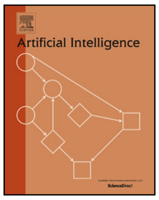
Artificial Intelligence 300 (2021) 103550

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint



A framework for step-wise explaining how to solve constraint satisfaction problems[☆]

Bart Bogaerts, Emilio Gamba[*], Tias Guns

Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussel, Belgium

ARTICLE INFO

Article history:
Received 27 April 2020
Received in revised form 20 June 2021
Accepted 22 June 2021
Available online 29 June 2021

Keywords:
Artificial intelligence
Constraint solving
Explanation

ABSTRACT

We explore the problem of step-wise explaining how to solve constraint satisfaction problems, with a use case on logic grid puzzles. More specifically, we study the problem of explaining the inference steps that one can take during propagation, in a way that is easy to interpret for a person. Thereby, we aim to give the constraint solver explainable agency, which can help in building trust in the solver by being able to understand and even learn from the explanations. The main challenge is that of finding a sequence of *simple* explanations, where each explanation should aim to be as cognitively easy as possible for a human to verify and understand. This contrasts with the arbitrary combination of facts and constraints that the solver may use when propagating. We propose the use of a cost function to quantify how simple an individual explanation of an inference step is, and identify the explanation-production problem of finding the best sequence of explanations of a CSP. Our approach is agnostic of the underlying constraint propagation mechanisms, and can provide explanations even for inference steps resulting from combinations of constraints. In case multiple constraints are involved, we also develop a mechanism that allows to break the most difficult steps up and thus gives the user the ability to *zoom in* on specific parts of the explanation. Our proposed algorithm iteratively constructs the explanation sequence by using an optimistic estimate of the cost function to guide the search for the best explanation at each step. Our experiments on logic grid puzzles show the feasibility of the approach in terms of the quality of the individual explanations and the resulting explanation sequences obtained.

1. Introduction

In the last few years, as AI systems employ more advanced reasoning mechanisms and computation power, it becomes increasingly difficult to understand why certain decisions are made. Explainable AI (XAI) research aims to fulfill the need for trustworthy AI systems to understand why the system made a decision for verifying the correctness of the system, as well as to control for biased or systematically unfair decisions.

Explainable AI is often studied in the context of (black box) prediction systems such as neural networks, where the goal is to provide insight into which part(s) of the input is important in the *learned* model. These insights (or local approximations thereof) can justify why certain predictions are made. In that setting, a range of techniques have been developed, ranging

[☆] This paper is part of the Special Issue on Explainable AI.
[*] Corresponding author.
E-mail addresses: bart.bogaerts@vub.be (B. Bogaerts), emilio.gamba@vub.be (E. Gamba), tias.guns@vub.be (T. Guns).

https://doi.org/10.1016/j.artint.2021.103550

from local explanations of predictions at the *feature level* [1,2] to **visual explanations** with *saliency maps* [3]. Adadi et al. [4], Guidotti et al. [5] and more recently Arrieta et al. [6], survey the latest trends and major research directions in this area. In contrast, in Constraint Satisfaction Problems (CSP) [7], the problem specification is an explicit *model-based representation* of the problem, hence creating the opportunity to explain the inference steps directly in terms of this representation.

Explanations have been investigated in constraint solving before, most notably for explaining overconstrained, and hence unsatisfiable, problems to a user [8]. Our case is more general in that it also works for satisfiable problems. At the solving level, in lazy clause generation solvers, explanations of a constraint are studied in the form of an implication of low-level Boolean literals that encode the result of a propagation step of an individual constraint [9]. Also, no-goods (learned clauses) in conflict-driven clause learning SAT solvers can be seen as explanations of failure during search [10]. These are not meant to be human-interpretable but rather to propagate effectively.

We aim to explain the process of propagation in a constraint solver, independent of the consistency level of the propagation and without augmenting the propagators with explanation capabilities. For problems that can — given a strong enough propagation mechanism — be solved without search, e.g. problems such as logic grid puzzles with a unique solution, this means explaining the entire problem solving process. For problems involving search, this means explaining the inference steps in one search node. It deserves to be stressed that we are not interested in the computational cost of performing an expensive form of propagation, but in explaining all consequences of a given assignment to the user in a way that is as understandable as possible.

More specifically, we aim to develop an explanation-producing system that is complete and interpretable. By *complete* we mean that it finds a *sequence* of small reasoning steps that, starting from the given problem specification and a partial solution, derives all consequences. Gilpin et al. [11] define *interpretable* explanations as 'descriptions that are simple enough for a person to understand, using a vocabulary that is meaningful to the user'. Our guiding principle is that of simplicity, where smaller and simpler explanations are better.

An open issue when presenting a sequence of explanations is the level of abstraction used. Starting from a sequence of as-simple-as-possible explanations as we do, there are two ways to change the level of abstraction. The first direction is that one could *group* multiple single steps into one larger step that can then be expanded on the fly. The second direction is to provide a *more detailed* explanation of harder steps which can not be broken into simpler steps with the standard mechanism. While the first direction can be useful when explanations get very long, from a theoretical perspective, it is less interesting. The second direction on the other hand is interesting for several reasons. First of all, we noticed that some of the generated steps are still too complicated to be understood easily and thus really require being explained in more detailed. Secondly, breaking them up in smaller steps is an interesting challenge. Indeed, since we start from the idea that steps should be as simple as possible, it should not be possible to break them up further. To still break them up further, we took inspiration from methods people use when solving puzzles, and mathematicians use when proving theorems. That is, for explaining a single step in more detail we work using reasoning by contradiction: starting from the negation of a consequence, we explain how a contradiction is obtained. In essence, we assume the negation of the derived fact, and can then reuse the principle of explanation steps to construct a sequence that leads to an inconsistency. This novel approach allows us to provide a mechanism for *zooming in* on the most difficult explanation step.

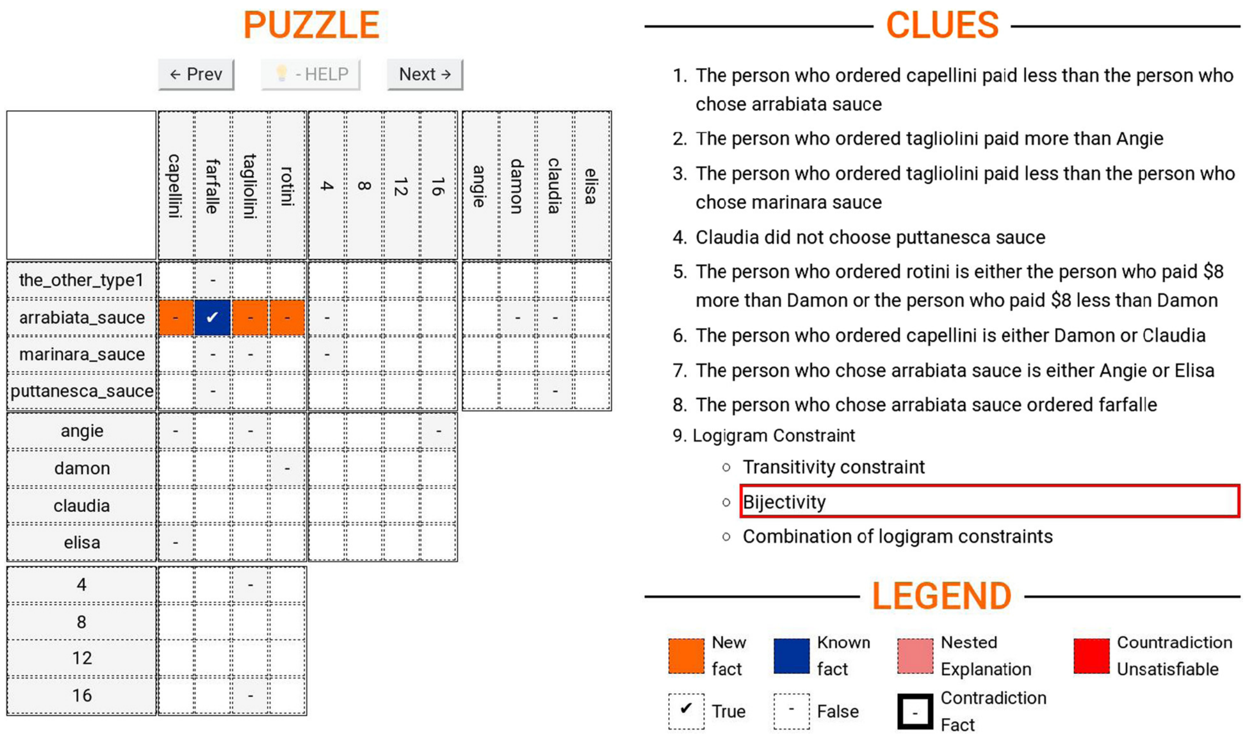
In practice, we choose to present the constraints in natural language, which is an obvious choice for logic grid puzzles as they are given in the form of natural language *clues*. We represent the previously and newly derived facts visually, as can be seen in the grid in Fig. 1. In this figure, the implicit 'Bijectivity' axiom present in each logic grid puzzle is used to derive the following: since Arrabiata sauce was eaten with Farfalle, it was not eaten with any of the other pasta types.

Our work, and more specifically the use case of logic grid puzzles, is motivated by the 'Holy Grail Challenge'[1] which had as objective to provide *automated* processing of logic grid puzzles, ranging from natural language processing, to solving, and explaining. While our integrated system, ZebraTutor, has the capability of solving logic grid puzzle starting from the natural language clues (see Section 8), the focus in this paper is on the explanation-producing part of the system.

The explanation-generating techniques we develop can be applied in a multitude of use cases. For instance, it can explain the entire sequence of reasoning, such that a user can better understand it, or in case of an unexpected outcome can debug the reasoning system or the set of constraints that specify the problem. As our approach starts from an arbitrary set of facts, it can also be used as a virtual assistant when a user is stuck in solving a problem. The system will explain the simplest possible next step, or in an interactive setting, the system can explain how it would complete a partial solution of a user. Such explanations can be useful in the context of *interactive configuration* [12] where a domain expert solves a problem (e.g., a complicated scheduling or rostering problem) but is assisted by an intelligent system. The system in that case typically does not have all the knowledge required to solve the problem since certain things are hard to formalize, especially when personal matters are involved. In such case, the system cannot find the optimal solution automatically, but it can help the expert for instance by propagating information that follows from its knowledge base. In case something undesired is propagated, the user might need an explanation about *why* this follows; this is where our methods can be plugged in. Finally, our measures of simplicity of reasoning steps can be used to estimate the difficulty of solving a problem for a human, e.g. for gradual training of experts.

Summarized, our main contributions are the following:

[1] https://freuder.wordpress.com/pthg-19-the- third-workshop-on-progress-towards-the-holy-grail/.



	capellini	farfalle	tagliolini	rotini	4	8	12	16	angie	damon	claudia	elisa
the_other_type1		-										
arrabiata_sauce	🟧 -	🟦 ✓	🟧 -	🟧 -	-					-	-	
marinara_sauce		-	-		-							
puttanesca_sauce		-									-	
angie	-		-					-				
damon				-								
claudia												
elisa	-											
4			-									
8												
12												
16			-									

CLUES

1. The person who ordered capellini paid less than the person who chose arrabiata sauce
2. The person who ordered tagliolini paid more than Angie
3. The person who ordered tagliolini paid less than the person who chose marinara sauce
4. Claudia did not choose puttanesca sauce
5. The person who ordered rotini is either the person who paid $8 more than Damon or the person who paid $8 less than Damon
6. The person who ordered capellini is either Damon or Claudia
7. The person who chose arrabiata sauce is either Angie or Elisa
8. The person who chose arrabiata sauce ordered farfalle
9. Logigram Constraint
   - Transitivity constraint
   - Bijectivity
   - Combination of logigram constraints

LEGEND

🟧	New fact
🟦	Known fact
▨	Nested Explanation
🟥	Countradiction Unsatisfiable
✓	True
-	False
☐	Contradiction Fact

Fig. 1. Demonstration of explanation visualization. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

- We formalize the problem of step-wise explaining the propagation of a constraint solver through a sequence of small inference steps;
- We propose an algorithm that is agnostic to the propagators and the consistency level used, and that can provide explanations for inference steps involving arbitrary combinations of constraints;
- Given a cost function quantifying human interpretability, our method uses an optimistic estimate of this function to guide the search to low-cost explanations, thereby making use of Minimal Unsatisfiable Subset extraction;
- We introduce nested explanations to provide additional explanations of complex inference steps using reasoning by contradiction;
- We experimentally demonstrate the quality and feasibility of the approach in the domain of logic grid puzzles.

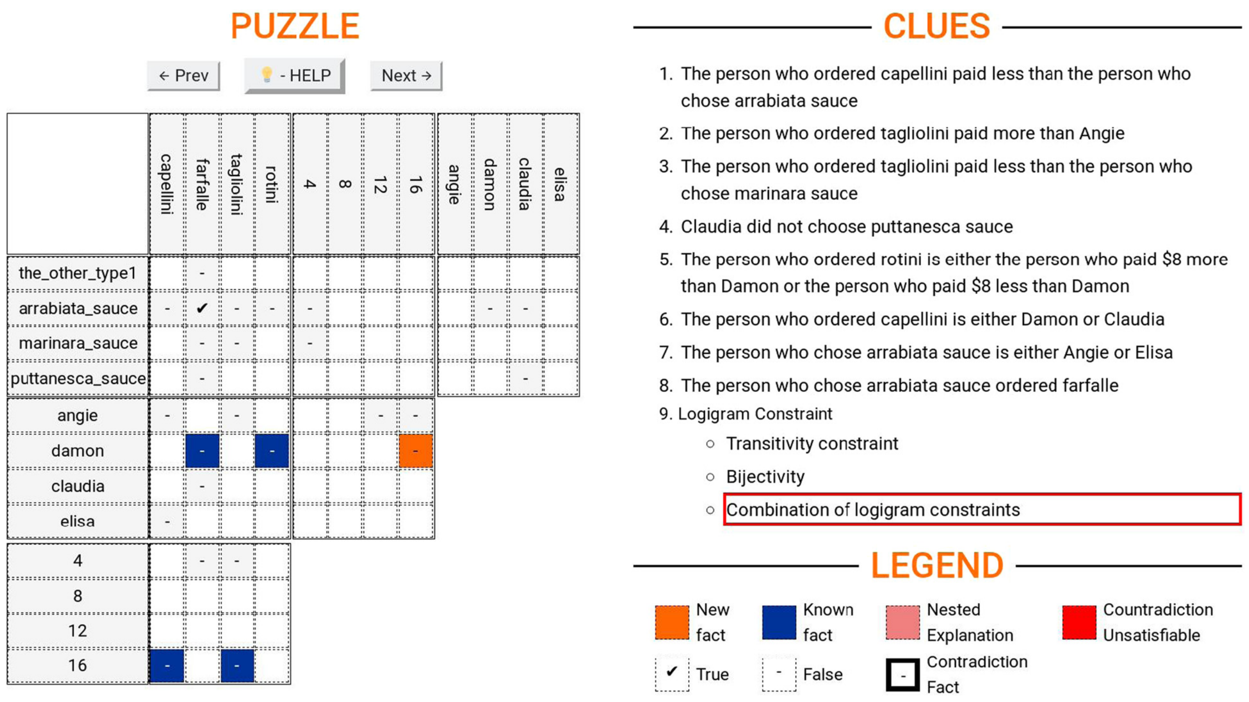
This paper is structured as follows. In Section 2, we discuss related work. Section 3, explains the rules of logic grid puzzles and presents background information. Sections 4 and 5, formalize the theory of the explanation-production problem on two abstraction levels, while Section 6 describes the algorithms needed to solve the explanation-production problem. In Section 7, we motivate our design decisions using observations from the ZebraTutor use case. Section 8 describes the entire information pipeline of the ZebraTutor integrated system. In Section 9, we experimentally study the feasibility of our approach. Finally, we conclude the paper in Section 10.

Publication history This paper is an extension of previous papers presented at workshops and conferences [13–15]. The current paper extends the previous papers with more detailed examples, additional experiments, as well as the formalization of what we call *nested explanation sequences*.

2. Related work

This research fits within the general topic of Explainable Agency [16], whereby in order for people to trust autonomous agents, the latter must be able to *explain their decisions* and the *reasoning* that produced their choices.

Explanations of Constraint Satisfaction Problems (CSP) have been studied most in the context of over-constrained problems. The goal then is to find a small conflicting subset. The QuickXplain method [8] for example uses a dichotomic approach that recursively partitions the constraints to find a minimal conflict set. Many other papers consider the same goal and search for explanations of over-constrainedness [17,18]. A minimal set of conflicting constraints is often called a *minimal unsatisfiable subset* (MUS) or *minimal unsatisfiable core* [19]. Despite the fact that we do not (specifically) aim to explain overconstrained problems, our algorithms will internally also make use of MUS extraction methods.



	capellini	farfalle	tagliolini	rotini	4	8	12	16	angie	damon	claudia	elisa
the_other_type1		-										
arrabiata_sauce	-	✔	-	-	-					-	-	
marinara_sauce		-	-		-							
puttanesca_sauce		-									-	
angie	-		-				-	-				
damon		🟦 -		🟦 -				🟧 -				
claudia		-										
elisa	-											
4		-	-									
8												
12												
16	🟦 -		🟦 -									

1. The person who ordered capellini paid less than the person who chose arrabiata sauce
2. The person who ordered tagliolini paid more than Angie
3. The person who ordered tagliolini paid less than the person who chose marinara sauce
4. Claudia did not choose puttanesca sauce
5. The person who ordered rotini is either the person who paid \$8 more than Damon or the person who paid \$8 less than Damon
6. The person who ordered capellini is either Damon or Claudia
7. The person who chose arrabiata sauce is either Angie or Elisa
8. The person who chose arrabiata sauce ordered farfalle
9. Logigram Constraint
   - Transitivity constraint
   - Bijectivity
   - Combination of logigram constraints

🟧	New fact
🟦	Known fact
▨	Nested Explanation
🟥	Countradiction Unsatisfiable
✔	True
-	False
▣	Contradiction Fact

Fig. 2. Demonstration of explanation visualization.

While explainability of constraint optimization has received little attention so far, in the related field of *planning*, there is the emerging subfield of *eXplainable AI planning* (XAIP) [20], which is concerned with building planning systems that can explain their own behavior. This includes answering queries such as ‘why did the system (not) make a certain decision?’, ‘why is this the best decision?’, etc. In contrast to explainable machine learning research [5], in explainable planning one can make use of the explicit *model-based representation* over which the reasoning happens. Likewise, we will make use of the constraint specification available to constraint solvers, more specifically typed first-order logic [21].

Our work was inspired by the Holy Grail Challenge [22] at the 2019 Constraint Programming conference (CP), which in turn has its roots in earlier work of E. Freuder on inference-based explanations [23]. In the latter, the authors investigate logic grid puzzles and develop a number of problem-specific inference rules that allow solving (most, but not all) such puzzles without search. These inference rules are equipped with explanation templates such that each propagation event of an inference rule also has a templated explanation, and hence an explanation of the solution process is obtained. We point out that the more complex inference rules (NCC and GNCC) are in fact inference rules over hard-coded combinations of (in)equality constraints. In contrast, our proposed method works for any type of constraint and any combination of constraints, and automatically infers a minimal set of facts and constraints that explain an inference step, without using any problem-specific knowledge. This powerful combination of constraints is able to automatically detect interesting consistency patterns that needed to be hand-coded in the Freuder’s seminal work, but also in the solutions submitted by other contestants in the challenge [24]. Fig. 2 shows an example of a non-trivial explanation that our approach automatically generated; a combination of a so-called bijectivity and transitivity constraints, which was hard-coded as the special-purpose ‘path consistency’ pattern in earlier logic-grid specific work [23].

There is a rich literature on automated and interactive theorem proving, recently focusing on providing proofs that are understandable for humans [25] and, e.g., on teaching humans – using interaction with theorem provers – how to craft mathematical proofs [26]. Our work fits into this line of research since our generated explanations can also be seen as proofs, but in the setting of finite-domain constraint solving.

In the context of interactive explanations, Caine and Cohen [27] introduce a *mixed-initiative* Intelligent Tutoring System (MITS). In that framework, the system maintains a model of the knowledge of the user (who is solving a sudoku), estimating how well certain strategies are understood. The tutor, based on its model of the student, decides whether the benefits outweigh the costs of interaction with the student (i.e. proposing a move or explaining a step) when the student lacks knowledge. The student also has the ability to ask for further guidance; in other words, the two involved parties can take the initiative to engage in interaction. In comparison, in our approach, we do not aim to model the user, but rather assume a cost-function is specified that quantifies difficulties of derivations. At each point in time, we then suggest the simplest next derivation.

3. Background

While our proposed method is applicable to constraint satisfaction problems in general, we use *logic grid puzzles* as an example domain, as it requires no expert knowledge to understand. Our running example is a puzzle about people having dinner in a restaurant and ordering different types of pasta, which is the hardest logic grid puzzle we encountered; it was sent to us by someone who got stuck solving it and wondered whether it was correct in the first place. The entire puzzle can be seen in Fig. 1; the full explanation sequence generated for it can be explored online at http://bartbog.github.io/zebra/pasta.

In this section, we first present logic grid puzzles as a use case, and afterwards introduce *typed first-order logic*, which we use as the language to express our constraint programs in. Here, it is important to stress that our definitions and algorithms work for any language with model-theoretic semantics, including typical constraint programming languages [28].

3.1. Logic grid puzzles

A logic grid puzzle (also known as 'Zebra puzzle' or 'Einstein puzzle') consists of natural language sentences (from hereon referred to as 'clues') over a set of *entities* occurring in those sentences. For instance, our running example in Fig. 1 contains as second clue 'The person who chose arrabiata sauce is either Angie or Elisa' and (among others) the entities 'arrabiata sauce', 'Angie' and 'Elisa'.

The set of entities is sometimes left implicit if it can be derived from the clues, but often it is given in the form of a grid. Furthermore, in such a puzzle the set of entities is partitioned into equally-sized groups (corresponding to *types*); in our example, 'person' and 'sauce' are two such types. The goal of the puzzle is to find relations between each two types such that

- *Each clue is respected*;
- *Each entity of one type is matched with exactly one entity of the second type*, e.g., each person chose exactly one sauce and each sauce is linked to one person (this type of constraint will be referred to as *bijectivity*); and
- *The relations are logically linked*, e.g., if Angie chose arrabiata sauce and arrabiata sauce was paired with farfalle, then Angie must also have eaten farfalle (from now on called *transitivity*).

3.2. Typed first-order logic

Our constraint solving method is based on *typed first-order logic*. Part of the input is a logical vocabulary consisting of a set of types, (typed) constant symbols, and (typed) relation symbols with associated type signature (i.e., each relation symbol is typed $T_1 \times \cdots \times T_n$ with n types T_i).[2] For our running example, the constant symbol *Angie* of type *person* is linked using relation *chose* with signature *person* $\times$ *sauce* to the constant symbol *arrabiata* of type *sauce*.

A *first-order theory* is a set of sentences (well-formed variable-free first-order formulas [29] in which each quantified variable has an associated type), also referred to as constraints.

In Section 8, we explain how we to obtain a vocabulary, as well as a theory in a mostly automated way from the clues of a logic grid puzzle using Natural Language Processing. For now, we assume them to be given.

Since we work in a fixed and finite domain, the vocabulary, the interpretation of the types (the domains) and the constants are fixed. This justifies the following definitions:

Definition 1. A *(partial) interpretation* is a finite set of literals, i.e., expressions of the form $P(\overline{d})$ or $\neg P(\overline{d})$ where P is a relation symbol typed $T_1 \times \cdots \times T_n$ and $\overline{d}$ is a tuple of domain elements where each d_i is of type T_i.

Definition 2. A partial interpretation is *consistent* if it does not contain both an atom and its negation, it is called a *full* interpretation if it either contains $P(\overline{d})$ or $\neg P(\overline{d})$ for each well-typed atom $P(\overline{d})$.

For instance, in the partial interpretation

$$I_1 = \{chose(Angie, arrabiata), \neg chose(Elisa, arrabiata)\}$$

it is known that Angie had arrabiata sauce while Elisa did not. This partial interpretation does not specify whether or not Elisa ate Farfalle.

The syntax and semantics of first-order logic are defined as usual (see e.g. [29]) by means of a satisfaction relation $I \models T$ between first-order theories T and full interpretations I. If $I \models T$, we say that I is a model (or solution) of T.

Definition 3. A partial interpretation I_1 is *more precise* than partial interpretation I_2 (notation $I_1 \geq_p I_2$) if $I_1 \supseteq I_2$.

[2] We here omit function symbols since they are not used in this paper.

Informally, one partial interpretation is more precise than another if it contains more information. For example, the partial interpretation

$$I_2 = \{chose(Angie, arrabiata), \neg chose(Elisa, arrabiata), \neg chose(Damon, arrabiata)\}$$

is more precise than I_1 ($I_2 \geq_p I_1$).

For practical purposes, since variable-free literals are also sentences, we will freely use a partial interpretation as (a part of) a theory in solver calls or in statements of the form $I \wedge T \models J$, meaning that everything in J is a consequence of I and T, or stated differently, that J is less precise than any model M of T satisfying $M \geq_p I$.

In the context of first-order logic, the task of finite-domain constraint solving is better known as *model expansion* [30]: given a logical theory T (corresponding to the constraint specification) and a partial interpretation I with a finite domain (corresponding to the initial domain of the variables), find a model M more precise than I (a partial solution that satisfies T).

If P is a logic grid puzzle, we will use T_P to denote a first-order theory consisting of:

- One logical sentence for each clue in P;
- A logical representation of all possible bijection constraints;
- A logical representation of all possible transitivity constraints.

For instance, for our running example, some sentences in T_P are:

$$\neg chose(Claudia, puttanesca).$$

$$\forall s \in sauce : \exists p \in pasta : paired(s, p)$$

$$\forall s \in sauce : \forall p_1 \in pasta, \forall p_2 \in pasta : paired(s, p_1) \wedge paired(s, p_2) \Rightarrow p_1 = p_2.$$

$$\forall s \in : sauce : \forall p \in person, \forall c \in price : chose(p, s) \wedge payed(p, c) \Rightarrow priced(s, c).$$

The first of these sentences is a translation of the fourth clue of our running example ('Claudia did not choose puttanesca sauce'). The second and the third express that every sauce was paired to exactly one pasta (bijectivity), and the last expresses a transitivity constrained between *chose*, *payed*, and *priced*.

4. Problem definition

The overarching goal of this paper is to generate a sequence of small reasoning steps, each with an interpretable explanation. We first introduce the concept of an explanation of a single reasoning step, after which we introduce a cost function as a proxy for the interpretability of a reasoning step, and the cost of a sequence of such steps.

4.1. Explanation of reasoning steps

We assume that a theory T and an initial partial interpretation I_0 are given and fixed.

Definition 4. We define the *maximal consequence* of a theory T and partial interpretation I (denoted $max(I, T)$) as the most precise partial interpretation J such that $I \wedge T \models J$.

Phrased differently, $max(I, T)$ is the intersection of all models of T that are more precise than I; this is also known as the set of *cautious consequences* of T and I, and corresponds to ensuring *global consistency* in constraint solving. Algorithms for computing cautious consequences without explicitly enumerating all models exist, such as for instance the ones implemented in clasp [31] or IDP [32] (in the latter system the task of computing all cautious consequences is called *optimal-propagate* since it performs the strongest propagation possible).

Weaker levels of propagation consistency can be used as well, leading to a potentially smaller maximal consequence interpretation $max_{otherConsistency}(I, T)$. The rest of this paper assumes we want to construct a sequence that starts at I_0 and ends at $max(I_0, T)$ for some consistency algorithm, i.e., that can explain all computable consequences of T and I_0.

Definition 5. A *sequence of incremental partial interpretations* of a theory T with initial partial interpretation I_0 is a sequence $\langle I_0, I_1, \ldots, I_n = max(I_0, T)\rangle$ where $\forall i > 0, I_{i-1} \leq_p I_i$ (i.e., the sequence is precision-increasing).

The goal of our work is not just to obtain a sequence of incremental partial interpretations, but also for each incremental step $\langle I_{i-1}, I_i\rangle$ we want an explanation (E_i, S_i) that justifies the newly derived information $N_i = I_i \setminus I_{i-1}$. When visualized, such as in Fig. 1, it will show the user precisely which information and constraints were used to derive a new piece of information.

Definition 6. Let I_{i-1} and I_i be partial interpretations such that $I_{i-1} \wedge T \models I_i$. We say that (E_i, S_i, N_i) *explains* the derivation of I_i from I_{i-1} if the following hold:

- $N_i = I_i \setminus I_{i-1}$ (i.e., N_i consists of all newly defined facts),
- $E_i \subseteq I_{i-1}$ (i.e., the explaining facts are a subset of what was previously derived),
- $S_i \subseteq T$ (i.e., a subset of the constraints is used), and
- $S_i \cup E_i \models N_i$ (i.e., all newly derived information indeed follows from this explanation).

The problem of simply checking whether (E_i, S_i, N_i) explains the derivation of I_i from I_{i-1} is in co-NP since this problem can be performed by verifying that $S_i \wedge \neg N_i$ has no models more precise than E_i. It is hence an instance of the negation of a first-order model expansion problem [33].

Part of our goal of finding easy to interpret explanations is to avoid redundancy. That is, we want a non-redundant explanation (E_i, S_i, N_i) where none of the facts in E_i or constraints in S_i can be removed while still explaining the derivation of I_i from I_{i-1}; in other word: the explanation must be *subset-minimal.*

Definition 7. We call (E_i, S_i, N_i) a *non-redundant explanation of the derivation of I_i from I_{i-1}* if it explains this derivation and whenever $E' \subseteq E_i$; $S' \subseteq S_i$ while (E', S', N_i) also explains this derivation, it must be that $E_i = E', S_i = S'$.

Definition 8. A *non-redundant explanation sequence* is a sequence

$$\langle (I_0, (\emptyset, \emptyset, \emptyset)), (I_1, (E_1, S_1, N_1)), \ldots, (I_n, (E_n, S_n, N_n)) \rangle$$

such that $(I_i)_{i \leq n}$ is a sequence of incremental partial interpretations and each (E_i, S_i, N_i) explains the derivation of I_i from I_{i-1}.

4.2. Interpretability of reasoning steps

While subset-minimality ensures that an explanation is non-redundant, it does not quantify how *interpretable* an explanation is. This quantification is problem-specific and is often subjective. In this paper, we will assume the existence of a cost function $f(E_i, S_i, N_i)$ that quantifies the interpretability of a single explanation.

In line with the goal of 'simple enough for a person to understand' and Occam's Razor.

4.3. Interpretability of a sequence of reasoning steps

In its most general form, we would like to optimize the understandability of the entire sequence of explanations. While quantifying the interpretability of a single step can be hard, doing so for a sequence of explanations is even harder. For example, is it related to the most difficult step or the average difficulty, and how important is the ordering within the sequence? As a starting point, we here consider the total cost to be an aggregation of the costs of the individual explanations, e.g. the average or maximum cost.

Definition 9. Given a theory T and an initial partial interpretation I_0, the *explanation-production problem* consists of finding a non-redundant explanation sequence

$$\langle (I_0, (\emptyset, \emptyset, \emptyset)), (I_1, (E_1, S_1, N_1)), \ldots, (I_n, (E_n, S_n, N_n)) \rangle$$

such that a predefined aggregate over the sequence $(f(E_i, S_i, N_i))_{i \leq n}$ is minimized.

Example aggregation operators are *max*() and *average*(), which each have their peculiarities: the *max*() aggregation operator will minimize the cost of the most complicated reasoning step, but does not capture whether there is one such step used, or multiple. Likewise, the *average*() aggregation operator will favor many simple steps, including splitting up trivial steps into many small components if the constraint abstraction allows this. Even for a fixed aggregation operator, the problem of holistically optimizing a sequence of explanation steps is much harder than optimizing the cost of a single reasoning step, since there are exponentially more sequences.

5. Nested explanations

If each explanation in the sequence is non-redundant, this means the steps cannot be further broken up in smaller substeps. Yet, in our earlier work we noticed that some explanations were still quite hard to understand, mainly since a clue had to be combined with implicit constraints and a couple of previously derived facts. All these things *together* implied a consequence, and they had to be taken into account at once. Such steps turned out to be too complicated to be understood easily and thus require being explained in more detail.

An example of a more difficult step is depicted at the top of Fig. 3. It uses a disjunctive clue ('The person who ordered Rotini is either the person who paid \$8 more than Damon or the person who paid \$8 less than Damon'), in combination with three previously derived facts to derive that Farfalle does not cost \$8. This derivation is non-trivial when only looking at the highlighted clue. It turns out that the derivation can be explained in a step-wise manner using only implicit constraints with the help of reasoning by contradiction:

- If Farfalle did cost \$8, then (since Damon did not eat Farfalle), Damon did not pay \$8;
- If Farfalle costs \$8, then it does not cost \$16;
- Since Farfalle does not cost \$16 and neither does Capellini or Tagliolini, Rotini must cost \$16;
- However, the fact that Rotini costs \$16, while Damon did not pay \$8 is in contradiction with the clue in question;
- Hence, Farfalle can not cost \$8.

We hence wish to provide a further, *nested* explanation of such difficult reasoning steps. We believe that an explanation using contradiction is a good tool for this for two reasons: *(i)* it is often used by people when solving puzzles, as well as by mathematicians when proving theorems; and *(ii)* adding the negation of a derived fact such as 'Farfalle does not cost \$8', allows us to generate a new sequence of non-redundant explanations up to inconsistency, hence reusing the techniques from the previous section. This novel approach allows us to provide a mechanism for *zooming in* into the more difficult explanation steps of the explanation sequence.

5.1. Nested explanation of a reasoning step

We propose the following principles for what constitutes a meaningful and simple nested explanation, given a non-trivial explanation (E, S, N):

- a nested explanation starts from the explaining facts E, augmented with the counterfactual assumption of a newly derived fact $n \in N$;
- at each step, it only uses clues from S;
- each step is easier to understand (has a *strictly* lower cost) than the parent explanation which has cost $f(E, S, N)$;
- from the counterfactual assumption, a contradiction is derived.

Note that if an explanation step derives multiple new facts, e.g. $|N| > 1$, then we can compute a nested explanation for each $n_i \in N$.

More formally, we define the concept of *nested explanation* as follows:

Definition 10. The *nested explanation* problem consists of — given a non-redundant explanation (E, S, N), and a newly derived fact $n \in N$ — finding a non-redundant explanation sequence

$$\langle\ (I'_0, (\emptyset, \emptyset, \emptyset)),\ (I'_1, (E'_1, S'_1, N'_1)), \ldots,\ (I'_n, (E'_n, S'_n, N'_n))\ \rangle$$

such that:

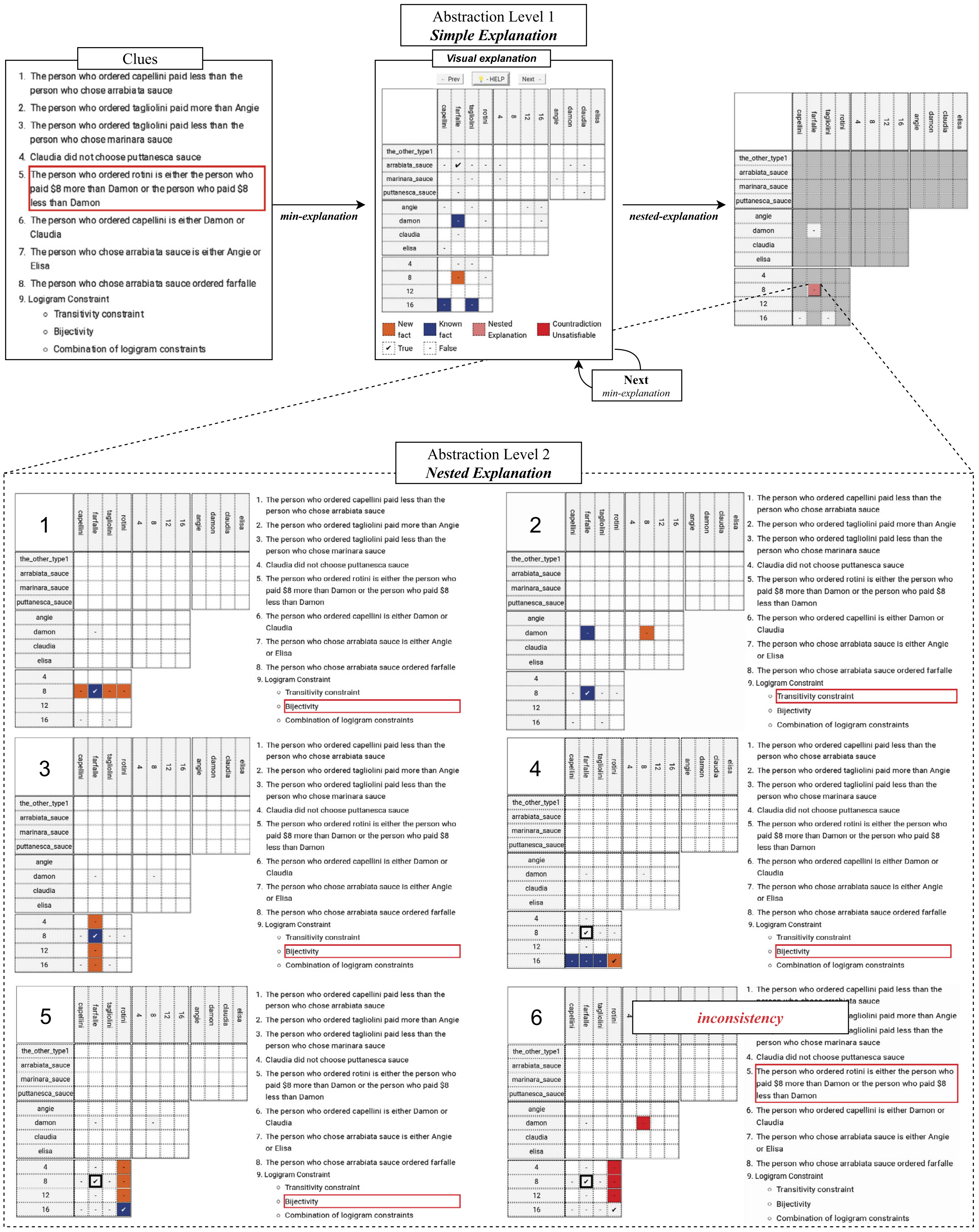
- I'_0 is the partial interpretation $\{\neg n_i \wedge E\}$;
- $S'_i \subseteq S$ for each i;
- $f(E'_i, S'_i, N'_i) < f(E, S, N)$ for each i;
- I'_n is inconsistent; and
- a predefined aggregate over the sequence $\left(f(E'_i, S'_i, N'_i)\right)_{i \leq n}$ is minimized.

We can hence augment each explanation (E, S, N) with a set of nested explanations if they exist. We next discuss algorithms for computing explanations and nested explanations.

6. Explanation-producing search

In this section, we tackle the goal of searching for a non-redundant explanation sequence that is as simple as possible to understand.

Ideally, we could generate all explanations of each fact in $max(I_0, T)$, and search for the lowest scoring sequence among those explanations. However, the number of explanations for each fact quickly explodes with the number of constraints, and is hence not feasible to compute. Instead, we will iteratively construct the sequence, by generating candidates for a given partial interpretation and searching for the smallest one among those.


Clues

1. The person who ordered capellini paid less than the person who chose arrabiata sauce
2. The person who ordered tagliolini paid more than Angie
3. The person who ordered tagliolini paid less than the person who chose marinara sauce
4. Claudia did not choose puttanesca sauce
5. The person who ordered rotini is either the person who paid $8 more than Damon or the person who paid $8 less than Damon
6. The person who ordered capellini is either Damon or Claudia
7. The person who chose arrabiata sauce is either Angie or Elisa
8. The person who chose arrabiata sauce ordered farfalle
9. Logigram Constraint
   - Transitivity constraint
   - Bijectivity
   - Combination of logigram constraints

Fig. 3. A difficult explanation step, including its nested explanation.

6.1. Sequence construction

We aim to minimize the cost of the explanations of the sequence, measured with an aggregate over individual explanation costs $f(E_i, S_i, N_i)$ for some aggregate like *max*() or *average*(). The cost function f could for example be a weighted sum of the cardinalities of E_i, S_i and N_i; in Section 7, we discuss a concrete cost function for the use case of logic grid puzzles.

Instead of globally optimizing the aggregated sequence cost, we encode the knowledge that we are seeking a sequence of small explanations in our algorithm. Namely, we will greedily and incrementally build the sequence, each time searching for the lowest scoring next explanation, given the current partial interpretation. Such an explanation always exists since the end point of the explanation process $max(I_0, T)$ only contains consequences of I_0 and T.

Algorithm 1 formalizes the greedy construction of the sequence, which determines $I_{end} = max(I_0, T)$ through propagation and relies on a *min-explanation*(I, T) function to find the next cost-minimal explanation.

Algorithm 1: High-level greedy sequence-generating algorithm.

```
  Input  : A initial interpretation $I_0$ and a set of constraints $T$
  Output: An explanation sequence $Seq$
1 $I_{end} \leftarrow$ propagate($I_0 \wedge T$)
2 Seq $\leftarrow$ empty sequence
3 $I \leftarrow I_0$
4 while $I \neq I_{end}$ do
5     $(E, S, N) \leftarrow$min-explanation($I, T$)
6     append $(E, S, N)$ to Seq
7     $I \leftarrow I \cup N$
8 end
```

6.2. Candidate generation

The main challenge is finding the lowest scoring explanation, among all reasoning steps that can be applied for a given partial interpretation I. We first study how to *enumerate* a set of candidate non-redundant explanations given a set of constraints.

For a set of constraints T, we can first use propagation to get the set of new facts that can be derived from a given partial interpretation I and the constraints T. For each new fact n not in I, we wish to find a non-redundant explanation $(E \subseteq I, S \subseteq T, \{n\})$ that explains n (and possibly explains more). Recall from Definition 8 that this means that whenever one of the facts in E or constraints in T is removed, the result is no longer an explanation. We now show that this task is equivalent to finding a Minimal Unsat Core (or Minimal Unsat Subset, MUS). To see this, consider the theory

$$I \wedge T \wedge \neg n.$$

This theory surely is unsatisfiable since n is a consequence of I and T. Furthermore, under the assumption that $I \wedge T$ is consistent (if it were not, there would be nothing left to explain), *any* unsatisfiable subset of this theory contains $\neg n$. We then see that each unsatisfiable subset of this theory is of the form $E \wedge S \wedge \neg n$ where $(E, S, \{n\})$ is a (not necessarily redundant) explanation of the derivation of $\{n\}$ from I. Vice versa, each explanation of $\{n\}$ corresponds to an unsatisfiable subset. Thus, the *minimal* unsatisfiable subsets (MUS) of the above theory are in one-to-one correspondence with the non-redundant explanations of n, allowing us to use existing MUS algorithms to search for non-redundant explanations.

We must point out that MUS algorithms typically find *an* unsatisfiable core that is *subset-minimal*, but not *cardinality-minimal*. That is, the unsat core can not be reduced further, but there could be another minimal unsat core whose size is smaller. That means that if size is taken as a measure of simplicity of explanations, we do not have the guarantee to find the optimal ones. And definitely, when a cost function kicks, optimality is also not guaranteed.

Algorithm 2 shows our proposed algorithm. The key part of the algorithm is on line 4 where we find an explanation of a single new fact n by searching for a MUS that includes $\neg n$. We search for subset-minimal unsat cores to avoid redundancy in the explanations. Furthermore, once a good explanation (E, S, N) is found, we immediately explain all implicants of E and S. In other words: we take N to be subset-maximal. The reason is that we assume that all derivable facts that use the same part of the theory and the same part of the previously derived knowledge probably require similar types of reasoning and it is thus better to consider them at once. Thus, we choose to generate candidate explanations at once for all implicants of (E, S) on line 7. Note that the other implicants $N \setminus \{n\}$ may have simpler explanations that may be found later in the for loop, hence we do not remove them from J.

We assume the use of a standard MUS algorithm, e.g., one that searches for a satisfying solution and if a failure is encountered, the resulting Unsat Core is shrunk to a minimal one [19]. While computing a MUS may be computationally demanding, it is far less demanding than enumerating all MUS's (of arbitrary size) as candidates.

Algorithm 2: candidate-explanations(I, T).

```
Input  : A partial interpretation $I$ and a set of constraints $T$
Output: A set of candidate explanations $Candidates$
1  Candidates ← {}
2  $J \leftarrow$ propagate($I \wedge T$)
3  for $n \in J \setminus I$ do
       // Minimal expl. of each new fact:
4      $X \leftarrow MUS(\neg n \wedge I \wedge T)$
5      $E \leftarrow I \cap X$                              // facts used
6      $S \leftarrow T \cap X$                        // constraints used
7      $N \leftarrow$ propagate($E \wedge S$)          // all implied facts
8      add ($E, S, N$) to Candidates
9  end
10 return $Candidates$
```

6.3. Cost functions and cost-minimal explanations

We use Algorithm 2 to generate candidate explanations, but in general our goal is to find cost-minimal explanations. In the following, we assume fixed a cost function f that returns a score for every possible explanation (E, S, N).

To guide the search to cost-minimal MUS's, we use the observation that typically a small (1 to a few) number of constraints is sufficient to explain the reasoning. A small number of constraints is also preferred in terms of easy to understand explanations, and hence have a lower cost. For this reason, we will not call *candidate-explanations* with the full set of constraints T, but we will iteratively grow the number of constraints used.

We make one further assumption to ensure that we do not have to search for candidates for all possible subsets of constraints. The assumption is that we have an optimistic estimate g that maps a subset S of T to a real number such that $\forall E, N, S : g(S) \leq f(E, S, N)$. This is for example the case if f is an additive function, such as $f(E, S, N) = f_1(E) + f_2(S) + f_3(N)$ where $g(S) = f_2(S)$ assuming f_1 and f_3 are always positive.

We can then search for the smallest explanation among the candidates found, by searching among increasingly worse scoring S as shown in Algorithm 3. This is the algorithm called by the iterative sequence generation (Algorithm 1).

Algorithm 3: min-explanation(I, T).

```
Input  : A partial interpretation $I$ and a set of constraints $T$
Output: A non-redundant explanation $(E, S, N)$ with $E \subseteq I$, $S \subseteq T$, $E \cup S \models N$
1  $N \leftarrow$ propagate($I \wedge T$)
2  $best \leftarrow (I, T, N)$
3  for $S \subseteq T$ ordered ascending by $g(S)$ do
4      if $g(S) > f(best)$ then
5          break
6      end
7      cand ← best explanation from candidate-explanations($I$, $S$)
8      if $f(cand) < f(best)$ then
9          $best \leftarrow cand$
10     end
11 end
12 return $best$
```

Every time *min-explanation*(I, T) is called with an updated partial interpretation I the explanations should be regenerated. The reason is that for some derivable facts n, there may now be a much easier and cost-effective explanation of that fact. There is one benefit in caching the *Candidates* across the different iterations, and that is that in a subsequent call, the cost of the most cost-effective explanation that is still applicable can be used as a lower bound to start from. Furthermore, in practice, we cache all candidates and when we (re)compute a MUS for a fact n, we only store it if it is more cost-effective than the best one we have previously found for that fact, across the different iterations.

6.4. Searching nested explanations

We extend Algorithm 1 to generate new candidate explanations with support for nested explanations as introduced in Section 5. Fundamentally, the generated candidate explanations are further decomposed in a nested explanation sequence provided that the sequence is easier than the candidate explanation according to the defined cost function f. We assess the possibility for a nested explanation for every newly derived fact in the candidate explanation. Similar to Algorithm 1, Algorithm 4 exploits the *min-explanation* function to generate the candidate nested explanations. The only difference is that after computing each explanation step, also a call to *nested-explanations* (which is found in Algorithm 5) is done to generate a nested sequence.

The computation of a nested explanation as described in Algorithm 5 also reuses *min-explanation*; but, the main differences with the high level explanation generating algorithm come from the fact that the search space for next easiest explanation-step is bounded by *(i)* the input explanation: it can use only constraints (and facts) from the original explanation, and *(ii)* the cost of the parent explanation is an upper bound on the acceptable costs at the nested level.

Given an explanation step (E, S, N) and a newly derived fact $n \in N$ for which we want a more detailed explanation, Algorithm 5 first constructs a partial interpretation I' formed by the facts in E and the negated new fact n. Then, we gradually build the sequence by adding the newly found explanations (E', S', N') as long as the interpretation is consistent and the explanation is easier than explanation step (E, S, N) (this is in line with Definition 10 and serves to avoid that the nested explanation is simply a single-step explanation that is equally difficult as the original step.

While Algorithm 5 tries to find a nested explanation sequence for each derived fact, it will not find one for each fact due to the if-check at Line 8. This check is present to avoid that the nested explanation is as difficult or harder than the high-level step it aims to clarify (also see Definition 10, where the third bullet point explicitly forbids such nested justifications). This check can kick in for two different reasons. The first reason is that the explanation step at the main level is simply too simple to be further broken up in pieces. For instance the explanation of Fig. 1 is of that kind: it uses a single bijectivity constraint with a single previously derived fact. Breaking this derivation up in strictly smaller parts would thus not be helpful.

This phenomenon can also occur for difficult steps: sometimes the best nested explanation of a difficult explanation step contains a step that is as difficult or harder than the high-level step itself. In that case, this is a sign that reasoning by contradiction is not simplifying matters in this step and other methods should be explored to further explain it.

Algorithm 4: greedy-explain(I_0, T).

```
   Input : A initial interpretation $I_0$ and a set of constraints $T$
   Output: An explanation sequence $Seq$
1  $I_{end} \leftarrow$ propagate($I_0 \wedge T$)
2  $Seq \leftarrow$ empty sequence
3  $I \leftarrow I_0$
4  while $I \neq I_{end}$ do
5      $(E, S, N) \leftarrow$min-explanation$(I, T)$
6      $nested \leftarrow$ nested-explanations$(E, S, N)$
7      append $((E, S, N), nested)$ to $Seq$
8      $I \leftarrow I \cup N$
9  end
```

Algorithm 5: nested-explanations(E, S, N).

```
   Input : Explanation facts $E$, constraints $S$ and newly derived facts $N$
   Output: A collection of nested explanation sequences $nested\_explanations$
1  $nested\_explanations \leftarrow$ empty set
2  for $n \in N$ do
3      $store \leftarrow true$
4      $nested\_seq \leftarrow$ empty sequence
5      $I' \leftarrow E \wedge \neg\{n\}$
6      while $consistent(I')$ do
7          $(E', S', N') \leftarrow$min-explanation$(I', S)$
8          if $f(E', S', N') \geq f(E, S, N)$ then
9              $store \leftarrow false$
10             break
11         end
12         append $(E', S', N')$ to $nested\_seq$
13         $I' \leftarrow I' \cup N'$
14     end
15     if $store$ then
           // only when all steps simpler than (E,S,N)
16         append $nested\_seq$ to $nested\_explanations$
17     end
18 end
19 return $nested\_explanations$
```

7. Explanations for logic grid puzzles

We instantiated the above described algorithm in the context of logic grid puzzles. In that setting, for T, we take T_P for some puzzle P, as described in Section 3. There are three types of constraints in T: *transitivity* constraints, *bijectivity* constraints and *clues*, where the first two follow the same structure in every puzzle and the clues are obtained in a mostly automatic way (see Section 8). Before defining a cost-function, and the estimation for g used in our implementation, we provide some observations that drove our design decision.

Observation 1: Propagations from a single implicit constraint are very easy to understand Contrary to the clues, the implicit constraints (transitivity/bijectivity) are very limited in form and propagations over them follow well-specified patterns. For instance in the case of bijectivity, a typical pattern that occurs is that when $X-1$ out of X possible values for a given function have been derived not to be possible, it is propagated that the last value should be true; this is visualized for instance in Fig. 1. Hence, in our implementation, we ensure that they are always performed first. Stated differently, g and f are designed in such a way that $g(S_1) \geq f(E, S_2, N)$ whenever S_2 consists of only one implicit constraint and S_1 does not.

Observation 2: Clues rarely propagate by themselves We observed that the automatically obtained logic representation of clues usually has quite weak (unit) propagation strength in isolation. This is not a property of the clues, but rather of the final obtained translation. As an example, consider the following sentence: 'The person who ordered capellini is either Damon or Claudia'. From this, a human reasoner might conclude that Angie did not order capellini. However, the (automatically) obtained logical representation is

$$\exists p \in person : ordered(p, capellini) \land (p = Damon \lor p = Claudia).$$

This logic sentence only entails that Angie did not order capellini *in conjunction with the bijectivity constraint on ordered*. In the natural language sentence, this bijectivity is implicit by the use of the article 'The' which entails that there is a unique person who ordered capellini.

We observed that there is rarely any propagation from sole clues, and that only few implicit constraints are active together with a clue at any time. Because of this last observation, in our implementation for logic grid puzzles we decided not to consider all subsets of implicit constraints in combination with a clue as candidate sets S in Line 3 in Algorithm 3. Instead, we combine each clue with the entire set of all implicit constraints, subsequently counting on the non-redundance of the explanation (the subset-minimality of the core) to eliminate most of the implicit constraints since they are not used anyway.

Observation 3: Clues are typically used independently from other clues A next observation is that in all the puzzles we encountered, human reasoners never needed to combine two clues in order to derive new information and that when such propagations are possible, they are quite hard to explain, and can be split up into derivations containing only single clues. The latter is of course not guaranteed, since one can artificially devise disjunctive clues that do not allow propagation by themselves. Our algorithms are built to handle this case as well, but it turned out to be not necessary in practice: in the puzzles we tested, we never encountered an explanation step that combined multiple clues.

Observation 4: Previously derived facts are easier to use than clues or implicit constraints Our final observation that drove the design of the cost functions is that using previously derived facts is often easier than using an extra clue or implicit constraint. This might be due to the fact that previously derived facts are of a very simple nature while, even implicit constraints contain quantification and are thus harder to grasp. An additional reason for this perceived simplicity is that the derived facts are visualized in the grid.

A cost function With these four observations in mind, we devised f and g as follows (where $nc(S)$ denotes the number of *clues* (not: implicit constraints) in S, M and N are explanation parameters):

$$f(E, S, N) = basecost(S) + |E| + N \cdot |S|$$

$$g(S) = basecost(S) = \begin{cases} 0 & \text{if } |S| = 1 \text{ and } nc(S) = 0 \\ M & \text{if } |S| > 1 \text{ and } nc(S) = 0 \\ M \cdot nc(S) & \text{otherwise} \end{cases}$$

The parameter M is taken here to be larger than any reasonable explanation size $|E| + N \cdot |S|$ and N to be relatively small in comparison, but slightly larger than a few facts $|E|$. In our experiments, we use the combination $M = 100$ and $N = 5$ which provided good explanation sequences for the tested puzzles. The effect of this, is that we can generate our subsets S in Line 3 of Algorithm 3 in the following order:

- First all S containing exactly one implicit constraint.
- Next, all S containing exactly all implicit constraints and (optionally) exactly one clue.
- Finally, all clue pairs, triples etc. though in practice this is never reached.

Summarized, our instantiation for logic grid puzzles differs from the generic methods developed in the previous section in that it uses a domain-specific optimization function f and does not consider all S in Line 3, but only promising candidates based on our observations.

For the complete non-redundant explanation sequence our tool produces on the running example using these scoring functions, we refer to http://bartbog.github.io/zebra/pasta. An example of the hardest derivation we encountered (with cost 108), as well as its nested explanation, is depicted in Fig. 3. It uses several bijectivity constraints for uniqueness of persons, but also for reasoning on the relation between costs and types of pasta, in combination with a clue and three assumptions.

8. Logic grid puzzles: from natural language clues to typed first-order logic

The demo system we developed, called ZebraTutor, is named after Einstein's zebra puzzle, which is an integrated solution for solving logic grid puzzles, and for explaining, *in a human-understandable way*, how the solution can be obtained from the clues. The input to ZebraTutor is a plain English language representation of the clues and the names of the *entities* that make up the puzzle, e.g. 'Angie', 'arrabiata'.

In typical logic grid puzzles, the entity names are present in the grid that is supplied with the puzzle. For some puzzles, not all entities are named or required to know in advance; a prototypical example is Einstein's Zebra puzzle, which ends with the question 'Who owns the zebra?', while the clues do not name the zebra entity and the puzzle can be solved without knowledge of the fact there is a zebra in the first place.

The complete specification undergoes the following steps, starting from the input:

- **A** **Part-Of-Speech tagging**: A Part-Of-Speech (POS) tag is associated with each word using an out-of-the-box POS tagger [34].
- **B** **Chunking and lexicon building**: A problem-specific lexicon is constructed.
- **C** **From chunked sentences to logic**: Using a custom grammar and semantics a logical representation of the clues is constructed
- **D** **From logic to a complete IDP specification**: The logical representation is translated into the IDP language and augmented with logic-grid-specific information.
- **E** **Explanation-producing search in IDP**: This is the main contribution of this paper, as explained in Section 6.
- **F** **Visualization**: The step-by-step explanation is visualized.

The first three of these steps are related to Natural Language Processing (NLP) and are detailed in section 8.1 hereafter. Next, we explain in section 8.2 how the IDP specification formed in step *Step D* is used to generate the explanations and visualizations in steps *Step E* and *Step F* respectively. An online demo of our system can be found on http://bartbog.github.io/zebra, containing examples of all the steps (bottom of demo page).

8.1. Natural Language Processing

Step A. POS tagging

The standard procedure in Natural Language Processing is to start by tagging each word with its estimated Part-Of-Speech tag (POS tag). We use the standard English Penn treebank Pos tagset [34] together with NLTK's built-in perceptron tagger[3] as POS tagger. It uses a statistical inference mechanism trained on a standard training set from the Wall Street Journal. Since any POS-tagger can make mistakes, we manually verify and correct the assigned POS-tags to make sure that all of the puzzle's entities are tagged as noun.

Step B. Chunking and lexicon building

From a natural language processing point of view, the hardest part is step B: automatically deriving the lexicon and building the grammar. The lexicon assigns a role to different sets of words, while the grammar is a set of rules describing how words can be combined into sentences. The goal of this second step is to group the POS-tagged words of the clues into *chunks* that are tagged with lexical categories of which 3 are puzzle-specific: proper nouns for the individual entities that are central to the puzzle, other nouns to groups of entities (like *pasta, sauce*) and transitive verbs that link two entities to each other (e.g. Claudia did not *choose* puttanesca sauce). The other categories are determiner, number, preposition, auxiliary verb etc. and contain a built-in list of possible members. We refer to [35] for full details on the categories.

This process of grouping words is referred to as *chunking*. We use NLTK and a custom set of regular expressions for chunking the proper nouns and different types of transitive verbs. The result is a lexicon where each word or set of words (chunk) is assigned a role based on the POS tags. On top of these roles, we defined a puzzle-independent grammar in the Blackburn and Bos framework [36,37]. The grammar itself was created based on 10 example training puzzles, and tested on 10 different puzzles to ensure genericity [35].

Step C. From chunked sentences to logic

Next in *Step C*, the lexicon, partly problem agnostic and partly puzzle-specific, is fed into a type-aware variant of the semantical framework of Blackburn & Bos [36,37], which translates the clues into Discourse Representation Theory [38]. The

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

typed extension allows us to discover the case where different verbs are used as synonyms for the same inherent relation between two types, e.g. '$ate(person, pasta)$' and '$ordered(person, pasta)$'.

In our system, this is a semi-automated method that suggests a lexicon and lets a user modify and approve it, to compensate for possible 'creativity' of the puzzle designers who tend to insert ambiguous words, or use implicit background knowledge such as using 'in the morning' when there is only one time slot before 12:00.

8.2. From logic to visual explanations

Once the types are built, we generate a complete specification which is used by the reasoner, the IDP system [32], to solve the problem.

Step D. From logic to a complete IDP specification

From the types built in *Step C*, we construct the IDP vocabulary containing: all the types and a relation for each transitive verb or preposition. For instance, if the clues contain a sentence 'Claudia did not choose puttanesca sauce', then the vocabulary will contain a binary relation chose(.,.) with the first argument of type *person* and the second argument of type *sauce*.

After the vocabulary, we construct IDP theories: we translate each clue into IDP language, and we add implicit constraints and present in logic grid puzzles. The implicit constraints are stemming from the translation of the clues: our translation might generate multiple relations between two types. For instance, if there are clues 'The person who ate taglioni paid more than Angie' and 'The person who ordered farfalle chose the arrabiata sauce', then the translation will create two relations *ate* and *ordered* between persons and pasta. However we know that there is only one relation between two types, hence we add a theory containing synonymy axioms; for this case concretely:

$$\forall\, x \in person\ \forall\, y \in pasta : ate(x, y) \leftrightarrow ordered(x, y)$$

Similarly, if two relations have an inverse signature, they represent inversion functions, for instance *liked_by* and *ordered* in the clues 'Taglioni is liked by Elisa' and 'Damon ordered capellini'. In this case we add constraints of the form

$$\forall\, x \in person\ \forall\, y \in pasta : liked\_by(y, x) \leftrightarrow ordered(x, y)$$

Next, we refer back to the end of section 3 for examples of the bijectivity and transitivity axioms that link the different relations.

The underlying solver, IDP [32] uses this formal representation of the clues both to solve the puzzle and to explain the solution. We chose the IDP system as an underlying solver since it natively offers different inference methods to be applied on logic theories, including model expansion (searching for solutions), different types of propagation (we used optimal-propagate here to find $max(I, T)$), unsat-core extraction and offers a Lua [39] interface to glue these inference steps together seamlessly [32].

9. Experiments

Using logic grid puzzles as a use-case, we validate the feasibility of finding non-redundant explanation sequences and generating nested explanation sequences. As data, we use puzzles from Puzzle Baron's Logic Puzzles Volume 3 [40]. The first 10 puzzles were used to construct the grammar; the next 10 to test the genericity of the grammar. Our experiments below are on test puzzles only; we also report results on the *pasta* puzzle, which was sent to us by someone who did not manage to solve it himself.

As constraint solving engine, we use IDP [32] due to the variety of inference methods it supports natively. The algorithms themselves are written in embedded LUA, which provides an imperative environment inside the otherwise declarative IDP system. The code was not optimized for efficiency and can at this point not be used in an interactive setting, as it takes between 15 minutes to a few hours to fully explain a logic grid puzzle. Experiments were run on an Intel(R) Xeon(R) CPU E3-1225 with 4 cores and 32 Gb memory, running linux 4.15.0 and IDP 3.7.1. The code for generating the explanations of the puzzles in the experiments is available at [41].

9.1. Sequence composition

We first investigate the properties of the puzzles and the composition of the resulting sequence explanations. The results are shown in Table 1. The first column is the puzzle identifier, where the puzzle identified as p is the pasta puzzle, our running example. The next 3 columns show the properties of each puzzle: $|type|$ is the number of types (e.g. person, sauce) while $|dom|$ is the number of entities of each type and $|grid|$ is the number of cells in the grid, i.e. the number of literals in the maximal consequence interpretation $I_n = max(\emptyset, T)$. Coincidentally, almost all the puzzles have 4 types with domain size 5, hence 150 cells, except for the pasta puzzle which has a domain size of 4, thus 96 cells.

Table 1
Properties of the puzzles, explanation sequences and constraints used in the explanations.

p	|types|	|dom|	|grid|	# steps	$\overline{cost}$	1 bij.	1 trans.	1 clue	1 clue+i.	mult i.	mult c.
1	4	5	150	112	27.06	31.25%	50.00%	0.89%	17.85%	0%	0%
2	4	5	150	119	27.77	23.53%	57.14%	1.68%	17.64%	0%	0%
3	4	5	150	110	23.93	32.73%	51.82%	0%	15.46%	0%	0%
4	4	5	150	115	24.54	27.83%	55.65%	2.61%	13.92%	0%	0%
5	4	5	150	122	24.97	24.59%	59.02%	0.82%	15.58%	0%	0%
6	4	5	150	115	22.58	26.96%	58.26%	2.61%	12.18%	0%	0%
7	4	5	150	110	26.79	35.45%	46.36%	0.91%	17.27%	0%	0%
8	4	5	150	118	26.81	33.90%	47.46%	3.39%	15.25%	0%	0%
9	4	5	150	114	24.75	28.95%	54.39%	3.51%	13.16%	0%	0%
p	4	4	96	83	34.45	33.73%	40.96%	1.20%	21.69%	2.41%	0%

Table 2
Statistics on number of previously derived facts $|E|$ used in the explanation steps.

p	**average nr. of facts used**					**% of explanations with a clue that use # facts**				
	all	Bij.	Trans.	Clue	multi i.	0 facts	1 facts	2 facts	3 facts	>3 facts
1	1.84	2.37	2.00	0.52	-	66.67%	28.57%	0%	0%	4.76%
2	1.85	2.50	2.00	0.61	-	47.83%	47.83%	0%	4.35%	0%
3	1.84	2.33	2.00	0.24	-	82.35%	11.76%	5.88%	0%	0%
4	1.89	2.50	2.00	0.47	-	68.42%	15.79%	15.79%	0%	0%
5	1.85	2.50	2.00	0.35	-	65.00%	35.00%	0%	0%	0%
6	1.84	2.35	2.00	0.29	-	76.47%	17.65%	5.88%	0%	0%
7	1.88	2.31	2.00	0.75	-	55.00%	25.00%	10.00%	10.00%	0%
8	1.86	2.58	2.00	0.18	-	81.82%	18.18%	0%	0%	0%
9	1.85	2.45	2.00	0.32	-	78.95%	15.79%	0%	5.26%	0%
p	1.73	2.07	2.00	0.53	4.00	68.42%	21.05%	0%	10.53%	0%

Columns 5 and 6 show the amount of steps (#$steps$) in the explanation sequences found, and $\overline{cost}$ is the average cost of an explanation step in the puzzle. The number of inference steps is around 110-120 for all but the pasta puzzle, which is related to the grid size.

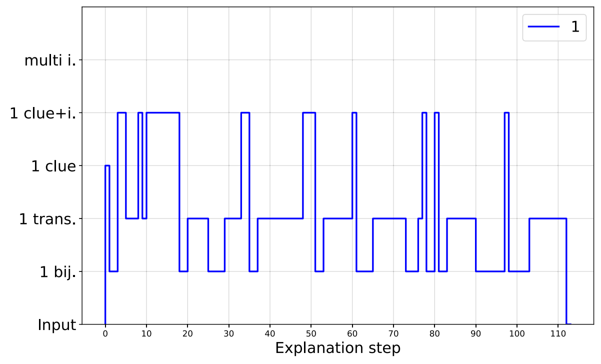
The rest of the columns investigate the proportion of inference steps in the explanation sequence, e.g. the trivial steps using just one bijection constraint (**1 bij.**), one transitivity constraint (**1 trans.**) or one clue (**1 clue**) and no other constraints; and the more complex inference steps using one clue and some implicit (bijectivity or transitivity) constraint (**1 clue+i**), multiple implicit constraints (**mult i.**). We can observe (see 7) around 30% of steps are simple bijectivity steps (e.g. completing a row or column in one relation), around 50% are transitivity steps (except for the pasta puzzle) and up to 3% use just a single clue (see 7). The majority of the steps involving clues use a more complex combination of a clue with other constraints. We see that while our method can combine multiple clues, the explanations generated never require combining multiple clues in one inference step (**mult c.** always 0, see 7), that is, the method always find simpler steps involving just one clue. Also notably, the puzzles from the booklet never require combining implicit constraints, while the harder pasta puzzle does. In general, less than 1/5th of the explanations actually need to use a clue or a combination of a clue and implicit constraints. Note that however, for puzzle 3, it is not possible to find new facts based on clue information only, it has to be combined with 1 or multiple constraints.

9.2. Sequence progression

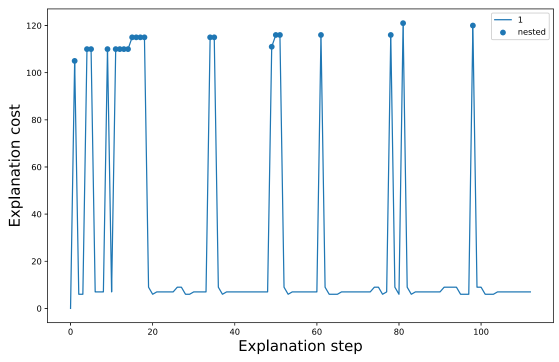
The left side of Fig. 4 shows a visualization of the type of explanation used in each of the explanation steps for the hardest puzzles 1,2 and p (puzzles with the highest average step cost). We can see that typically at the beginning of the sequence, individual clues (3rd line) and some individual bijectivity (1st line) and transitivity (2nd line) constraints are used, i.e., trivial ones. This is then followed by a series of clues that also involve bijectivity/transitivity constraints, after which a large fraction of the table can be completed with bijectivity/transitivity, followed by a few last clue/implicit constraint combinations and another round of completion. The exception to this is the pasta puzzle. We can see that after around 20 steps where mostly clues have been used, twice a combination of implicit logigram constraints must be used to derive a new fact, after which the table can be easily completed with bijectivity/transitivity and twice the use of a clue.

9.3. Explanation size

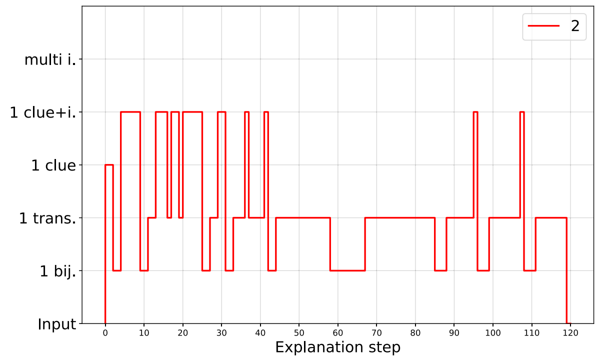
Our cost-function is constructed to favor few (if any) clues and constraints in the explanations, and a small number of previously derived facts $|E|$. Table 2, first 5 columns, shows the average number of facts used per type of constraints used in the explanation. We can observe that the average number of facts used is indeed low, less than two (column 'all'). The breakdown per type of constraint shows that bijectivity typically uses more facts: it either uses three 'negative' facts in



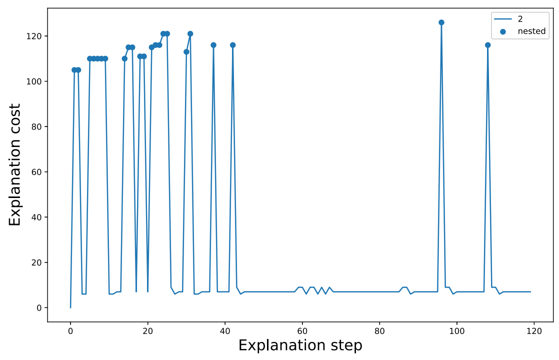
(a) Puzzle 1.



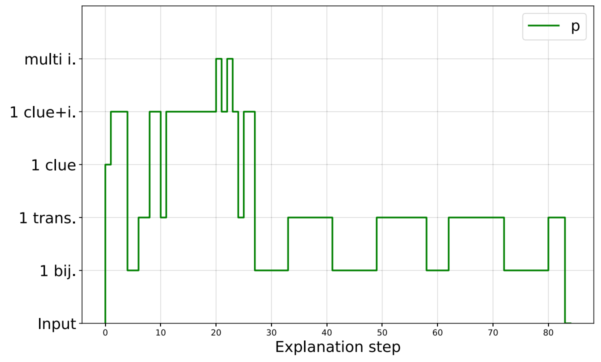
(b) Puzzle 1.



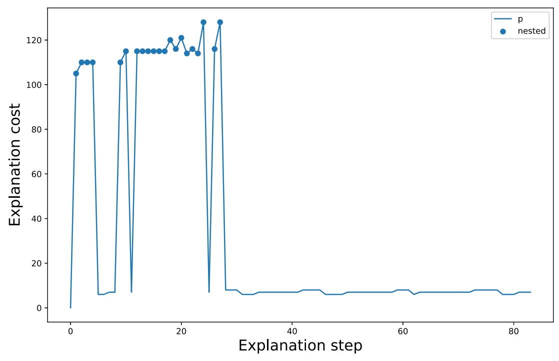
(c) Puzzle 2.



(d) Puzzle 2.



(e) Pasta puzzle.



(f) Pasta puzzle.

Fig. 4. Side-by-side comparison of puzzle composition (left) and puzzle complexity with nested explanation steps highlighted (right).

one row to infer a 'positive' fact, as in Fig. 1 or it uses one 'positive' fact to infer three negative facts. Note that for such an intuitive constraint, the number of facts used does not matter much. Transitivity, by nature, always uses two previously derived facts. When an explanation involves a clue, few facts are involved on average.

The rest of the columns take a closer look at the number of facts used when the explanation involves a clue. We can see that our approach successfully finds small explanations: many clues (the trivial ones) use no facts, while some use 1 fact and only occasionally 2 or more facts are needed. The puzzles 1, 2, 9 and pasta require the use of 3 facts or more together with a clue corresponding to the puzzles with the highest amounts of clues with implicit constraints. Only puzzle 1 requires clues to be combined with more than 3 facts. Even though puzzle 9 has a lower cost, the fact it uses more than 3 facts combined with a clue is linked to a complex clue formulation.

Notably the amount of facts for explanations requiring clues, is equal to 0 half of the time, which means that the new information can be derived independently of other new facts. Part of the clues with 1 fact are linked to the clue involving constraints, namely clues combined with bijectivity as they can derive 3 new facts from only 1 fact. Altogether clues with 0 or 1 facts form more than 80% of the explanations using clues with facts.

Table 3
Properties of the nested explanations.

p	# steps	% steps with nested			average steps nested expl.	Composition of nested explanation				
		of all	of clue+i.	of m-i		1 bij.	1 trans.	1 clue	1 clue+i.	mult i.
1	112	14.29%	100%	-	2.94	36.99%	32.88%	12.33%	17.81%	0%
2	119	14.29%	100%	-	2.65	48.28%	10.34%	20.69%	20.69%	0%
3	110	7.27%	100%	-	2.00	50.00%	0%	0%	50.00%	0%
4	115	13.04%	100%	-	4.04	40.26%	29.87%	20.78%	9.09%	0%
5	122	13.11%	100%	-	2.29	47.83%	0%	8.70%	43.48%	0%
6	115	10.43%	100%	-	2.56	50.00%	8.00%	20.00%	22.00%	0%
7	110	15.45%	100%	-	3.29	37.50%	32.69%	18.27%	11.54%	0%
8	118	9.32%	100%	-	2.36	43.75%	9.38%	25.00%	21.88%	0%
9	114	10.53%	100%	-	3.50	31.25%	28.12%	20.31%	20.31%	0%
p	83	16.87%	100%	100%	3.07	48.44%	20.31%	7.81%	17.19%	6.25%

9.4. Nested explanations

We next investigate the explanation cost of the different steps in an explanation sequence, and which ones we can find a nested explanation for. Fig. 4, right side, shows for a few puzzles the explanation cost of each explanation step; for each step it is also indicated whether or not a nested explanation can be found (those where one was found are indicated by a dot in the figure). The first observation we can draw from the side-by-side figures, is peaks with nested explanations (on the right) overcome with almost all peaks on the left. Simply put, for each of the non-trivial explanations, we are most often able to find a nested explanation that can provide a more detailed explanation using contradiction. Secondly, we observe that the highest cost steps involve more than one constraint: either a clue and some implicit constraints, or a combination of implicit constraints (only in the pasta puzzle). The harder pasta puzzle also has a higher average cost, as was already reported in Table 1.

As the explanation sequence progresses, the cost of difficult explanation steps also increases, meaning that we can find easy explanations in the beginning, but we will require more complex reasoning to find new facts that unlock more simpler steps afterwards. This is especially visible from the trend line that fits all each puzzle's nested explanation dots on the right side of Fig. 4.

We now have a closer look at the composition of these nested explanations in Table 3. When looking at the percentage of reasoning steps that have a nested explanation, we see that only a fraction of all steps have a nested explanation. As the figures already indicated, when looking at the non-trivial steps, we see that for all of those our method is able to generate a nested sequence that explains the reasoning step using contradiction.

When looking at the number of steps in a nested sequence, we see that the nested explanations are rather small: from 2 to 4 steps. The subsequent columns in the table show the composition of the nested explanations, in terms of the types of constraints used. The majority of those are simple steps involving just a single constraint (a single bijectivity, transitivity or clue). Interestingly, even in nested explanations which typically explain a '1 clue+i.' step, the nested explanation also has a step involving a clue and at least one other implicit constraint. Detailed inspection showed this was typically the final step, which is a contradiction in the clue as in Fig. 3.

The composition of the nested sequence also shows that the nested explanations were so simple that they cannot be further decomposed into a second (deeper) level of nested explanations. This is due to the way our cost function is defined, i.e. promoting the use of *simple* constraints and heavily penalizing combinations of clues and/or constraints.

We can further look at the *cost* of the different steps in a nested explanation, which we use as a proxy for difficulty of the steps. Fig. 5 displays a violin plot of the ratio of the cost of a nested step divided by the cost of the original parent step that it is contributing to explain; a wider region indicates a higher density of steps having that ratio. Note that by Definition 10, a step in a nested explanation can never be as costly or more costly than its parent step (always below the top orange line, though it often comes close). The figure shows us that there are many steps with a cost of only a fraction of the parent step (near 0-10% of the original cost), but also some steps closer but still a bit simpler than the parent step. Due to the construction of the cost function in Section 7 with each clue contributing a value of '100' and each fact only a value of '1', this means it typically involves fewer previously derived facts, which should make it easier to understand.

10. Discussion, future work, and conclusions

In this paper, we formally defined the problem of step-wise explanation generation for satisfaction problems, as well as presenting a generic algorithm for solving the problem. We extended the mechanism so that it can be used in a *nested* way, to further explain an explanation. We developed the algorithm in the context of logic grid puzzles where we start from natural language clues and provide a human-friendly explanation in the form of a visualization.

When investigating the nested explanation in Fig. 2 as it is produced by the system, one can observe that this explanation does not entirely match an explanation that would be produced by a human reasoner for a couple of reasons:

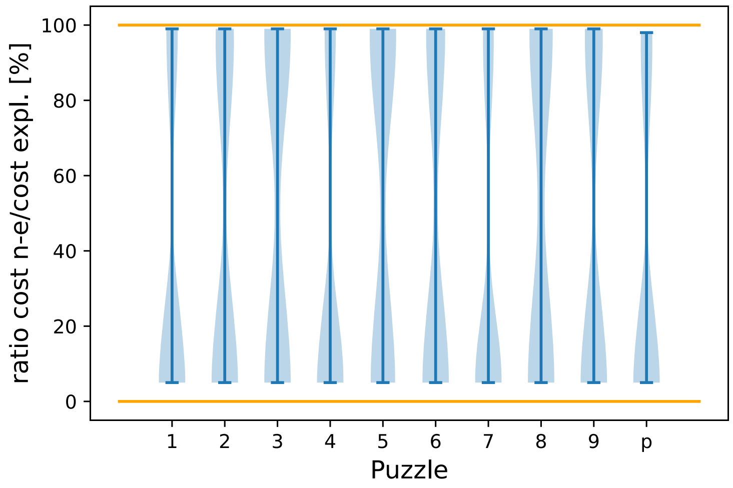


Fig. 5. Distribution of the ratio of the cost of a nested explanation sequence step and the cost of original explanation step for all puzzles.

- In the generation of the nested explanation, as well as in the high-level sequence, we used the greedy algorithm from section 6. While at the high level, this yields good results, at the nested level, this results in sometimes propagating facts that are not used afterwards. The very first propagation in the nested explanation is of this kind. While this would be easy to fix by postprocessing the generated explanation, we left it in our example to highlight this difference between the nested and non-nested explanation.
- It sometimes happens that the system finds, as a minimal explanation, one in which $X - 1$ negative facts are used instead of the corresponding single positive fact. This can be seen in the last step. For human reasoners the positive facts often seem to be easier to grasp. A preference for the system towards these negative facts might be incidentally due to formulation of the clues or it can incidentally happen due to the way the MUS is computed (only subset-minimality is guaranteed there). In general, observations of this kind should be taken into account when devising a cost function.
- A last observation we made (but that is not visible in the current figure) is that sometimes the generated nested explanations seem to be unnecessarily hard. In all cases we encountered where that was the case, the explanation was the same: the set of implicit constraints contains a lot of redundant information: a small number of them would be sufficient to imply all the others. Our cost function, and the subset-minimality of the generated MUS entails that in the explanation of a single step, implicit constraints will never be included if they follow from other included implicit constraints. However, when generating the nested explanations, it would actually be preferred to have those redundant constraints, since they allow breaking up the explanation in simpler parts, e.g., giving a simple step with a single bijectivity, rather than a complex step that uses a combination of multiple implicit constraints.

These observations suggest that further research into the question *what constitutes an understandable explanation for humans* is needed with respect to using appropriate interpretability metrics [42,43]. Additional directions to produce easier-to-understand explanations would be *(i)* to optimize the sequence as a whole, rather than only individual steps, and *(ii)* to learn the cost function based on user traces.

Additionally, on our puzzles, the nested explanations were so simple that no further refinement was needed. For more difficult problems, this may no longer be the case; as such, a possible avenue for future work is a generic approach to multi-level nested explanations. Another interesting direction is to research whether nesting can be related to existing notions of abstraction and refinement [44–46].

With respect to *efficiency*, the main bottleneck of the current algorithm is the many calls to MUS, which is a hard problem by itself. For this reason, in the time between submission and acceptance of this paper, we have investigated methods to perform *unsatisfiable subset optimization* [47], building on existing algorithms for searching cardinality-minimal MUSs [48].

From a systems point of view, a direction for future work is to make our approach (near) real-time, essentially allowing ZebraTutor to be called *while* a user is solving the puzzle. This will become especially important when we implement our ideas in more critical domains, such as *interactive configuration*, where a human and a search engine cooperate to solve a configuration problem and the human can often be interested in understanding *why* the system did certain derivations [49,50].

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This research received funding from the Flemish Government under the 'Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen' programme and the FWO Flanders project G070521N. We thank Jens Claes (and his master thesis supervisor Marc Denecker) for the implementation of a typed extension of the Blackburn & Bos framework as part of his master's thesis, as well as Rocsildes Canoy for his help with the NLP aspect of the information pipeline.

References

[1] M.T. Ribeiro, S. Singh, C. Guestrin, "Why should I trust you?" Explaining the predictions of any classifier, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 1135–1144.

[2] S.M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: Advances in Neural Information Processing Systems, 2017, pp. 4765–4774.

[3] R.R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, D. Batra, Grad-cam: visual explanations from deep networks via gradient-based localization, in: Proceedings of ICCV, 2017, pp. 618–626.

[4] A. Adadi, M. Berrada, Peeking inside the black-box: a survey on explainable artificial intelligence (XAI), IEEE Access 6 (2018) 52138–52160, https://doi.org/10.1109/access.2018.2870052.

[5] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi, A survey of methods for explaining black box models, ACM Comput. Surv. 51 (5) (2018) 1–42.

[6] A.B. Arrieta, N. Díaz-Rodríguez, J.D. Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, F. Herrera, Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI, Inf. Fusion 58 (2020) 82–115.

[7] F. Rossi, P. van Beek, T. Walsh (Eds.), Handbook of Constraint Programming, Foundations of Artificial Intelligence, vol. 2, Elsevier, 2006, http://www.sciencedirect.com/science/bookseries/15746526/2.

[8] U. Junker, Quickxplain: conflict detection for arbitrary constraint propagation algorithms, in: IJCAI'01 Workshop on Modelling and Solving Problems with Constraints, 2001.

[9] T. Feydy, P.J. Stuckey, Lazy clause generation reengineered, in: International Conference on Principles and Practice of Constraint Programming, Springer, 2009, pp. 352–366.

[10] J. Marques-Silva, I. Lynce, S. Malik, Conflict-driven clause learning sat solvers, in: Handbook of Satisfiability, Ios Press, 2009, pp. 131–153.

[11] L.H. Gilpin, D. Bau, B.Z. Yuan, A. Bajwa, M. Specter, L. Kagal, Explaining explanations: an overview of interpretability of machine learning, in: F. Bonchi, F.J. Provost, T. Eliassi-Rad, W. Wang, C. Cattuto, R. Ghani (Eds.), 5th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2018, Turin, Italy, October 1-3, 2018, IEEE, 2018, pp. 80–89.

[12] A. Felfernig, L. Hotz, C. Bagley, J. Tiihonen, Knowledge-based configuration: from research to business cases, 2014.

[13] J. Claes, B. Bogaerts, R. Canoy, T. Guns, User-oriented solving and explaining of natural language logic grid puzzles, in: The Third Workshop on Progress Towards the Holy Grail, 2019.

[14] J. Claes, B. Bogaerts, R. Canoy, E. Gamba, T. Guns, Zebratutor: explaining how to solve logic grid puzzles, in: Beuls et al. [51], http://ceur-ws.org/Vol-2491/demo96.pdf.

[15] B. Bogaerts, E. Gamba, J. Claes, T. Guns, Step-wise explanations of constraint satisfaction problems, in: 24th European Conference on Artificial Intelligence (ECAI), 2020, https://doi.org/10.3233/FAIA200149, in press.

[16] P. Langley, B. Meadows, M. Sridharan, D. Choi, Explainable agency for intelligent autonomous systems, in: Twenty-Ninth IAAI Conference, 2017.

[17] K. Leo, G. Tack, Debugging unsatisfiable constraint models, in: International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer, 2017, pp. 77–93.

[18] K. Zeighami, K. Leo, G. Tack, M.G. de la Banda, Towards semi-automatic learning-based model transformation, in: Proceedings of CP, 2018, pp. 403–419.

[19] J. Marques-Silva, Minimal unsatisfiability: models, algorithms and applications, in: 2010 40th IEEE International Symposium on Multiple-Valued Logic, IEEE, 2010, pp. 9–14.

[20] M. Fox, D. Long, D. Magazzeni, Explainable planning, in: IJCAI'17 workshop on Explainable AI, arXiv:1709.10256.

[21] J. Wittocx, M. Denecker, M. Bruynooghe, Constraint propagation for first-order logic and inductive definitions, ACM Trans. Comput. Log. 14 (2013).

[22] E.C. Freuder, Progress towards the holy grail, Constraints 23 (2) (2018) 158–171.

[23] M.H. Sqalli, E.C. Freuder, Inference-based constraint satisfaction supports explanation, in: AAAI/IAAI, vol. 1, 1996, pp. 318–325.

[24] G. Escamocher, B. O'Sullivan, Solving logic grid puzzles with an algorithm that imitates human behavior, arXiv preprint, arXiv:1910.06636.

[25] M. Ganesalingam, W.T. Gowers, A fully automatic theorem prover with human-style output, J. Autom. Reason. 58 (2) (2017) 253–291, https://doi.org/10.1007/s10817-016-9377-1.

[26] K. Yang, J. Deng, Learning to prove theorems via interacting with proof assistants, in: K. Chaudhuri, R. Salakhutdinov (Eds.), Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, in: Proceedings of Machine Learning Research, PMLR, vol. 97, 2019, pp. 6984–6994, http://proceedings.mlr.press/v97/yang19a.html.

[27] A. Caine, R. Cohen, Mits: a mixed-initiative intelligent tutoring system for sudoku, in: Conference of the Canadian Society for Computational Studies of Intelligence, Springer, 2006, pp. 550–561.

[28] F. Rossi, P. Van Beek, T. Walsh, Handbook of Constraint Programming, Elsevier, 2006.

[29] H. Enderton, H.B. Enderton, A Mathematical Introduction to Logic, Elsevier, 2001.

[30] D.G. Mitchell, E. Ternovska, F. Hach, R. Mohebali, Model expansion as a framework for modelling and solving search problems, Tech. Rep. TR 2006-24, Simon Fraser University, Canada, 2006.

[31] M. Gebser, B. Kaufmann, T. Schaub, The conflict-driven answer set solver clasp: progress report, in: E. Erdem, F. Lin, T. Schaub (Eds.), Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings, in: Lecture Notes in Computer Science, vol. 5753, Springer, 2009, pp. 509–514.

[32] B.D. Cat, B. Bogaerts, M. Bruynooghe, G. Janssens, M. Denecker, Predicate logic as a modeling language: the IDP system, in: M. Kifer, Y.A. Liu (Eds.), Declarative Logic Programming: Theory, Systems, and Applications, ACM / Morgan & Claypool, 2018, pp. 279–323.

[33] A. Kolokolova, Y. Liu, D.G. Mitchell, E. Ternovska, On the complexity of model expansion, in: C.G. Fermüller, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings, in: Lecture Notes in Computer Science, Springer, 2010, pp. 447–458.

[34] M.P. Marcus, B. Santorini, M.A. Marcinkiewicz, Building a large annotated corpus of English: the penn treebank, Comput. Linguist. 19 (2) (1993) 313–330.

[35] J. Claes, Automatic translation of logic grid puzzles into a typed logic, Master's thesis, KU Leuven, Leuven, Belgium, June 2017.

[36] P. Blackburn, J. Bos, Representation and Inference for Natural Language. A First Course in Computational Semantics. Volume 1, ISBN 9781575867731, 2005, http://www.coli.uni-saarland.de/publikationen/softcopies/Blackburn:1997:RIN.pdf.

[37] P. Blackburn, J. Bos, Working with discourse representation theory, an Advanced Course in Computational Semantics.
[38] H. Kamp, Discourse representation theory: what it is and where it ought to go, in: A. Blaser (Ed.), Natural Language at the Computer, Scientific Symposium on Syntax and Semantics for Text Processing and Man-Machine-Communication, Heidelberg, FRG, February 25, 1988, Proceedings, in: Lecture Notes in Computer Science, vol. 320, Springer, 1988, pp. 84–111.
[39] R. Ierusalimschy, L.H. De Figueiredo, W.C. Filho, Lua—an extensible extension language, Softw. Pract. Exp. 26 (6) (1996) 635–652.
[40] S. Ryder, Puzzle Baron's Logic Puzzles, Alpha Books, Indianapolis, Indiana, 2016.
[41] G. Emilio, B. Bart, G. Tias, C. Jens, A framework for step-wise explaining how to solve constraint satisfaction problems, Jun. 2021, https://doi.org/10.5281/zenodo.4982025.
[42] R.R. Hoffman, S.T. Mueller, G. Klein, J. Litman, Metrics for explainable ai: challenges and prospects, arXiv preprint, arXiv:1812.04608.
[43] A. Rosenfeld, Better metrics for evaluating explainable artificial intelligence, in: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, 2021, pp. 45–50.
[44] M. Leuschel, M. Butler, Automatic refinement checking for b, in: International Conference on Formal Engineering Methods, Springer, 2005, pp. 345–359.
[45] Z.G. Saribatur, P. Schüller, T. Eiter, Abstraction for non-ground answer set programs, in: European Conference on Logics in Artificial Intelligence, Springer, 2019, pp. 576–592.
[46] D.G. Mitchell, E. Ternovska, Expressive power and abstraction in essence, Constraints 13 (3) (2008) 343–384.
[47] G. Emilio, B. Bart, G. Tias, Efficiently explaining csps with unsatisfiable subset optimization, in: Proceedings of IJCAI, 2021.
[48] A. Ignatiev, A. Previti, M. Liffiton, J. Marques-Silva, Smallest mus extraction with minimal hitting set dualization, in: Proceedings of CP, Springer, 2015, pp. 173–182.
[49] P.V. Hertum, I. Dasseville, G. Janssens, M. Denecker, The KB paradigm and its application to interactive configuration, Theory Pract. Log. Program. 17 (1) (2017) 91–117, https://doi.org/10.1017/S1471068416000156.
[50] P. Carbonnelle, B. Aerts, M. Deryck, J. Vennekens, M. Denecker, An interactive consultant, in: Beuls et al. [51], http://ceur-ws.org/Vol-2491/demo45.pdf.
[51] K. Beuls, B. Bogaerts, G. Bontempi, P. Geurts, N. Harley, B. Lebichot, T. Lenaerts, G. Louppe, P.V. Eecke (Eds.), Proceedings of the 31st Benelux Conference on Artificial Intelligence (BNAIC 2019) and the 28th Belgian Dutch Conference on Machine Learning (Benelearn 2019), Brussels, Belgium, November 6-8, 2019, CEUR Workshop Proceedings, vol. 2491, 2019, CEUR-WS.org, 2019, http://ceur-ws.org/Vol-2491.