Crosswords with Constraint Satisfaction Problems

September 17, 2023

Introduction

Constraint Satisfaction Problems (CSPs) are a class of mathematical problems where variables are subject to constraints that must be satisfied to find a valid solution. CSPs are commonly used in artificial intelligence and computer science to model and solve a wide range of real-world problems, including puzzles, scheduling, and optimization tasks.

In this exercise, you will be implementing a CSP solver for crossword puzzles. The code template includes gaps (indicated by ___) that need to be filled in. The solver uses techniques such as node consistency, arc consistency (AC3), and backtracking to find a valid assignment of words to crossword puzzle variables.

Instructions

Please follow these instructions to complete the provided code template accurately. Ensure that you do not modify the existing code structure, function names, or method signatures unless explicitly instructed to do so. The following methods need to be completed by you:

1. **enforce_node_consistency**
    - In the `enforce_node_consistency` method, remove words from each variable's domain that are not node-consistent. Node consistency is given here by the length of the words in the variable's domain.
    - Iterate over each variable in `self.domains` and, for each word in `self.domains[variable]`, check if its length matches the length of the variable. If not, remove it from the domain.
2. **revise**
    - In the `revise` method, make variable `x` arc consistent with variable `y`.

- Use the overlap information provided by `self.crossword.overlaps[x, y]` to determine overlap positions (indices `v1` and `v2`).
- For each word `x_i` in the domain of `x`, check if there exists a word `y_j` in the domain of `y` such that `x_i[v1] == y_j[v2]`. If not, remove `x_i` from the domain of `x`.

3. **ac3**

   - In the `ac3` method, enforce arc consistency on the CSP. In this case, the binary constraints on a variable are given by its overlap with neighboring variables (words sharing common letters).
   - Initialize a queue of arcs (edges) to process. If `arcs` is None, create a list of all arcs in the problem (all possible pairs of variables that share an edge).
   - Implement the AC3 algorithm using a while loop, processing each arc with the `revise` function. Ensure that you return `True` if arc consistency is enforced and no domains are empty; otherwise, return `False`.

4. **assignment_complete**

   - In the `assignment_complete` method, check if the assignment is complete.
   - Iterate over all variables in `self.crossword.variables` and, for each variable, check if it exists in the `assignment` dictionary and if the assigned word is among the available words. Return `True` if all variables are assigned valid words; otherwise, return `False`.

5. **consistent**

   - In the `consistent` method, check if the assignment is consistent.
   - Iterate over each assigned variable `variable_x` and its assigned word `word_x` in the `assignment` dictionary.
   - Check for word length consistency and ensure that assigned words are unique and overlap correctly. Return `True` if consistent; otherwise, return `False`.

6. **select_unassigned_variable**

   - In the `select_unassigned_variable` method, select an unassigned variable for assignment.
   - Calculate the remaining domain size and degree (number of neighbors) for each unassigned variable, and use these criteria to select the next variable for assignment.

7. **backtrack**

- In the `backtrack` method, implement the Backtracking Search algorithm.
- Check if the assignment is complete. If not, select an unassigned variable and iterate over its domain, trying each value recursively.
- Use heuristics for variable selection and value ordering to improve search efficiency.

8. **solve**
   - In the `solve` method, select the appropriate methods to create your CSP crossword solver.

9. **Run your CSP crossword solver**
   - Test your implementation with crossword puzzle instances to verify that it works correctly.
   - To run your program, you can run a command like python generate.py data/structure1.txt data/words1.txt, specifying a structure file and a words file. If an assignment is possible, you should see the resulting assignment printed in your terminal.

Conclusion

By following these instructions, you can complete the code template to create a robust CSP solver for crossword puzzles. CSPs and related algorithms are essential tools for solving complex problems in various domains. More detailed instructions about the logic implemented in this exercise can be found in this page

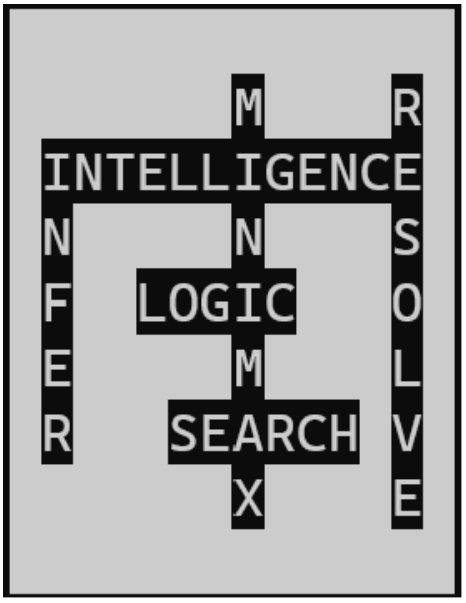

Figure 1: Example of CSP crossword solver output