Applying the AC-3 Algorithm to Graph Coloring: Ensuring Different Colors for Neighboring States

Ishwak Sharda, Kshitij Goyal, Joshua Lepon

October 19, 2024

1 Introduction

The AC-3 algorithm ensures arc consistency between variables in a Constraint Satisfaction Problem (CSP). In a CSP, each variable has a domain of values, and constraints exist between variables. AC-3 operates by eliminating values from the domains that do not satisfy constraints. A CSP is defined by:

- **Variables**: $X = \{X_1, X_2, \ldots, X_n\}$
- **Domains**: $D = \{D_1, D_2, \ldots, D_n\}$ where each D_i represents the possible values for variable X_i
- **Constraints**: C, a set of binary constraints between pairs of variables that restrict their simultaneous values.

The goal of the AC-3 algorithm is to enforce *arc consistency*. A variable X_i is said to be arc-consistent with a neighboring variable X_j if for every value $x \in D_i$, there is some value $y \in D_j$ such that (x, y) satisfies the constraint between X_i and X_j.

2 Pseudocode

Pseudocode for AC-3

Pseudocode for Remove Inconsistent Values

3 Mathematical Definition and Analysis

Given a CSP defined by:

- A set of variables $X = \{X_1, X_2, \ldots, X_n\}$

Algorithm 1 AC-3 Algorithm

Input: A CSP problem with variables, domains, and constraints.
Output: True if CSP is arc-consistent, False if inconsistent.
Initialize the queue with all arcs (X_i, X_j) in CSP.
while queue is not empty **do**
  $(X_i, X_j) \leftarrow$ dequeue from queue
  if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**
    if D_i is empty **then**
      return False
    end if
    for all X_k in NEIGHBORS(X_i) **do**
      Enqueue (X_k, X_i)
    end for
  end if
end while
return True =0

Algorithm 2 Remove Inconsistent Values

Input: Variables X_i and X_j, domains D_i and D_j, and a constraint between them.
Output: True if domain of X_i is revised, False otherwise.
revised $\leftarrow$ False
for all $x \in D_i$ **do**
  if no $y \in D_j$ allows $(X_i = x, X_j = y)$ to satisfy the constraint **then**
    Remove x from D_i
    revised $\leftarrow$ True
  end if
end for
return revised =0

- Domains $D = \{D_1, D_2, \ldots, D_n\}$, where each D_i is the domain of variable X_i
- Binary constraints C_{ij} between pairs of variables X_i and X_j

We define a CSP as **arc-consistent** if for every pair of variables (X_i, X_j), for every value $x \in D_i$, there exists a value $y \in D_j$ such that the constraint $C_{ij}(x, y)$ is satisfied:

$$\forall x \in D_i, \exists y \in D_j \text{ such that } C_{ij}(x, y) = \text{True}$$

If no such value y exists for some x, x is removed from D_i, and we revise the domain as:

$$D'_i = D_i \setminus \{x \mid \forall y \in D_j, C_{ij}(x, y) = \text{False}\}$$

The AC-3 algorithm works by iterating over all arcs and enforcing arc consistency. If a domain becomes empty during this process, the CSP is deemed *inconsistent*.

3.1 Time Complexity Analysis

The time complexity of the AC-3 algorithm is $O(ed^3)$, where:

- e is the number of arcs (or binary constraints) between the variables.
- d is the size of the largest domain.

Simplified Derivation:

- For each arc (X_i, X_j), the algorithm checks all values in the domain of X_i (which has size d).
- For each value in D_i, it compares with all values in D_j (also of size d).
- This means each consistency check takes $O(d^2)$ time.
- Since each arc might be added back to the queue up to d times, the worst-case complexity for an arc is $O(d^3)$.

Therefore, the total time complexity is:

$$O(e \cdot d^3)$$

where e is the number of arcs in the CSP.

3.2 Space Complexity Analysis

The space complexity of AC-3 is $O(e + d)$, where:

- e is the number of arcs (constraints) stored in the queue.
- d is the size of the largest domain, which we must store for each variable.

4 Application to Graph Coloring

In the Graph Coloring Problem, variables represent the nodes, and the domains represent possible colors. Constraints exist between adjacent nodes that must not share the same color. AC-3 helps by pruning the domain of possible colors for each node based on its neighbors.

Example

Consider a node X_i with a domain of colors:

$$D_i = \{R, G, B\}$$

If one of its neighbors is assigned the color R, AC-3 will remove R from the domain of X_i, leaving:

$$D_i = \{G, B\}$$

This pruning of the search space helps reduce the number of possibilities, thereby making it easier to find a valid coloring for the entire graph. The use of AC-3 in this context significantly improves the efficiency of the search process for a solution.

5 Conclusion

The AC-3 algorithm is a powerful technique for enforcing arc consistency in graph coloring problems, such as ensuring that neighboring states in a map have different colors. Its efficiency in terms of both time and space makes it ideal for practical applications where constraints between adjacent regions must be satisfied. By iteratively revising the color domains for each state, AC-3 ensures that neighboring states remain properly distinguished or detects potential conflicts early in the process.

6 References

- Zhang, Y. "A New Strategy for AC-3." International Joint Conference on Artificial Intelligence, 2001, http://redwood.cs.ttu.edu/ yuazhang/publications/ac3-1-ijcai01.pdf. Accessed 18 Oct. 2024.
- "AC-3 Algorithm." Wikipedia, Wikimedia Foundation, 17 Sept. 2023, https://en.wikipedia.org/wiki/AC-3_algorithm. Accessed 18 Oct. 2024.
- Roy, Satyam. "How to Solve Constraint Satisfaction Problems (CSPs) with AC-3 Algorithm in Python." Medium, The Startup, 19 Apr. 2020, https://medium.com/swlh/how-to-solve-constraint-satisfaction-problems-csps-with-ac-3-algorithm-in-python-f7a9be538cfe. Accessed 18 Oct. 2024.

- Russell, Stuart. "Constraint Satisfaction Problems." CS 188: Introduction to Artificial Intelligence, University of California, Berkeley, Fall 2005, https://people.eecs.berkeley.edu/ russell/classes/cs188/f05/slides/chapter05-6pp.pdf. Accessed 18 Oct. 2024.
- "AC-3 Algorithm Explanation." ChatGPT, OpenAI, https://chatgpt.com/share/67131118-07a8-800c-acd4-fd6b5ef475fa. Accessed 18 Oct. 2024.