24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Step-wise explanations of constraint satisfaction problems

Bart Bogaerts[1] and **Emilio Gamba**[1] and **Jens Claes**[2] and **Tias Guns**[1]

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. 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. We propose an approach that is agnostic of the underlying constraint propagation mechanisms, and that can provide explanations even for inference steps resulting from combinations of constraints. 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 sequences obtained.

1 Introduction

Explainable AI research aims to fulfil the need for trustworthy AI systems that can explain their reasoning in a human-understandable way. As these systems employ more advanced reasoning mechanisms and computation power, it becomes increasingly difficult to understand why certain decisions are made. Understanding the decisions is important 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) machine learning systems such as neural networks, where the goal is to provide insight into what part of the input is important in the *learned* model. These insights (or local approximations thereof) can justify why certain predictions are made. In contrast, in constraint satisfaction problems, 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 [15]. 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 [23]. These are not meant to be human-interpretable but rather to propagate efficiently.

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. [13] 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. We choose to represent the constraints in natural language, which is an obvious choice for logic grid puzzles which are given as natural language *clues*. We representing the previously and newly derived facts visually, as can be seen in the grid in Figure 1.

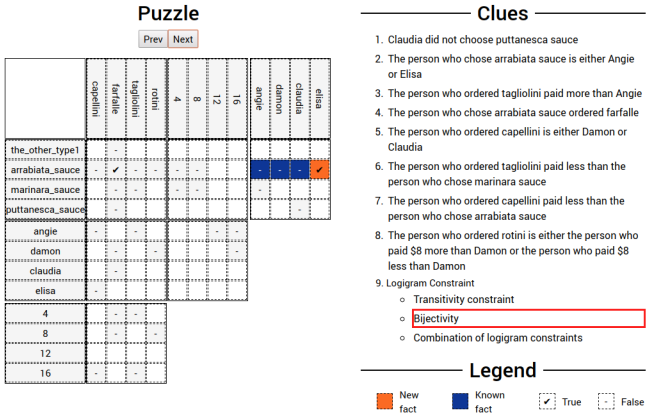

Figure 1: Demonstration of explanation visualisation.

Our work is motivated by the "Holy Grail Challenge"[3] which had as objective to provide automated processing of logic grid puzzles, ranging from natural language processing, to solving, and explaining.

[1] Vrije Universiteit Brussel, firstname.lastname@vub.be
[2] jensclaes33@gmail.com

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

An earlier version of our system won the challenge at the workshop. While our system has the capability of solving logic grid puzzle starting from the natural language clues (see Section 7), the focus of this paper is on the novel explanation-producing part of the system.

The explanation-generating techniques we develop can be applied in a multitude of use cases. For instance, our tool can explain the entire sequence of reasoning, such that a user can debug the reasoning system or the set of constraints that specify their 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 move, or in an interactive setting where a system can explain how it would complete a partial solution of a user. 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.

Our contributions are the following:

- 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 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 experimentally demonstrate the quality and feasibility of the approach in the domain of logic grid puzzles.

2 Related work

This research fits within the general topic of Explainable Agency [19], 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. For example, explainable planning [10] is concerned with building planning systems that can explain their own behaviour. 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, 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.

Explanation of Constraint Satisfaction Problems (CSP) has been studied mostly in the context of overconstrained problems. The goal then is to find which constraints conflict with each other. The QuickXplain method [15] for example uses a dichotomic approach that recursively partitions the constraints to find a minimal conflict set. This is also known as the Minimum Unsat Subset (MUS) problem or Minimal Unsat Core extraction [22]. Many algorithm exist for finding a MUS or enumerating all MUS's [22]. We will use MUS extraction for finding a minimal explanation of an individual inference step.

Our work is inspired by the holy grail challenge at CP'2019, which in turn has its roots in earlier work of E. Freuder on inference-based explanations [26]. In that work, the authors investigate logic grid puzzles and develop a number of problem-specific inference rules that allow solving 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 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.

There is a rich literature on automated and interactive theorem proving, recently focussing on providing proofs that are understandable for humans [11] and, e.g., on teaching humans – using interaction with theorem provers – how to craft mathematical proofs [27]. 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.

Our approach also relates to the work of Belahcene et. al. [1] who — in the context of decision-aiding — aim to build incremental explanations using preference relations. In our case, this would correspond to preferring simple constraints to more complex combinations of constraints through a cost function.

3 Background

Logic grid puzzles. While our proposed method is applicable to constraint satisfaction problems in general, we use *logic grid puzzles* as example domain, as it requires no expert knowledge to understand.

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 Figure 1 contains as second clue "The person who chose arrabiata sauce is either Angie or Elisa" and (among others) 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*).

In Section 7, we explain how we obtain a vocabulary and first-order theory in a mostly automated way from the clues. The result is a vocabulary with types corresponding to the groups of entities in the clues, and the names and types of the binary relations to find (e.g *chose(person, sauce)*, *paired(sauce, pasta)*, *eaten(person, pasta)*); as well as constraints (first-order sentences) corresponding to the clues, and the bijectivity and transitivity constraints. Let T_P be a theory containing all of these constraints for a given puzzle P.

Our running example is a puzzle about people having dinner in a restaurant and ordering different types of pasta. It is the hardest logic grid puzzle we encountered (as a reference, at a recent AI conference, when presenting our tool [8], only four out of 80 researchers who tried managed to solve it). The entire puzzle can be seen in Figure 1; the full final explanation generated for it can be found at `http://bartbog.github.io/zebra/pasta`.

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 type symbols, (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). [4] For example, type *person* with constant symbol *Angie* of type *person* and a relation *chose(.,.)* with signature *person* $\times$ *sauce*.

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

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.*

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 $I_{Ang-Ar} = \{chose(Angie, arrabiata),\ \neg chose(Elisa, arrabiata)\}$ it is known that $Angia$ had arrabiata sauce and Elisa did not.

A partial interpretation I_1 is *more precise* than partial interpretation I_2 (notation $I_1 \geq_p I_2$) if $I_1 \supseteq I_2$. The partial interpretation $\{chose(Angie, arrabiata), \neg chose(Elisa, arrabiata), \neg chose(damon, arrabiata)\}$ is more precise than I_{Ang-Ar}.

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* [24]: 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).

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 reasoning step, after which we introduce a cost function for a reasoning step and the cost of a sequence of reasoning steps.

Explanation of reasoning steps. We assume that a theory T_P and an initial partial interpretation I_0 are given and fixed.

Definition 2. *We define the **maximal consequence** of a theory T_P and partial interpretation I (denoted $max(I, T)$) as the precision-maximal partial interpretation J such that $I \wedge T_P \models J$.*

Phrased differently, $max(I, T)$ is the intersection of all the models of T 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 [12] or IDP [6] (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 $max_{other-consistency}(I, T)$. The rest of this paper assumes we want to construct a sequence that starts at I_0 and ends at $max(I_0, T_P)$ for some consistency algorithm, i.e., that can explain all computable consequences of T_P and I_0.

Definition 3. *A **sequence of incremental partial interpretations** of a theory T_P with initial partial interpretation I_0 is a sequence $\langle I_0, I_1, \ldots, I_n = max(I_0, T_P)\rangle$ 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 Figure 1, it will show the user precisely which information and constraints were used to derive a new piece of information.

Definition 4. *Let I_{i-1} and I_i be partial interpretations such that $I_{i-1} \wedge T_P \models I_i$. 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),*
- *$E_i \subseteq I_i$ (i.e., the explaining facts are a subset of what was previously derived),*
- *$S_i \subseteq T_P$ (i.e., a subset of the clues and implicit constraints are 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 model expansion problem [18].

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}; that is: the explanation must be *subset-minimal*.

Definition 5. *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 6. *A **non-redundant explanation sequence** is a sequence*

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

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

Interpretability of a reasoning steps. While subset-minimality ensures that an explanation is non-redundant, it does not quantify how *interpretable* a explanation is. This quantification is a problem-specific and often subjective manner.

We will assume the existence of a cost function $f(E_i, S_i, N_i)$ that quantifies the interpretability of a single explanation. This is typically specific to the family of problems considered.

In line with the goal of "simple enough for a person to understand" and Occam's Razor, we reason that smaller explanations are easier

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

to interpret than explanations that use a larger number of facts or constraints. In Section 6 we provide a size-based cost function for use in our logic grid puzzle tool, though others can be used as well.

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 7. *Given a theory T_P and initial partial interpretation I_0, the **explanation-production problem** consist of finding a non-redundant explanation sequence*

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

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

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 favour 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 Explanation-producing search

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

Ideally, we could generate all explanations of each fact in I_n, 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.

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; see Section 6 for the cost function we will use for 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 $max(I_0, T_P)$ only contains consequences of I_0 and T_P.

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

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

```
1 I_end ← propagate(I_0 ∧ T_P);
2 Seq ← empty sequence;
3 I ← I_0;
4 while I ≠ I_end do
5 |   (E, S, N) ←min-explanation(I, T_P);
6 |   append (E, S, N) to Seq;
7 |   I ← I ∪ N;
8 end
```

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 look at how to *enumerate* a set of candidate non-redundant explanations given a set of constraints.

For a set of constraints C (later algorithms will not always use T_P for this C!), 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 C. For each new fact a not in I, we wish to find a non-redundant explanation $(E \subseteq I, S \subseteq C, \{a\})$ that explains a. Recall from Definition 6 that this means that whenever one of the facts in E or constraints in S 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) of a derived this. To see this, consider the theory

$$I \wedge C \wedge \neg a.$$

This theory surely is unsatisfiable since a is a consequence of I and C. Furthermore, under the assumption that $I \wedge C$ is consistent (if it were not, there would be nothing left to explain), *any* unsatisfiable subset of this theory contains $\neg a$. We then see that each unsatisifiable subset of this theory is of the form $E \wedge S \wedge \neg a$ where $(E, S, \{a\})$ is a (not necessarily redundant) explanation of the derivation of $\{a\}$ from I. Vice versa, each explanation of $\{a\}$ corresponds to an unsatisifiable subset. Thus, the *minimal* unsatisifiable subsets (MUS) of the above theory are in one-to-one correspondence with the non-redundant explanations of a, 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 a by searching for a MUS that includes $\neg a$. 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 $A \setminus \{a\}$ may have simpler explanations that may be found later in the for loop, hence we do not remove them from J.

Algorithm 2: candidate-explanations(I, C)

input : A partial interpretation I and a set of constraints C

Candidates $\leftarrow \{\}$;
$J \leftarrow$ propagate$(I \wedge C)$;
for $a \in J \setminus I$ **do**
  // Minimal expl. of each new fact:
  $X \leftarrow MUS(\neg a \wedge I \wedge C)$;
  $E \leftarrow I \cap X$; // facts used
  $S \leftarrow C \cap X$; // constraints used
  $A \leftarrow$ propagate$(E \wedge S)$; // all implied facts
  add (E, S, A) to Candidates
end
return *Candidates*

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 [22]. While computing a MUS may be computationally demanding, it is far less demanding than enumerating all MUS's (of arbitrary size) as candidates.

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 that we have a cost function $f(E, S, N)$ 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_P, 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_P 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, C)

input : A partial interpretation I and a set of constraints C

Candidates $\leftarrow \{\}$;
for $S \subseteq C$ *ordered by* $g(S)$ **do**
  if $g(S) < min(\{f(cand_i) | cand_i \in$ *Candidates*$\})$ **then**
    break;
  end
  cand $\leftarrow$ candidate-explanations(I, S);
  add to Candidates all cand$_i$ with corresp. value $f(cand_i)$;
end
return $cand_i \in$ *Candidates with minimal* $f(cand_i)$

Every time *min-explanation*(I, C) is called with an updated partial interpretation I the explanations should be regenerated. The reason is that for some derivable facts a, 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 a, 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 Explanations for logic grid puzzles

We instantiated the above described algorithm in the context of logic grid puzzles. In that setting there are basically three types of constraints in T_P: 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 7). Before defining a cost-function, and the estimation for g used in our implementation, we provide some observation 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 Figure 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 $g(S_1) \geq f(I, S_2)$ whenever S_2 consists of only one implicit constraint and S_1 does not.

Observation 2: clues propagate rarely 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 obtained logical representation is

$$\exists p : ordered(p, capellini) \wedge (p = Damon \vee p = Claudia).$$

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** person which entails that there is only a single person ordering 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. Hence, when pairing clues to other constraints we always pair it with the set of all implicit (bijectivity, transitivity) constraints.

Observation 3: clues are typically used independently from other clues A final 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.

With these three observations in mind, we devised f and g as fol-

lows (where $nc(C)$ denotes the number of clues in C):

$$f(I, C) = basecost(C) + |I| + |C|$$

$$g(C) = basecost(C) = \begin{cases} 0 & \text{if } |C| = 1 \text{ and } nc(C) = 0 \\ 20 & \text{if } |C| > 1 \text{ and } nc(C) = 0 \\ 20 \cdot nc(C) & \text{otherwise} \end{cases}$$

The number 20 is taken here to be larger than any reasonable explanation size. The effect of this, is that we can generate our subsets S in Line 2 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 considering all S in Line 2, but only promising candidates based on our observations.

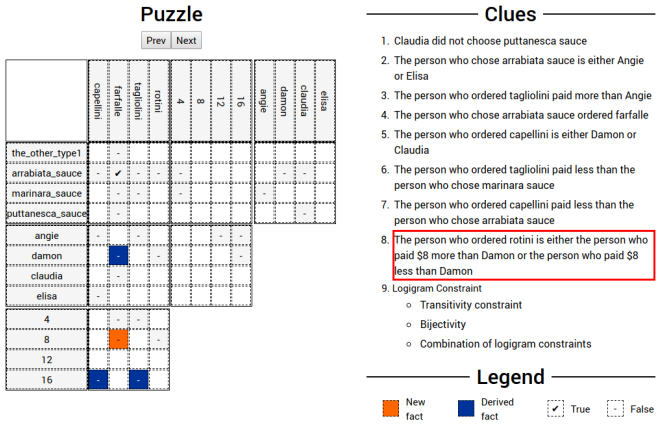

Figure 2: Demonstration of hard explanation.

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 28) is depicted in Figure 2. 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. Intuitively, the reasoning happening here can be explained as follows: if *farfalle* were to cost \$8, then due to the assumptions and bijectivity *rotini* would cost 16. However, since Damon did not take *farfelle* (which we assumed costs \$8), this is in contradiction with the highlighted clue. Hence *farfalle* does not cost \$8.

7 Logic grid puzzles: From natural language clues to typed first-order logic

We developed a demo system, called ZEBRATUTOR, 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 a list of all the *entities* present in the puzzle. It then applies NLP techniques to build a puzzle-specific lexicon. This lexicon is fed into a type-aware variant of the semantical framework of Blackburn & Bos [3, 4], which translates the clues into Discourse Representation Theory [16]. The logic is further transformed to a specification in the IDP language, a typed extension of first-order logic. The underlying solver, IDP [6] 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 optimalpropagate here to find $max(I, T_P)$), and unsat-core extraction and offers a lua interface to glue these inference steps together seamlessly [6]. The complete specification undergoes the following **steps**:

A **Part-Of-Speech (POS) tagging**: A part-of-speech tag is associated with each word using an out-of-the-box POS tagger [21].

B **Chunking and lexicon building**: A problem-specific lexicon is constructed; 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 [3, 4]. The grammar was created based on 10 example training puzzles, and tested on 10 different puzzles to ensure genericity [7].

C **Parsing**: We use a typed variant of the Blackburn and Bos framework to use the lexicon and grammar to derive a logical formulation of the clues in Discourse Representation Theory. The 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 '$ordered(person, pasta)$'. This is then translated into the IDP language and the bijectivity and transitivity constraints are automatically added.

D **Explanation-producing search in IDP**: this is the main contribution of this paper, as explained in Section 5.

E **Visualisation**: all explanation steps (E_i, S_i, N_i) are visualized by means of a color-coded logic grid, where different colors are used to highlight E_i and N_i, and S_i. Figures 1 and 2 contain examples.

An online demo of our system can be found on `http://bartbog.github.io/zebra`, containing examples of all the steps. A more detailed explanation of steps B and C of the information pipeline can be found in [7]. From a natural language processing point of view, the hardest part is step B: automatically deriving the lexicon. 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 timeslot before 12:00.

8 Experiments

Using logic grid puzzles as a use-case, we validate the feasibility of finding a non-redundant explanation sequence. As data we use puzzles from Puzzle Baron's Logic Puzzles Volume 3 [25]. 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 [6] for the reasons explained in Section 7. The algorithm itself is 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.

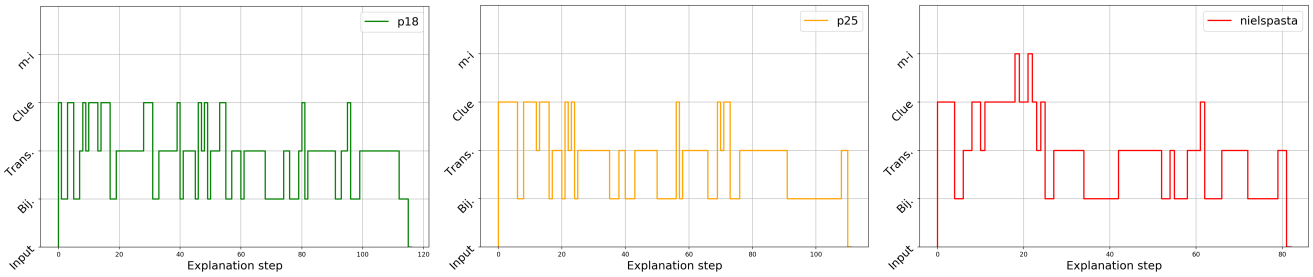

Figure 3: Type of constraints used in each step

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 puzzle identified as p is our running example. $|type|$ is the number of types of entities (e.g. person, sauce) while $|dom|$ is the number of entities of each type. $|grid|$ is the number of cells in the grid, i.e. the number of literals in the maximal consequence interpretation $I_n = max(I_0, T_P)$. Coincidentally,

puzzle	|type|	|dom|	|grid|	steps	% Bij.	% Trans.	% 1 Clue	% m-i	% m-c
1	4	5	150	113	30.97	49.56	18.58	0.0	0
2	4	5	150	122	21.31	59.02	18.85	0.0	0
3	4	5	150	115	28.7	53.91	16.52	0.0	0
4	4	5	150	116	27.59	54.31	17.24	0.0	0
5	4	5	150	123	24.39	58.54	16.26	0.0	0
6	4	5	150	116	26.72	57.76	14.66	0.0	0
7	4	5	150	111	36.94	45.05	17.12	0.0	0
8	4	5	150	119	33.61	47.06	18.49	0.0	0
p	4	4	96	82	34.15	40.24	21.95	2.44	0

Table 1: Composition of puzzle explanations

almost all the puzzles have 4 types with domain size 5, hence 150 cells, except for the pasta puzzle which has a domain size 4, thus 96 cells. We notice that the number of inference steps is around 120 for all but the pasta puzzle. When investigating the proportion of the inference steps that use bijections (only), transitivity (only) or a clue, we can see that around 50% the explanations typically use a transitivity constraint, around 25% typically a trivial bijectivity constraint (e.g. completing a row or column in one relation), and less then $1/5$th of the explanations actually need to use a clue. In the table, #m-i and #m-c refer to the use of multiple implicit constraints and multiple clues respectively. We can see that it is never necessary to combine multiple constraints in one inference step. Also, notably, the puzzles from the booklet never require combining implicit constraints, while the anecdotally hard pasta puzzle is the only one that does: it cannot be solved by focussing on the clues but requires combining facts and knowledge in the table alone to crack it.

2. Sequence progression Figure 3 shows a visualisation the type of explanation used at every step of the sequence, for some different puzzles. The red line is the pasta puzzle. We can see that typically at the beginning of the sequence, clues (4th line) and bijectivity (2nd line) are used, i.e., the trivial ones. This is then followed by a round of clues and some bijectivity/transitivity, after which a large fraction of the table can be completed with bijectivity/transitivty, followed by a few last clues 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.

	avg. facts				% of clue expl. with facts			
p	all	Clues	Trans.	Bij.	0 facts	1 fact	2 facts	>2 facts
1	1.82	0.52	2.0	2.37	66.6%	28.5%	0.0%	4.7%
2	1.80	0.61	2.0	2.38	47.8%	47.8%	0.0%	4.3%
3	1.83	0.31	2.0	2.45	78.9%	15.7%	0.0%	5.2%
4	1.85	0.45	2.0	2.50	70.0%	15.0%	15.0%	0.0%
5	1.87	0.40	2.0	2.60	65.0%	30.0%	5.0%	0.0%
6	1.82	0.29	2.0	2.35	76.4%	17.6%	5.8%	0.0%
7	1.93	0.73	2.0	2.46	57.8%	26.3%	5.2%	10.5%
8	1.84	0.18	2.0	2.57	81.8%	18.1%	0.0%	0.0%
p	1.76	1.05	2.0	2.07	60.0%	15.0%	0.0%	25.0%

Table 2: Puzzle explanation cost based on the cost function $f(I, C)$ and statistics on puzzle constraints

3. Explanation size. Our cost-function is constructed to favour few (if any) clues and constraints in the explanations, and few previously derived facts $|E|$. Table 2 shows the average number of facts used per explanation. We also show the average number of facts used when using only bijectivity or transitivity, or when a clue is used.

We can observe that the average number of facts used is indeed low, less than two. Furthermore, the breakdown shows that bijectivity typically uses more facts: it uses three 'negative' facts in one row to infer a 'positive' fact, as in Figure 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. Finally, when looking at the number of facts used together with 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 exception is again the difficult pasta puzzle.

9 Discussion, future work, and conclusions

In this paper, we formally defined the problem of step-wise explanation generation and presented a generic algorithm for solving this problem. We implemented this 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 visualisation. We used this implementation to investigate puzzle properties and difficulty.

The main bottleneck of the current algorithm is the many calls to MUS, which is a hard problem by itself. Therefore, in future work we want to investigate unsat-core *optimization* with respect to a cost-function, either by taking inspiration for instance from the MARCO algorithm [20] but adapting it to prune based on cost-functions instead of subset-minimality, or alternatively by reduction to QBF [17].

Secondly, we want to dig deeper into the question *what constitutes an understandable explanations for humans*, either by optimizing the entire sequence instead of step by step, by learning the cost function based on user traces, or by reusing our developed algorithms to explain reasoning steps in more detail and as such develop an expla-

nation mechanism that operates at different levels of abstraction. To illustrate this last point: in the setting of very difficult puzzles, some of the explanation steps still require some effort to understand. When explained by a human, this is often done using some form of proof by contradiction using an explanation of the form "suppose this [the derived fact] would not hold, then [some simpler reasoning steps], which is not possible". This is also how we explained Figure 2 at the end of Section 6. This explanation process is of *exactly the same form* as what we generate now. The only difference is that I_0 is not the empty interpretation, but one in which a wrong value is assigned.

A final direction for future work is to make our approach interactive, essentially allowing ZEBRATUTOR to be called *while* a user is solving the puzzle and to implement in more serious domains such as for instance interactive configuration in which a human and a search engine cooperate to solve some configuration problem and the human can often be interested in understanding *why* the system did certain derivations [14, 5].

Acknowledgements

This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

REFERENCES

[1] Khaled Belahcene, Christophe Labreuche, Nicolas Maudet, Vincent Mousseau, and Wassila Ouerdane, 'Explaining robust additive utility models by sequences of preference swaps', *Theory and Decision*, **82**(2), 151–183, (2017).

[2] Katrien Beuls, Bart Bogaerts, Gianluca Bontempi, Pierre Geurts, Nick Harley, Bertrand Lebichot, Tom Lenaerts, Gilles Louppe, and Paul Van 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*, volume 2491 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019.

[3] Patrick Blackburn and Johan Bos. Representation and inference for natural language, 2005.

[4] Patrick Blackburn and Johan Bos, 'Working with discourse representation theory', *An Advanced Course in Computational Semantics*, (2006).

[5] Pierre Carbonnelle, Bram Aerts, Marjolein Deryck, Joost Vennekens, and Marc Denecker, 'An interactive consultant', In Beuls et al. [2].

[6] Broes De Cat, Bart Bogaerts, Maurice Bruynooghe, Gerda Janssens, and Marc Denecker, 'Predicate logic as a modeling language: the IDP system', in *Declarative Logic Programming: Theory, Systems, and Applications*, eds., Michael Kifer and Yanhong Annie Liu, 279–323, ACM / Morgan & Claypool, (2018).

[7] Jens Claes, *Master thesis: Automatic Translation of Logic Grid Puzzles into a Typed Logic*, Master's thesis, KU Leuven, Leuven, Belgium, June 2017.

[8] Jens Claes, Bart Bogaerts, Rocsildes Canoy, Emilio Gamba, and Tias Guns, 'Zebratutor: Explaining how to solve logic grid puzzles', In Beuls et al. [2].

[9] Thibaut Feydy and Peter J Stuckey, 'Lazy clause generation reengineered', in *International Conference on Principles and Practice of Constraint Programming*, pp. 352–366. Springer, (2009).

[10] Maria Fox, Derek Long, and Daniele Magazzeni, 'Explainable planning', in *IJCAI'17 workshop on Explainable AI (arXiv:1709.10256)*.

[11] M. Ganesalingam and W. T. Gowers, 'A fully automatic theorem prover with human-style output', *Journal of Automated Reasoning*, **58**(2), 253–291, (Feb 2017).

[12] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub, 'The conflict-driven answer set solver clasp: Progress report', in *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*, eds., Esra Erdem, Fangzhen Lin, and Torsten Schaub, volume 5753 of *Lecture Notes in Computer Science*, pp. 509–514. Springer, (2009).

[13] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal, 'Explaining explanations: An overview of interpretability of machine learning', in *5th IEEE International Conference on Data Science and Advanced Analytics, DSAA 2018, Turin, Italy, October 1-3, 2018*, eds., Francesco Bonchi, Foster J. Provost, Tina Eliassi-Rad, Wei Wang, Ciro Cattuto, and Rayid Ghani, pp. 80–89. IEEE, (2018).

[14] Pieter Van Hertum, Ingmar Dasseville, Gerda Janssens, and Marc Denecker, 'The KB paradigm and its application to interactive configuration', *TPLP*, **17**(1), 91–117, (2017).

[15] Ulrich Junker, 'Quickxplain: Conflict detection for arbitrary constraint propagation algorithms', in *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, (2001).

[16] Hans Kamp, 'Discourse representation theory: What it is and where it ought to go', in *Natural Language at the Computer, Scientific Symposium on Syntax and Semantics for Text Processing and Man-Machine-Communication, Heidelberg, FRG, February 25, 1988, Proceedings*, ed., Albrecht Blaser, volume 320 of *Lecture Notes in Computer Science*, pp. 84–111. Springer, (1988).

[17] Hans Kleine Büning and Uwe Bubeck, 'Theory of quantified boolean formulas', in *Handbook of Satisfiability*, eds., Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, volume 185 of *Frontiers in Artificial Intelligence and Applications*, 735–760, IOS Press, (2009).

[18] Antonina Kolokolova, Yongmei Liu, David G. Mitchell, and Eugenia Ternovska, 'On the complexity of model expansion', in *Logic for Programming, Artificial Intelligence, and Reasoning - 17th International Conference, LPAR-17, Yogyakarta, Indonesia, October 10-15, 2010. Proceedings*, eds., Christian G. Fermüller and Andrei Voronkov, volume 6397 of *Lecture Notes in Computer Science*, pp. 447–458. Springer, (2010).

[19] Pat Langley, Ben Meadows, Mohan Sridharan, and Dongkyu Choi, 'Explainable agency for intelligent autonomous systems', in *Twenty-Ninth IAAI Conference*, (2017).

[20] Mark H Liffiton and Ammar Malik, 'Enumerating infeasibility: Finding multiple muses quickly', in *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pp. 160–175. Springer, (2013).

[21] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz, 'Building a large annotated corpus of english: The penn treebank', *Computational Linguistics*, **19**(2), 313–330, (1993).

[22] Joao Marques-Silva, 'Minimal unsatisfiability: Models, algorithms and applications', in *2010 40th IEEE International Symposium on Multiple-Valued Logic*, pp. 9–14. IEEE, (2010).

[23] Joao Marques-Silva, Inês Lynce, and Sharad Malik, 'Conflict-driven clause learning sat solvers', in *Handbook of satisfiability*, 131–153, ios Press, (2009).

[24] David G. Mitchell, Eugenia Ternovska, Faraz Hach, and Raheleh Mohebali, 'Model expansion as a framework for modelling and solving search problems', Technical Report TR 2006-24, Simon Fraser University, Canada, (2006).

[25] Stephen Ryder, *Puzzle Baron's logic puzzles*, Alpha Books, Indianapolis, Indiana, 2016.

[26] Mohammed H Sqalli and Eugene C Freuder, 'Inference-based constraint satisfaction supports explanation', in *AAAI/IAAI, Vol. 1*, pp. 318–325, (1996).

[27] Kaiyu Yang and Jia Deng, 'Learning to prove theorems via interacting with proof assistants', in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, eds., Kamalika Chaudhuri and Ruslan Salakhutdinov, volume 97 of *Proceedings of Machine Learning Research*, pp. 6984–6994. PMLR, (2019).