# Arc-Consistency in Dynamic Constraint Satisfaction Problems

## Understanding AC-4 and DnAC-4

Muhammad Mohsin Siddiqi

Seminar on Artificial Intelligence 2
Faculty of Mathematics and Physics
Charles University

# Outline

# Introduction

A **Constraint Satisfaction Problem (CSP)** consists of:

- **Variables**
- **Domains**
- **Constraints**

**Goal:** assign values to variables such that all constraints are satisfied.

**Examples:**

- scheduling
- timetabling
- Sudoku
- map coloring

# Background

- Constraint Satisfaction Problems are widely used in AI.
- Many real problems are dynamic rather than static.
- Constraints may be added or removed over time.
- Recomputing everything from scratch is inefficient.

**Goal:** understand how arc consistency can be maintained efficiently when the problem changes.

# Sources

This presentation is based mainly on:

**Christian Bessière (1991)**
**Bessière, C. (1991). Arc-Consistency in Dynamic Constraint Satisfaction Problems. AAAI-91, pp. 221–226.**

Main topics:

- Dynamic CSPs
- AC-4
- limitation of AC-4 in dynamic settings
- DnAC-4

# Map Coloring Example

**Variables:**

$WA,\ NT,\ SA,\ QLD,\ NSW,\ VIC$

**Domain:**

$$\{Red,\ Green,\ Blue\}$$

**Constraint:** Neighboring regions must have different colors.

# Why Map Coloring Is a CSP

In the map-coloring example:

- each region is a variable
- each color is a possible value
- each shared border gives a constraint

The goal is to find a valid coloring where neighboring regions have different colors.

# Arc Consistency

A value is **arc-consistent** if it has at least one supporting value in every neighboring variable.

**Main idea:** If a value can never satisfy a neighboring constraint, remove it.

This reduces the search space and makes solving easier.

# Example

Suppose:

$$A \neq B$$

**Domains:**

$$A = \{Red\}, \qquad B = \{Red, Blue\}$$

Since $A$ can only be Red, Red in $B$ has no support.

**After filtering:**

$$B = \{Blue\}$$

# Motivation

After arc consistency:

- domains become smaller
- impossible values are removed early
- search becomes faster

Arc consistency does not solve the whole CSP, but it simplifies the problem.

# AC-4 Intro

AC-4 is a classical algorithm for enforcing arc consistency in a static CSP.

Its basic idea is:

- every value must have support
- AC-4 stores how many supports each value has
- if the number of supports becomes zero, the value is deleted

# AC-4 Algorithm

## AC-4

```
Algorithm AC-4
procedure AC-4
  Q <- INITIALIZE;
  while not Q empty
    select and delete any pair <j,b> from Q;
    for each <i,a> from Sj,b do
      counter[(i,j),a] <- counter[(i,j),a] - 1;
      if counter[(i,j),a]=0 and a is still in Di then
        delete a from Di;
        Q <- Q union {<i,a>};
      endif
    endfor
  endwhile
end AC-4
```

**Main idea:** delete unsupported values and propagate deletions.

# How AC-4 Works

1. Start with values that already have no support.
2. Put them into a queue.
3. Remove one deleted value from the queue.
4. Update the counters of values that depended on it.
5. If another value loses all support, delete it too.

Repeat until no unsupported values remain.

# Why AC-4 Is Efficient

AC-4 is efficient because:

- supports are stored in advance
- counters are updated incrementally
- it avoids repeated checking from scratch

So AC-4 is very good for static CSPs.

# Dynamic CSP Intro

A Dynamic CSP is a sequence of CSPs where the problem changes over time.

**Two common updates:**

- **Constraint addition** (restriction)
- **Constraint removal** (relaxation)

# Why AC-4 Is Not Enough

AC-4 works well when constraints are added.

But when a constraint is removed:

- some deleted values may become valid again
- AC-4 does not remember why a value was deleted

So AC-4 does not know which values should be restored.

# Main Limitation of AC-4

The key limitation is:

**AC-4 forgets the reason for past deletions**

Because of this, when a constraint is relaxed, AC-4 often has to restart from the initial domain.

# Main Idea of DnAC-4

The paper proposes:

**DnAC-4**

**Key idea:** When a value is deleted, remember why it was deleted.

# Justification Concept

Each removed value stores a **justification**.

**Example:** If value $(i, a)$ becomes unsupported because of $\{i, j\}$, store:

$$\text{justif}(i, a) = j$$

This allows the algorithm to trace the cause of deletion later.

# Why Justification Helps

When a constraint is removed:

- DnAC-4 can find values deleted because of that constraint
- those values are reconsidered first

So the algorithm restores values incrementally instead of recomputing everything.

# Example CSP Used in the Paper

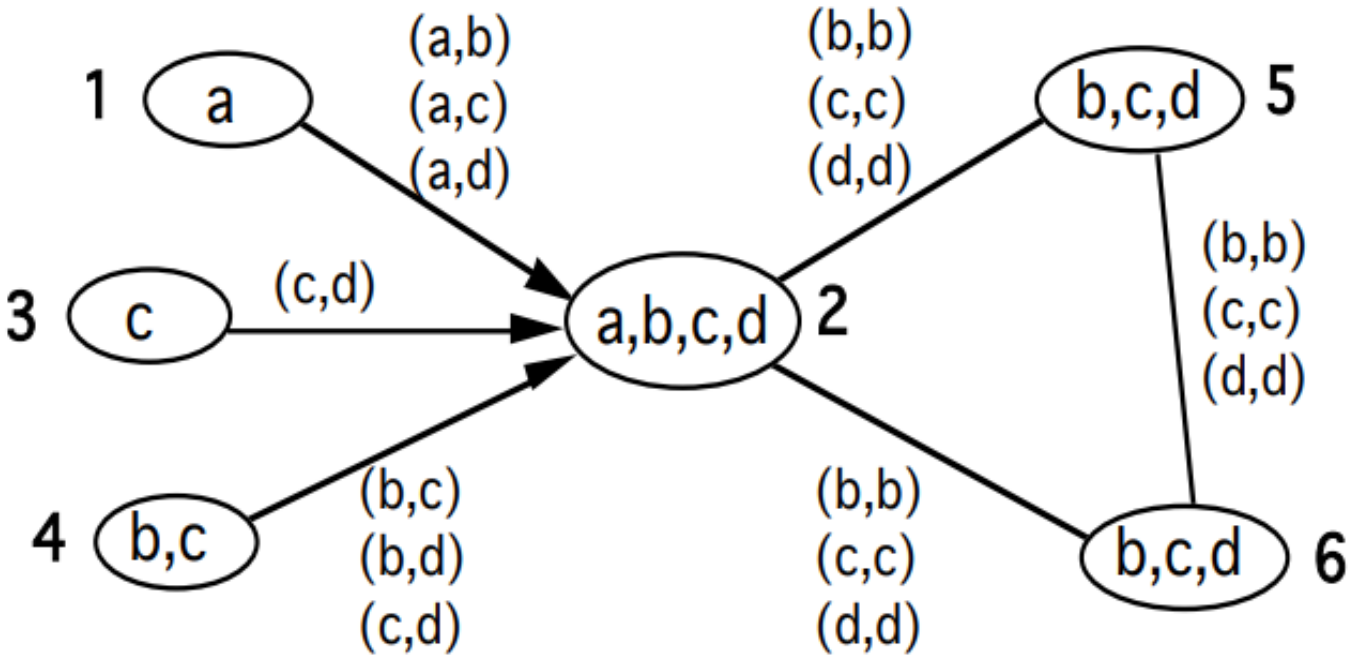


Figure 1: An example of CSP

# Example After Arc Consistency

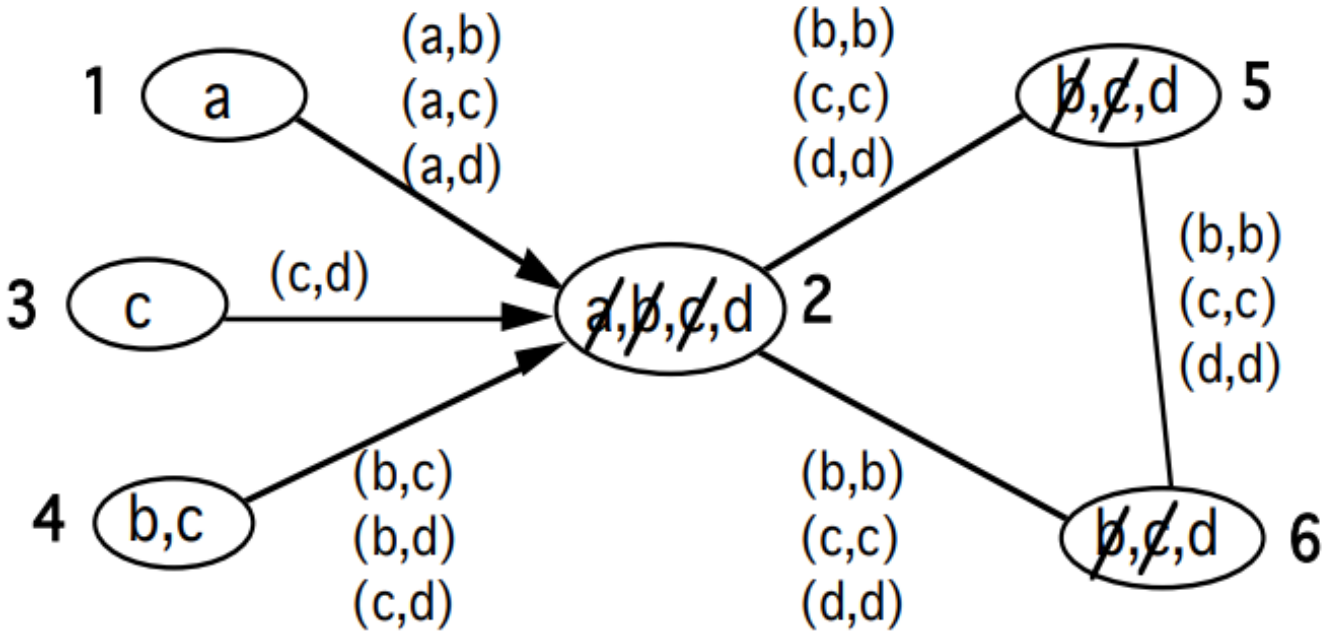


Figure 2: The CSP of fig.1 after application of an arc-consistency algorithm

# What These Two Figures Show

These two figures illustrate:

- the original CSP used in the paper
- the same CSP after applying arc consistency

They help show how values are removed before dynamic updates are considered.

# Main Data Structures of DnAC-4

DnAC-4 uses:

- support counters
- domain table $D$
- justification table
- suppression list (SL)
- relaxation list (RL)

# Constraint Addition in DnAC-4

When adding a constraint:

1. compute supports
2. find unsupported values
3. remove them
4. propagate deletions

This is similar to AC-4, but DnAC-4 also stores justifications.

# Addition Procedures from the Paper

```
procedure Beg-Add ((i, j ) ; var SL );
   begin
1  for each b ∈ dom(j ) do S_jib , ∅;
2  for each a ∈ dom(i )do
      begin
3     Total , 0;
4     for each b ∈ dom(j ) do
5        if ((i, a ), (j, b )) ∈ R_ij then
            begin
6           if D(j, b )=true then Total , Total + 1;
7           Append(S_jib , (a ));
            end;
8     counter[(i, j ), a ] , Total ;
9     if counter[(i, j ), a ]=0 then Append(SL, [(i, j ), a ]);
      end;
   end;
```

Figure 3

# Propagation and Relaxation Procedures

```
procedure Propag-Suppress (var SL );
   begin
1  while SL ≠ ∅ do
      begin
2     choose [(i, m ), a ] from SL and remove it from SL;
3     if D(i, a )=true and counter[(i, m ), a ]=0 then
         begin
4        justif(i, a ) , m ;
5        D(i, a ) , false;
6        for each j / {i, j } ∈ c do
7           for each b ∈ S_ija do
               begin
8              counter[(j, i ), b ] , counter[(j, i ), b ] - 1;
9              if counter[(j, i ), b ]=0 then
                  Append(SL, [(j, i ), b ] ) ;
               end;
         end;
      end
   end;
```

Figure 4

```
procedure Init-Propag-Relax ({k, m }; var SL );
   begin
   { Step 1: values whose justification was {k, m } are
     put in RL }
1  RL , ∅ ;
2  for each a ∈ dom(k ) do
      if justif(k, a )=m then
         begin
         Append(RL, (k, a )); justif(k, a ) , nil ;
         end;
3  for each b ∈ dom(m ) do
      if justif(m, b )=k then
         begin
         Append(RL, (m, b )); justif(m, b ) , nil ;
         end;
4  Delete {k, m } from the set of constraints c and remove
     its counters and sets of supported values;

   { Step 2: values in RL are added to D and
     consequences are propagated }
5  while RL ≠ ∅ do
      begin
6     choose (i, a ) from RL and remove it from RL;
7     D(i, a ), true ;
8     for each j / {i, j } ∈ c do
         begin
9        for each b ∈ S_ija do
            begin
10          counter[(j, i ), b ] , counter[(j, i ), b ] + 1;
11          if justif(j, b )=i then
               begin
               Append(RL, (j, b )); justif(j, b ) , nil ;
               end;
            end;
12       if counter[(i, j ), a ]=0 then
            Append(SL, [(i, j ), a ] );
         end;
      end;
   end;
```

Figure 5

# Explanation of Figures 4 and 5

- **Figure 4 (Propag-Suppress):** This procedure removes values that have lost all valid support after a constraint update.
- It takes unsupported values from the suppression list (`SL`), deletes them from the domain, and records the reason in the justification table.
- **Figure 5 (Init-Propag-Relax):** This procedure is used when a constraint is removed and some previously deleted values may become valid again.
- It first restores candidate values into the relaxation list (`RL`), then propagates these restorations through related variables.
- Together, Figures 4 and 5 allow DnAC-4 to efficiently handle both deletion and restoration of values while maintaining arc consistency.

# Constraint Retraction in DnAC-4

When removing a constraint:

1. restore values deleted because of that constraint
2. propagate restored values
3. delete again the values that are still unsupported

This is the main advantage of DnAC-4 over AC-4.

# Worked Relaxation Example

The paper shows what happens when constraint $\{2, 3\}$ is removed.

**Step 1:** values deleted because of $\{2, 3\}$ are restored.

**Step 2:** some restored values are removed again if they are still unsupported on another constraint.

# Graph After Relaxation

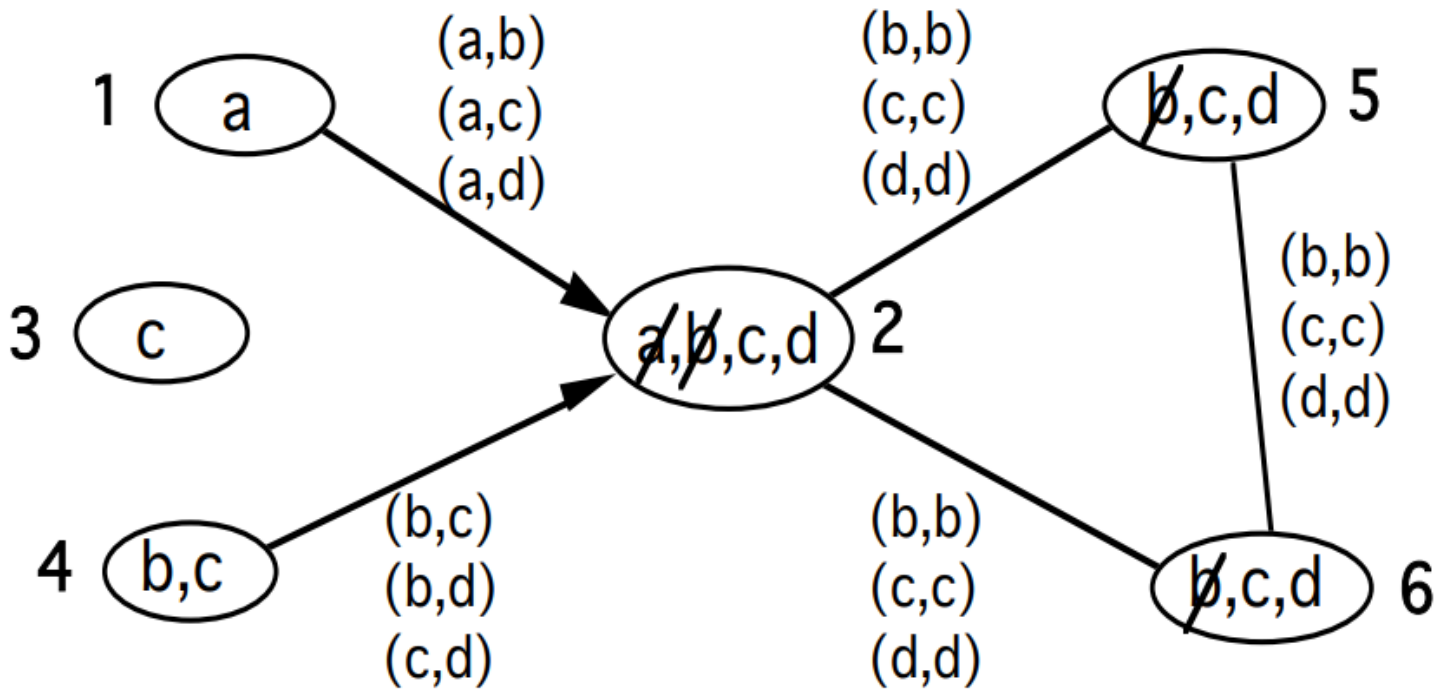


Figure 6: The CSP of fig.2 after the relaxation of the constraint {2, 3}

# Correctness and Complexity

The paper proves that after:

- **Add**
- **Relax**

the domain remains **maximal arc-consistent**.

**Time Complexity:**

$$O(ed^2)$$

**Space Complexity:**

$$O(ed^2)$$

# Results from the Paper

| | n=16 d=8 | | | n=12 d=12 | | | n=8 d=16 | | |
|---|---|---|---|---|---|---|---|---|---|
| | pc=35 pu=65 | pc=50 pu=50 | pc=65 pu=35 | pc=35 pu=65 | pc=50 pu=50 | pc=65 pu=35 | pc=35 pu=65 | pc=50 pu=50 | pc=65 pu=35 |
| restrictions AC-4 | 3611 | 3866 | 4498 | 4140 | 4683 | 4223 | 3128 | 3654 | 3298 |
| restrictions DAC-4 | 3622 | 4010 | 6968 | 4140 | 4698 | 4449 | 3128 | 3654 | 3337 |
| relaxation AC-4 | 3536 | 3818 | 4526 | 3946 | 4541 | 4136 | 2793 | 3401 | 3142 |
| relaxation DAC-4 | 4 | 28 | 266 | 0 | 11 | 33 | 0 | 0 | 12 |
| total AC-4 | 7147 | 7684 | 9025 | 8086 | 9224 | 8359 | 5921 | 7055 | 6440 |
| total DAC-4 | 3626 | 4038 | 7234 | 4140 | 4709 | 4482 | 3128 | 3654 | 3349 |
| DAC-4 gain | 49% | 47% | 20% | 48% | 48% | 46% | 47% | 48% | 47% |

Figure 7: Results of comparison tests between AC-4 and DnAC-4

# Conclusion

- CSPs are a useful AI framework.
- Arc consistency removes impossible values.
- AC-4 is strong for static CSPs.
- DnAC-4 may do slightly more work during restriction
- but it saves much more work during relaxation
- overall it performs better for Dynamic CSPs
- DnAC-4 improves AC-4 for dynamic updates by storing justifications.
- The key idea is to remember why values were deleted.

# References


- Bessière, C. (1991). *Arc-Consistency in Dynamic Constraint Satisfaction Problems*. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pp. 221–226.
- Mackworth, A. K. (1977). *Consistency in Networks of Relations*. *Artificial Intelligence*, 8(1), 99–118.
- Mohr, R., & Henderson, T. C. (1986). *Arc and Path Consistency Revisited*. *Artificial Intelligence*, 28(2), 225–233.

# Thank You