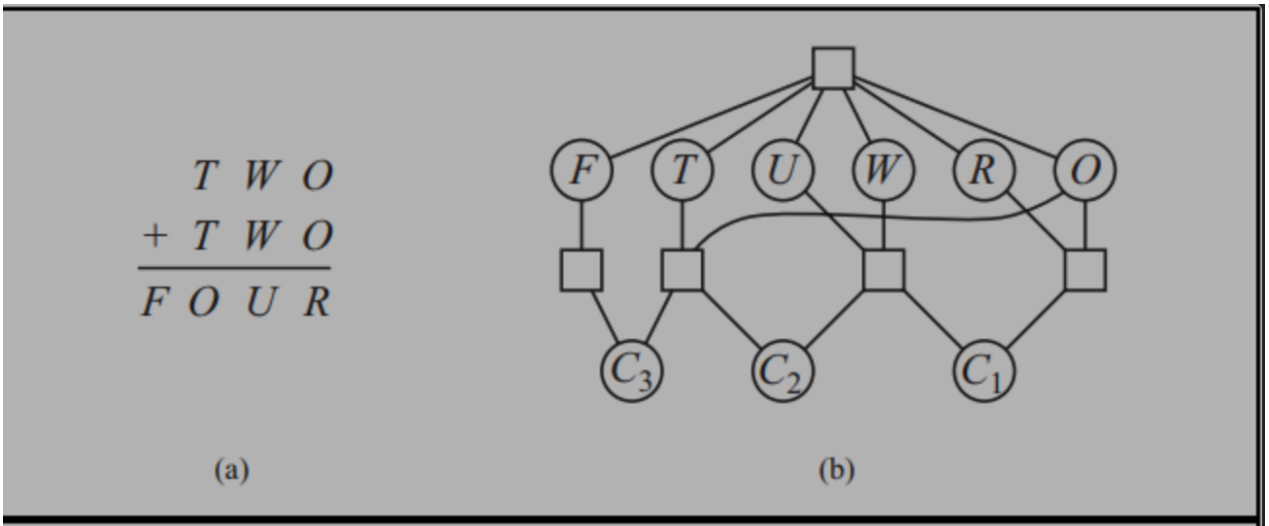
hw3

张芷苒 PB21081601

- 6.5
- 6.11
- 6.12

> **6.5** Solve the cryptarithmetic problem in Figure 6.2 by hand, using the strategy of backtracking with forward checking an
>
> 
>
>
> **Figure 6.2** (a) A cryptarithmetic problem. Each letter stands for a distinct digit; the aim is to find a substitution of digits for letters such that the resulting sum is arithmetically correct, with the added restriction that no leading zeroes are allowed. (b) The constraint hypergraph for the cryptarithmetic problem, showing the *Alldiff* constraint (square box at the top) as well as the column addition constraints (four square boxes in the middle). The variables C_1, C_2, and C_3 represent the carry digits for the three columns.

process:

1. **MRV Heuristic:** select the variable that has the fewest legal values remaining. In this case, 'F' is the most restricted since it must be 1 because adding any two same numbers can't be more than 1 on the thousands place.
2. **LCV Heuristic:** choose the value that rules out the fewest choices for the neighboring variables in the constraint graph.
3. **Forward Checking:** as we assign each value, we eliminate values from the domains of adjacent unassigned variables to prevent future conflicts.
4. **Backtracking:** If encounter a conflict that we can't resolve, backtrack to the previous variable and try a different value for it.

Step 1: Assign 'F' = 1

- MRV → start with 'F' because its value can only be '1' to satisfy the constraint that the sum must not have leading zeros.
- This does not constrain any other variables yet, don't need to apply LCV here.

Step 2: Assign a value to 'T'

- 'T' cannot be '1', and since it's being added to itself, it must be a value that when doubled does not exceed 9 and is even. The possible values for 'T' are therefore 2, 4, 6, or 8. Start with the lowest even value 'T' = 2 and move up if needed.
- Forward checking eliminates '2' as an option for other variables.

Step 3: Assign a value to 'W'

- 'W' can't be '1' or '2'. Pick the next lowest value, 'W' = 3.
- Forward checking now eliminates '3' for other variables.

Step 4: Assign a value to 'O'

- 'O' can't be '1', '2', or '3'. Need to consider that 'O' + 'O' will result in a carry, meaning 'O' must be 5 or greater to ensure 'U' is not 0, which would not comply with the constraint of all digits being different. Let's pick 'O' = 5, which is the least constraining because it allows for more options for 'R' and 'U'.
- Forward checking eliminates '5' for other variables.

Step 5: Check the 'U' and 'R' values

- 'O' is 5, so 'O' + 'O' = 10, and 'U' must be 0, which violates the no leading zero constraint, so that 'O' = 5 is incorrect. We will have to backtrack and choose a different value for 'O'.

Step 6: Backtrack and reassign 'O'

- try the next possible value for 'O', which is '6', the constraints we have for 'U' and 'R'. '6' + '6' = 12, so 'U' = 2, but since 'T' = 2, this is not allowed. Continue to 'O' = 7.

Step 7: Final Assignments

- With 'O' = 7, we have 'O' + 'O' = 14, so 'U' = 4 and 'R' = 8.

- Forward checking eliminates '4', '7', and '8' for other variables.
- Continue with these values and if at any point we encounter a violation of the constraints, backtrack and try the next least constraining value.

> **6.11** Use the AC-3 algorithm to show that arc consistency can detect the inconsistency of the partial assignment $WA = green, V = red$ for the problem shown in Figure 6.1

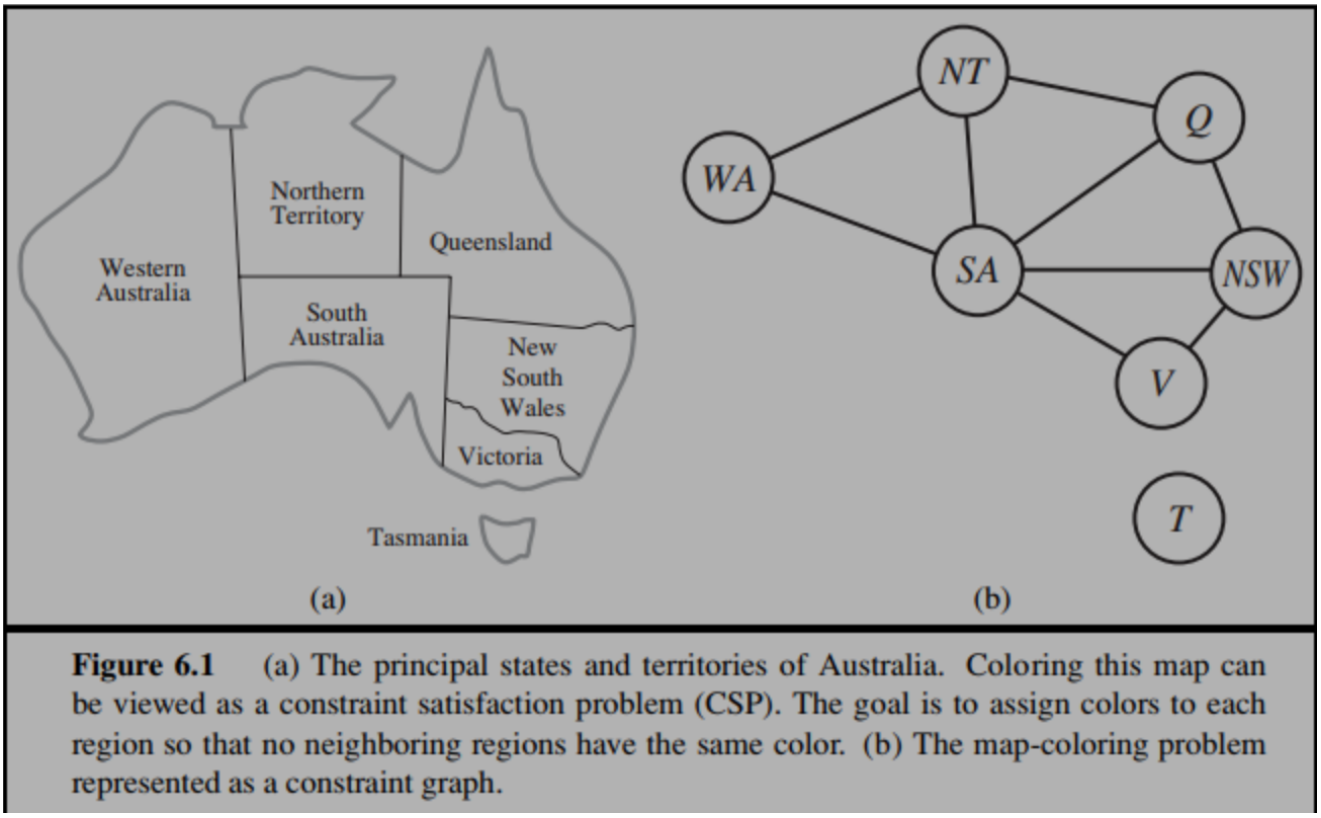


Figure 6.1 (a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

The AC-3 (Arc Consistency Algorithm #3) algorithm can be used to check if a given partial assignment is consistent in a Constraint Satisfaction Problem (CSP) such as map coloring. The AC-3 algorithm works by making arcs consistent, which means for every value of one variable, there should be a corresponding legal value of an adjacent variable.

In the case of the map coloring of Australia provided, if Western Australia (WA) is assigned green and Victoria (V) is assigned red, we need to check if this partial assignment is consistent with the given constraints.

The algorithm maintains a queue of arcs to check and continually removes an arc and checks for consistency between the variables at either end of the arc. If the domain of a variable is reduced, all its neighbors, except for the one that caused the reduction, are added to the queue.

The arcs would initially be:

- (WA, NT), (WA, SA)

- (NT, WA), (NT, Q), (NT, SA)
- (SA, WA), (SA, NT), (SA, Q), (SA, NSW), (SA, V)
- (Q, NT), (Q, SA), (Q, NSW)
- (NSW, Q), (NSW, SA), (NSW, V)
- (V, SA), (V, NSW), (V, T)
- (T, V)

Given the partial assignment WA = green and V = red, the constraints tell us that NT, SA adjacent to WA cannot be green, and NSW, SA, and T adjacent to V cannot be red.

The AC-3 algorithm proceeds to enforce consistency:

1. Starting with WA = green, it removes green from the domains of NT and SA.
2. Then it considers V = red, removing red from the domains of NSW, SA, and T.
3. If any domain becomes empty, the algorithm stops and reports an inconsistency because there's no valid color left for that region which respects the current partial assignment.

The algorithm will detect inconsistency if, after considering all constraints, there's no valid assignment possible for the neighboring states. For example, if NT and SA have no colors left after WA = green, or if NSW, SA, and T have no colors left after V = red, the algorithm will indicate an inconsistency.

Applying the AC-3 algorithm in this specific problem, however, would not immediately detect an inconsistency since there are more than two colors to choose from (typically at least three are used in map coloring problems). Given that only WA and V are assigned colors, NT, SA, Q, NSW, and T still have at least one color in their domains that would not conflict with the green WA or the red V, assuming there are at least three colors in total to choose from.

In conclusion, with the partial assignment WA = green and V = red, and assuming at least three colors to choose from, the AC-3 algorithm would not detect an inconsistency simply with this information. Further assignments or a restriction to only two colors could lead to a situation where AC-3 would detect an inconsistency.

> **6.12** What is the worst-case complexity of running AC-3 on a tree-structured CSP?

O(n).

每个节点最多只有一个父节点和多个子节点。运行AC-3算法时，每个节点会与它的父节点和子节点进行一次弧一致性检查。因为树没有环，每个弧只需要检查一次。

算法从树的叶节点开始，朝着根节点进行弧一致性检查，每次检查将可能影响到的值从变量的域中排除。由于每个节点至多只有一个父节点，因此对于每个节点，检查并更新其与父节点的关系最多只需要进行一次。同样，节点与其子节点之间的关系也是如此。一旦对某个节点的域进行了更新，就不需要再次回到该节点，因为树形结构保证了不会有进一步的回溯。

所以对于树形CSP，AC-3算法的每个弧只需要访问一次，总共需要的操作步骤与CSP中的变量数n成线性关系。→ 最坏时间复杂度O(n)。