Project 2: Constraint Satisfaction Techniques

Luke Runnels

March 4, 2023

1 Introduction

1.1 Constraint Satisfication Problems

Constraint Satisfication Problems(CSPs) are highly studied problems in the field of Artificial Intelligence. According to Russell et al.[1], CSPs are typically modeled as a set of variables χ, a set of domains D for each variable in χ, and a set of constraints C that constrain the possible assignments for each variable in χ. A CSP is considered *complete* if all variables in χ have been assigned a value from D that satisfy all constraints in C.

2 Problems Used

2.1 Map Coloring Problem

In the Map Coloring Problem(MCP), the most common representation is a graph $G = (V, E)$, where V is the total number of nodes and E is the total number of edges between the nodes. Given a set of colors

$$Cr = \{Cr_1...Cr_n\} \tag{1}$$

the goal of the MCP is to assign a color Cr_i to every node in G, where, given an arbitrary node V_i, all of the *neighbors* connected to V_i by the edges in E are colored differently.
In terms of a CSP, the set of variables are the set of all nodes

$$V = \{V_1...V_m\} \tag{2}$$

from the graph G. The set of domains are the set of all colors from (1) assigned to each variable in (2). The set of constraints are given by

$$C = V_1 \neq V_2 \neq ... \neq V_k \tag{3}$$

where $\{V_1...V_k\}$ are connected by the edges E.

2.2 Sudoku

In Sudoku, a partially fill 9x9 matrix represents nine 3x3 regions, 9 rows, and 9 columns. The matrix contains 81 cells, where each cell is assigned a value between 1 and 9. The general constraint in this problem is that every cell must have a distinct value from all of the corresponding cells in the row, column, and region.
In terms of a CSP, the set of variables are cells

$$V = \{V_{11}...V_{ij}...V_{99}\} \tag{4}$$

where i represents the row number and j represents the column number.
The set of domains are the values 1 through 9 assigned to each variable in (4).
The set of constraints, according to Russell et al.[1], is

$$\begin{gathered}
V_{11} \neq V_{12}... \neq V_{19} \\
V_{21} \neq V_{22}... \neq V_{29} \\
... \\
V_{11} \neq V_{21}... \neq V_{91} \\
V_{12} \neq V_{22}... \neq V_{92} \\
... \\
V_{11} \neq V_{12} \neq V_{13} \neq V_{21} \neq V_{22} \neq V_{23} \neq V_{31} \neq V_{32} \neq V_{33} \\
...
\end{gathered}$$

where the first grouping represents row distinction, the second grouping represents column distinction, and the third grouping represents region distinction.

3 Algorithm Explanation

3.1 Depth first search with backtracking

The most common algorithm for solving CSPs is using a modified version of depth first search called depth first search with backtracking. It will pick a variable V_i from the CSP and test every possible domain value D_j from V_i in an attempt to build a complete assignment for the CSP. If all domain values from V_i are exhausted without the assignment of V_i, the algorithm will return a *"failure"*, where it will *"backtrack"* by removing the *"failed"* variable from the assignment. This process is repeated multiple times until a complete assignment for the CSP is found.

A rough implementation in pseudocode based on Figure 6.5 from Russell el al.[1] is sketched out below. This algorithm factors in ordering and inference heuristics with the goal of finding solutions faster by pruning the search. These heuristics will be described in the next section.

Procedure 1 *DFS_Backtrack*

```
Input: $CSP, Assignment$
Output: $solution, failure$
  if Assignment is complete then return Assignment
  end if
  $Variable \leftarrow Ordering\_Heuristic(CSP, Assignment)$
  for every value of Variable's domain do
      if value is consistent with $CSP's$ constraints then
          Assign value to $Variable$ and add $Variable$ to $Assignment$
          $Inferences \leftarrow Inference\_Heuristic(CSP, Assignment, Variable)$
          if $Inferences \neq$ "failure" then
              Add $Inferences$ to $CSP$
              $Result \leftarrow DFS\_Backtrack(CSP, Assignment)$
              if $Result \neq$ "failure" then return $Result$
              end if
              Remove $Inferences$ from $CSP$
          end if
          Remove value assignment from $Variable$ and remove $Variable$ from $Assignment$
      end if
  end for
    return "failure"
```

4 Heuristics Used

It is common for the search space of a CSP to be exponential. Thus, depth first search with backtracking alone is often impractical for solving most CSPs, especially both the Map Coloring Problem for a large number of nodes and edges as well as Sudoku with a large number of cell assignments missing. To help the search process, various heuristics for selecting an unassigned variable are employed. There are two types of heuristics that will be experimented with: Ordering and Inference. These heuristics work together to *prune* depth first search in order to detect an assigned variable that will lead to a dead-end, or *failure*.

4.1 Ordering Heuristics

The goal of an ordering heuristic is to select an unassigned variable based on the state of its domain. It should be noted that ordering heuristics factor in the current state of the CSP's domain. That is, given every unassigned variable in the CSP, it takes into account the current *legal* values in each of the variable's domain. How the current domain state changes is dependent on an inference heuristic, which will be described in the next subsection.

4.1.1 Random

This heuristic will be used as a basis for comparison with the rest of the ordering heuristics. Essentially, the random heuristic selects an unassigned variable randomly from the CSP, without factoring in the current state of the CSP's domain. It can be interpreted as applying no ordering heuristic to the CSP for selecting an unassigned variable.

4.1.2 Minimum Remaining Value

The idea behind this heuristic is choosing an unassigned variable with the $fewest$ legal values left in its domain. For example, if an arbitrary CSP has the unassigned variables V_1, V_2, and V_3, with the current domain state given by $D_1 = \{1\}$, $D_2 = \{1, 2\}$, and $D_3 = \{1, 2, 3\}$, then the minimum remaining value heuristic will select D_1. If the heuristic detects multiple unassigned variables that have the same number of *legal* values left, then it will randomly select one of those variables. This heuristic works to prune the search by returning a *failure* back to Procedure 1 if it detects a variable with no *legal* values left.

4.1.3 Minimum Remaining Value with Degree

This idea behind this heuristic is combining the Minimum Remaining Value heuristic described previously with a new heuristic called degree. The degree heuristic works by selecting unassigned variables that are involved in the largest number of constraints with other unassigned variables in the CSP. For example, if an arbitrary CSP has the unassigned variables V_1, V_2, and V_3, with the two binary constraints $V_1 \neq V_2$ and $V_2 \neq V_3$, then V_2 will be selected because it is involved in the highest number of constraints. This combined heuristic works to prune the search through the Minimum Remaining Value heuristic only.

4.2 Inference Heuristics

To prune the search for finding a solution to a CSP, Procedure 1 must detect a variable with no legal values left in its associated domain. The ordering heuristics selects unassigned variables based on the number of legal values left. However, for ordering to be utilized efficiently, an inference heuristic needs to be factored in to reduce the domain state.

4.2.1 Default

The option for applying *no* inference heuristic will be called the default heuristic. Essentially, the domain state will remain unaltered. This has the consequence of removing pruning from Procedure 1.

4.2.2 Forward Checking

The idea behind this heuristic is eliminating legal values from the immediate neighbors of an unassigned variable. For example, if an arbitrary CSP has the unassigned variables V_1, V_2, and V_3, a current domain state $D_1 = \{1, 2, 3\}$, $D_2 = \{1, 2, 3\}$, and $D_3 = \{1, 2, 3\}$, constraining relationships between all three variables, and an incoming assignment of 1 to V_1, then forward checking will update both D_2 and D_3 to $\{2, 3\}$. A failure will be returned to Procedure 1 if this heuristic attempts to delete a value from a domain that has no *legal* values left.

4.2.3 Arc-3 Consistency

The idea behind this heuristic is propagating domain changes throughout the CSP. For example, if an arbitrary CSP has the unassigned variables $V = \{V_1 ... V_N\}$, the domain state $D_i = \{d_1 ... d_N\}$ for every variable V_i, and the variable V_i is in a binary constraint with every other variable in the CSP, then this heuristic will propagate domain changes through all of the *arcs* in the CSP.

Arcs are pairings between unassigned variables (X_i, X_j) where X_i is referred to as the tail of the arc and X_j is referred to as the head of the arc. This heuristic will first consider every possible *arc* in the CSP. It will scan through all possible domain values in the tail variable(X_i). If there is no domain value in the head variable(X_j) that satisfies the constraint between X_i and X_j, then the domain value in X_i that causes this inconsistency will be removed. If a domain value is removed from X_i, then all of the arcs (X_j, X_k) have to be checked as well, where X_j is a neighbor variable of X_i and X_k is the neighbor variable of X_j.

5 Performance and Analysis

Most of the heuristics combinations presented here were implemented according to the examples in section 4. However, the arc consistency heuristic was initialized with arcs between a single selected unassigned variable and its unassigned neighbor variables. This decision, suggested by Russell et al. [1], was used to cut down on the higher runtime of this heuristic, which would have been present if the heuristic was initialized with all possible arcs in the CSP.

All of the runtimes presented below were measured in seconds.

5.1 Map Coloring Problem

For testing the performance of every ordering and inference combination against the Map Color Problem, ten graphs of the problem containing 10 nodes, 50 nodes, and 100 nodes were inclusively generated, which totaled thirty graphs of varying search space size. The best average runtime for the following tables are highlighted in green.

	Inference Heuristic		
Order Heuristic	Default	For. Check	AC-3
Random	1.08	0.62	0.28
MRV	0.73	0.38	0.55
MRV-Degree	0.85	0.73	0.25

Table 1: Average runtime(seconds) for 10 MCPs with 10 nodes and six possible colors.

	Inference Heuristic		
Order Heuristic	Default	For. Check	AC-3
Random	4.20	4.82	5.12
MRV	3.09	4.61	3.80
MRV-Degree	5.64	3.50	3.40

Table 2: Average runtime(seconds) for 10 MCPs with 50 nodes and six possible colors.

	Inference Heuristic		
Order Heuristic	Default	For. Check	AC-3
Random	7.67	6.08	7.36
MRV	6.62	6.70	7.30
MRV-Degree	8.40	5.95	6.83

Table 3: Average runtime(seconds) for 10 MCPs with 100 nodes and six possible colors.

The results are quite interesting. On average, it seems that no matter what ordering heuristic got chosen, the default inference performed the worst. This makes sense as the default inference will not support any pruning, so depth first search had to exhaust the entire search space for all of the graphs. Interesting, applying a meaningful ordering heuristic with the default inference seemed to produce worse runtimes. This does make sense, as the ordering heuristics are not able to effectively exploit the domain state with the default heuristic, so they are effectively overhead.

In addition, it appears that the Arc-3 Consistency inference heuristic performed the best for smaller graphs and the forward checking heuristic performed the best on average for the larger graphs. This does seem reasonable because for larger graphs, the variables will have more neighbors, so the Arc-3 heuristic is less impactful. But, somewhat surprisingly, forward checking only calculated the absolute minimum average runtime for the largest graph, and not the *mid-sized* graph. However, it should be noted that the difference in performance between the heuristic combinations become much more pronounced as the size of a CSP grows. So this anomaly does not seem concerning, as depth first search most likely got lucky with the generated graphs.

The difference in performance between the smaller and larger graphs is also prevalent with the ordering heuristics. It appears that the MRV-Degree heuristic performed the best on average, with MRV coming in second and Random coming in third. Of course, Table 2 has a major anomaly with a MRV and Default heuristic combination performing the best on average. However, combinations involving the MRV-Degree heuristic seemed to perform the best overall, so this anomaly is most likely caused by the favorable graphs that were generated.

5.2 Sudoku

Similarly to the Map Color Problem, to adequately test the performance of every ordering and inference heuristic combination, numerous puzzles of the Sudoku board had to be generated. The results presented below are the average runtimes of 100 puzzles containing 10, 20, and 30 cells missing, which totaled 300 puzzles. The best average runtimes are highlighted in green.

	Inference Heuristic		
Order Heuristic	Default	For. Check	AC-3
Random	1.89	1.93	1.82
MRV	2.02	1.63	1.66
MRV-Degree	2.19	1.82	1.81

Table 4: Average runtime(seconds) for 100 sudoku puzzles with 10 cells missing

	Inference Heuristic		
Order Heuristic	Default	For. Check	AC-3
Random	4.50	4.18	4.14
MRV	4.12	3.35	3.58
MRV-Degree	4.94	3.59	3.92

Table 5: Average runtime(seconds) for 100 sudoku puzzles with 20 cells missing

	Inference Heuristic		
Order Heuristic	Default	For. Check	AC-3
Random	14.36	12.57	16.85
MRV	13.23	5.12	4.97
MRV-Degree	25.16	6.61	7.05

Table 6: Average runtime(seconds) for 100 sudoku puzzles with 30 cells missing

On average, it seems that any combination involving Random ordering or Default inference performed worse. Just like the Map Coloring problem, any combination involving these heuristics would need to exhaust the search space, so it is reasonable that these combinations are worse.

For the performance of the inference heuristics, it seems that the forward checking heuristic performed better with less cells missing, while the arc consistency heuristic performed better with more cells missing. In addition, the MRV ordering heuristic performed best for all of the boards generated. These results seem be reflect the fact that every variable will always have 24 neighbors. So, with the relatively small number of neighbors, the arc consistency heuristic can detect failures relatively quickly and the degree heuristic does not provide as much optimization.

5.3 Similarities and Differences between MCP and Sudoku

There are some interesting similarities and differences with the performance of solving the Map Coloring Problem compared to solving Sudoku.

The major similarity between the two domains is that both the default inference heuristic and random ordering heuristic performed the worst. These performances are expected as the search space must be expanded entirely to find a solution.

The major difference between the two domains is the performance of the forward checking and arc consistency heuristic. For smaller CSPs, the arc consistency heuristic performed better on average with the Map Coloring Problem, while the forward checking heuristic performed better on average with Sudoku. Conversely, for larger CSPs, forward checking performed better on average with Map Coloring, while arc consistency performed better on average with Sudoku. This contrast does seem reasonable, because the number of constraints in the Map Coloring Problem is exponentially proportional to its size, while the number of constraints in Sudoku is constantly proportional to its size.

The performance contrast with the number of binary constraints between the domains seem to be explained with the ordering heuristics. In the Map Coloring Problem, the MRV-Degree heuristic seemed to perform the best on average, while in Sudoku, the MRV heuristic alone seemed to perform the best on average. Just like the contrast between the inferences heuristics, this does seem reasonable. The degree heuristic directly dependent on the number of constraints between the variables. So, it seems that the degree heuristic provided a lot of optimization to the Map Coloring Problem and not Sudoku.

6 Conclusion

Depth first search with backtracking is one of the most simple, yet effective techniques for deterministically solving constraint satisfication problems. However, the search space for most CSPs grow exponentially to the number of variables and constraints. So, ordering and inference heuristics are applied in an attempt to help prune the search space by detecting inconsistent assignments.

This paper has shown the general benefits of applying meaningful ordering and inference heuristics. Otherwise, depth first search with backtracking may run exponentially to find a solution. Whether depth first search applies which heuristic combination is dependent on the size of the CSP. For CSPs with less variables, it may be beneficial to use Arc-3 consistency. However, for CSPs with more variables, it's better to use forward checking only. For CSPs with less constraints, it is better to just use MRV alone. However, for CSPs with more constraints, it may be beneficial to use MRV with degree.

References

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearsons, 4 edition, 2021.