Relating the Structure of a Problem and Its Explanation

Eugene C. Freuder

Insight Centre for Data Analytics, School of Computer Science & Information Technology, University College Cork, Cork, Ireland
eugene.freuder@insight-centre.org

Abstract

As AI becomes more ubiquitous there is increasing interest in computers being able to provide explanations for their conclusions. This paper proposes exploring the relationship between the structure of a problem and its explanation. The nature of this challenge is introduced through a series of simple constraint satisfaction problems.

Introduction

As AI becomes more ubiquitous there is increasing interest in computers being able to provide explanations for their conclusions (Aha et al. 2018; Magazenni et al. 2018), and the European GPDR provides special impetus (Goodman and Flaxmanar 2016). Many years ago I began exploring the relationship between the structure of a problem and the complexity of its solution (Freuder 1982). *This paper proposes exploring the relationship between the structure of a problem and its explanation.* The nature of this challenge is introduced through a series of simple constraint satisfaction problems.

A constraint satisfaction problem (CSP) involves choosing a *value* for each problem *variable*, subject to restrictions (*constraints*) on allowable combinations of values. CSPs have many uses in AI and in real-world applications. In fact, in an earlier AAAI Senior Track paper I argued that constraints can serve as a unifying force in AI (Freuder 2006).

In constraint satisfaction, on the one hand we are fortunate in that an explanation for a successful solution is very straightforward: "see, the constraints are satisfied". (Though one may want a further explanation as to how the solution was obtained, or a characterization of a set of solutions). However, when a constraint satisfaction problem is unsolvable, explanations are difficult, as there can be an exponential number of reasons for failure, corresponding to every way that the constraints cannot be satisfied, and there can be many different routes to arriving at the conclusion that satisfaction is impossible. To misquote Tolstoy: Solvable CSPs are all alike; every unsolvable CSP is unsolvable in its own way.

Thus the primary focus here is on explaining failure. A number of approaches have been taken to providing explanations for constraint satisfaction failure. We restrict ourselves here to efforts to provide explanations to users as opposed to explanations intended to make algorithms more efficient or to aid programmers (Freuder 2017). Given the exponential threat, and to address specific needs, these efforts generally start with an abstracted or higher-level form of explanation, e.g. sets of unsatisfiable constraints, and then quickly limit their focus, e.g. to minimal sets of unsatisfiable constraints.

The position taken here is to start with truly complete explanations (Freuder 2018) and abstract and limit from there. The approach taken here is straightforward, and at least at this point, not very deep or technical; but one that, perhaps for that reason, may be well-suited to the goal of providing a high-level "big picture" of the structure of the problem, in a form readily meaningful to a human user. The hope is that this approach may, as well, lead to general insights into the structure of constraint satisfaction problems.

Complete

A CSP can be represented by a *constraint network*, with a node for each variable, each with a set of possible values, and edges (or more generally hyperedges) representing constraints. We will use the problem represented by the following constraint network as an initial example. It represents a simple, unsolvable "coloring problem". The problem is to choose a color (r)ed or (b)lue for each node (X, Y, Z) such that any two nodes connected by a constraint (A, B, or C) have different colors. Each constraint is a "not equals" constraint.

Copyright © 2019, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

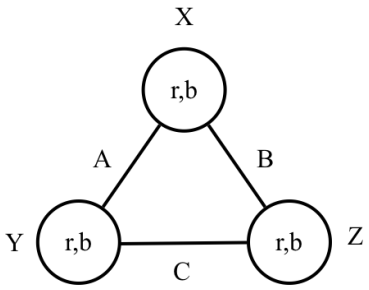

Figure 1: Example Problem

Of course, this is an unsolvable problem. Just as the obvious way to "completely explain" a solution is simply to point at the constraints and say "see they are all satisfied", the obvious way to provide a *complete explanation* for failure is to list all the possible instantiations, assignments of values to variables, and point out for each the constraints that are not satisfied.

There are 8 possible assignments of values red or blue to the variables X, Y, and Z in our example, none of which satisfy all the constraints. Each of these is listed in the table below, along with the set of constraints that the assignment does not satisfy. E.g. (r b r) {B} indicates that assigning red to X, blue to Y and red to Z violates just constraint B.

(r r r) {A B C}	(r r b) {A}
(r b r) {B}	(r b b) {C}
(b r r) {C}	(b r b) {B}
(b b r) {A}	(b b b) {A B C}

Figure 2: A Complete Explanation

Scalability

The reader may be immediately concerned that listing all the possible assignments to obtain a complete explanation will not be an approach that scales well. Research in constraint programming on explanation has understandably grappled with scaling issues. Abstractions and limitations derived from complete explanations, and associated visualization tools, may address scaling concerns. Clever algorithms or sampling methods may reduce effort. We can seek to take advantage of symmetry and specialized structure. Most importantly, relationships we observe, prove, or mine with machine learning methods, between problem structure and explanation structure may permit us to make at least approximate predictions or generalizations without large scale computational effort.

If a large effort is still required, an initial off-line effort may be amortized by its use for repeated, efficient on-line queries. However, our presumption is also that, as AI becomes increasingly pervasive in everyday life, attention will shift to some degree to smaller scale problems where the issue is not so much how to minimize explanation size, as how to maximize explanation utility, and the need may be less for scalable algorithms and more for effective human-computer interfaces.

Abstraction

Complete explanations may be useful in themselves, e.g. for machine learning purposes, but human users will generally appreciate higher level views of the problem "landscape". We can look for, and automate extraction of, patterns, abstractions, properties of the complete solution set.

We focus here on the sets of constraints that are violated by each instantiation. The constraint violations in our example's complete explanation can be summarized by the set:

{{A} {A} {B} {B} {C} {C} {A B C} {A B C}}

where each of the elements corresponds to the set of constraints violated by an assignment.

We have "abstracted out" some information here. This says that there are two assignments that just violate constraint A, but it does not specify which they are (though this information might be "attached" to be pulled up when asked for).

This can be rewritten as:

$$2\{A\} \cup 2\{B\} \cup 2\{C\} \cup 2\{A\ B\ C\}$$

and then as:

$$2([\boldsymbol{C}]^1 \cup \{\boldsymbol{C}\})$$

where $[S]^k$ is the set of all cardinality k subsets of S, $\boldsymbol{C}$ is the set of all constraints, and nS denotes a set containing n copies of every member of S.

While removing all constraints is, of course, *sufficient* to allow any solution, we are indicating here that it is only *necessary* for two of the solutions.

We could limit such a representation further, e.g. by reducing just to a term that represents removing the fewest constraints or one that represents the most partial solutions. Such limitations could simplify the generation or handling of the explanation.

There is, of course, a long history of work on "partial constraint satisfaction" (Freuder and Wallace 1992), but this generally focuses on finding an instantiation that satisfies a maximal number of constraints. In some applications, users might want a broader picture of their options. We might want to define and study concepts like k-unsatisfiability, for instantiations where k constraints are unsatisfied or k-unsolvability, for problems that can be solved by removing k constraints.

The sets {A}, {B}, {C} and {A B C} correspond to what have been called exclusion sets (O'Sullivan et al. 2004). We define an *exclusion set* here as a subset of the

problem constraints that when removed permits at least one solution. It is important to note that the sets employed here do *not* correspond to the common use of sets in explanations to represent, in effect, unsatisfiable subproblems, often "minimal" unsatisfiable subproblems, where removing any constraint makes the subproblem satisfiable. The representation here is both finer-grained, and arguably more comprehensive.

The symmetry of the problem is reflected in the explanation and helps make it more compact, but we could also try to make use of symmetry up front to simplify the derivation of the formula.

The formula compactly describes the "landscape" of the problem, and corresponds to a simple, high-level, natural language "story":

> We can get a solution by removing just one constraint, and in fact any constraint will do, and will admit two more solutions. Removing two constraints at a time will not buy us any more solutions. There are two solutions that cannot be obtained without removing all constraints.

The formula should be straightforward to generate automatically (though there may be room for cleverness in making the process more efficient) and provides a compact representation for the computer and for expert human users. The English explanation should be easy to produce automatically from the formula by a rudimentary English-language generator. I claim that it constitutes a satisfying high-level explanation for a human user, and one that provides some insights that are not immediately obvious from the statement of even a simple problem like this.

We can, of course, look for other forms of abstraction, we can investigate other ways of characterizing or classifying complete explanations, other features or patterns, perhaps even an *explanation taxonomy*.

Questions

The formula could also be used to automatically generate answers to common questions that users might have. For example:

Q: How many more solutions do I gain if I allow two constraints at a time to be violated rather than just one.
A: None actually. There are only two solutions that require removing more than one constraint, and those require removing all of them.

We can also use this approach to answer "what if" or "how" questions. For example, "How much do I have to give up to obtain a solution with r(ed) for X?" or "Can I get a solution with r(ed) for X by removing just one constraint?" If we limit the values for X to r(ed), the explanation formula for the resulting problem is:

$$[\boldsymbol{C}]^1 \cup \{\boldsymbol{C}\}$$

which readily provides the answer. Of course, ideally we could "annotate" the explanation for the original problem to permit us to derive this explanation without starting from scratch.

Solvable

We can also look at the unsolvable portion of a problem that does have solutions. Suppose that constraint C, between Y and Z, is an equality constraint rather than an inequality constraint. This changes the results to:

(r r r) {A B}	(r r b) {A C}
(r b r) {B C}	(r b b) solution
(b r r) solution	(b r b) {B C}
(b b r) {A C}	(b b b) {A B}

Figure 3: Complete Explanation for Modified Problem

The complete explanation for the unsatisfiable portion of this problem is:

$$2[\boldsymbol{C}]^2$$

If we like, we can add the solvable assignments back in, represented by the empty set of unsatisfied constraints, to obtain the formula:

$$2([\boldsymbol{C}]^2 \cup \{\emptyset\})$$

Or in English:

> This problem has two solutions. Eliminating one constraint will not add any new solutions. Eliminating each combination of two constraints at a time yields two additional solutions. There is no assignment that fails to satisfy all three constraints at once.

Comparisons

These explanations also can be used as a tool to compare problems or view the effects of modifying a problem. For example, here we see that changing the one constraint from inequality to equality changed the explanation to its "complement": the second explanation involves exactly those subsets of constraints that are not in the first. You may find this to be a bit of a surprise. Why is this so, under what general conditions will this happen? More generally, how

will changes in the problem change the explanation? Studying the structure and relationships of complete explanations may yield new insights into CSPs.

This complementarity is strongly reflected visually in the following image, which distinguishes which of the subsets of {A B C} in the the traditional "power set lattice" appear in the explanation of the original or modified problem.

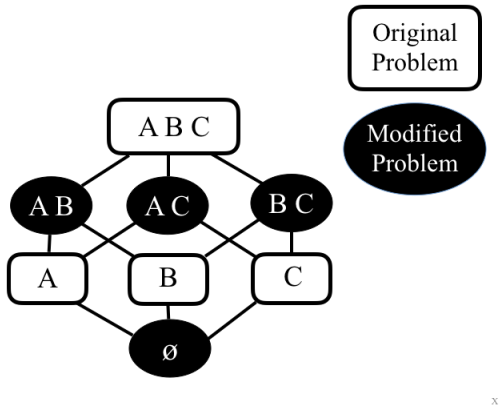

Figure 4: Complementarity

Structure

For our final example we consider the Queens problem. The classic Queens problem is to place 8 chess queens on a chessboard such that no two attack each other. We will look at the 3 Queens problem, placing 3 chess queens on a 3 by 3 mini chessboard, such that no two attack each other. This is unsolvable. It is a larger problem than our coloring problem, there are 27 possible instantiations. The abstracted explanation is:

$$2[\boldsymbol{C}]^1 \cup 2[\boldsymbol{C}]^2 \cup 6\{C_{12}\ C_{23}\} \cup 9\boldsymbol{C}$$

In English:

> Removing any one of the constraints enables two solutions. Removing any two of the constraints at a time enables two other solutions, but removing C_{12} and C_{23} will enable 6 more. To obtain the remaining 9 solutions you need to remove all the constraints.

Partly thanks to symmetry, the formula is still fairly compact even though we have more than triple the number of possible instantiations that we had for the coloring problem. And again the English explanation tells us an interesting story about the "structure" of the unsatisfiability, one which is not immediately obvious from the statement of the problem. In this case, for example, in another form of "complementarity", unlike for the coloring problem, only a relatively small proportion of the assignments become solutions when a single constraint is removed, and a relatively large proportion of assignments only become solutions when all the constraints are removed.

Again, this suggests a potential line of inquiry: can we predict based on problem structure what proportion of the assignments will be only "one constraint away" from satisfiability? More generally, can we relate the structure of a problem to the structure of its explanation?

The question of "distance from satisfiability" leads, for example, to the following conjecture:

The Good News / Bad News Conjecture:

The harder a constraint satisfaction problem
is to solve,
the easier it is to come close to a solution,
and vice versa.

By "coming close" is meant finding an instantiation that violates few constraints. If one is familiar with the basic approaches to solving CSPs, backtracking and local search, this conjecture will seem a natural one. The existence of many "almost solutions" can lead to costly, frustrating forays deep into the search tree or up to local maxima. We could investigate conjectures like this experimentally. What kind of theoretical machinery might enable us to prove conjectures like this formally?

Conclusion

We have identified:

- An Opportunity: Studying the structure and relationships of explanations may yield new insights into problems.
- A Challenge: Can we relate the structure of a problem and the structure of its explanation?

These have been raised here, and might be further pursued, in the context of basic constraint satisfaction problems, but might also be generalized or specialized. For example, still in the context of constraint satisfaction, we could expand to consider constraint optimization problems, or specialize to scheduling problems. There are many variants and extensions of CSPs that might merit bespoke approaches to explanation. More generally, we can look for corresponding opportunities and challenges in explaining other forms of AI.

Acknowledgments

This material is based upon works supported by the Science Foundation Ireland under Grant No. 12/RC/2289, which is co-funded under the European Regional Development Fund.

References

Aha, D.; Darrell, T.; Doherty, P.; and Magazzeni, D. eds. 2018. *Proceedings of the 2nd Workshop on Explainable Artificial Intelligence*. https://tinyurl.com/yxh7vpsl.

Freuder, E. 1982. A Sufficient Condition for Backtrack-Free Search. *J. ACM* 29(1): 24-32.

Freuder, E. 2006. Constraints: The Ties That Bind. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, 1520-1523. Menlo Park, Calif.: AAAI Press.

Freuder, E. 2017. Explaining Ourselves: Human-Aware Constraint Reasoning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, 4858-4862. Palo Alto, Calif.: AAAI Press.

Freuder, E. 2018. Complete Explanations. At the web page of the Second Workshop on Progress Towards the Holy Grail. https://tinyurl.com/yxcwjonj.

Freuder, E., and Wallace, R. 1992. Partial Constraint Satisfaction. *Artificial Intelligence* 58(1-3): 21-70.

Goodman, B., and Flaxmanar, S. 2016. European Union regulations on algorithmic decision-making and a "right to explanation". Xiv:1606.08813v3 [stat.ML].

Magazzeni, D.; Smith, D.; Langley, P.; and Biundo, S. eds. 2018. *Proceedings of the 1st Workshop on Explainable Planning*. https://tinyurl.com/y2ybbhtq.

O'Sullivan, B.; Papadopoulos, A.; Faltings, B.; and Pu, P. 2007. Representative Explanations for Over-Constrained Problems. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, 323-328. Menlo Park, Calif.: AAAI Press.