

Kantenkonsistenz und AC-3

Carsten Gips (HSBI)

Unless otherwise noted, this work is licensed under CC BY-SA 4.0.

Problem bei BT-Suche

Zuweisung eines Wertes an Variable X :

- Passt zu aktueller Belegung
- Berücksichtigt aber nicht **restliche** Constraints
=> macht weitere Suche u.U. unmöglich/schwerer

Lösung: Nach Zuweisung alle *nicht zugewiesenen Nachbarvariablen* prüfen

INFERENCE: Vorab-Prüfung (Forward Checking)

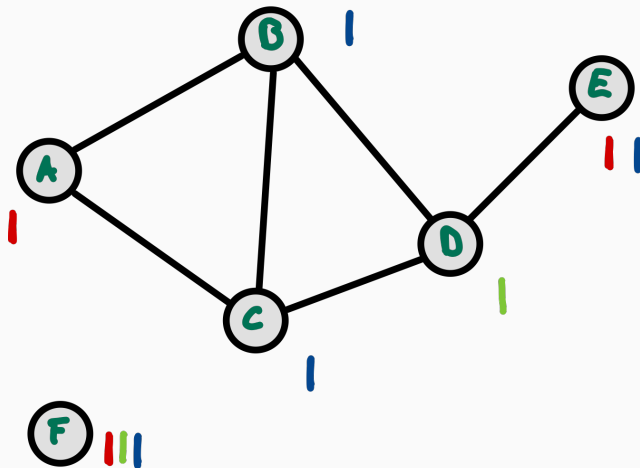
```
def BT_Search(assignment, csp):  
    if complete(assignment): return assignment  
  
    var = VARIABLES(csp, assignment)  
  
    for value in VALUES(csp, var):  
        if consistent(value, var, assignment, csp):  
            assignment += {var = value}  
  
            if INFERENCE(csp, assignment, var) != failure:  
                result = BT_Search(assignment, csp)  
                if result != failure: return result  
  
            assignment -= {var = value}  
  
    return failure
```

Quelle: Eigener Code basierend auf einer Idee nach (Russell und Norvig 2020, S. 176, Fig. 5.5)

Nach Zuweisung eines Wertes an Variable X :

- Betrachte alle nicht zugewiesenen Variablen Y :
 - Falls Constraints zw. X und Y , dann ...
 - ... entferne alle inkonsistenten Werte aus dem Wertebereich von Y .

Forward Checking findet nicht alle Inkonsistenzen!



- Nach $\{A = \text{red}, D = \text{green}\}$ bleibt für B und C nur noch blue
- B und C sind aber benachbart

Übergang von Forward Checking zu Kantenkonsistenz

- Forward Checking erzeugt Konsistenz für alle Constraints der **gerade betrachteten (belegten) Variablen**.
- Idee: Ausdehnen auf alle Kanten ... => Einschränken der Wertemengen

Definition Kantenkonsistenz (Arc Consistency)

*Eine Kante von X nach Y ist "**konsistent**", wenn für jeden Wert $x \in D_X$ und für alle Constraints zwischen X und Y jeweils ein Wert $y \in D_Y$ existiert, so dass der betrachtete Constraint durch (x, y) erfüllt ist.*

Ein CSP ist kanten-konsistent, wenn für alle Kanten des CSP Konsistenz herrscht.

Beispiel Kantenkonsistenz

$$V = \{a, b, c, d, e\}$$

$$C = \{((a, b), \neq), ((b, c), \neq), ((a, c), \neq), ((c, d), =), ((b, e), <)\}$$

$$D_a = D_b = D_c = \{1, 2, 3\}, D_d = \{1, 2\}, D_e = \{1, 2, 3\}$$

Tafelbeispiel Kantenkonsistenz

Beispiel Kantenkonsistenz

$$V = \{a, b, c, d, e\}$$

$$C = \{((a, b), \neq), ((b, c), \neq), ((a, c), \neq), ((c, d), =), ((b, e), <)\}$$

$$D_a = D_b = D_c = \{1, 2, 3\}, D_d = \{1, 2\}, D_e = \{1, 2, 3\}$$

Tafelbeispiel Kantenkonsistenz

Einschränkung der Ausgangswertemengen (kanten-konsistent)

$$D_a = \{1, 2, 3\}, D_b = \{1, 2\}, D_c = \{1, 2\}, D_d = \{1, 2\}, D_e = \{2, 3\}$$

=> Kantenkonsistenz ist nur **lokale** Konsistenz!

Beispiel Kantenkonsistenz

$$V = \{a, b, c, d, e\}$$

$$C = \{((a, b), \neq), ((b, c), \neq), ((a, c), \neq), ((c, d), =), ((b, e), <)\}$$

$$D_a = D_b = D_c = \{1, 2, 3\}, D_d = \{1, 2\}, D_e = \{1, 2, 3\}$$

Tafelbeispiel Kantenkonsistenz

Einschränkung der Ausgangswertemengen (kanten-konsistent)

$$D_a = \{1, 2, 3\}, D_b = \{1, 2\}, D_c = \{1, 2\}, D_d = \{1, 2\}, D_e = \{2, 3\}$$

=> Kantenkonsistenz ist nur **lokale** Konsistenz!

Anmerkung: $((a, b), \neq)$ ist Kurzform für $((a, b), \{(x, y) \in D_a \times D_b \mid x \neq y\})$

AC-3 Algorithmus: Herstellen von Kantenkonsistenz

```
def AC3(csp):  
    queue = Queue(csp.arcs)  
    while not queue.isEmpty():  
        (x,y) = queue.dequeue()  
        if ARC_Reduce(csp,x,y):  
            if D_x.isEmpty(): return false  
            for z in x.neighbors(): queue.enqueue(z,x)  
    return true  
  
def ARC_Reduce(csp, x, y):  
    change = false  
    for v in D_x:  
        if not (any w in D_y and csp.C_xy(v,w)):  
            D_x.remove(v); change = true  
    return change
```

1. Vorverarbeitung: Reduktion der Wertemengen *vor* BT-Suche
 - Nach AC-3 evtl. bereits Lösung gefunden (oder ausgeschlossen)
2. Propagation: Einbetten von AC-3 als Inferenzschritt in BT-Suche
(**MAC** – Maintaining Arc Consistency)
 - Nach jeder Zuweisung an X_i Aufruf von AC-3-Variante:
 - Initial nur Kanten von X_i zu allen noch nicht zugewiesenen Nachbarvariablen
 - Anschließend rekursiver Aufruf von BT-Suche

- Anwendung von Forward Checking und ...
- ... die Erweiterung auf alle Kanten: AC-3, Kantenkonsistenz



Unless otherwise noted, this work is licensed under CC BY-SA 4.0.