CSC384
Constraint Satisfaction Problems
Part 3

Bahar Aameri & Sonya Allin

Winter 2020

Constraint Propagation: Generalized Arc Consistency

A constraint $C(V_1, V_2, V_3, ..., V_n)$ is **GAC wrt** a variable V_i iff for **every domain value** of V_i, there exist domain values for $V_1, V_2, ..., V_{i-1}, V_{i+1}, ..., V_n$ that satisfy $C(V_1, V_2, V3, ..., V_n)$.

$C(V_1, V_2, V3, ..., V_n)$ is **GAC** iff it is GAC with respect to **all variables** in its scope.

A **CSP** is **GAC** if and only if **all of its constraints** are GAC.

Say we find a value d of variable V_i that is **not consistent** wrt a constraint: that is, there is **no assignments** to the other variables that satisfy the constraint when $V_i = d$:

- d is said to be **Arc Inconsistent**.
- We can **remove** d from the domain of V_i as this value cannot lead to a solution (much like Forward Checking, but more powerful).

Example: $C(X, Y) : X > Y$

$Dom[X] = \{\cancel{1}, 5, 11\}, Dom[Y] = \{3, 8, \cancel{15}\}$

$$\left.\begin{array}{l} X = 1 \text{ is arc inconsistent} \\ Y = 15 \text{ is arc inconsistent} \end{array}\right\} \Rightarrow \begin{array}{l} Dom[X] = \{5, 11\} \\ Dom[Y] = \{3, 8\} \end{array}$$

GAC-Based Propagation

Pruning the domain of a variable to make a constraint GAC can make a different constraint **no longer GAC**.

Example: $C_1(X,Y) : X > Y$, $C_2(Y,Z) : Y > Z$

$Dom[X] = \{\cancel{1}, \cancel{5}, 11\}, Dom[Y] = \{\cancel{3}, 8, \cancel{15}\}, Dom[Z] = \{4, 6\}$

- To make C_1 GAC we must prune 1 from Dom[X]
- To make C_2 GAC we must prune 3 from Dom[Y]
- Now C_1 is no longer GAC since $X=5$ is arc inconsistent

  ⇒ must remove 5 from Dom[X]

Need to **re-achieve GAC** for some constraints whenever a domain value is **pruned**.

GAC: Considerations

- All constraints must be GAC at every node of the search space.
  This is accomplished by **removing** from the domains of the variables all **arc inconsistent values**:
  - Every time we **assign** a value to a variable V, we check all constraints over V and prune **arc inconsistent** values from the **current domain** of the other variables of the constraints.
- Removing a value from a variable domain may trigger **further inconsistency**.
  We have to **repeat** the procedure until **everything is consistent**:
  - Have a queue of constraints that need to be made **GAC**.
  - Constraints are added (back) to the queue if the domain of one of their variables is changed.
  - The procedure **stops** when the queue is empty.
- After backtracking from the current assignment the values that were **pruned** (as a result of that assignment) must be **restored**.
  Some **bookkeeping** needs to be done to remember which values were pruned by which assignment.

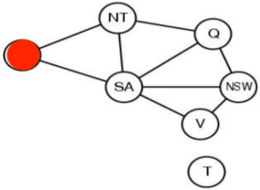
2

$C_1(SA, WA) : SA \neq WA,$ $C_2(NT, WA) : NT \neq WA,$ $C_3(SA, NT) : SA \neq NT$
$C_4(SA, Q) : SA \neq Q,$ $C_5(SA, NSW) : SA \neq NSW,$ $C_6(SA, V) : SA \neq V$
$C_7(NT, Q) : NT \neq Q,$ $C_8(Q, NSW) : Q \neq NSW,$ $C_9(NSW, V) : NSW \neq V$

Value Assignments: $WA = R$
Then, for SA and NT, R becomes arc inconsistent wrt C_1 and C_2.

Current Domains:

$Dom[SA] = \{\cancel{R}, G, B\}$ $Dom[NT] = \{\cancel{R}, G, B\}$
$Dom[Q] = \{R, G, B\}$ $Dom[NSW] = \{R, G, B\}$
$Dom[V] = \{R, G, B\}$ $Dom[T] = \{R, G, B\}$



GAC Queue

$\cancel{C_7}$ | $\cancel{C_6}$ | $\cancel{C_5}$ | $\cancel{C_4}$ | $\cancel{C_3}$ | $\cancel{C_2}$ | $\cancel{C_1}$

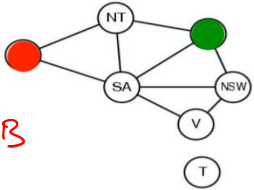
✓ ✓ ✓ ✓ ✓

$C_1(SA, WA) : SA \neq WA$, $C_2(NT, WA) : NT \neq WA$, $C_3(SA, NT) : SA \neq NT$
$C_4(SA, Q) : SA \neq Q$, $C_5(SA, NSW) : SA \neq NSW$, $C_6(SA, V) : SA \neq V$
$C_7(NT, Q) : NT \neq Q$, $C_8(Q, NSW) : Q \neq NSW$, $C_9(NSW, V) : NSW \neq V$

Value Assignments: $WA = R, Q = G$
Then, for SA, NT and NSW, G becomes arc inconsistent wrt C_4, C_7, and C_8.

Current Domains:

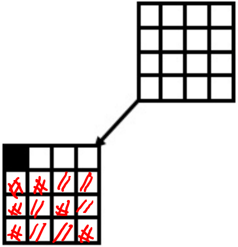
$Dom[SA] = \{\cancel{G}, \cancel{B}\}$ $Dom[NT] = \{\cancel{G}, B\}$
$Dom[Q] = \{R, G, B\}$ $Dom[NSW] = \{R, \cancel{G}, B\}$
$Dom[V] = \{R, G, B\}$ $Dom[T] = \{R, G, B\}$



SA = B

C_9 | C_8 | C_7 | C_6 | C_5 | C_4 | C_3 | $\cancel{C_2}$ ✓ | $\cancel{C_1}$ ✓

must remove B from Dom[SA] to make C_3 GAC $\Rightarrow$ Dom[SA] = {} DWO!



Value Assignments: $Q_1 = 1$
Then $Q_2 = 1, Q_2 = 2, Q_3 = 1, Q_3 = 3, Q_4 = 1, Q_4 = 4$
become arc inconsistent.

Current Domains:

$$\underbrace{Dom[Q_2] = \{\cancel{1}, \cancel{2}, 3, 4\} \qquad Dom[Q_3] = \{\cancel{1}, 2, \cancel{3}, 4\} \qquad Dom[Q_4] = \{\cancel{1}, 2, 3, \cancel{4}\}}_{\text{Put all constraints on the queue}}$$

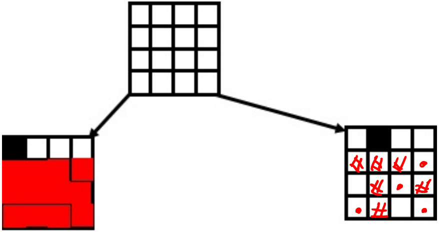
$Q_2 = 3$: no consistent Q_3 value $\Rightarrow Dom[Q_2] = \{\cancel{3}, 4\}$

$Q_3 = 4$: no consistent Q_2 value $\Rightarrow Dom[Q_3] = \{2, \cancel{4}\}$

$Q_3 = 2$: no consistent Q_4 value $\Rightarrow Dom[Q_3] = \{\cancel{2}\}$

DWO

Value Assignments: $Q_1 = 2$
Then $Q_2 = 1, Q_2 = 2, Q_2 = 3, Q_3 = 2,$
$Q_3 = 4, Q_4 = 2$ become arc inconsistent.



Current Domains:

$$\underbrace{Dom[Q_2] = \{\cancel{1}, \cancel{2}, \cancel{3}, 4\} \qquad Dom[Q_3] = \{1, \cancel{2}, 3, \cancel{4}\} \qquad Dom[Q_4] = \{1, \cancel{2}, 3, 4\}}$$

$\Rightarrow$ Put all constraints on the queue

$Q_3 = 3$: no consistent Q_2 Value $\Rightarrow Dom[Q_3] = \{1, \cancel{3}\}$

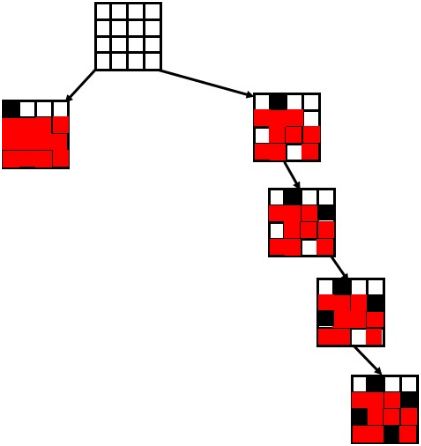
$Q_4 = 1$: no consistent Q_3 value $\Rightarrow Dom[Q_4] = \{\cancel{1}, 3, 4\}$

$Q_4 = 4$: no consistent Q_2 value $\Rightarrow Dom[Q_4] = \{3, \cancel{4}\}$

$\Rightarrow Dom[Q_2] = \{4\}$ $Dom[Q_3] = \{1\}$

$Dom[Q_4] = \{3\}$

Current Domains: $Dom[Q_2] = \{4\}, Dom[Q_3] = \{1\}, Dom[Q_4] = \{3\}$.
Now search no longer has to branch since only one value left for each variable.
It just walks down to a solution assigning each variable in turn.



GAC-Based Propagation

- **Plain Backtracking** check a constraint only when it has **zero** unassinged variables.
- **Forward checking** checks a constraint only when it has **one** unassinged variables.
- **GAC** checks **all constraints**, leading to much more pruning in general.
  - Even at the **root** before any variables have been assigned, we can get **some pruning** by making the constraints GAC consistent.
  - Checking for consistency can be done as a **pre-processing step**, or it can be **directly integrated** into a search algorithm.
  - If we apply arc consistency propagation **during** search the search tree's size will typically be **much reduced** in size.
  - **Note:** GAC enforce **does NOT** find a solution! (why?)
    To find **a solution** we must use do **search** while enforcing GAC.

Example: $Dom[X] = \{a_1, a_2\}$

$Dom[Y] = \{b_1, b_2\}$

$C_1(X,Y)$	X	Y
T	a_1	b_1
T	a_2	b_2
F	a_1	b_2
F	a_2	b_1

$C_2(X,Y)$	X	Y
T	a_1	b_2
T	a_2	b_1
F	a_1	b_1
F	a_2	b_2

C_1 and C_2 are GAC, but the CSP has no solution!

```
def GAC_Enforce()
// GAC-Queue contains all constraints one of whose variables has
// had its domain reduced. At the root of the search tree we can
// first run GAC_Enforce with all constraints on GAC-Queue
1.  while GACQueue not empty
2.     C = GACQueue.extract()
3.     for V := each member of scope(C)
4.        for d := CurDom[V]
5.           Find an assignment A for all other variables in scope(C)
             such that C(A ∪ V=d) is True
6.           if A not found
7.              CurDom[V] = CurDom[V] - d   # remove d from the domain of V
8.              if CurDom[V] == {}     # DWO for V
9.                  empty GACQueue
10.                 return DWO        # return immediately
11.           else
12.               push all constraints C' such that V ∈ scope(C')
                  and C' ∉ GACQueue on to GACQueue
13.  return TRUE   # loop exited without DWO
```

GAC: The Algorithm

```
def GAC(Level)
    if all Variables assigned
       PRINT Value of each Variable
       EXIT or RETURN                    # EXIT for only one solution
                                         # RETURN for more solutions
    V := PickUnassignedVariable()
    Assigned[V] := TRUE
    for d := each member of CurDom(V)
       Value[V] := d
       Prune all values other than d from CurDom[V]
       for each constraint C whose scope contains V
             Put C on GACQueue
       if(GAC_Enforce() != DWO)
             GAC(Level+1) # all constraints were ok
    RestoreAllValuesPrunedByFCCheck()
     Assigned[V] := FALSE     # UNDO as we have tried all of V's values
     RETURN
```

GAC: Outcomes

When all constraints are GAC three outcomes are possible:

1. Each domain has a single value.

2. At least one domain is empty.

3. Some domains have more than one value.
   Need to solve this new CSP (usually) **simpler** problem: same constraints, domains have been reduced

GAC: Complexity

- **BT worst-case running time:** $\mathcal{O}(d^N)$, where d is the max size of a variable domain, and N is the number of variables.

- **Worst-case complexity** of arc consistency procedure on a problem with N variables, c **binary constraints**, and d be the max size of a variable domain:

  - How often will we prune the domain of variable V? $O(d)$

  - How many constraints will be put on the queue when pruning domain of a variable V? $O(\text{degree of } V)$

  - Sum of degrees of all variables: $2 \times C$

  - Overall, how many constraints will be put on the queue? $O(cd)$

  - Checking consistency of each constraint: d^2

  - **Overall Complexity:** $O(cd^3)$

- For CSP with **higher-order constraints**:
  - Checking consistency of a constraint C with arity k (i.e., $|scope(C)| = k$): $O(d^k)$
  - It can be shown that the **Overall Complexity** is: $O(d^k \times T(c))$

number of forbiden values for C

More readings:

Bessiere, C., and Regin, J.C. 1997. Arc consistency for general constraint networks: preliminary results. In Proceedings of IJCAI97, 398-404.

GAC: Improving Efficiency

A **support** for a value assignment $V = d$ in a constraint C is an **assignment** A to all of the other variables in $scope(C)$ s.t. $A \cup \{V = d\}$ satisfies C.

A constraint C is **GAC** if for **every** variable V_i in its scope, **every** value $d_i \in CurDomain(V_i)$ has a **support** in C.

GAC: Improving Efficiency

- Smarter implementations keep track of **supports** to avoid having to search though all possible assignments to the other variables for a satisfying assignment.

- Rather than search for a satisfying assignment to C containing $V = d$, they check if the **current support** is **still valid**.

- Also they take advantage that a support for $V = d$, e.g. $\{V = d, X = a, Y = b, Z = c\}$ is also a support for $X = a, Y = b$, and $Z = c$.

- Another key development in practice is that for some constraints this computation can be done in polynomial time.
  Example: Ideas from graph matching theory are used to find support for variables in $All - diff(V_1, .., V_n)$ in **polynomial time**.

The special purpose algorithms for achieving GAC on particular types of constraints are very important in practice.