Solving Logic Grid Puzzles with an Algorithm that Imitates Human Behavior[*]

Guillaume Escamocher and Barry O'Sullivan

Insight Centre for Data Analytics
University College Cork, Cork, Ireland
{guillaume.escamocher,barry.osullivan}@insight-centre.org

Abstract. We present in this paper our solver for logic grid puzzles. The approach used by our algorithm mimics the way a human would try to solve the same problem. Every progress made during the solving process is accompanied by a detailed explanation of our program's reasoning. Since this reasoning is based on the same heuristics that a human would employ, the user can easily follow the given explanation.

1 Introduction

Most research work done in Computer Science aims to brings us closer to a machine that can solve any problem stated by any user. The most promising advancements towards this Holy Grail [1] have come so far from Constraint Programming.

Nowadays Constraint Programming occurs in very different contexts, both in terms of the problems it is solving, and in terms of the people using it. Constraint Programming has been used by computer scientists to solve a multi-billion auction problem for the U.S. government involving thousands of television stations [2]. It is also used, often unknowingly, by many people every day to solve the daily Sudoku puzzle from their favorite newspaper.

In this paper we focus on logic grid puzzles which, not unlike Sudoku, are casual problems. Following the instructions of the Challenge, we looked at the puzzles from the logicgridpuzzles.com website. Throughout this paper, we will refer to individual puzzles from the website by their difficulty, their ID and their title in that order. So for example, the Dragon Slayer puzzle is Hard 211 "Dragon Slayer".

A logic puzzle is composed of k *categories*, of $k \times n$ *elements* and of *clues*. Categories are sets of elements, and all categories in the same puzzle contain the same number n of elements. A *solution* for a logic grid puzzle is a set of n k-tuples, such that each tuple contains one element from each category, and such that no element occurs in distinct tuples. The clues are restrictions on the possible matchings. A valid logic grid puzzle has exactly one solution.

As an example, consider the following puzzle:

[*] This publication has emanated from research conducted with the financial support of Science Foundation Ireland (SFI) under Grant Number SFI/12/RC/2289, co-funded under the European Regional Development Fund.

- 3 categories:
  - “First Name”, containing 3 elements (“Angela”, “Donald” and “Leo”).
  - “Country”, containing 3 elements (“Germany”, “Ireland” and “United States”).
  - “Year of Birth”, containing 3 elements (“1946”, “1954” and “1979”).
- 3 clues:
  - The person from the “United States” was born in “1946”.
  - “Leo” is younger than the person from “Germany”.
  - “Donald” was born in “1946”, or he is from “Ireland”.

The only way to fulfill the clues is to match “Angela” with “Germany” and “1954”, “Donald” with “United States” and “1946”, and “Leo” with “Ireland” and “1979”. This is the solution to the puzzle. Note that the solution can be found even though not all labels appear in the clues.

Our contribution is a constraint program that takes as input a logic grid puzzle and solves it while explaining the reasoning behind each variable assignment. Throughout the process, our algorithm behaves as a human trying to solve the same puzzle would.

The remainder of the paper is laid out as follows. Section 2 describes how we implemented our approach. Its outline follows the four steps enumerated in the Challenge statement, with a fifth subsection presenting some other functions that we added to increase the customization possibilities of the explanation part. Section 3 then details the characteristics of the puzzles we tested our program on, as well as the difficulties we encountered and how we dealt with them.

2 Implementation

Each of the following four subsections deals with the steps enumerated in the Challenge statement, in order. The last subsection of the current section presents features that we added to our program.

2.1 Input

We are only partially addressing Step 1 of the Challenge. Instead of taking as input the website pages exactly as they are, our program asks the user to enter the clues as sets of constraints. For each constraint the user is adding, they are asked the clue to which the constraint belongs, which type of constraint is being added, and further information that is specific to each constraint type. We have implemented the following 14 constraints:

1. *yes*: “X” is “Y”. The user is asked to give two labels, corresponding to “X” and “Y” respectively. Example: Clue 2 in Hard 211 “Dragon Slayer”.
2. *no*: “X” is not “Y”. The user is asked to give two labels, corresponding to “X” and “Y” respectively. Example: Clue 7 in Hard 107 “Bird Rescue 101”.
3. *or*: “X” is “Y” or “Z”. The user is asked to give three labels, corresponding to “X”, “Y” and “Z” respectively. Example: Clue 3 from Section 1’s example.

4. *xor*: “X” is either “Y” or “Z”. The user is asked to give three labels, corresponding to “X”, “Y” and “Z” respectively. Example: Clue 6 in Hard 107 “Bird Rescue 101”.
5. *alldiff*: “X_1”, “X_2”,..., “X_n” are all distinct. The user is first asked for the value of n, then asked to give n labels, corresponding to the n “X_i”. Example: Clue 7 in Hard 71 “Home Sick”.
6. *twobytwo*: Out of “X” and “Y”, one is “W” and the other is “Z”. The user is asked to give four labels, corresponding to “X”, “Y”, “W” and “Z” respectively. Example: Clue 4 in Hard 74 “Class is in”.
7. *before*: “X” is before “Y” in the “C” category. The user is asked to give three labels, corresponding to “X”, “C” and “Y” respectively. Example: Clue 6 in Hard 74 “Class is in”.
8. *after*: “X” is after “Y” in the “C” category. The user is asked to give three labels, corresponding to “X”, “C” and “Y” respectively. Example: Clue 2 in Hard 93 “Pasta Night”.
9. *beforefixed*: “X” is exactly n elements before “Y” in the “C” category. The user is first asked for the value of n, then asked to give three labels, corresponding to “X”, “C” and “Y” respectively. Example: Clue 2 in Hard 107 “Bird Rescue 101”.
10. *afterfixed*: “X” is exactly n elements after “Y” in the “C” category. The user is first asked for the value of n, then asked to give three labels, corresponding to “X”, “C” and “Y” respectively. Example: Clue 3 in Hard 74 “Class is in”.
11. *beforeatleast*: “X” is at least n elements before “Y” in the “C” category. The user is first asked for the value of n, then asked to give three labels, corresponding to “X”, “C” and “Y” respectively. Example: “Laura paid at least 3 fewer dollars than the woman from Cork”.
12. *afteratleast*: “X” is at least n elements after “Y” in the “C” category. The user is first asked for the value of n, then asked to give three labels, corresponding to “X”, “C” and “Y” respectively. Example: “The movie that Emily rented came out at least 10 years after *Fight Club*”.
13. *distance*: “X” is exactly n elements from “Y” in the “C” category. The user is first asked for the value of n, then asked to give three labels, corresponding to “X”, “C” and “Y” respectively. Example: “The Norwegian lives next to the blue house” in the Zebra puzzle.
14. *disjunction*: “X_1” is “Y_1” or “X_2” is not “Y_2” or ... or “X_n” is “Y_n”. The user is first asked for the number of disjuncts, then for the polarity of each of them (“is” is positive, “is not” is negative) then finally they are asked to give $2n$ labels, corresponding to “X_1”, “Y_1”,...,“Y_n” in order. Example: “If the 23-year-old person is wearing a blue shirt, then Bill did not order a burger”.

For each constraint, the order in which the labels are requested mirrors the order in which they generally appear in clues of this type. In particular, this is the reason why the category label must be entered between the element labels in Constraints 7-13.

One might think there is too much redundancy with that many types of constraints. Some types are particular cases of others (for example 7 is the same as 11 with n set to 1), some are dual of others (9 is the same as 10 with the order of the labels reversed) and some can be written as a conjunction of others (all types can be written as a conjunction of type 14 constraints). We chose to implement that many constraints in order to be as close as possible to the original formulation of the clues. The only constraint type that deviates from this intent is 14, which was added to cover as many possible clues as possible.

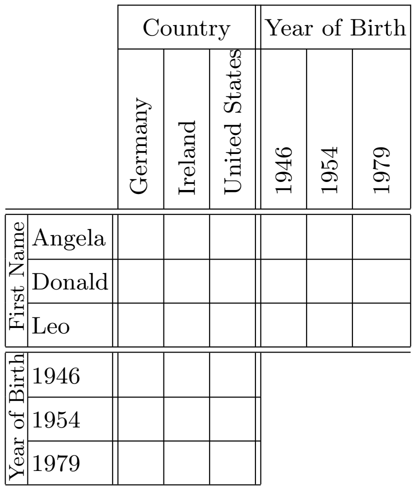
To illustrate this part of our implementation, this is what the constraints from Section 1's example look like after input is taken:

- {1,yes,"United States","1946"}
- {2,after,"Leo","Year of Birth","Germany"}
- {3,or,"Donald","1946","Ireland"}

Each constraint contains as part of its description the clue it belongs to. Indeed, clues can contain several constraints (see for example Easy 8 "A Michigan Adventure"), and by referring to the clue instead of the constraint the explanation is more clear for the user.

2.2 Model

The "grid" part in the name of this puzzle genre refers to the representation most often used to solve them. Usually for a logic grid puzzle with k categories, the elements from Categories 2 to k label the columns, while the elements from Category 1 and Categories k to 3 label the rows. The exact positioning of the categories does not actually matter, as long as each pair of elements from different categories is represented by exactly one cell. Figure 1 shows the grid corresponding to the running example.



		Country			Year of Birth		
		Germany	Ireland	United States	1946	1954	1979
First Name	Angela						
	Donald						
	Leo						
Year of Birth	1946						
	1954						
	1979						

Fig. 1. The running example represented as a grid.

Solving a puzzle consists in filling each cell of its grid by either *yes* if the label of the row is matched with the label of the column in the solution, or by *no* otherwise. In our model, we represent the grid by an array of integers. Each cell is initially set to 0. Once a pair of elements is determined to be matched in the solution, the corresponding cell in the array is set to 1. Similarly, if two elements from different categories are determined to not be matched together in the solution, then the corresponding cell in the array is set to -1.

2.3 Solving

Our main goal when addressing Step 3 of the Challenge was to create a solver that behaves exactly as a human would. Therefore our solver uses the same inference rules, in the same order, as the user would if trying to solve the puzzle.

At each step of the process, our algorithm picks a rule from a set of inference rules and applies it to fill one cell. The rules can be divided into two kinds: consistency rules that only use information from the current state of the grid, and clues rules that also use information from the constraints forming the clues. Consistency rules can be further divided between basic consistency rules that are easy to used for humans, and advanced consistency rules that are more complicated and will be avoided by humans unless necessary.

Basic consistency rules use the meta-information of logic grid puzzles: for any two distinct categories C and D, there is a bijection between the elements of C and the elements of D. So if in the running example we know that the cell (“1946”,“United States”) is filled with *yes*, then the cells (“1954”,“United States”) and (“1979”,“United States”) must be filled with *no*. Also, if the cells (“Leo”,“1946”) and (“Leo”,“1954”) have already been filled with *no*, then (“Leo”, “1979”), the last cell for “Leo” in the “Year of Birth” category, must be filled with *yes*.

Our most frequently used advanced consistency rule relies on the transitivity inherent from the grid format. So if in the running example both (“Donald”,“1946”) and (“1946”,“United States”) are filled with *yes*, then (“Donald”,“United States”) must also be filled with *yes*. Another one of our advanced consistency rules bears similarities with path consistency. If we have two elements from different categories labelled “e” and “f”, and a third category C such that for each element in C labelled “g” either the cell (“e”,“g”) or the cell (“f”,“g”) is filled with *no*, then it means that no element of C can be matched with both “e” and “f”, so “e” and “f” cannot be matched together in the solution, and therefore the cell (“e”,“f”) can be filled with *no*.

Each rule that uses information from the clues is associated with one of our fourteen constraints. Some of the constraints are only associated with one trivial rule. For example the rule associated with Constraint 1 (“X” is “Y”) directly fills the cell (“X”,“Y”) with *yes*. Other constraints are associated with several, more elaborate rules that not only use the information from the clue the constraint is part of, but also take into account the current state of the grid. For example one of the rules for Constraint 3 (“X” is “Y” or “Z”) fills the cell (“X”,“Z”) with *yes* if the cell (“X”,“Y”) contains *no*, and another rule for the same constraint fills the cell (“X”,“W”) with *no* for every label “W” such that both (“W”,“Y”) and (“W”,“Z”) contain *no*.

Humans will naturally read the clues in order and fill what they can from that information. Then they will complete rows and columns in categories where either one cell is filled with *yes* or all cells but one are filled with *no*. Once they have exhausted all easy ways to make progress, and only then, they will use the more complicated reasoning found in our advanced consistency rules. When deciding which rule to try next, our solver reflects that method of thinking.

```
Data: A logic grid puzzle.
Result: The solution to the puzzle in the form of a filled grid.
progress ← true;
while progress do
    progress ← false;
    for i ← 1 to |Cons| do
        rule ← nextrule(Cons);
        if canapply(rule) then
            apply(rule);
            progress ← true;
        end
    end
    easyprogress ← true;
    while easyprogress do
        for i ← 1 to |BCR| do
            rule ← nextrule(BCR);
            if canapply(rule) then
                apply(rule);
                progress ← true;
            end
        end
    end
    if progress == false then
        difficultprogress ← false;
        while difficultprogress == false do
            rule ← nextrule(ACR);
            if canapply(rule) then
                apply(rule);
                difficultprogress ← true;
            end
        end
    end
end
```

Algorithm 1: Logic grid puzzle solver.

The priority order between the different kinds of inference rules is illustrated in Algorithm 1. *Cons* is the set of rules using information from the cluse, *BCR* is the set of basic consistency rules and ACR is the set of advanced consistency rules. As can be seen in Lines 4-10, the algorithm tries the constraints in order, without immediately trying earlier constraints again when progress is made. On the other hand, basic consistency rules are used over and over because of their extreme simplicity, which makes them very attractive for a human user. Finally, advanced consistency rules are only used as a last resort, when no further progress is possible from any other rule.

2.4 Output

Every time a cell is filled, our program outputs the reasoning that led it to determine the correct value for that particular cell. So if called on the running example, our program writes the following when using the information from the clues to start solving the puzzle:

- “United States” is “1946” (Clue 1).
- “Leo” is after “Germany” in the “Year of Birth” category (Clue 2), so “Leo” is not “Germany”.
- “Leo” is after “Germany” in the “Year of Birth” category (Clue 2), so “Leo” is not the first element in that category, so “Leo” is not “1946”.
- “Leo” is after “Germany” in the “Year of Birth” category (Clue 2), so “Germany” is not the last element in that category, so “Germany” is not “1979”.

Note that Clue 3 cannot be exploited yet. The state of the grid at this point is pictured in Figure 2, with “Y” representing *yes* and a dot representing *no*.

		Country			Year of Birth		
		Germany	Ireland	United States	1946	1954	1979
First Name	Angela						
	Donald						
	Leo	•			•		
Year of Birth	1946			Y			
	1954						
	1979	•					

Fig. 2. Using the clues to start the solving process.

The solver now has enough information to complete the bottom left block:

– 7 cells can be filled from basic consistency.

To avoid cluttering the explanation, the solver groups together consecutive lines of basic consistency application. This is entirely optional, and can be turned off. Now that basic consistency has filled more cells, the solver tries using the clues again to check whether some new information can be inferred:

– "Germany" is not one of the first 1 element in the "Year of Birth" category, and "Leo" is after "Germany" in that category (Clue 2), so "Leo" is not one of the first 2 elements in the "Year of Birth" category, so "Leo" is not "1954".
– "Donald" is "1946" or "Ireland" (Clue 3), and "Germany" is neither "1946" nor "Ireland", so "Donald" is not "Germany".
– "Donald" is "1946" or "Ireland" (Clue 3), and "1954" is neither "1946" nor "Ireland", so "Donald" is not "1954".
– 9 cells can be filled from basic consistency.

		Country			Year of Birth		
		Germany	Ireland	United States	1946	1954	1979
First Name	Angela	Y	•	•	•	Y	•
	Donald	•			Y	•	•
	Leo	•			•	•	Y
Year of Birth	1946	•	•	Y			
	1954	Y	•	•			
	1979	•	Y	•			

Fig. 3. The solution is almost complete.

Figure 3 shows that at this point the grid is almost completely filled. However the clues are now all fully satisfied, and all possibilities of progress from basic consistency rules have been exhausted. Therefore the algorithm must use an advanced transitivity rule to take the next step towards the solution:

– "Donald" is "1946" and "1946" is "United States", so "Donald" is "United States".
– 3 cells can be filled from basic consistency.

By default our program outputs the full explanation at once, however the option to only write one line at a time has also been implemented. This can be useful if the user wants to solve the puzzle by themselves, but is stuck at one stage and desires a hint.

2.5 Additional Features

We have enhanced our program with some additional functions to offer a more customizable explanation to the user. One of these lets the user know every time they can discard a clue. While technically this corresponds to the last time that clue is used, which can be easily determined by checking the last line that the clue appears in the explanation, we instead look for the moment when already filled cells of the grid explicitly indicate that the clue has been fully satisfied. This choice keeps with the general intent of our work, which is to build a solver that reasons as a human would.

In most logic grid puzzles, the aim is to fill the entire grid. It is however conceivable that a user would be interested in only knowing the value of one single cell. In fact, the famous Zebra puzzle does not ask for the matching of all elements, but only for who in the "Nationality" category is paired with the eponymous "Zebra". If requested to do so, our program can restrict the explanation to only the part which is relevant to the determination of a particular cell value.

Finally, our program can convert logic grid puzzles into Conjunctive Normal Form (CNF). The resulting CNF files can then be used by any SAT solver or model counter. We primarily implemented this feature as a debugging tool, to check the validity of new puzzles.

3 Results

3.1 Scope

Our tests encompass the puzzles that are presented as either easy or hard on the website, as well as the Zebra puzzle, probably the first and most well-known puzzle of this type.

We removed 8 puzzles from our consideration. One of them (Hard 72 "Expensive Coffee") has a clue that refers to a label not present in the corresponding category. Editing this label to one of the five elements in that particular category led to no solution in three cases, and to a unique but rejected solution in the other two cases. The clues in the seven other puzzles that we removed (Easy 23 "Football fanatics", Easy 57 "Special Delivery", Easy 75 "Three Friends", Easy 108 "Movie Buffs Associated Week of Films - Helen Mirren", Easy 133 "Easter Eggs", Hard 73 "For sale... sold!", Hard 127 "Secret Santa") were not enough to reduce the number of solutions to 1. This can be easily checked manually for the easy puzzles, while for the hard ones we used our CNF conversion feature and applied a SAT solver on the resulting file for confirmation.

After removing these defective puzzles, we were left with 69 puzzles: 55 of easy difficulty, 13 of hard difficulty, and the Zebra puzzle. All but 3 of them have 4 categories, 2 of them have 5 categories and the last one (the Zebra puzzle) has 6 categories. The number of elements in each category is 3 for 50 puzzles, 4 for 6 puzzles, and 5 for the other 13 puzzles.

For six puzzles (Easy 22 “Baggage Mishaps”, Easy 60 “Holiday Decision”, Easy 64 “Robbery at Millionaire’s Mansion”, Easy 70 “The racehorses”, Easy 76 “Three little boys”, Easy 83 “The Enchanted Forest”), the information contained in the opening statement was needed to have no more than one solution. To address this, we simply treated the statement as an additional clue (Clue 0).

3.2 Successes and Challenges

Out of the 69 puzzles considered, 67 are straightforward to model by our set of fourteen constraints. The clues in one of the other two puzzles, Hard 119 “A New Personal Computer”, contain the label “Andrew” which does not fit in any of the existing categories, despite being directly part of the puzzle main objective (“Which computer has been chosen by Andrew?”). We resolved this issue by adding a new “Andrew” category containing the labels “Andrew”, “NotAndrew1”, “NotAndrew2”, “NotAndrew3” and “NotAndrew4”. This allowed us to fully model all clues in that puzzle, and subsequently to solve it. We also added ordering constraints (by using Constraint 7) on the four “NotAndrew” labels, in order to keep the uniqueness of the solution. This has no effect on the answer to the original puzzle question.

The clues in the last puzzle, Easy 65 “Sporting Excellence”, contain cross-referencing meta-information. Our representation of a clue as a conjunction of constraints was not able to model them individually.

Our algorithm managed to find the unique solution, accompanied by a human-readable explanation, for all 68 puzzles we could model in the input part of the Challenge. This gives us a success rate of 98.6%. Our only failure was on an Easy puzzle with only four categories and three elements in each categories, which proves that the issue has nothing to do with scale.

On a Dell laptop with an Ubuntu 18.04 operating system and an Intel i7-5600U processor, solving all 68 puzzles takes a combined time of 91 milliseconds with file logging, 71 milliseconds without. This shows that our method is extremely computationally efficient.

References

1. Freuder, E. C.: In Pursuit of the Holy Grail. Constraints **2**(1), 57–61 (1997)
2. Newman, N., Fréchette, A., Leyton-Brown, K.: Deep optimization for spectrum repacking. Communications of the ACM **61**(1), 97–104 (2018)