Artificial Intelligence Project: Generating and Solving Crossword Puzzles as CSP

Mukul Shingwani, Jaimin Gajjar, Riyanshu Jain, Mitul Vashista
B20AI023, B20AI014, B20AI060, B20AI022

November 6, 2022

1 Abstract

Constraint satisfaction problems (CSPs) are a representation of mathematical problems defined as a set of objects whose state must fulfil several constraints or limitations. CSPs are the focus of artificial intelligence research and commonly show significant levels of complexity, necessitating a combination of heuristics and search techniques to be solved in an acceptable or reasonable amount of time. In this paper, we propose a simple yet novel technique that aims to generate a crossword puzzle implemented using a constraint satisfaction problem (CSP) search technique, which begins with identifying suitable variables and values (domains). In order to determine the appropriate variables to store in the crossword, we further solve the generated crossword puzzle using a backtracking algorithm and inference.
Keywords - Constraint Satisfaction Problem, Backtracting, Arc Consistency, Forward Checking, Crossword puzzle

2 Introduction

Crossword problems offer a useful testing ground for constraint satisfaction algorithms. Despite being one of the difficult problems for computers to tackle, there are still relatively simple solutions exist which provide quicker ways to solve them. In this study, we present several Constraint Satisfaction Problem (CSP) techniques, such as arc consistency, forward checking, and Depth First Search (DFS). Previous research has shown these methods to be significantly more efficient and effective CSP solvers. Even reasonably large instances of constraint satisfaction problems can be solved quickly by CSP, despite the fact that it belongs to the class of NP-complete problems.
CSPs are mathematical problems where one must find states which satisfy a number of constraints [1]. CSPs usually consist of a state defined by a variable

X_i with values from domain D_i and a goal test, which is a set of constraints specifying allowable combinations of values for subsets of variables. The way assigning values to a variable works is, first a value is assigned to a variable which takes the assignment to a new state, then another assignment takes place, and so on. But when such a case arises where no proper assignment is found for a variable we need to backtrack and hence, a Depth-first search technique is applied. DFS for CSPs with single-variable assignments is called a backtracking search. DFS is preferred because of the low cost of memory since we only need to take assignments to a single variable into account at each node.
To choose a variable that is to proceed next, we need to apply proper heuristics as to reduce the number of backtracks. They are defined as the most constrained variable, most constraining variable, and least constraining value. Combining all these heuristics can improve backtracking efficiency but to further shorten the search space we use forward checking and arc consistency. Using all of these techniques we present an efficient and quick approach to finding the solution to a crossword puzzle with a minimum number of backtracks.
The crossword puzzle exhibits relevant applications in entertainment, educational and rehabilitation contexts.

3 Literature Survey

The first research in which a crossword puzzle was viewed as a CSP was done by Mazlack in 1976 [2] where he analyzed the construction of a crossword puzzle generator. As pointed out by [Dechter and Pearl, 1981] [3], when solving a constraint satisfaction problem, there are four choices to be made: which variable to instantiate next, what value to use as the instantiation, how to deal with backtracking and what kind of preprocessing to do. Based on this approach Ginsberg, in 1990 [4] , analyzed ”smart backjump” and arc consistency to construct a generator for crossword. They concluded that arc consistency and the cheapest-first heuristic are essential to solve such kinds of puzzles. Since then, the technique of Constraint Programming has been of great interest and has attracted wide research in this area. Gary Meehan and Peter Gray [5] showed the comparison between the use of wide variety of heuristics and constraint programming in constructing the puzzle. In solving the crossword puzzles, most successful automated crossword solvers used A* search for inferencing, modified depth-first search or search engine to integrate external knowledge. Eric Wallace et al. [6] presented new methods for crossword solving based on neural question answering, structured decoding, and local search. Shazeer, Littman and Keim proposed a solution for crossword puzzles using probabilistic constraint satisfaction by maximizing the probability of a correct solution and maximizing the number of correct words (variable values) in the solution [7] Leonardo Rigutini et al. [8] presents a fully automated crossword generator with no human intervention and also attempts at compiling a crossword schema with the extracted definitions using a constrained satisfaction programming (CSP) solver. Our approach is also fundamentally built on the top of CSP but differs in the sense that it can generate a crossword and then also solve the generated crossword.

4 Methodology

Our approach focused on how to treat the space and the time well, because of this, every step must be accurately measured and the decision of choosing the variable or the value must be done using heuristics. In the forthcoming subsections, we'll explain the methodology adopted by us that'll help one understand the workflow of the project.

4.1 Steps & control strategy

A block in the crossword will consist of the following information :

- Starting point in the grid.
- Ending point in the grid.
- Current word filled between them.
- Link to values of the same length and to all of its neighbors.
- Normal Constraints and Arc Consistency Constraints.
- Heuristics for Variable (word) Choosing and Value Choosing.

4.2 Choosing the variables

We used a sorted list, that is sorted based on two heuristics, the main heuristic is the most constraining variable heuristic, and the second one will be used when the two or more variables have the same most constraint variable number, which was the minimum remaining value heuristic, after choosing the variable, if there's a value that applies both the forward checking and the arc consistency, the heuristics of the variable will be set to ZERO so it won't be used again in the sorted set. If there was no appreciation value, backtracking will occur.

4.3 Choosing the value

After selecting the variables, the next step is to select the values. We would like to choose a value that maximizes the value of the other neighbors. Not only does it maximize them globally, but it doesn't maximize their null neighbors. This can be easily done by using the linked array list of values and knowing how each character occurs at each position. This is the value heuristic. If there is no such value, the response is sent to the DFS loop, thus backtracking.

4.4 Backtracking Algorithm

We use the basic backtracking strategy of attempting to label each variable, un-labeling and retrying if labeling fails, and return true once all variables have been assigned values or false once all variables have failed. The pseudo-code for basic backtracking search is given here:-

```
def BacktrackSearch (variables V):
    consistent = TRUE
    level = 1
    Variable x_i = initialize()
    Loop
        If (consistent):
            (x_i, consistent) = label (x_i)
        else:
            (x_i, consistent) = un-label (x_i)
        If (No more unassigned variables):
            return  Solution  -Found
        else if(All variables have been tried):
            return  No  -Solution
    End loop
# end backtrack search
```

Listing 1: Algorithm for backtracking

4.5 Forward Checking

The forward check algorithm uses the current instantiations of variables to truncate the domains of variables that have not yet been instantiated. They actually allow you to look ahead through CSP's search tree to examine the status of a variable's domains, and if one of those domains is destroyed, all possible values get removed, then backtracking happens. Forward checking has proven to be one of the most effective methods for speeding up CSP's.
A forward check is done after every successful allocation (an allocation that maximizes the neighbor's options, no null options for anyone). A forward check iterates over all neighbors and adds a constraint resulting from assigning a value. Each zero-valued neighbor is indicative enough to choose a different value to assign. The other value is the correct next value that satisfies the criteria for selecting a value. If no other value exists, backtracking occurs.

4.6 Arc consistency

After the forward check, some of the neighbor's values are removed. If so, the arc consistency phase begins. Arc consistency traverses all neighbors and each gets the intersection cell of them and its neighbors. After knowing the intersection cell and the location of both neighbors and neighbors of neighbors, we add a new type of constraint than the labeled arc consistency constraint. Every variable has that kind of constraint, and what it does is basically OR the values available from the normal constraint with the arc consistency values.

```
def REVISE(csp, X_i, X_j):
    returns true iff we revise the domain of Xi
    revised = false
    for each x in D_i do:
        if (no value  of y in D_j allows (x,y) to satisfy the
        constraint between X_i and X_j):
            delete x from D_i
        revised = true
    return revised
```

```
def AC-3(csp):
#inputs: csp, a binary CSP with components (X, D, C)
#local variables: queue, a queue of arcs, initially all the arcs in
     csp
    if (inconsistency):
        return false
    else:
        return true
    while queue is not empty do:
        (X_i, X_j ) = GET_FIRST_ELEMENT(queue)
        if REVISE(csp, X_i, X_j):
            if (size of D_i = 0):
                return false
        for each X_k in X_i.NEIGHBORS - {X_j} do:
            add (X_k, X_i) to queue
    return true
```

Listing 2: Algorithm for AC-3

Arc consistency reduces CSP's search space by removing unsupported values from the domain of variables. If a variable has a constraint, the value v_k of variable x is not supported, so if x is set to v_k, there is no instantiation of any other variable that satisfies the constraint. AC-3 works in two steps. First iterate over each variable xi times and check if its value is supported by all conditions of x_i. On the other hand, AC-3 queues unsupported values that it has removed. Next, remove the previously removed unsupported values and check to see if the removal resulted in additional unsupported values. New unsupported values are placed on the queue and continue to be dequeued until a point is reached where the queue is empty. This happens when there are no more deprecated values and only a finite number of values can be removed, so termination is guaranteed.

4.7 Depth First Search

In this approach we started by pushing a state containing all the variables unassigned, then in the loop body of the DFS, we will choose a variable, and then a value, if some constraints do not satisfy while choosing the value then a backtrack will occur, when the stack is empty then there are no such word combinations that can fit the grid, otherwise, the solution will be found during the DFS.

5 Experiment settings

In this project, we show a crossword generator and from the output of that generator, we made a crossword solver. The input is just the matrix size of the crossword puzzle, for example, 5 rows and 5 columns. Then we convert the output of this crossword to a .txt file containing the horizontal and vertical words which need to be filled in the puzzle. The solver needs a .txt file that shows # as a black box and _ as a white box, another file with answer words as a data file also needs to be provided as input. The AI Program will try to solve the crossword by enforcing node consistency, arc consistency, and backtracking algorithm to find the right variables and solve the board.

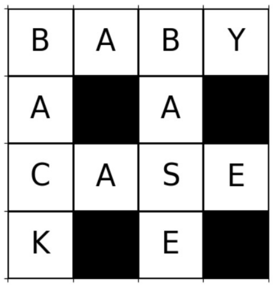
Given below are various functions along with what they do:-

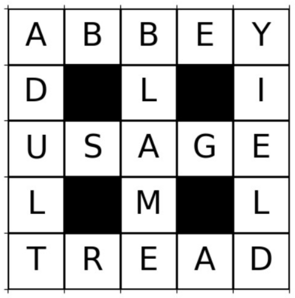
- **ensure_node_consistency:-** Update domains such that each variable is node-consistent. (Remove any values that are inconsistent with a variable's unary constraints; in this case, the length of the word.)
- **revise:-** Make variable x arc consistent with variable y. To do so, remove values from the domain of x for which there is no possible corresponding value for y in domains of y Return True if a revision was made to the domain of x; return False if no revision was made.
- **ac3:-** Update domains such that each variable is arc consistent. If arcs are None, begin with an initial list of all arcs in the problem. Otherwise, use arcs as the initial list of arcs to make it consistent. Return True if arc consistency is enforced and no domains are empty; return False if one or more domains end up empty.
- **order_domain_values:-** Return a list of values in the domain of var, in order by the number of values they rule out for neighboring variables. The first value in the list, for example, should be the one that rules out the fewest values among the neighbors of var.
- **select_unassigned_variables:-** Return an unassigned variable not already part of an assignment. Choose the variable with the minimum number of remaining values in its domain. If there is a tie, choose the variable with the highest degree. If there is a tie, any of the tied variables are acceptable to return values.
- **backtrack:-** Using Backtracking Search, take as input a partial assignment for the crossword and return a complete assignment if possible to do so. an assignment is a mapping from variables (keys) to words (values). If no assignment is possible, return None.

All of the inputs and outputs will be shown in the results section below.

6 Results

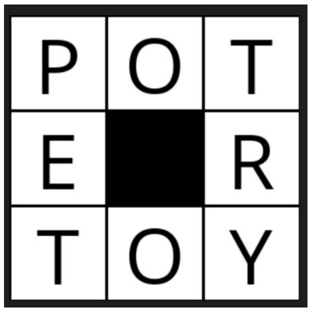
- **Generator**

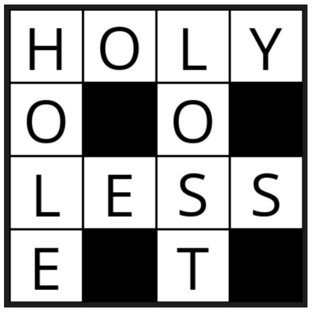




We pass input as the number of rows and columns, which randomly generates a crossword by randomly putting black and white boxes. one of the possible sets of words satisfying the puzzle is selected from a data file containing 3000+ words.

- **Solver**





Here, the input is a txt file that has been obtained from the generator and converted to a combination of # & _ which signifies black and white boxes respectively. Another txt file containing the clues (both across and down) is also provided in the input using which our AI solver generates a possible answer if it exists, otherwise returns ”No solution”.
sample input to the crossword solver for a 4x4 puzzle:-

```
----
-#-#
----
-#-#
```

Github repository link

7 Conclusion & Future Work

Constraints Satisfaction Problems can sometimes be the best way to solve a problem, by looking at a problem we can immediately find a formulation that fits the CSP approach. The challenge is that not all formulations can lead to a solution in simple steps, so the formulation needs to be simple to program and require the least effort in terms of space and time complexities. In this report, we have considered different strategies for generating and solving the constraint satisfaction problem. Forward-checking algorithms have been found to be particularly efficient for crossword puzzles. CSP is an NP-complete class of problems, but a fairly large instance can be solved using effective heuristics and strategies. Some algorithmic techniques we didn't consider were dynamic backtracking and dynamic variable ordering which might be of value. Dynamic backtracking may improve local lookups, as variables once assigned are not required to be unassigned and only need to be backtracked instead of deallocated. In the domain of crosswords, this has proven effective because black squares often separate different parts of the crossword and they do not affect each other much except for one or two connections.

References

[1] Frank Langbein et al. "Constraint satisfaction problems". In: (Mar. 2001).

[2] Lawrence J. Mazlack. "Computer construction of crossword puzzles using precedence relationships". In: *Artificial Intelligence* 7.1 (1976), pp. 1–19. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(76)90019-9. URL: https://www.sciencedirect.com/science/article/pii/0004370276900199.

[3] Rina Dechter and Judea Pearl. "Network-Based Heuristics for Constraint-Satisfaction Problems". In: *Artif. Intell.* 34 (1987), pp. 1–38.

[4] Matthew L. Ginsberg et al. "Search Lessons Learned from Crossword Puzzles". In: *AAAI*. 1990.

[5] Gary Meehan and Peter Gray. "Constructing Crossword Grids : Use of Heuristics vs Constraints". In: 1997.

[6] Eric Wallace et al. *Automated Crossword Solving*. 2022. DOI: 10.48550/ARXIV.2205.09665. URL: https://arxiv.org/abs/2205.09665.

[7] Noam Shazeer, Michael Littman, and Greg Keim. "Solving Crossword Puzzles as Probabilistic Constraint Satisfaction". In: *Proc. AAAI '99* (May 1999).

[8] Leonardo Rigutini et al. "A Fully Automatic Crossword Generator". In: *2008 Seventh International Conference on Machine Learning and Applications* (2008), pp. 362–367.