:10.1007/978-3-319-10428-7_2.

34 Bochra Rabbouch, Foued Saâdaoui, and Rafaa Mraihi. Constraint programming based algorithm for solving large-scale vehicle routing problems. In *HAIS 2019*. Springer, 2019. doi:10.1007/978-3-030-29859-3_45.

35 S I Robertson. *Problem Solving*. Psychology Press, 2001.

36 Maxim Shishmarev, Christopher Mears, Guido Tack, and Maria Garcia de la Banda. Visual search tree profiling. *Constraints An Int. J.*, 21(1):77–94, 2016. doi:10.1007/s10601-015-9202-1.

37 Herbert A Simon and John R Hayes. The understanding process: Problem isomorphs. *Cognitive psychology*, 8(2):165–190, 1976.

38 Mohammed H. Sqalli and Eugene C. Freuder. Inference-based constraint satisfaction supports explanation. In *AAAI/IAAI 1996*. AAAI Press / The MIT Press, 1996. URL: http://www.aaai.org/Library/AAAI/1996/aaai96-048.php.

39 Maarten Van Someren, Yvonne F. Barnard, and J. Sandberg. The think aloud method: A practical approach to modelling cognitive processes. *London: AcademicPress*, 11, 1994.

40 Xu Zhu, Miguel A. Nacenta, Özgür Akgün, and Peter Nightingale. How people visually represent discrete constraint problems. *IEEE Trans. Vis. Comput. Graph.*, 26(8):2603–2619, 2020. doi:10.1109/TVCG.2019.2895085.