Kapitel MK:IV

IV. Modellieren mit Constraints

- Einführung und frühe Systeme
- Konsistenz I
- Binarization
- Generate-and-Test
- Backtracking-basierte Verfahren
- Konsistenz II
- Konsistenzanalyse
- Weitere Analyseverfahren
- FD-CSP-Anwendungen
- Algebraische Constraints
- Intervall Constraints
- Optimierung und Überbestimmtheit

Binarization

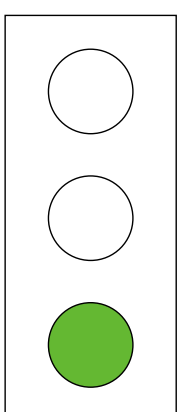
Definition 6 (CSP-FD [vgl. Constraint-Definition]**)**

Sei $X = \{x_1, x_2, \ldots, x_n\}$ eine Menge von n Variablen mit den Wertebereichen $D_1, D_2, \ldots, D_n$, und sei $\mathcal{C}$ ein über X definiertes Constraint-Netz.

Sind die Wertebereiche (Domains) der Variablen X endlich, dann bezeichnet man ein so definiertes Constraint-Problem als CSP-FD: „Constraint Satisfaction Problem with Finite Domains“

Beispiel:

- $X = \{x_1, x_2, x_3\}$
- $D_1 = D_2 = D_3 = \{\texttt{green, off, red, yellow}\}$



$C(x_1, x_2, x_3) = \{$ `(red, off, off),`
`(red, yellow, off),`
`(off, yellow, off),`
`(off, off, green)` $\}$

Binarization

Definition 7 (Binarization)

Unter Binarization versteht man die Umwandlung eines Constraint-Netzes $\mathcal{C}$ mit m-stelligen Constraints, $m > 2$, in ein Constraint-Netz $\hat{\mathcal{C}}$, das nur unäre und binäre Constraints enthält.

Zwei mögliche Ansätze zur Binarization:

1. Hidden-Variable-Encoding.
   Basis: Kanten-Constraint-Graph
   Einführung von zusätzlichen, neuen Variable x_C für jeden m-stelligen Constraint C mit $m > 2$.

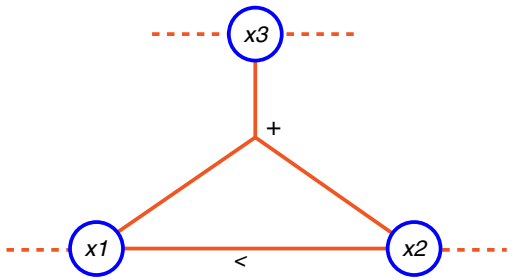
2. Dual-Encoding.
   Basis: Knoten-Constraint-Graph
   Einführung neuer Variablen x_C für jeden Constraint $C \in \mathcal{C}$; die originalen Variablen werden verworfen.

Bemerkungen:

- Weitere Umformungsschritte bei beiden Ansätzen:
  1. Konstruktion der Elemente des Wertebereiches D_{x_C} von x_C als Teilmenge des Kartesischen Produktes der Wertebereiche der Variablen von C so, dass gilt: $t \in D_{x_C} \Leftrightarrow t \in C$
  2. Einführung zusätzlicher Projektions-Constraints.
- Binarization ist von theoretischem Interesse: Bei Analysen braucht man sich nur auf unäre und binäre Constraints zu beschränken. In der Constraint-Lösungs-Praxis ist Binarization eher unüblich.

Binarization

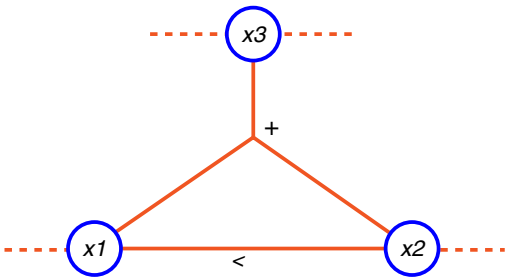
Beispiel (als Kanten-Constraint-Graph):



- $X = \{x_1, x_2, x_3\}$
- $D_1 = \{1, 2\},\ D_2 = \{3, 4\},\ D_3 = \{5, 6\}$
- $C_1(x_1, x_2, x_3) : x_1 + x_2 = x_3$,
  $C_2(x_1, x_2) : x_1 < x_2$

Binarization

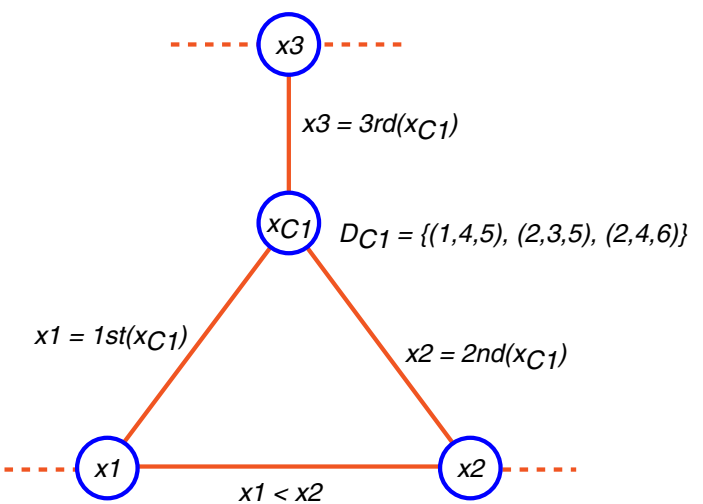
Beispiel (als Kanten-Constraint-Graph):



- $X = \{x_1, x_2, x_3\}$
- $D_1 = \{1, 2\},\ D_2 = \{3, 4\},\ D_3 = \{5, 6\}$
- $C_1(x_1, x_2, x_3) : x_1 + x_2 = x_3$,
  $C_2(x_1, x_2) : x_1 < x_2$

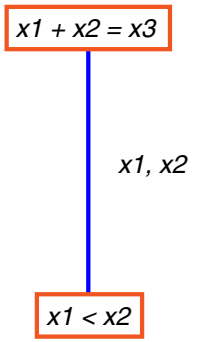
Binarization mit Hidden-Variable-Encoding:

- neue Variable x_{C_1} mit $D_{C_1} = \{(1, 4, 5), (2, 3, 5), (2, 4, 6)\}$
- neue Projektions-Constraints $C_{p_i},\ i = 1, \ldots, 3 :\ x_i\ =\ x_{C_1}\ |_i$



Binarization

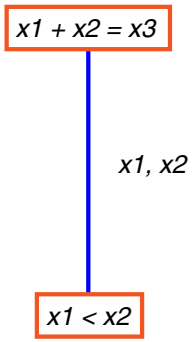
Beispiel (als Knoten-Constraint-Graph):



- $X = \{x_1, x_2, x_3\}$
- $D_1 = \{1, 2\},\ D_2 = \{3, 4\},\ D_3 = \{5, 6\}$
- $C_1(x_1, x_2, x_3) : x_1 + x_2 = x_3,\ C_2(x_1, x_2) : x_1 < x_2$

Binarization

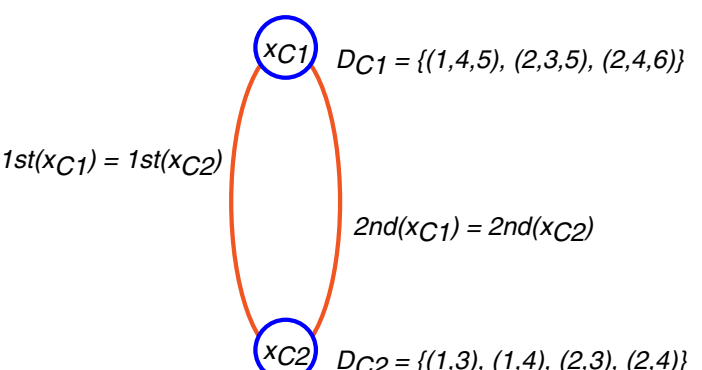
Beispiel (als Knoten-Constraint-Graph):



- $X = \{x_1, x_2, x_3\}$
- $D_1 = \{1, 2\},\ D_2 = \{3, 4\},\ D_3 = \{5, 6\}$
- $C_1(x_1, x_2, x_3) : x_1 + x_2 = x_3,\ C_2(x_1, x_2) : x_1 < x_2$

Binarization mit Dual-Encoding:

- neue Variablen $x_{C_1},\ x_{C_2}$ mit
  $D_{C_1} = \{(1, 4, 5), (2, 3, 5), (2, 4, 6)\}$ und $D_{C_2} = \{(1, 3), (1, 4), (2, 3)(2, 4)\}$
- neue Constraints $C_{p_i},\ i = 1, 2 :\quad x_{C_1}\,|_i = \ x_{C_2}\,|_i$



Generate-and-Test (GT)

Lösungsverfahren für ein CSP auf endlichen Wertebereichen

1. Generate-and-Test.
   Basis: systematische Suche
2. Propose-and-Improve.
   Basis: Heuristiken
3. Backtracking.
   Basis: systematische Suche
4. Backtracking mit Konfliktanalyse.
   Basis: systematische Suche + Konfliktverwaltung
5. Konsistenzanalyse.
   Basis: Grundbereichseinschränkung
6. Kombination von Konsistenzanalyse und Backtracking.
7. Variablensortierung.
   Basis: Heuristiken
8. Constraint-Netz-Reformulierung.
   Basis: Graphanalyse

Bemerkungen:

- Die Konsistenzanalyse ist auch mit anderen Verfahren kombinierbar.

Generate-and-Test

Schema eines GT-Algorithmus:

1. Generierung einer Belegung für alle Variablen.
2. Test aller Constraints mit dieser Belegung.
3. Falls Constraints nicht erfüllt sind, weiter bei (1).

Fragen:

- Was kann man über die Laufzeit sagen?
- Was kann man hinsichtlich der Systematik sagen?

Zusammenhang von GT zu anderen Lösungsprinzipien:

- Informieren des Generators, um die Wahrscheinlichkeit eines Konflikts klein zu halten
  ➜ Propose-and-Improve
- Kopplung von Generator und Tester, um Konflikte früh zu entdecken
  ➜ Backtracking

Bemerkungen:

- Weiterhin können die genannten Lösungsverfahren hinsichtlich einer *konstruktiven* und einer *destruktiven* Vorgehensweise beurteilt werden. Konstruktive Verfahren versuchen, die Lösungsmenge $\hat{\mathcal{D}}$ eines CSP direkt aufzubauen; destruktive Verfahren versuchen, möglichst viel über die Menge $\mathcal{D} \setminus \hat{\mathcal{D}}$ herauszufinden.
- Beispiel EL: Das Propagieren von Werten in einem elektronischen Schaltkreis entpricht einer konstruktiven Sicht.
- Beispiel Waltz: Das Streichen von Tupeln bei Ecken-Constraints, die mit einer Kante verbunden sind, entspricht einer destruktiven Sicht.

Generate-and-Test

Propose-and-Improve

Propose-and-Improve-Verfahren sind Spezialisierungen des Generate-and-Test-Prinzips:

- Aus einem gescheiterten Test wird Information für eine neue Variablenbelegung gewonnen.
- Diese Informationsgewinnung ist (notwendigerweise) heuristischer Natur. Warum?

Bekannte Heuristiken:

- Min-Conflicts
- Random-Walk
- Tabu-Search
- Connectionist-Approach

Generate-and-Test

Propose-and-Improve

Die Min-Conflicts-Heuristik:

1. Wähle eine Variable x_C eines nicht-erfüllten Constraints C.
2. Setze x_C auf einen Wert $d \in D_{x_C}$, so dass die Zahl der nicht-erfüllten Constraints kleiner wird.
   Bei einer Auswahlsituation (tie), treffe zufällige Entscheidung.
3. Falls Constraints unerfüllt, weiter bei (1).

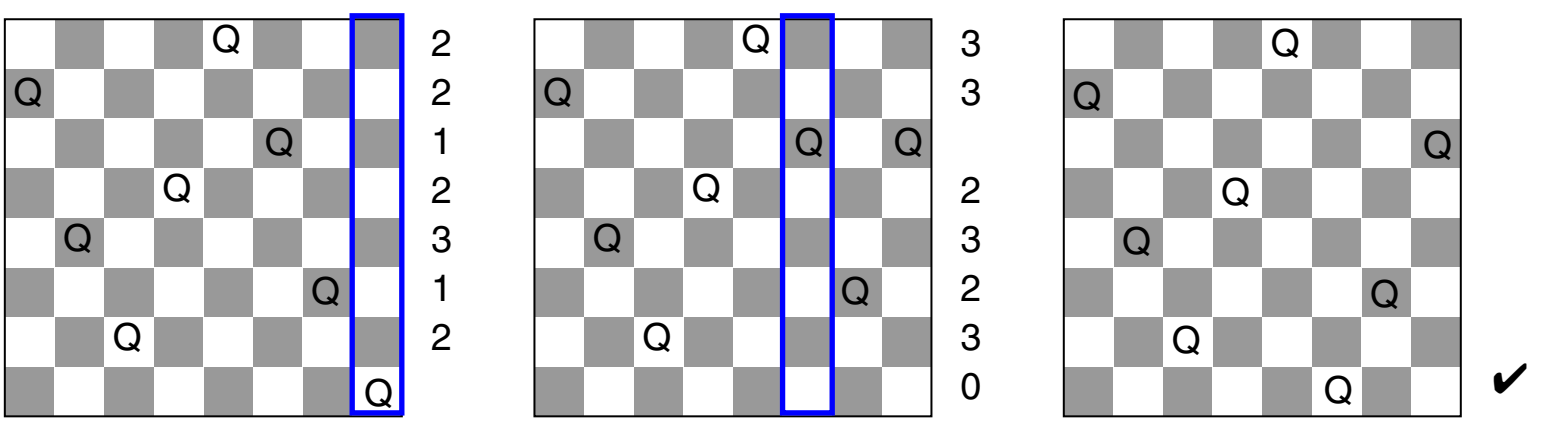
Frage: Ist die Min-Conflicts-Heuristik vollständig?

Generate-and-Test

Propose-and-Improve

Beispiel für die Min-Conflicts-Heuristik [Russell/Norvig 1995]:

- eine Variable pro Spalte mit Wertebereich $\{1, \ldots, 8\}$
- Variable codiert die Position (Zeile) einer Dame
- Die Zahlen geben die Anzahl der Konflikte an, falls die Dame in der entsprechenden Zeile positioniert würde.



Backtracking (BT) -basierte Verfahren

Lösungsverfahren für ein CSP auf endlichen Wertebereichen

1. Generate-and-Test.
   Basis: systematische Suche
2. Propose-and-Improve.
   Basis: Heuristiken
3. Backtracking.
   Basis: systematische Suche
4. Backtracking mit Konfliktanalyse.
   Basis: systematische Suche + Konfliktverwaltung
5. Konsistenzanalyse.
   Basis: Grundbereichseinschränkung
6. Kombination von Konsistenzanalyse und Backtracking.
7. Variablensortierung.
   Basis: Heuristiken
8. Constraint-Netz-Reformulierung.
   Basis: Graphanalyse

Backtracking-basierte Verfahren

Schema eines BT-Algorithmus:

- Eine partielle Lösung wird sukzessive ausgebaut.
- Liegt ein unerfüllter Constraint vor, wird für die letzte Variable (allgemein: Choice-Point) ein neuer Wert gewählt.

Vergleich von Generate-and-Test und Backtracking:

	Generate-and-Test	Backtracking
Suche	generatorgesteuert	operatorgesteuert
Basis	Menge vollständiger Lösungsobjekte	Menge von Teillösungsobjekten
Suchraum	unstrukturiert	Operatoren definieren Struktur
Systematik	✓	✓

Backtracking-basierte Verfahren

Algorithm: BT

Input: $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$. Domains of the n constraint variables.
$\mathcal{C}$. Constraint net.
$i = 1$. Index of the constraint variable under investigation.

Output: Array with constraint variable assignments.

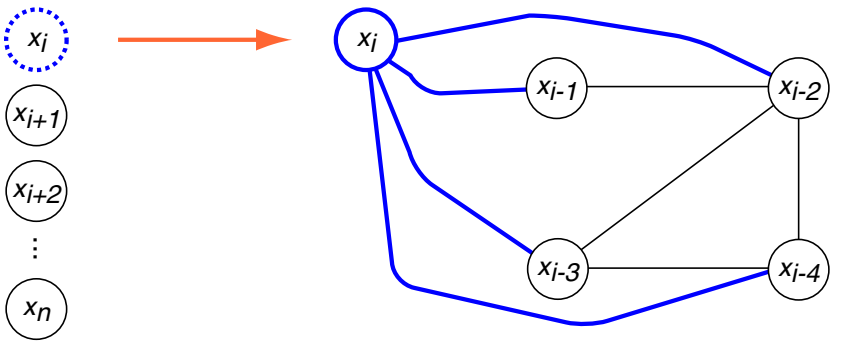
```
BT(D,C,i)
 1. FOREACH d in D_i DO
      assign[i] := d;
      consistent := TRUE;
      FOR h := 1 TO i-1 DO
        consistent := ((C(x_h,x_i) ∉ C) ∨ (assign[h],assign[i]) ∈ C(x_h,x_i));
        IF (consistent = FALSE) THEN BREAK;
      ENDDO
      IF (consistent = TRUE)
      THEN
        IF (i = n)
        THEN PRINT(assign[])
        ELSE BT(D,C,i+1)
      ENDIF
    ENDDO
```

Bemerkungen:

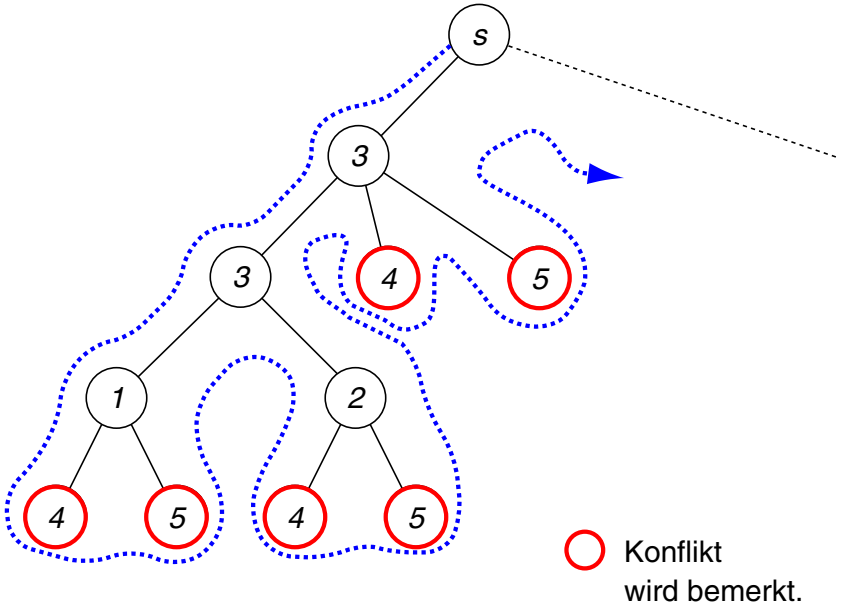
- Der Algorithmus BT ist spezialisiert für die folgende Situation:
  1. n Variablen mit geordneten Wertebereichen $D_i, \ldots, D_n$.
  2. Die Constraints in $\mathcal{C}$ sind binär.
  3. *assign*$[i]$ enthält die aktuelle Belegung für Variable i.

Backtracking-basierte Verfahren

Durch die sukzessive Hinzunahme von Variablen wird das Constraint-Netz bei der Suche mit BT aufgebaut:



Suchraumsicht:



Startknoten

x1, D1 = {3, 5}

x2, D2 = {3, 4, 5}

x3, D3 = {1, 2}

x4, D4 = {4, 5}

Backtracking-basierte Verfahren

Probleme bei Backtracking

1. Thrashing.
   Generierung von Variablenbelegungen und Berechnung von Constraints, die nichts mit einem existierenden Widerspruch zu tun haben.

2. Redundancy.
   Entdeckung der Widersprüchlichkeit von widersprüchlichen Variablenbelegungen für Constraints immer wieder auf's Neue.

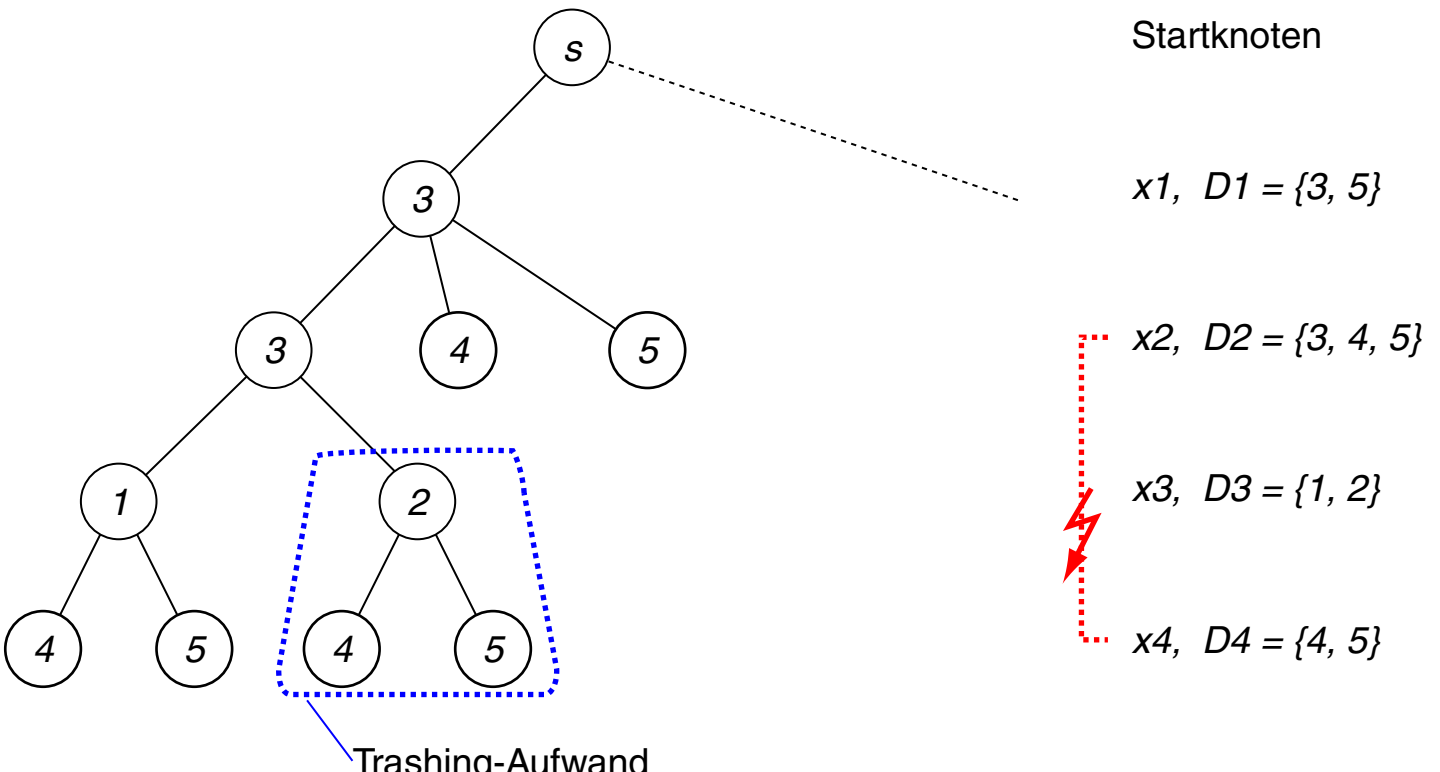
3. Late-Detection.
   Entdeckung der Widersprüchlichkeit erst bei der Erzeugung einer konkreten Wertebelegung.
   Early-Detection bedeutet dagegen, dass widersprüchliche Wertebelegung in einem Preprocessing erkannt werden.

Backtracking-basierte Verfahren

Probleme bei Backtracking

Beispiel für Trashing:

Sei $C(x_2, x_4)$ gegeben als $x_2 = x_4$.



Bemerkungen:

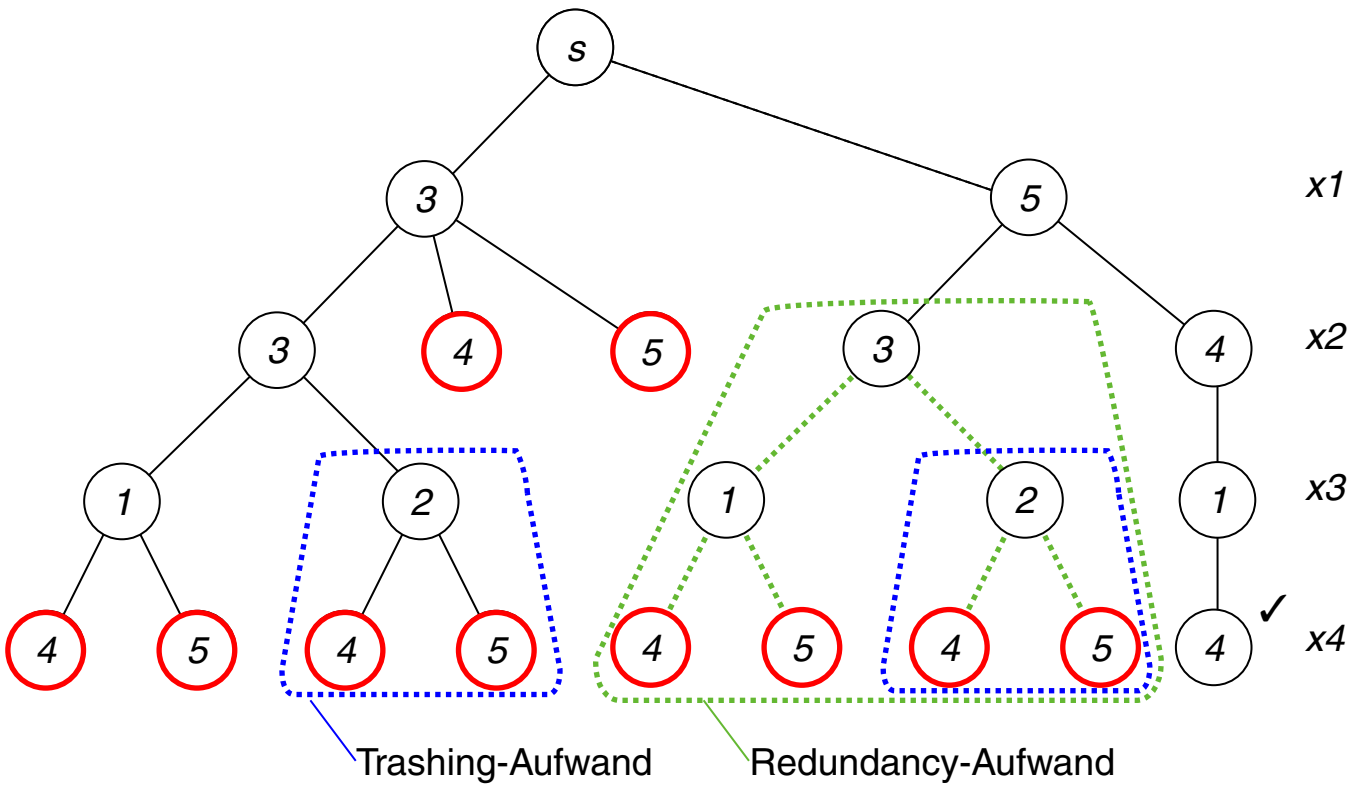
- Ursache für Thrashing: Existenz von Constraint(s) $C(x_i, x_j)$, so dass die Belegung(en) für x_i im Konflikt mit allen Elementen aus D_j steht.
- Der Trashing-Aufwand e berechnet sich aus der Größe der involvierten Grundbereiche und wächst exponentiell in $j - i$. $e = O(|D_{i+1} \times D_{i+2} \times \ldots D_j|)$.
- Verminderung des Thrashing-Aufwands von Backtracking: Konfliktanalyse, um zu einem Verursacher des Widerspruchs zurückzuspringen.

Backtracking-basierte Verfahren

Probleme bei Backtracking

Beispiel für Redundancy:

Sei $C(x_1, x_2)$ gegeben als $x_1 > x_2$.



Bemerkungen:

- Ursache für Redundancy: Nach einem Backtracking stellt man erneut fest, dass bestimmte Variablenbelegungen nicht zusammen passen.
- Der Redundancy-Aufwand hängt von zwei Dingen ab:
  (a) Wie aufwendig die zugehörigen Constraint-Auswertungen sind.
  (b) Wie kompakt sich widersprüchliche Variablenbelegungen (*Nogoods*) repräsentieren lassen; im Beispiel: $\perp (x_2, x_3, x_4) = \{(3, *, *)\}$
- Verminderung des Redundancy-Aufwands:
  zu (a) Caching von Berechnungen – auch bekannt als *Backmarking*.
  zu (b) Speichern von Nogoods – auch bekannt als *Conflict-Recording*.

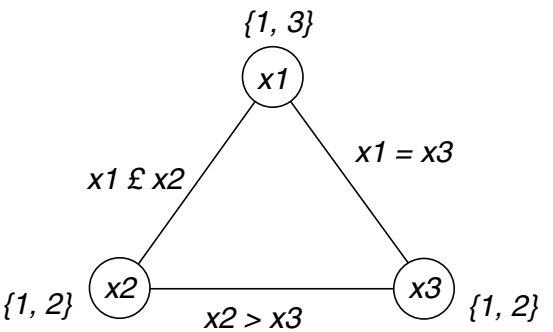
Backtracking-basierte Verfahren

Backtracking mit Konfliktanalyse: Backjumping (BJ)

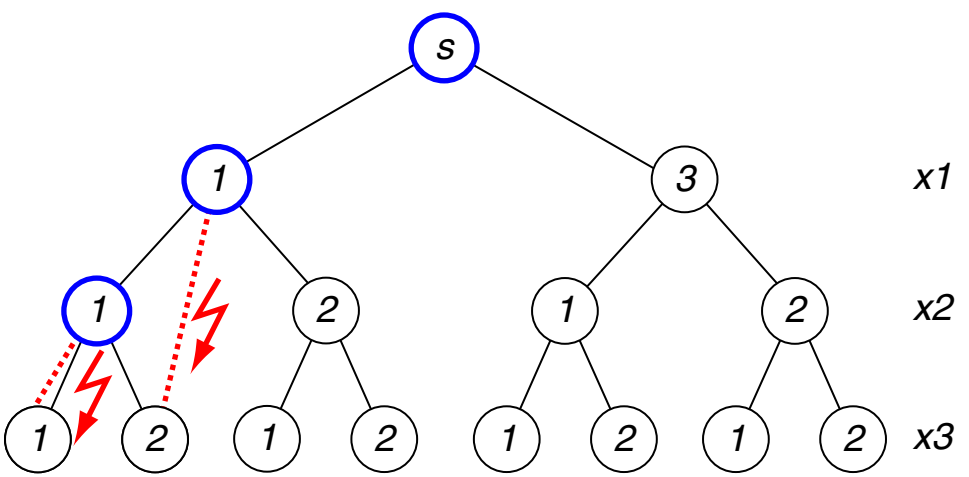
Idee: Rücksprung zu einem Verursacher des Konfliktes.

Problematik 1: Wie weit zurückspringen?

Beispiel:



aktuelle Variable	Konflikt
$x_3 = 1$	$x_2 = 1$
$x_3 = 2$	$x_1 = 1$



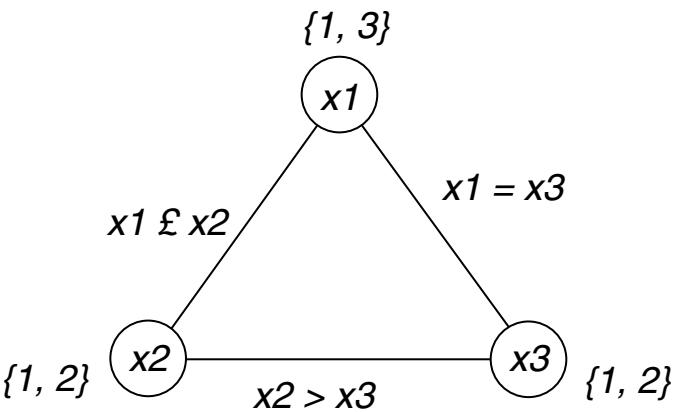
Bemerkungen:

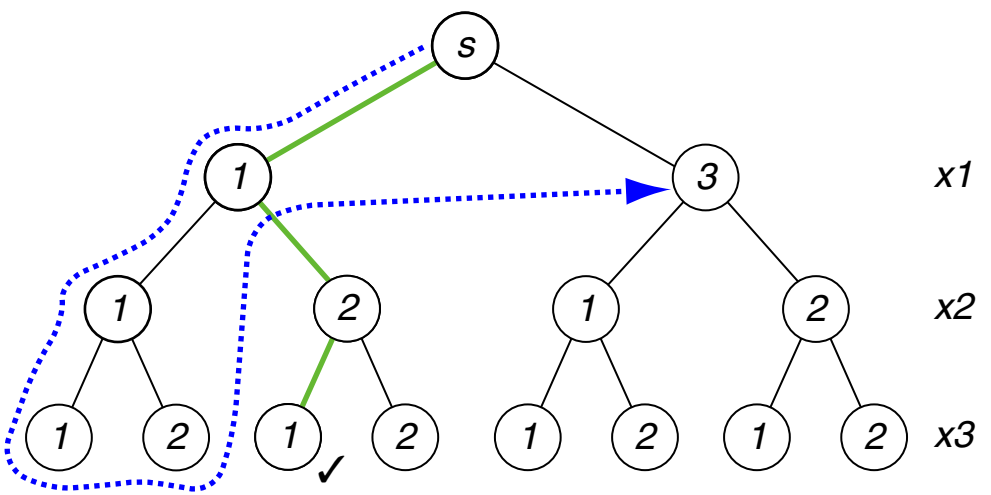
- Sind für eine Variable x alle Elemente ihres Grundbereichs D in Konflikt mit den Belegungen anderer Variablen $X' \subset X$, dann muss die Belegung einer Variablen in X' geändert werden, um eine konfliktfreie Belegung für x zu finden. Im folgenden bezeichnen wir die Elemente in X' auch als *Konfliktvariablen*.

Backtracking-basierte Verfahren

Backtracking mit Konfliktanalyse: Backjumping (BJ)

Beispiel (Fortsetzung):





Bemerkungen:

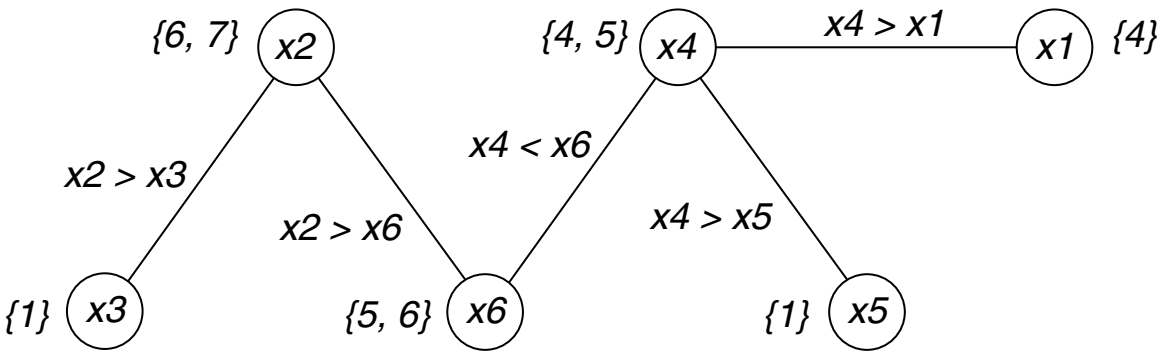
- Wegen des Rücksprungs nach x_1 wird die Lösung nicht gefunden.
- Sind für eine Variable x alle Elemente ihres Grundbereichs D in Konflikt mit den Belegungen anderer Variablen, dann bleibt BT vollständig, wenn ein Rücksprung zu derjenigen Konfliktvariable erfolgt, die am wenigsten weit zurückliegt, d. h., die am tiefsten im Suchraum ist.
- Im folgenden Algorithmus BJ wird sich diese Suchraumtiefe mittels *returnDepth* gemerkt.

Backtracking-basierte Verfahren

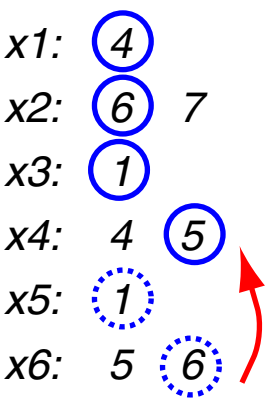
Backtracking mit Konfliktanalyse: Backjumping (BJ)

Problematik 2: Mehrfaches Zurückspringen.

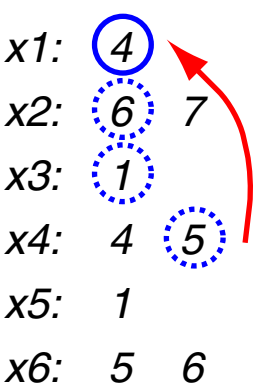
Beispiel:



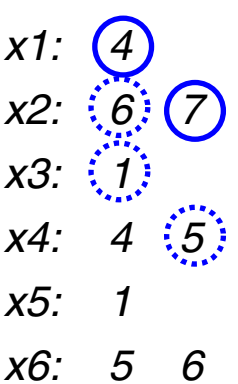
1. Rücksprung



2. Rücksprung im 1. Rücksprung



Backtracking nach 1. Rücksprung



aktuelle / vorige Belegung

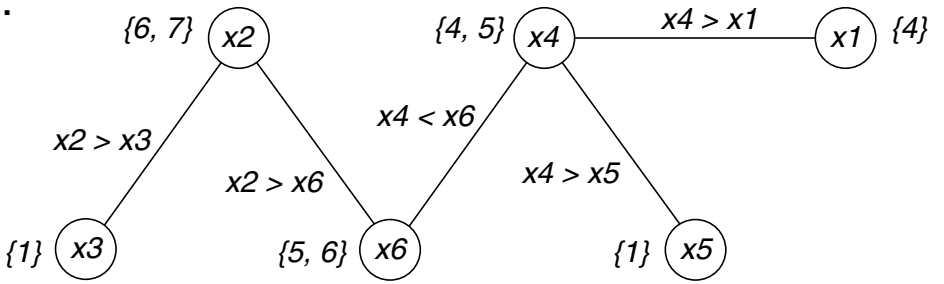
Bemerkungen:

- Findet man bei einem Rücksprung zur Variable x keinen Wert in ihrem Grundbereich D, der die Constraints erfüllt, kann weiter zurückgesprungen werden. Beachte: *Hierfür reicht eine Analyse der Konflikte von x nicht aus.* (siehe Beispiel)
- $x_6 = 5$ und $x_6 = 6$ ist in Konflikt mit x_4 bzw. x_2; deshalb der erste Rücksprung nach x_4 (links). Desweiteren ist $x_4 = 4$ in Konflikt mit x_1; deshalb der zweite Rücksprung nach x_1 (mitte). Hier ist die Suche zu Ende; eine existierende Lösung wird nicht gefunden. Im Bild rechts wird Backtracking anstatt eines zweiten Rücksprungs gemacht.

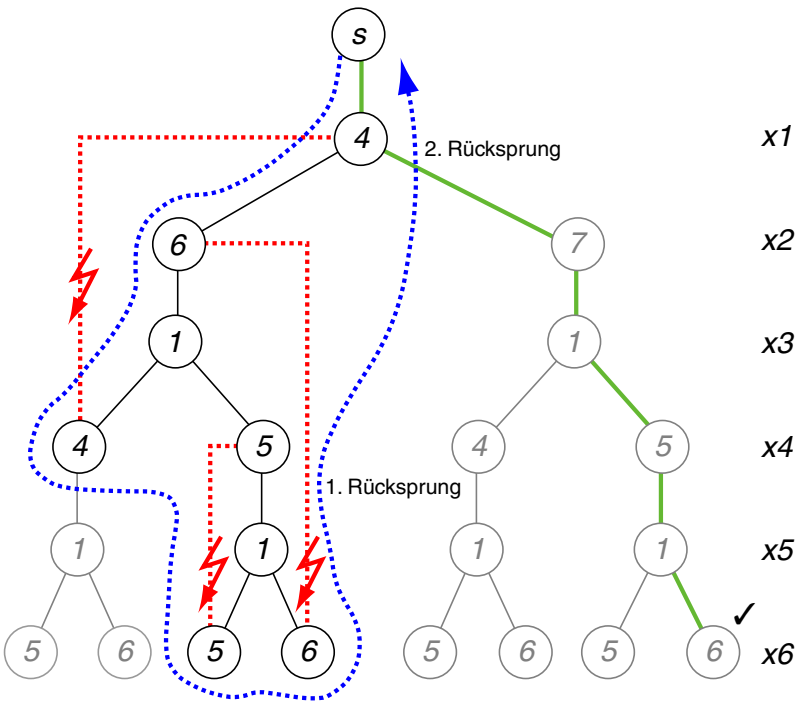
Backtracking-basierte Verfahren

Backtracking mit Konfliktanalyse: Backjumping (BJ)

Beispiel (Fortsetzung):



aktuelle Variable	Konflikt
$x_6 = 5$	$x_4 = 5$
$x_6 = 6$	$x_2 = 6$
$x_4 = 4$	$x_1 = 1$



Bemerkungen:

- Der 2. Rücksprung von x_4 zu x_1 verhindert, dass die Lösung (grün) gefunden wird, denn für x_1 existiert keine weitere Alternative. Beachte, dass in diesem Beispiel die Rücksprungbedingung verletzt ist: Nicht alle Elemente aus D_4 (die 5 nicht) stehen in Konflikt mit einer der schon belegten Variablen x_1, x_2, x_3.

Backtracking-basierte Verfahren

Backtracking mit Konfliktanalyse: Backjumping (BJ)

Diskussion des mehrfachen Zurückspringens im Beispiel:

- Der 1. Rücksprung von x_6 zu x_4 erfolgt gemäß der Rücksprungbedingung. Er verletzt nicht die Vollständigkeit, weil von den Konfliktvariablen $\{x_2, x_4\}$ die Variable x_4 am wenigsten weit zurückliegt. Der Grundbereich von x_4 ist jedoch erschöpft. Nun hat man zwei Möglichkeiten weiterzumachen: Ein weiterer Rücksprung oder Backtracking. Ein weiterer Rücksprung von x_4 ist möglich, bedarf aber einer genaueren Analyse.
- Von der Variablen x_4 stehen nicht alle Elemente des Grundbereichs D_4 (die 5 nicht) in Konflikt mit einer der schon belegten Variablen x_1, x_2 oder x_3. Genauer: Auf Basis der Constraints $C(x_i, x_4), i = 1, \ldots, 3$, steht x_4 nur für den Wert $4 \in D_4$ in einem Konflikt mit x_1; über das Element $5 \in D_4$ kann keine Aussage gemacht werden. (Mit $x_4 = 5$ landet man – wie gesehen – später durch $C(x_2, x_6)$ in einer Sackgasse; $x_4 = 5$ steht also „indirekt“ mit x_2 im Konflikt.) Die Konfliktinformation für D_4 ist somit unvollständig, und ein Rücksprung nach x_1 auf Basis dieser unvollständigen Information kann im allgemeinen Fall nicht zu einem vollständigen Algorithmus führen.

Backtracking-basierte Verfahren

Backtracking mit Konfliktanalyse: Backjumping (BJ)

Diskussion des mehrfachen Zurückspringens im Beispiel (Fortsetzung):

- Erkenntnis: Ist der Grundbereich D_i einer Variablen x_i erschöpft, so sind zwei Situationen zu unterscheiden: (a) alle Elemente in D_i stehen im (direkten) Konflikt mit den Belegungen anderer Variablen; Ursache sind Constraints $C(x_h, x_i), h < i$. (b) nicht alle Elemente stehen im (direkten) Konflikt mit den Belegungen anderer Variablen.
  Situation (b) kann nach einem Rücksprung zu Variable x_i eintreten. Will man erneut zurückzuspringen, muss eine Konfliktanalyse auch die indirekten Abhängigkeiten berücksichtigen, um die am wenigsten weit zurückliegende Konfliktvariable zu identifizieren. Im vorherigen Beispiel ist das die Variable x_2; neben x_4 steht auch sie im Konflikt mit x_6.
- Allgemein: Ist nach dem Rücksprung von einer Variablen x_k zu einer Variablen x_i der entsprechende Grundbereich D_i erschöpft und möchte man weiter zurückspringen, so sind alle Konfliktvariablen x_h der Constraints $C(x_h, x_k), C(x_h, x_i), h < i$, zu berücksichtigen. Hieraus ist die am wenigsten weit zurückliegende Variable zu wählen. Der Algorithmus CBJ verfährt so.
- Einfache Lösung: Verzichtet man auf einen Rücksprung im Rücksprung und macht nur mit Backtracking weiter, so kann man sich die Konfliktanalyse der indirekten Abhängigkeiten sparen. Der Algorithmus BJ verfährt so. Nachteil: Beim Backtracking kann es zu Trashing kommen.

Backtracking-basierte Verfahren

Algorithm: BJ

Input: $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$. Domains of the n constraint variables.
$\mathcal{C}$. Constraint net.
$i = 1$. Index of the constraint variable under investigation.

Output: Array with constraint variable assignments.

```
BJ(D, C, i)
1.  returnDepth := 0;
2.  FOREACH d in D_i DO
      assign[i] := d;
      consistent := TRUE;
      FOR h := 1 TO i − 1 DO
        consistent := ((C(x_h, x_i) ∉ C) ∨ (assign[h], assign[i]) ∈ C(x_h, x_i));
        IF (consistent = FALSE) THEN BREAK;
      ENDDO
      maxCheckLevel := h − 1;
      IF (consistent = TRUE)
      THEN
        IF (i = n)
        THEN PRINT(assign[])
        ELSE
          maxCheckLevel := BJ(D, C, i + 1);
          IF (maxCheckLevel < i) THEN RETURN(maxCheckLevel);
        ENDIF
      ENDIF
      returnDepth := MAX({returnDepth, maxCheckLevel});
    ENDDO
```

```
3.  RETURN(returnDepth);
```

Bemerkungen:

- *returnDepth* definiert die Tiefe, zu der zurückgesprungen werden darf.
- Lösung für die diskutierten Problematiken beim Zurückspringen:
  1. Wie weit zurückspringen?
     Wegen *returnDepth* $:=$ MAX$(\{$*returnDepth*, *maxCheckLevel*$\})$ wird bei Konflikten mit der gerade untersuchten Variable die Rücksprungtiefe zu derjenigen Variable festgelegt, die am wenigsten weit zurückliegt.
  2. Mehrfaches Zurückspringen.
     returnDepth kann *während eines Backtrackings* nicht kleiner werden.

     ➜ *Während eines Backtrackings* ist kein Rücksprung möglich.

     Kann eine Variable x_i wieder konsistent belegt werden – d. h., es gibt keinen Widerspruch mit den Belegungen der Variablen $x_h, h < i$ – so wird im Aufruf BJ$(\mathcal{D}, \mathcal{C}, i+1)$ der Wert für *returnDepth* auf 0 gesetzt.

     ➜ Ein Rücksprung zur Variable $x_h, h \in \{0, \ldots, i\}$ ist möglich.

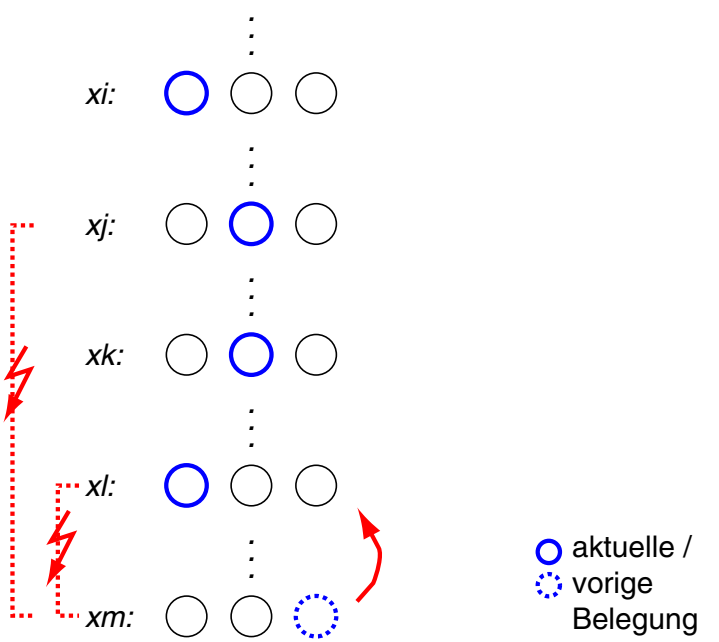
Backtracking-basierte Verfahren

Verbessertes Backjumping: Conflict-Directed Backjumping (CBJ)

Seien i, j, k, l und m Variablen-Indizes mit $i < j < k < l < m$. Die Variablen x_i, x_j, x_k, x_l, x_m werden gemäß ihres Indizes in der entsprechenden Suchraumtiefe zugewiesen.

Annahme: Alle Werte in D_m stehen in Konflikt mit x_l oder x_j.

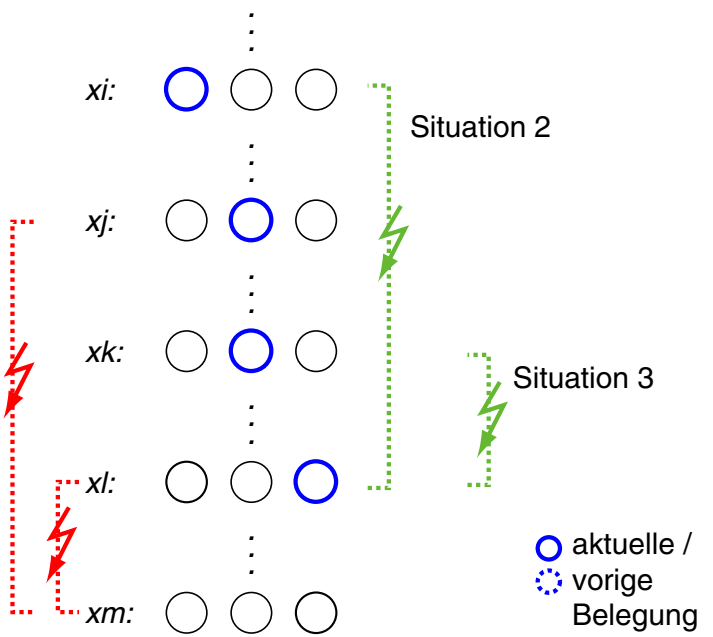
➜ Rücksprung zur Variable x_l, um anderen Wert aus D_l zu wählen. Illustration:



Backtracking-basierte Verfahren

Verbessertes Backjumping: Conflict-Directed Backjumping (CBJ)

Illustration (Fortsetzung):



Folgende Situationen sind denkbar:

1. Es existiert ein Wert in D_l, der mit keiner Variablen $x_h, h < l$, in Konflikt steht.
   ➜ x_l wird entsprechend gesetzt und weiter geht es mit x_{l+1}.
2. Die anderen Werte aus D_l stehen mit x_i in Konflikt.
   Wohin darf zurückgesprungen werden?
3. Die anderen Werte aus D_l stehen mit x_k in Konflikt.
   Wohin darf zurückgesprungen werden?

Backtracking-basierte Verfahren

Algorithm: CBJ

Input: $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$. Domains of the n constraint variables.
$\mathcal{C}$. Constraint net.
$i = 1$. Index of the constraint variable under investigation.

Output: Array with constraint variable assignments.

```
CBJ(D,C,i)
1.  cSet[i] := {0};
2.  FOREACH d in D_i DO
      assign[i] := d;  consistent := TRUE;
      FOR h := 1 TO i-1 DO
        consistent := ((C(x_h,x_i) ∉ C)  ∨  (assign[h],assign[i]) ∈ C(x_h,x_i));
        IF (consistent = FALSE) THEN BREAK;
      ENDDO
      IF (consistent = FALSE) THEN cSet[i] := cSet[i] ∪ {h-1};
      IF (consistent = TRUE)
      THEN
        IF (i = n)
        THEN PRINT(assign[]);  cSet[i] := cSet[i] ∪ {n-1};
        ELSE
          returnDepth := CBJ(D,C,i+1);
          IF (returnDepth < i) THEN RETURN(returnDepth);
        ENDIF
      ENDIF
    ENDDO
3.  returnDepth := MAX(cSet[i]);
    cSet[returnDepth] := cSet[returnDepth] ∪ (cSet[i] \ {returnDepth});
    RETURN(returnDepth);
```

Bemerkungen:

- Prinzip von Conflict-Directed Backjumping (CBJ): Betrachte bei einem Rücksprung die Vereinigungsmenge der relevanten Konfliktvariablen und wähle daraus diejenige, die am wenigsten weit zurückliegt.
- Der Algorithmus CBJ verwaltet in *cSet*$[i]$ für jede Variable x_i die Menge der Konfliktvariablen, die aus der Analyse der Constraints $C(x_h, x_i), h < i$, stammen. Bei einem Rücksprung von einer Variable x_k zu x_i wird *cSet*$[i]$ um die Menge der Konfliktvariablen ergänzt, die aus der Analyse der Constraints $C(x_h, x_k), h < i$, stammen.
- Kann eine Variable x_i wieder konsistent belegt werden – d. h., es gibt keinen Widerspruch mit den Belegungen der Variablen $x_h, h < i$ – so wird im Aufruf CBJ($\mathcal{D}, \mathcal{C}, i + 1$) der Wert von *cSet*$[i + 1]$ wieder auf die leere Menge gesetzt.
- Falls eine Lösung gefunden wurde, so wird $\{n - 1\}$ zur Menge der Konfliktvariablen hinzugenommen, um Backtracking zu garantieren, wenn der Grundbereich D_n erschöpft ist.

Backtracking-basierte Verfahren

Weitere Ansätze zur Effizienzsteigerung bei der Suchraumexploration

- Backmarking.

  Falls sich die Belegungen für zwei Variablen x_i und x_j nicht geändert haben, so kann das Ergebnis der Constraint-Auswertung wiederverwendet werden.

  ➜ Merken derjenigen Variablen, bei denen (a) ein Constraint nicht erfüllt war oder (b) sich eine Belegung geändert hat.

- Conflict-Recording.

  Jede widersprüchliche Belegung *assign*[] enthält eine minimal widersprüchliche Teilmenge, einen sogenannten *Nogood*.

  ➜ Nogoods können gelernt und gespeichert werden.

Bemerkungen:

- Backmarking verfolgt exakt den Suchpfad von Backtracking, macht aber weniger Constraint-Auswertungen.
- Conflict-Recording basiert auf der Theorie der *Truth-Maintenance-Systeme* (TMS) oder auch *Reason-Maintenance-Systeme* (RMS). Allgemein dienen solche Systeme dazu, einen nicht-montonen Schlussfolgerungsprozess nachzubilden. Im Zusammenhang mit einer Nogood-Verwaltung ist das Assumption-based TMS (ATMS) der wichtigste Vertreter.