Crosswords Generation As CSP

Department of Electrical & Computer Engineering
Birzeit University
Ahmad Yahya (1150276)
Maha Hajja (1150144)

Supervised By: Aziz Qaroush

Abstract

This paper discusses our project that introduces the problem of Crosswords puzzle, the main focus of the project was on the generation part of the puzzle, the generation was implemented using Constraints Satisfaction Problems (CSPs) search technique, which started from finding appropriate variables and values (domains), then finding the best data structures to satisfy the problems specification in terms of both space and time, our aim was not just to create a generator to fill the grid, but a reasoning UI that is easy to use for normal users and for those whom in real need of it like the newspapers editors, they can choose the topic(s) and the file they want, then they can choose the generation style (American, British, random or just manually), then if there's a possible words combination that can fit the constraints a new UI will pop-up with hints and a numbering link to the grid so they can print/publish it.

1. Introduction

Dating back to the 1960s and 1970s the earliest ideas leading to Constraints Programming (CP) may be found in the Artificial Intelligence (AI), where a goal based problem back then was to recognize the objects in a 3D scene by interpreting lines in the 2D drawings is probably the first CSP that was formalized! [1]

Constraint satisfaction problems (CSPs) are mathematical problems where one must find states or objects that satisfy a number of constraints or criteria. And are defined as a triple <X, D, Ci>, where X is a set of variables, D is a domain of values, and C is a set of constraints C1(S1)..Cn(Sn) where each Si is a set of variables. A constraint Ci is a combination of valid values for the variables Si. A Solution to the CSP is an assignment of values to S1..Sn that satisfies all constraints. [2]

After assigning a variable with a value a new variable need to be chosen leading to form a new state, then a variable will be chosen again from the new state, and so on and so forth, the best searching algorithm to handle this is the Depth First Search (DFS) because the ease of backtracking when no values found for a variable, and low cost of the memory, which is as max a stack full of all the variables.

Choosing a variable isn't always random, there's a various types of heuristics to choose a variable, one of them is the minimum remaining value heuristic, which chooses the variable with the minimum available values, other one is the most constraining variable heuristic that chooses the variable that affect the larger number of other variables, those are not the only heuristics, and there can be found a new heuristic that depends on the problem itself. The same thing applies to the values, there's bunch of heuristics to choose a value like the least constraining value heuristic which chooses the value that eliminates the minimum number of values from its neighbor variables.

The backtracking is a major problem, even with those heuristics there will be backtracks in the most cases. Several ways are used to cut the search space the most famous one is the forward checking, which is nothing but eliminating a variable neighbors values that won't satisfy the constraints after assigning the variable.

To cut the search space more and more, a robust technique is used which is the Arc Consistency, this technique is combined with the forward checking, that is after eliminating a value from a variable, we will check if the arc between that variable (the neighbor of the assigned variable) is consistent with all its neighbors (the assigned variable neighbors neighbor), what's meant by a constant arc is the arc that for every value for the starting point of it there's at least one value that satisfies the constraints in the ending point of it and vice versa, otherwise the value will be eliminated, if so we will test the arc consistency again because a value is again was eliminated, and so on and so forth. If all the arcs are consistent and with some values (not empty domain) we will continue the search else, we will backtrack.

Using all of the above together can lead to efficient and fast solution with very low number of backtracks.

Crossword puzzle is a popular form of word puzzle. A crossword puzzle consists of a diagram, usually rectangular, divided into blank (white) and cancelled (black, shaded, or crosshatched) squares. This diagram is accompanied by two lists of numbered definitions or clues, one for the horizontal and the other for the vertical words, the numbers corresponding to identical numbers on the diagram. Into each of the blank squares of the diagram a certain letter of the alphabet is to be inserted, forming the words fitting the numbered definitions or clues. The words cross each other, or interlock, which gives the puzzle its name. [3]

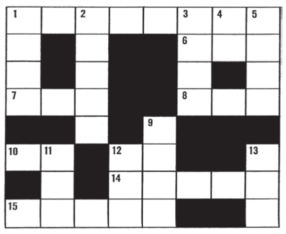

Figure – empty crosswords grid

2. Related Work

The first person ever who viewed the crosswords puzzle as a CSP was Mazlack in 1976 [4], but however he only test his approach on a small dictionary (2000 words maximum) and 2 – 3 letters each. His approach was choosing the cell as a variable and the letters as the values.

Fourteen years later in 1990 [5], Ginsberg et al, which is the first one who viewed the variables as the available slots in the grid and the values are the words of the same length of the slot.

This approach was way better than Mazlacks approach in terms of speed and memory, because he used fewer number of variables and also fewer number of values, he also used the arc consistency and what he called “smart backtracking”.

Now, all the papers and related works related to the crosswords puzzle references Ginsberg et al paper for its astounding results.

3. Project Specifications

First things first, we need to put a formulation to the problem, that is easy to understand, to code and to develop, first challenge that faced us was choosing the variable, what variable should we consider? the word itself, the cell or the available slots in the grid, after reading lots of related works, all of them agreed on Ginsberg et al variable which was the available slots in the grid, which by this we will have a minimum number of variables and easy constraints to apply to each variable, after this stage, we will choose the values, obviously the values are the words that can fit in the variable which is the available slot.

The constraints for assigning the values to the variables are as the problem says, the word must be read both directions across and down. So we can’t assign a value to a variable that eliminates all the options for other variable values.

To handle all of this we need an efficient data structures to represent each component. Also we need an Object Oriented Programming approach, hence we used the JAVA language for its amazing OOP and the strong Collections library.

3.1. The State

Since the solution is done by DFS, we need to represent a state to push and pop from the stack, the state will have all the variables in the grid and their status/properties.

3.2. The Variable

A variable will have the starting point and the ending point of it in the grid, the current word filled in it, a link to the values of the same length of the slot, a link to all its neighbors, the constraints applied from the neighbors (normal constraints) and the constraints applied from the arc consistency and a bunch of heuristics for both the variable choosing and the value choosing.

3.3. The Value

We will have a value for each length provided in the dataset, so we will have an instance of the value for all the words of length 3 for example, and so on. An instance will have (besides all the words of the same size) a two dimensional array list of size
26 * the length of this instance, this array list is what makes our approach special because it greatly saves the space because one value instance will be created for each length in the data provided. Its idea is for each letter in the English alphabet we will have a list telling us what words have this letter are in the ith positon, so the constraints in variable will be an intersection of those lists, note that the constraints are also a link to the array list in the value instance, so by this we incredibly reduced the space complexity, leaving the process of finding the available words to just intersection between those lists.

Back to the variables, as mentioned above the constraints will be just links to the lists of the value instance which created once linked to the variable. The constraints will be updated after each assignment, so a new entry will be added to the constraints of the neighbor of a variable after assigning that variable.

3.4. Forward Checking

For the forward checking, adding constraints to the neighbors will just be enough, because when we choose a neighbor we will do the intersection process on the constraints and the false values will be eliminated. When adding the constraints to the neighbors we will also check the number of available values (a temporally intersection process) if the number is zero, we will look for another value, if there’s no values left, then we will backtrack.

3.5. Arc Consistency

After the forward checking phase is done, it’s time for the arc consistency, we will traverse each neighbor and check the arc consistency of each one, if a variable is done with 0 values so we will choose another value, if they are done then we will backtrack.

This was just the project specifications, more details on how the things exactly done are discussed in the Approach section.

4. Data

The input data was a file containing the words, each word has a list of synonyms, a description to it or a clue, and the topic.

The user can choose what topics to fill the grid with. After filling the grid a pop-up window will appear containing a grid with the clues leading to the filled words, each one is numbered and linked to the grid. The clues are for both across and down words so the user can publish or print it.

5. Evaluating Criteria

The project was at first very challenging and confusing, a good formulation was the key to get our hands dirty with the code.

Even after the formulation the challenges started to occur one by one, from what data structure to use in each stage and how to reduce the memory of the values, and so on. The memory used at first was large because we used trivial and brute force techniques to implement our formulation, by spending more time on each stage we reduced the space in an amazing manner.

The first results were without the arc consistency, but yet they were so good. After using the arc consistency the grid was filled up even with 0 backtracks sometimes, which means little effort in terms of time complexity.

6. Approach

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.

6.1. Choosing the variable

We used a sorted list, that is sorted based on two heuristics, the main heuristic is the most constraining variable heuristic, 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 appreciate value a backtracking will occur.

6.2. Choosing the value

After choosing the variable, the next stage will be the value picking, we want to choose a value that maximize others neighbors values. Not just maximize them in total but also no zero neighbor values. This is done simply by knowing how each letter frequency in each position using the array list in the value which we are linking to, and that's our heuristic for the value. If there's no such a value a feedback will be sent to the DFS loop so that a backtracking will occur.

6.3. Forward Checking

The forward checking will happen after each successful assignment (the assignment that maximize neighbor's options with no zero options for any one), in forward checking we will iterate over all the neighbors and add the constraints that came from assigning the value. Any neighbor that its values became zero will just be enough indicator to choose other value to assign. The other value will be the right next value that satisfy the choosing the value condition. If there's no other value, then a backtracking will occur.

6.4. Arc Consistency

After the forward checking some of the neighbors values are eliminated, when so, the arc consistency stage will start. In arc consistency we will traverse all the neighbors, and for each one we will take the intersection cell of them and their neighbors, after knowing the intersection cell and its location for both the neighbor and the neighbor's neighbors we will add a new type of constraints called the arc consistency constraints, each variable will have this type of constraints, and what it does is basically ORing the available values from the normal constraints with the arc consistency values, so after the intersection phase we had for example {"cat", "car", "bar", "cur"} and the arc consistency list says that the character at position 0 must be 'c' and the character at position 2 must be 'r', the new available words will be {"car", "cur"} if the new available values were a null set then we have to choose other value, the arc consistency technique is done for both the starting point and the ending point, that is for the variable neighbor and for the variable neighbor's neighbor.

6.5. Depth First Search (DFS)

The depth first search is done just like any DFS approach, we started from pushing an 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 does not satisfy while choosing the value then a backtrack will occur, when the stack is empty then there's no such word combinations that can fit the grid, otherwise the solution will be found during the DFS.

Choosing the variables and the values is all done by the heuristics, the forward checking and the arc consistency just like discussed above.

All of this will be displayed in the UI, from the grid to all its components to the list of words. a log will be displayed to the user telling him what happend each iteration in the DFS, and finally a summary window button will appear that displays all the main events that happened during filling the grid, like the number of iterations, the number of backtracks, the time spent and the space used.

7. Results and analysis

The results can vary from input file to another, but on average the results were so good and fast. The real problem will occur when the file is so small that doesn't contain a possible combination to fill the grid, when so the dfs will keep searching for the solution on all the combination, and that takes some time. on average file sizes like 5000 ~ 5K words our program behaves so good and can find the solution in a low number of backtracks (sometimes ZERO)!, but when the input file constrains all the dictionary (~220K words) there may be tons and tons of backtracks, because the forward checking and arc consistency will fail the predict the wrong assigning when there's always a word that can be found to fit the constraints.

8. Development

Many steps can be implemented to improve the project, but all of them need to be well studied and understood, in the class we haven't take care of all the aspects of the CSPs, there are tons of optimization and solving techniques like the phase transition phenomena, which was first discussed in the fashion of crosswords puzzle in Anbulagan and Adi Botea paper.[6] Their paper has the best results so far in the crosswords puzzle as a CSP approach.

Speaking about us, if we had enough time we could learn the phase transition and other techniques, also we could implement a generator for the 3D crosswords puzzle.

So it's a matter of time to do all of the discussed above, maybe in our next vacation we will have time to handle all of this.

9. Conclusion

Constraints Satisfaction Problems are sometimes the best way to solve a problem with, we can look at a problem and right away find a formulation to it that fits the CSP approach, what's challenging is not all formulating can lead to a solution in easy steps, so the formulation needs to be easy to program, also the formulation must have a minimum effort in the space and the time complexities. The code for CSP problem is pretty much Object Oriented Approach, but anyway one can use the general CSP solvers to handle the problem. The code writing is an incremental process, for example you can't program the values and the variables aren't ready yet. Besides it's an incremental process it was very buggy (every modification generates a bunch of edge cases) so we had to write everything very carefully and slowly.

For us it was a powerful experience, we haven't ever programmed something as strong as the CSP. We gained professionality with the JAVA Collections library as well as the OOP in real life problem, also we gained new skills with using the UI development in JAVA and how to move the data between each window of the UIs.

10. References

[1]: Mauro, Jacopo. Constraints Meet Concurrency - A new approach to combine constraint programming and concurrency theory.

[2]: Ariel Procaccia - Carnegie Mellon University – Mathematical Foundations of AI.

[3]: www.britannica.com/topic/crossword-puzzle

[4]: L. J. Mazlack. Computer Construction of Crossword Puzzles Using Precedence Relationships Artiial Intelligence 7, pp 1-19, 1976.

[5]: M. L. Ginsberg et al. Search Lessons Learned From Crossword Puzzles Automated Reasoning, pp 210-215, 1990.

[6]: Anbulagan Botea and Adi Botea. Crossword puzzles as a constraint problem. In CP'08: Proceedings of the 14th international conference on Principles and Practice of Constraint Programming, pages 550–554, Berlin, Heidelberg, 2008. Springer-Verlag.