# 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

# Konsistenz II

Ziel der Konsistenzanalyse ist die Entfernung inkonsistenter Werte in den Grundbereichen der Variablen.

Ausgangspunkt der folgenden Betrachtungen:

- alle Constraints sind entweder unär oder binär
- zwischen Constraint-Netz und Constraint-Graph wird nicht explizit unterschieden
- der Graph ist ein Kanten-Constraint-Graph: Knoten entsprechen Variablen, Kanten entsprechen Constraints

# Konsistenz II

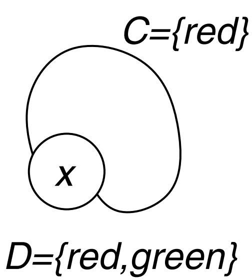
## Definition 8 (Knotenkonsistenz)

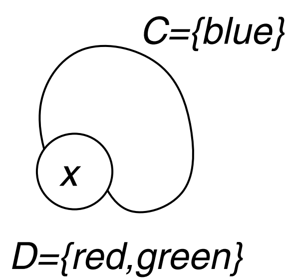
Sei $x$ eine Variable und $D$ der zugehörige Grundbereich. Sei $C(x)$ ein auf $x$ definierter (unärer) Constraint. Dann heißt die Variable $x$ knotenkonsistent (node consistent), falls gilt:

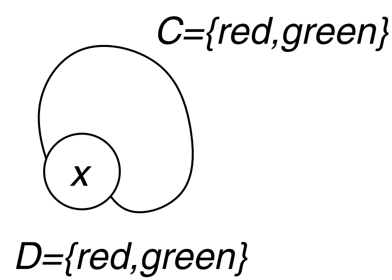
$$\forall d \in D : d \in C(x)$$

Ein Constraint-Netz heißt knotenkonsistent, falls jede Variable des Constraint-Netzes knotenkonsistent ist.

Beispiele:



# Konsistenz II

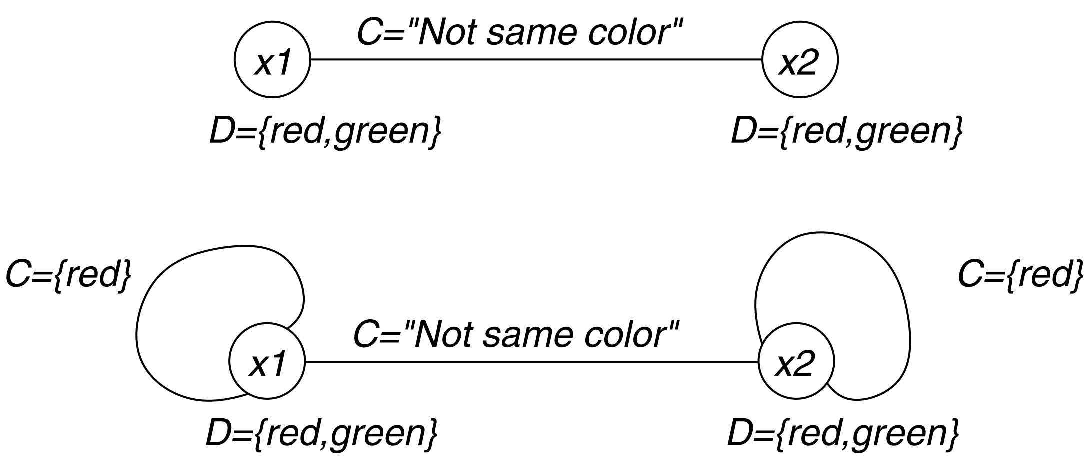
## Definition 9 (Kantenkonsistenz)

Seien $x_i$ und $x_j$ Variablen mit den Grundbereichen $D_i$ und $D_j$. Sei $C(x_i, x_j)$ ein auf $x_i$ und $x_j$ definierter (binärer) Constraint. Dann heißt die Kante $(x_i, x_j)$ kantenkonsistent (arc consistent), falls gilt:

$$\forall d_i \in D_i \; \exists d_j \in D_j : (d_i, d_j) \in C(x_i, x_j)$$

Ein Constraint-Netz $\mathcal{C}$ heißt kantenkonsistent, falls jede Kante des Constraint-Netzes kantenkonsistent ist.

Beispiele:



Bemerkungen:

- Algorithmen zur Konsistenzanalyse sind *lokale* Verfahren. Was bedeutet das?
- Knotenkonsistenz wird oft durch die Wahl der Grundbereiche garantiert.
- Ist es egal oder sinnvoll oder notwendig, die Knotenkonsistenz eines Constraint-Netzes vor seiner Kantenkonsistenz zu herzustellen?

# Konsistenz II

**Lemma 10**

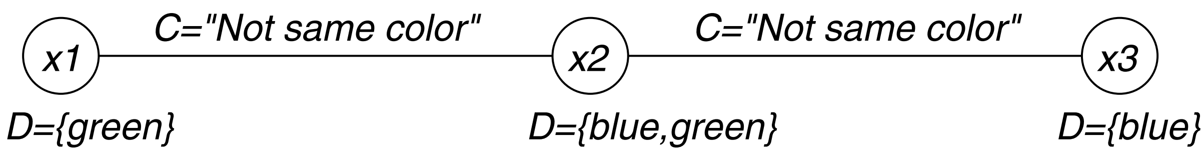
Sei $\mathcal{C}$ ein Constraint-Netz über den Variablen $x_1, x_2, \ldots, x_n$ mit den zugehörigen Grundbereichen $D_1, D_2, \ldots, D_n$. Enthält $\mathcal{C}$ nur unäre und binäre Constraints, dann gilt:

$$(D_1, D_2, \ldots, D_n) \text{ ist eine lokal konsistente Lösung für } \mathcal{C}$$

$$\Rightarrow$$

$$\mathcal{C} \text{ ist knoten- und kantenkonsistent.}$$

Beispiel:



# Konsistenz II

## Lemma 10

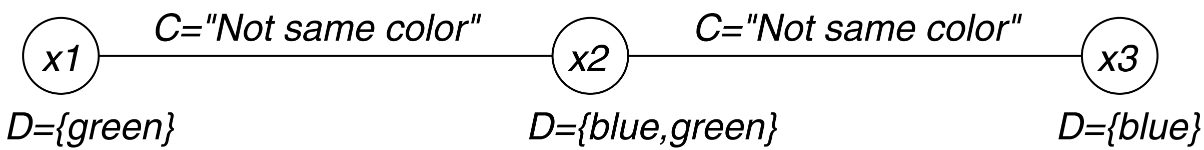
Sei $\mathcal{C}$ ein Constraint-Netz über den Variablen $x_1, x_2, \ldots, x_n$ mit den zugehörigen Grundbereichen $D_1, D_2, \ldots, D_n$. Enthält $\mathcal{C}$ nur unäre und binäre Constraints, dann gilt:

$$(D_1, D_2, \ldots, D_n) \text{ ist eine lokal konsistente Lösung für } \mathcal{C}$$

$$\Rightarrow$$

$$\mathcal{C} \text{ ist knoten- und kantenkonsistent.}$$

Beispiel:




Wenn $(D_1, D_2, \ldots, D_n)$ eine lokal konsistente Lösung für ein Constraint-Netz $\mathcal{C}$ ist, so nennen wir $\mathcal{C}$ auch lokal konsistent.

Ist das Constraint-Netz im Beispiel lokal konsistent bzw. kantenkonsistent?

# Konsistenzanalyse

## 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

# Konsistenzanalyse

## Kantenkonsistenz in CSP-FD

**Definition 11 (Constraint-Propagierung, Filtern)**

Constraint-Propagierung bedeutet, die durch Constraints definierten Relationen mittels lokaler Analyse zur Verkleinerung der Grundbereiche der Variablen auszunutzen.

Sprachgebrauch: Filtern (Filtering) der Grundbereiche.

Beispiel für Constraint-Propagierung: Herstellung der Kantenkonsistenz mittels der Funktionen „Revise“ and „AC“ (arc consistency).

- Revise.

  Betrachtet einen binären Constraint und entfernt die Elemente aus dem Grundbereich einer Variablen, für die keine Unterstützung (support) im Grundbereich der adjazenten Variablen gegeben ist.

- AC.

  Wendet Revise für ein Constraint-Netz an.

# Konsistenzanalyse

## Kantenkonsistenz in CSP-FD

Algorithm: Revise

Input: $D_i, D_j$. Domains of the constraint variables $x_i, x_j$.
$C(x_i, x_j)$. Constraint between $x_i$ and $x_j$.

Output: *delete*. Indicates whether or not $D_i$ has been reduced.

Side effect: Reduction of $D_i$ by unsupported elements respecting $C(x_i, x_j)$.

Revise($D_i, D_j, C(x_i, x_j)$)

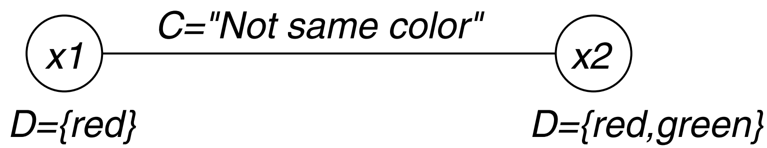
1. *delete* $:=$ FALSE;
2. FOREACH $d_i$ in $D_i$ DO
   IF NOT ($\exists\ d_j \in D_j : (d_i, d_j) \in C(x_i, x_j)$)
   THEN
   $D_i := D_i \setminus \{d_i\}$;
   *delete* $:=$ TRUE;
   ENDIF
   ENDDO
3. RETURN(*delete*)

Wie ist die Laufzeit von Revise im $O$-Kalkül?

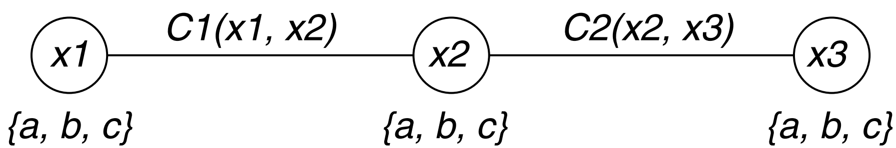
Bemerkungen:

- Das Konzept der Kantenkonsistenz ist gerichtet. Beispiel:




- Um ein Constraint-Netz kantenkonsistent zu machen, reicht es nicht aus, Revise einmal für jede Kante auszuführen. Beispiel:

$C_1(x_1, x_2) = \{(a, b), (a, c), (b, a), (c, a)\}$
$C_2(x_2, x_3) = \{(c, b)\}$



# Konsistenzanalyse

## Kantenkonsistenz in CSP-FD

Algorithm: AC1

Input: $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$. Domains of $n$ constraint variables.
$\mathcal{C}$. Constraint net.

Side effect: Reduction of the domains in $\mathcal{D}$ such that $\mathcal{C}$ is arc consistent.

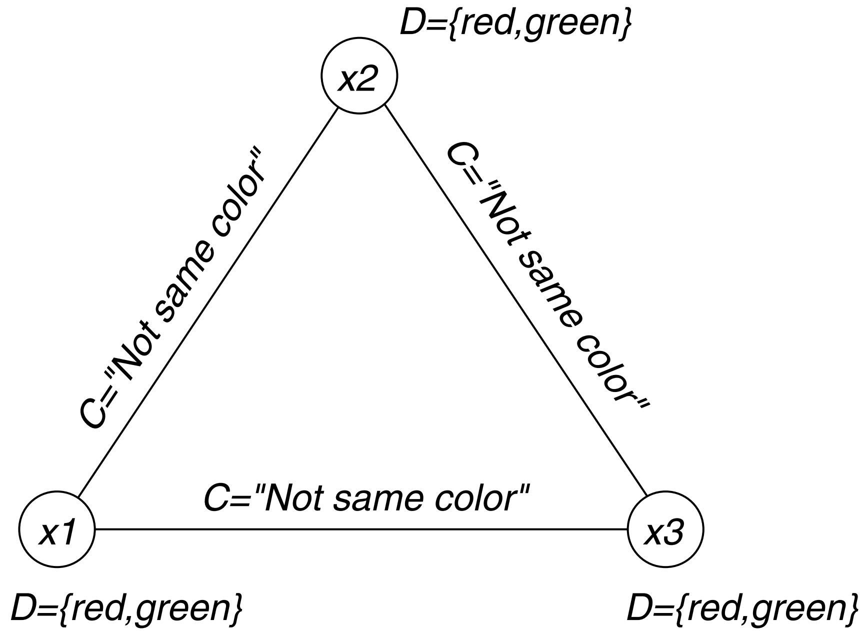
AC1($\mathcal{D}$,$\mathcal{C}$)

1. FOR $i := 1$ TO $n$ DO
   **makeNodeConsistent**($x_i$);
   ENDDO
2. REPEAT
   *change* := FALSE;
   FOREACH $C(x_i, x_j)$ in $\mathcal{C}$ DO
     *change* := (*change* $\vee$ **Revise**($D_i, D_j, C(x_i, x_j)$));
   ENDDO
   UNTIL (NOT (*change*));

Wie ist die Laufzeit von AC1 im $O$-Kalkül?

Bemerkungen:

- Die Kantenkonsistenz eines Constraint-Netzes garantiert nicht seine globale Erfüllbarkeit (globale Konsistenz). Beispiel:




- Trotzdem ist die Herstellung der Kantenkonsistenz eines Constraint-Netzes sinnvoll:
  1. Sie dient zur Verkleinerung des Lösungsraums.
  2. Berechnungsverfahren lokal ➜ effizient und parallel ausführbar.

# Konsistenzanalyse

## Kantenkonsistenz in CSP-FD

Verbesserung von AC1 [Mackworth 1977]: Es werden nur solche Kanten auf ihre Konsistenz überprüft, die inzident zu einer Variablen sind, deren Wertebereich reduziert wurde.

Algorithm: AC3

Input: $\mathcal{D} = \{D_1, D_2, \ldots, D_n\}$. Domains of $n$ constraint variables.
$\mathcal{C}$. Constraint net.

Side effect: Reduction of the domains in $\mathcal{D}$ such that $\mathcal{C}$ is arc consistent.

AC3($\mathcal{D}$, $\mathcal{C}$)

1. FOR $i := 1$ TO $n$ DO
   **makeNodeConsistent**($x_i$);
   ENDDO
2. $\mathcal{Q} := \mathcal{C}$;
3. WHILE (NOT ($\mathcal{Q} = \emptyset$)) DO
   CHOOSE $C(x_i, x_j)$ FROM $\mathcal{Q}$;
   $\mathcal{Q} := \mathcal{Q} \setminus \{C(x_i, x_j)\}$;
   IF **Revise**($D_i, D_j, C(x_i, x_j)$)
   THEN $\mathcal{Q} := \mathcal{Q} \cup \{C(x_h, x_i) \mid h \neq i \wedge h \neq j\}$;
   ENDDO

Bemerkungen:

- Die Laufzeitkomplexität von AC3 ist $\mathcal{O}(m \cdot d^3)$. Hierbei ist $m = |\mathcal{C}|$ und $d = \max\{|D_i| \mid i = 1 \ldots n\}$.
[Mackworth/Freuder 1985]

- AC4.
Für jeden Constraint $C(x_i, x_j)$ und für alle Elemente der Grundbereiche $D_i$ werden die erfüllenden (= Support) Elemente aus $D_j$ gespeichert. Die Laufzeitkomplexität sind (optimale) $\mathcal{O}(m \cdot d^2)$; der Speicheraufwand ist jedoch beträchtlich.
[Mohr/Henderson 1986]
Es kann gezeigt werden, dass AC3 im Durchschnitt effizienter ist als AC4.
[Wallace 1993]

- AC6.
Ähnlich wie AC4; Verbesserung der Platzkomplexität bei gleicher Laufzeitkomplexität dadurch, dass zu jedem Zeitpunkt nur ein Support-Element gespeichert wird.
[Bessiere 1994]

- AC7.
Weitere Verbesserung von AC6 durch effizientere Konsistenz-Checks: Ausnutzung spezieller Support-Eigenschaften.
[Bessiere et al. 1995, Schiex et al. 1996]

# Konsistenzanalyse

## Pfadkonsistenz in CSP-FD

Wie kann der Konsistenzbegriff weiter verstärkt werden?

**Definition 12 (Pfadkonsistenz, path consistency)**

Sei $\mathcal{C}$ ein Constraint-Netz über den Variablen $x_1, x_2, \ldots, x_n$ mit den zugehörigen Grundbereichen $D_1, D_2, \ldots, D_n$. Dann heißt $\mathcal{C}$ pfadkonsistent, falls für jede Kante $(x_i, x_j)$ und für jeden Pfad der Länge $l$ zwischen $x_i$ und $x_j$ gilt:

1. $(x_i, x_j)$ ist kantenkonsistent

2. $\forall (d_i, d_j) \in C(x_i, x_j),\ C(x_i, x_j) \in \mathcal{C}$
$\forall (x_{p_1}, x_{p_2}, \ldots, x_{p_{l-1}}, x_{p_l}),\ x_{p_1} = x_i,\ x_{p_l} = x_j :$
$\exists (d_i, d_{p_2}) \in C(x_i, x_{p_2})\ \ldots\ \exists (d_{p_{l-1}}, d_j) \in C(x_{p_{l-1}}, x_j)$

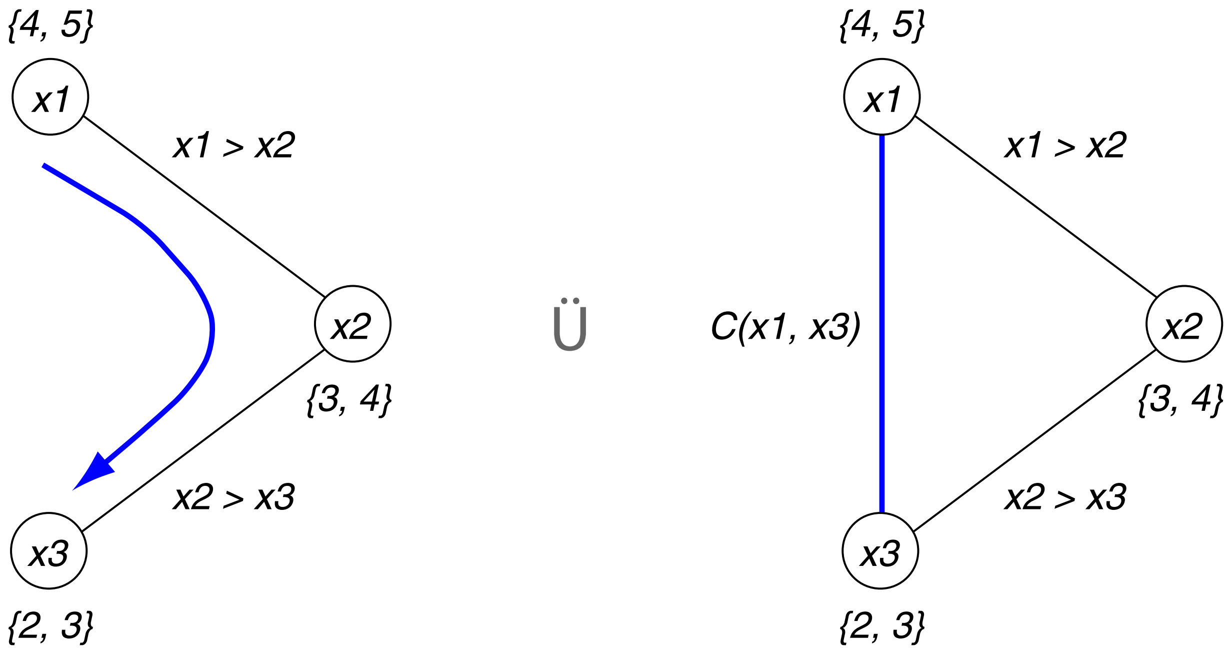
Bemerkungen:

- Jeder Constraint-Graph kann als vollständiger Graph interpretiert werden.
- Falls kein Constraint zwischen $x_i$ und $x_j$ besteht, kann ein spezieller True-Constraint $C(x_i, x_j)$ als Menge der Tupel des kartesischen Produktes von $D_i$ und $D_j$ vereinbart werden.
- Bei Herstellung der Pfadkonsistenz wird dieser True-Constraint wird unter Berücksichtigung der aktuellen Elemente in $D_i$ und $D_j$ angepasst.

# Konsistenzanalyse

## Pfadkonsistenz in CSP-FD

Beispiel:




Vor Herstellung der Pfadkonsistenz gab es kein $C(x_1, x_3)$ bzw. nur ein *True*$(x_1, x_3) = \{(4,2),(4,3),(5,2),(5,3)\}$. Nach Herstellung der Pfadkonsistenz sind für $(x_1, x_3)$ nur die Tupel $\{(4,2),(5,2),(5,3)\}$ vereinbar. Dies entspricht der Einführung eines neuen Constraints $C(x_1, x_3)$.

# Konsistenzanalyse

## Pfadkonsistenz in CSP-FD [Montanari 1974]

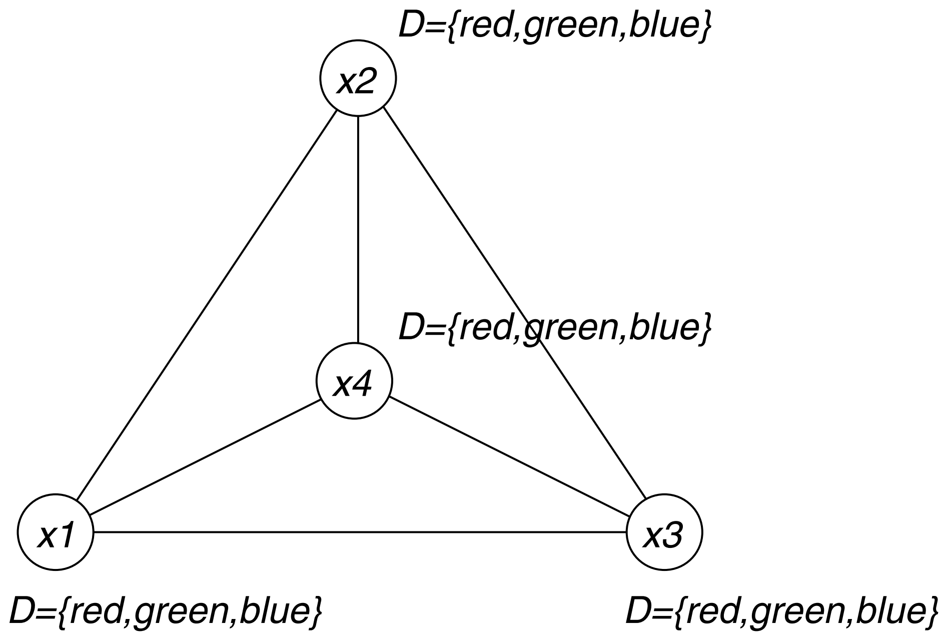
**Satz 13 (Pfadkonsistenz im vollständigen Constraint-Netz)**

Gegeben sei ein Constraint-Problem auf endlichen Grundbereichen (CSP-FD). Weiterhin sei das Constraint-Netz vollständig, also zwischen je zwei Variablen sei ein Constraint definiert. Dann gilt:

Das Constraint-Netz ist pfadkonsistent gdw. das Constraint-Netz für alle Pfade der Länge Zwei pfadkonsistent ist.

Bemerkungen:

- Die Pfadkosistenz eines Constraint-Netzes garantiert nicht seine globale Erfüllbarkeit (globale Konsistenz). Das abgebildete Constraint-Netz zeigt ein Beispiel; alle Constraints seien als „*Not same color*“ vereinbart:



# Konsistenzanalyse

**Definition 14 ($k$-Konsistenz, strenge $k$-Konsistenz)**

Sei $\mathcal{C}$ ein Constraint-Netz über einer Variablenmenge $X$ und seien $\{x_1, \ldots, x_{k-1}\} \subset X$ Variablen mit den Grundbereichen $D_1, \ldots, D_{k-1}$. Weiterhin seien Werte $d_1, \ldots, d_{k-1}$ in $D_1, \ldots, D_{k-1}$ gegeben, so dass alle Constraints zwischen den Variablen $x_1, \ldots, x_{k-1}$ erfüllt sind.

Existiert nun für jede weitere Variable $x_k$ mit Grundbereich $D_k$ ein Wert $d_k \in D_k$, so dass alle Constraints zwischen den $k$ Variablen erfüllt sind, dann heißt $\mathcal{C}$ $k$-konsistent.

Ein Constraint-Netz $\mathcal{C}$ heißt streng $k$-konsistent, falls es $\kappa$-konsistent für alle $\kappa \leq k$ ist.

Bemerkungen:

- Knotenkonsistenz ist äquivalent zu strenger 1-Konsistenz.
- ungerichtete Kantenkonsistenz ist äquivalent zu strenger 2-Konsistenz.
- Pfadkonsistenz ist äquivalent zu strenger 3-Konsistenz.

# Konsistenzanalyse

## $k$-Konsistenz und Backtracking-freie Suche

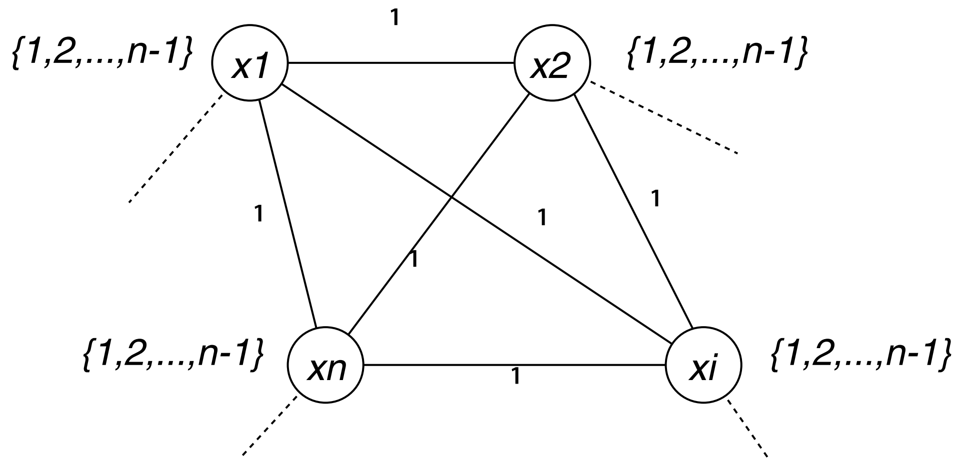
Gibt es spezielle Constraint-Probleme, bei denen (die Herstellung einer) $k$-Konsistenz ein Backtracking bei der Lösungssuche überflüssig macht?

Antworten:

- Ein kantenkonsistentes Constraint-Netz auf $n$ Variablen mit $|D_i| = 1, i = 1, \ldots, n$ ist global konsistent.
- Ein streng $n$-konsistentes Constraint-Netz auf $n$ Variablen ist global konsistent.
- In einem streng $(n-1)$-konsistentem Constraint-Netz auf $n$ Variablen ist im allgemeinen Suche nicht zu vermeiden.

Bemerkungen:

- Ein streng $(n-1)$-konsistentem Constraint-Netz auf $n$ Variablen muss keine Lösung besitzen.
- Beispiel: ein vollständiger Constraint-Graph mit „$\neq$ Constraints“ und dem Grundbereich $\{1, \ldots, n-1\}$ für alle Variablen.



# Konsistenzanalyse

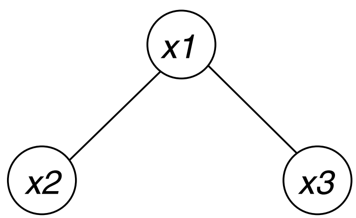
## $k$-Konsistenz und Backtracking-freie Suche

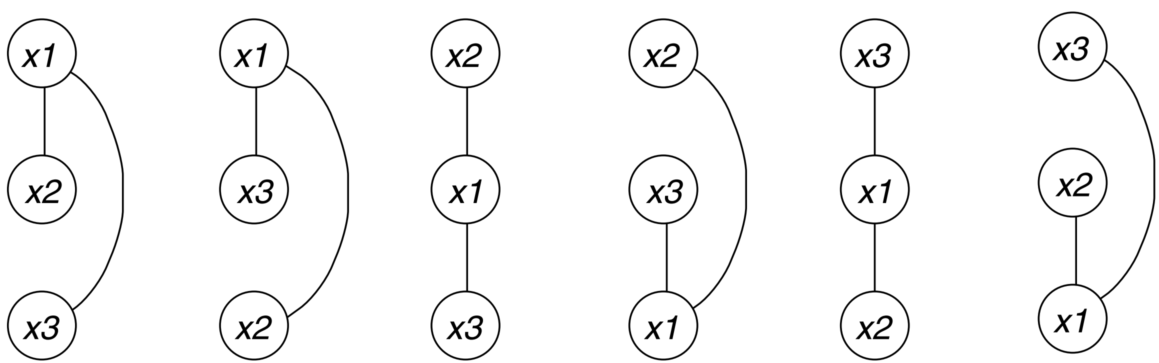
Es existiert ein Zusammenhang zwischen strenger $k$-Konsistenz und der Weite eines Constraint-Netzes.

**Definition 15 (Ordnung eines Constraint-Netzes)**

Ein Constraint-Netz heißt geordnet, wenn auf seinen Knoten eine lineare Ordnung definiert ist.

Beispiel:



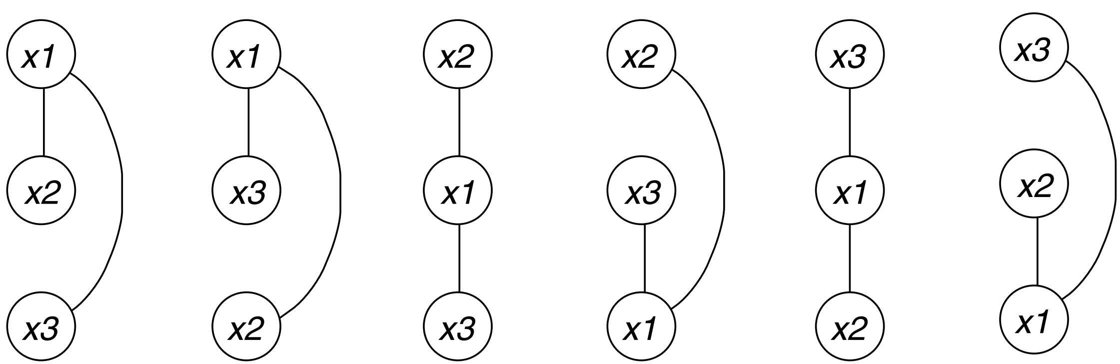
# Konsistenzanalyse

## $k$-Konsistenz und Backtracking-freie Suche

**Definition 16 (Weite eines Constraint-Netzes)**

1. Die Weite eines Knoten $v$ in einem geordneten Constraint-Netz ist die Anzahl der Kanten von $v$ zu Vorgängerknoten von $v$ (bezüglich der linearen Ordnung).
2. Die Weite eines geordneten Constraint-Netzes ist gleich dem Maximum aller Weiten seiner Knoten.
3. Die Weite eines Constraint-Netzes ist gleich der minimalen Weite aller linearen Ordnungen des Constraint-Netzes.

Beispiel:




Weite: 1 1 1 2 1 2

Bemerkungen:

- Jedes Suchverfahren definiert implizit eine Ordnung auf den Constraint-Variablen.
- Die Ordnung eines Constraint-Netzes beschreibt die Reihenfolge, in der ein Lösungsverfahren Werte für die Constraint-Variablen instanziiert.

# Konsistenzanalyse

## $k$-Konsistenz und Backtracking-freie Suche

**Satz 17 (Backtracking-freie Suche** [Freuder 1982]**)**

Gegeben sei ein Constraint-Problem auf endlichen Grundbereichen (CSP-FD). Weiterhin sei das Constraint-Netz streng $k$-konsistent und besitze die Weite $w$. Dann gilt:

Ist $k > w$, dann existiert bei der Suche nach einer erfüllenden Belegung für CSP-FD eine Instanziierungsreihenfolge, die Backtracking-frei ist.

Bemerkungen:

- Die Instanziierungsreihenfolge ist durch die Ordnung des Constraint-Netzes gegeben, bei der die Weite gleich $w$ ist.
- Dass eine erfüllende Belegung ohne Backtracking gefunden werden kann, impliziert die Existenz einer erfüllende Belegung.

# Konsistenzanalyse

## $k$-Konsistenz und Backtracking-freie Suche

Diskussion von Freuders Satz.

- Die Ordnung eines Constraint-Netzes mit minimaler Weite $w$ lässt sich polynomiell berechnen. Man braucht „nur“ noch das Constraint-Netz streng $(w + 1)$-konsistent zu machen, und kann eine Lösung ohne Suche bestimmen.

- Problem: Für $k > 2$ ist eine strenge $k$-Konsistenz nur mit Einführung zusätzlicher Kanten erzielbar, wodurch sich auch die Weite $w$ des Constraint-Netzes erhöhen kann.

- Typischerweise werden bei der Herstellung strenger $3$-Konsistenz so viele Kanten eingeführt, dass die Weite des Constraint-Netzes $n$ (Anzahl der Variablen) wird.

Fragen:

- Wie ist die Weite eines Constraint-Netzes, das einem Baum entspricht?
- Welche Konsequenz hat das?

# Konsistenzanalyse

Viele Suchverfahren instanziieren die Variablen in einer festen Reihenfolge (Stichwort: Tree-Search). Dieses Wissen über die Reihenfolge kann bei der Konsistenzanalyse ausgenutzt werden und führt zum Prinzip der gerichteten Konsistenzanalyse.

Beobachtungen:

1. In einer Tree-Search-Situation reicht es aus, zu garantieren, dass der Grundbereich einer zukünftig zu instanziierenden Variable einen Wert enthält, der mit einer aktuellen Belegung verträglich ist:
   Wird $x_i$ vor $x_j$ instanziiert, garantiert man mittels Revise($x_i, x_j, C(x_i, x_j)$) den Support für $x_i$ bzgl. $x_j$. Umgekehrt braucht jedoch nicht der Support für $x_j$ bzgl. $x_i$ garantiert zu werden – das wird durch Backtracking erledigt: Die Elemente von $D_j$ werden der Reihe nach durchprobiert, bis ein passendes gefunden ist.
   Würde man mittels Revise($x_j, x_i, C(x_i, x_j)$) den Support für $x_j$ bzgl. $x_i$ garantieren, so wäre der hiermit verbundene Aufwand dann als zuviel investiert, wenn es gar nicht zu einem Backtracking über diese Elemente von $D_j$ kommt.

2. Im Verlauf der Suche werden die Wertebereiche der Variablen eines kantenkonsistenten Constraint-Netzes verkleinert. Dabei bleibt in einer Tree-Search-Situation eine hergestellte Kantenkonsistenz erhalten:

$$\forall d_i \in D_i \; \exists d_j \in D_j : \; (d_i, d_j) \in C(x_i, x_j)$$

$$\Rightarrow \quad \forall D_i', \; D_i' \subseteq D_i : \; \forall d_i \in D_i' \; \exists d_j \in D_j : \; (d_i, d_j) \in C(x_i, x_j)$$

➜ Konsistenzanalyse gemäß AC$n$ macht zuviel des Guten.

# Konsistenzanalyse

## Gerichtete Verfahren

Schema von DAC (*directed arc consistency*) [Dechter/Pearl 1988]:

1. Bestimmung einer Instanziierungsreihenfolge $\pi : X \to \{1, \ldots, n\}$ der $n$ Variablen $X$. $\pi$ ist bijektiv.

2. Anwendung von Revise($D_i, D_j, C$) für Kante $(x_i, x_j)$ gdw.

   (a) $\pi(x_i) < \pi(x_j)$ und

   (b) Revise für folgende Kanten durchgeführt wurde:

$$\{(x_k, x_l) \mid ((\pi(x_k) < \pi(x_l)) \wedge (\pi(x_k) \geq \pi(x_j))\}$$

3. Anwendung von Backtracking entsprechend der durch $\pi$ festgelegten Reihenfolge.

Bemerkungen:

- DAC gibt an zwei Stellen eine Ordnung vor:
  (a) Die Reihenfolge, in der Constraints an Revise übergeben werden;
  (b) die Reihenfolge der Grundbereiche in jedem Revise-Aufruf.

# Konsistenzanalyse

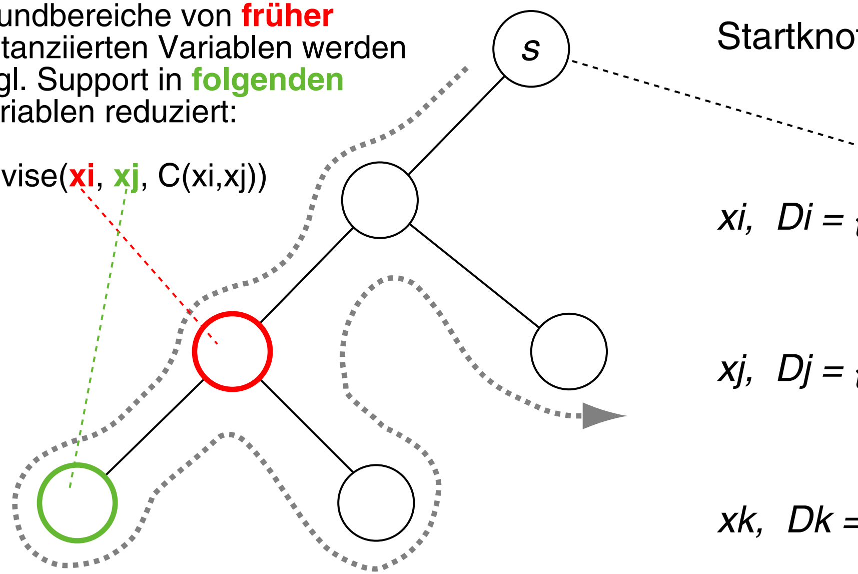
## Gerichtete Verfahren

Beispiel:

Grundbereiche von **früher** instanziierten Variablen werden bzgl. Support in **folgenden** Variablen reduziert:

Revise(**xi**, **xj**, C(xi,xj))

Reihenfolge, in der Constraints an Revise übergeben werden.




Startknoten

*xi, Di = {1, 5}*

*xj, Dj = {2, 3, 8}*

*xk, Dk = {4, 6}*

$C(xi,xj) := xi < xj$

$C(xj,xk) := xj < xk$

Bemerkungen:

- Reihenfolge der Revise-Anwendung von oben kommend:
  Revise$(D_i, D_j, C(x_i, x_j))$ vor Revise$(D_j, D_k, C(x_j, x_k))$.
  Die Grundbereiche nach dem Filtern sind: $D_i = \{1, 5\}$, $D_j = \{2, 3\}$, $D_k = \{4, 6\}$. Durch den Constraint $C(x_j, x_k)$ wurde Wert $8$ aus $D_j$ gestrichen; als Folge kann für Wert $5 \in D_i$ der Constraint $C(x_i, x_j))$ nicht mehr erfüllt werden.

- Reihenfolge der Revise-Anwendung von unten kommend (DAC):
  Revise$(D_j, D_k, C(x_j, x_k))$ vor Revise$(D_i, D_j, C(x_i, x_j))$.
  Die Grundbereiche nach dem Filtern sind: $D_i = \{1\}$, $D_j = \{2, 3\}$, $D_k = \{4, 6\}$. Beide Constraints sind kantenkonsistent.

- Gerichtete Kantenkonsistenz lässt sich in kanonischer Weise hinsichtlich gerichteter Pfadkonsistenz erweitern.

# Konsistenzanalyse

## Praktikabilität der Verfahren

Die wichtigsten Aussagen zur Konsistenzanalyse im Überblick:

- Die Herstellung der Kantenkonsistenz für ein CSP-FD ist in vielen Fällen angesagt.
- Für Baumsuchverfahren (feste Reihenfolge der Variablen) ist die Herstellung der gerichteten Kantenkonsistenz ausreichend.
- Höhere Werte als 2 für strenge $k$-Konsistenz haben sich in der Praxis nicht bewährt. Gründe:
  - hoher Speicheraufwand
  - Laufzeit für Konsistenzanalyse ist oft unverhältnimäßig hoch im Vergleich zur erzielten Vereinfachung des CSP-FD.

# Konsistenzanalyse

## 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

# Konsistenzanalyse

## Kombination mit Backtracking

Durch eine Konsistenzanalyse allein lässt sich oft keine Lösung eines CSP-FD bestimmen.

→ Kombination einer Konsistenzanalyse mit Suchverfahren

Zwei Möglichkeiten der Kombination:

(a) Konsistenzanalyse als Preprocessing (klar)

(b) Integration der Konsistenzanalyse in Backtracking: Look-Ahead

# Konsistenzanalyse

## Kombination mit Backtracking

Durch eine Konsistenzanalyse allein lässt sich oft keine Lösung eines CSP-FD bestimmen.

➜ Kombination einer Konsistenzanalyse mit Suchverfahren

Zwei Möglichkeiten der Kombination:

(a) Konsistenzanalyse als Preprocessing (klar)

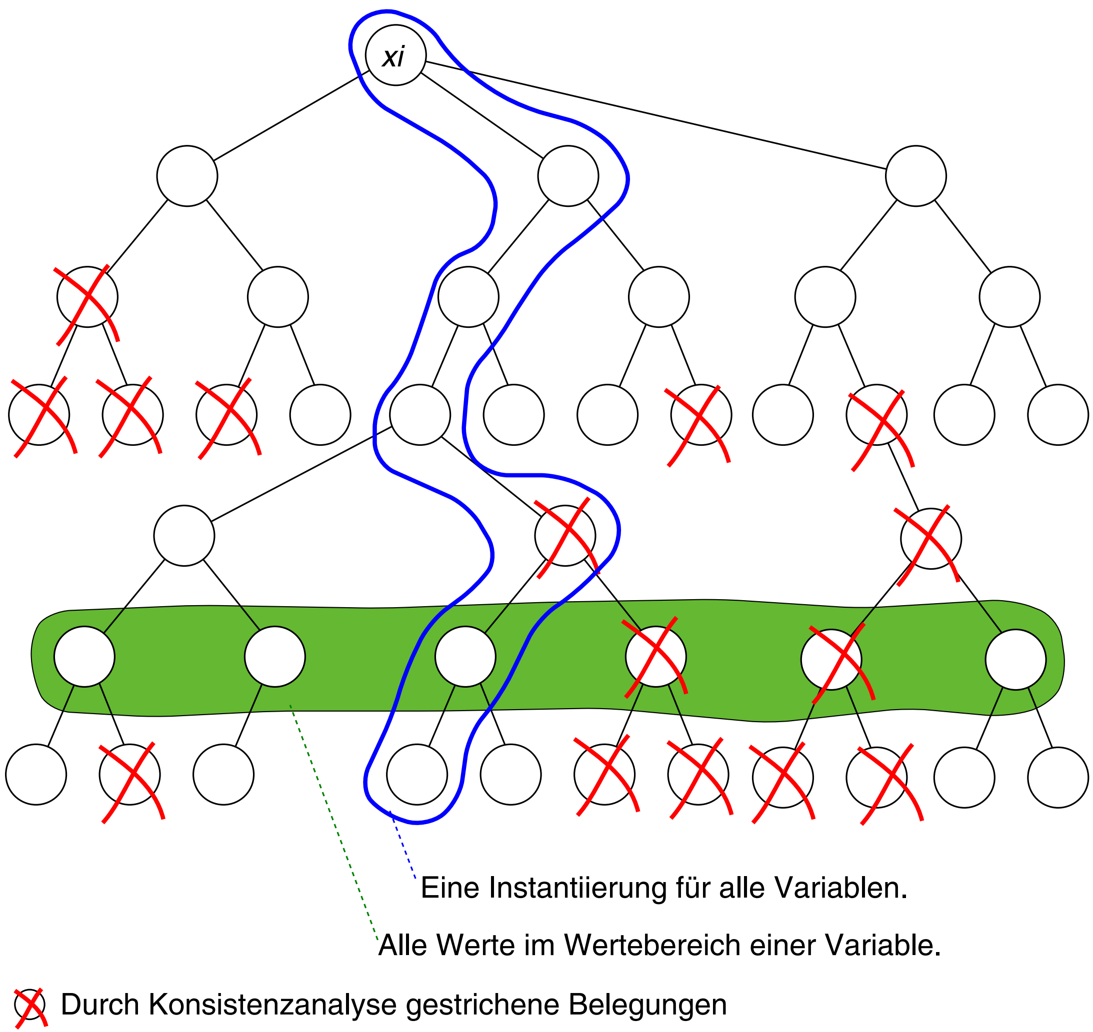
(b) Integration der Konsistenzanalyse in Backtracking: Look-Ahead

zu (b) Prinzip einer Look-Ahead-Strategie:

1. Instanziierung: Auswahl einer Variable $x_i \in X$ mit $|D_i| > 1$ und Auswahl eines Elementes $d_i \in D_i$.
2. Herstellung eines bestimmten Konsistenzlevels im aktuellen Constraint-Problem. Folgende Sonderfälle sind interessant:
   - Die Wertebereiche aller Variablen enthalten nur ein Element und das aktuelle Constraint-Problem ist kantenkonsistent. ➜ Lösung gefunden
   - Der Wertebereich einer Variablen ist leer. ➜ Backtracking
3. $X := X \setminus \{x_i\}$. Weiter bei 1.

# Konsistenzanalyse

## Kombination mit Backtracking



Bemerkungen:

- Generate-and-Test erzeugt eine Instanziierung für alle Variablen; Konsistenzanalyse (Filtern) analysiert alle Werte im Grundbereich einer Variablen.
- Die Kombination dieser beiden „orthogonalen“ Konzepte führt zu einem Spektrum von CSP-FD-Algorithmen: Backtracking, Forward-Checking, Partial-Look-Ahead, Full-Look-Ahead.

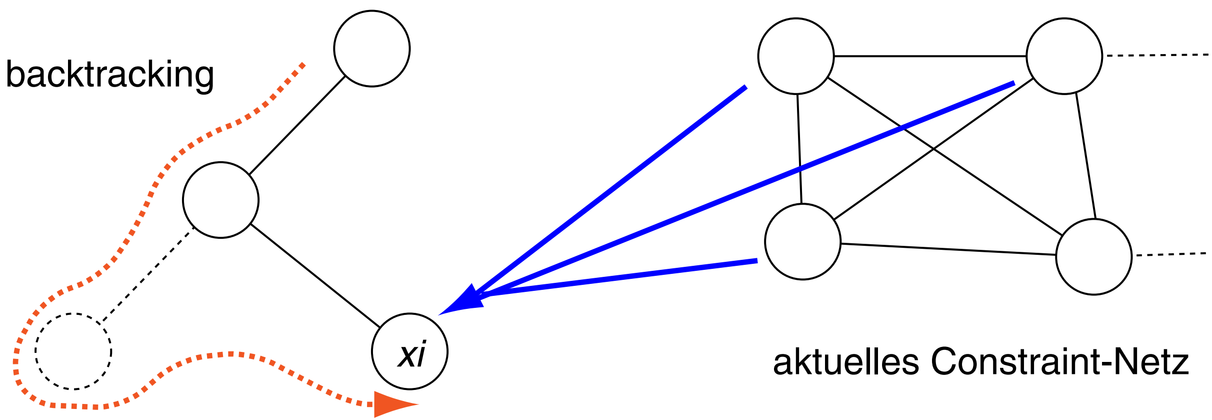
# Konsistenzanalyse

## Kombination mit Backtracking

Bei den Look-Ahead-Strategien werden drei Konsistenzlevel unterschieden:

1. Forward-Checking.

   Herstellung der Kantenkonsistenz nur zwischen der zuletzt instanziierten Variable und den nicht instanziierten Variablen.



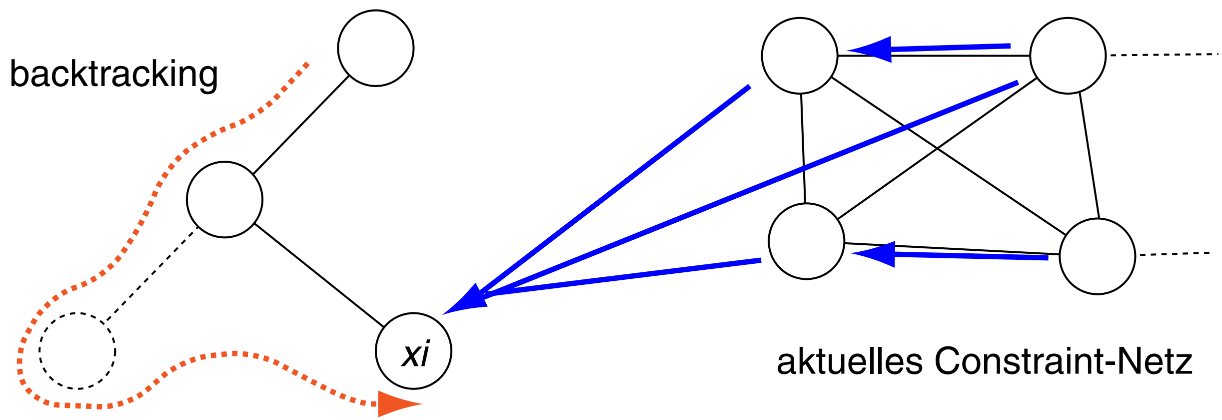

 

# Konsistenzanalyse

## Kombination mit Backtracking

Look-Ahead-Strategien (Fortsetzung):

2. Partial-Look-Ahead.
   Herstellung der gerichteten Kantenkonsistenz.




3. Full-Look-Ahead.
   Herstellung der Kantenkonsistenz im aktuellen Constraint-Netz.

# Weitere Analyseverfahren

## 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

# Weitere Analyseverfahren

## Variablensortierung

Die Reihenfolge, in der Variablen instanziiert werden, kann entscheidenden Einfluss auf die Komplexität der Suche haben.

Wichtige Heuristiken sind:

- Least-Commitment.
  „Verschiebe Entscheidungen so, dass wenig Verpflichtungen entstehen.“
  Anders ausgedrückt: „Instanziiere Variablen derart, dass der Freiheitsgrad bei zukünftigen Entscheidungen hoch bleibt.“

- Maximum-Pervasion.
  „Instanziiere diejenigen Variablen zuerst, die in der höchsten Anzahl von Constraints vorkommen.“

- Small-is-Quick. [Dechter/Pearl 1988]
  „Instanziiere Variablen derart, dass das resultierende Constraint-Problem möglichst klein wird.“

Bemerkungen:

- Least-Commitment-Heuristik von [Freuder/Quinn 1985]:
  „Instanziiere Variablen eines Stable-Sets zuletzt.“ Ein Stable-Set ist eine Menge von Variablen, zwischen denen kein Constraint definiert ist.

# Weitere Analyseverfahren

## Constraint-Netz-Reformulierung

Einteilung der Ansätze zur Constraint-Netz-Reformulierung:

1. Einführung von Constraints.

   Beispiel: Herstellung der $k$-Konsistenz in einem Constraint-Netz.

2. Entfernung von Constraints.

   Beispiel Cycle-Cut-Set-Analyse: Reduktion des Graphen des Constraint-Netzes auf einen Wald.

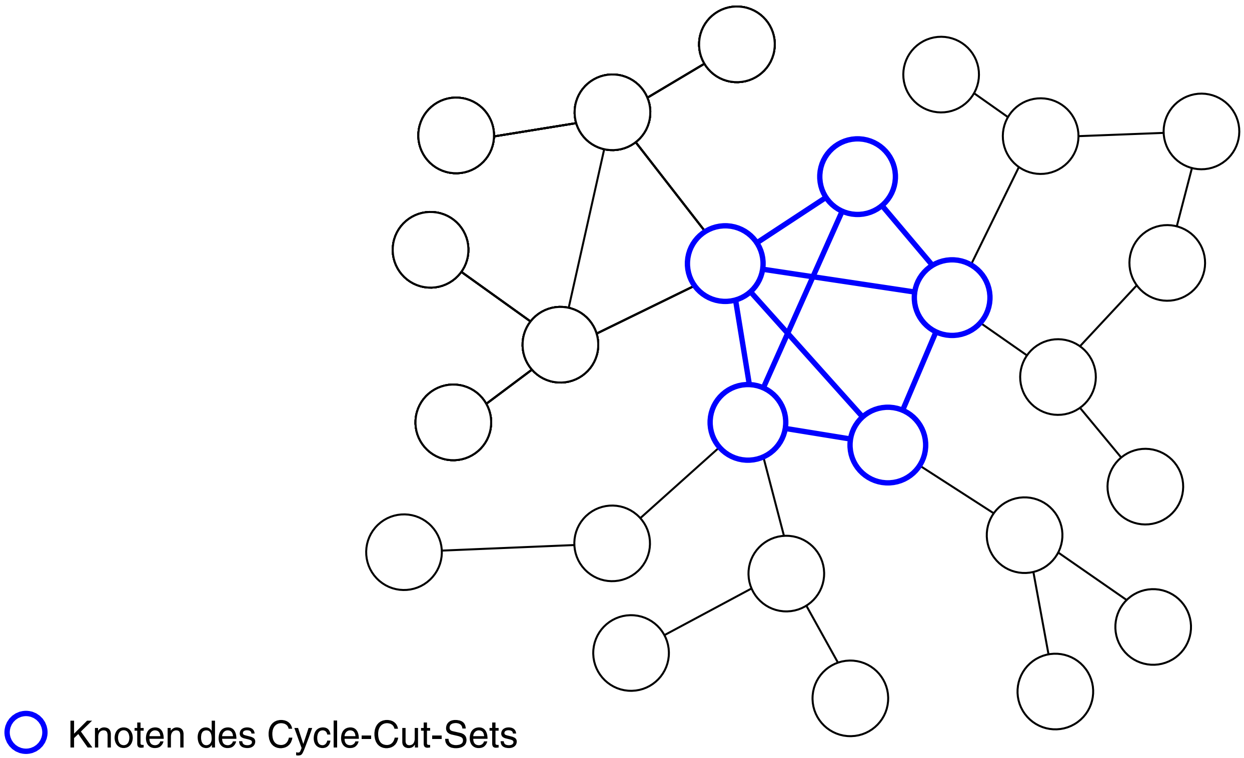
**Definition 18 (Cycle-Cut-Set)**

Sei $G = \langle V, E \rangle$ ein Graph, der kein Wald ist. Weiterhin sei $V_c \subset V \neq \emptyset$ und $G_T := G[V \setminus V_c]$ ein auf der Menge $V \setminus V_c$ induzierter Subgraph. Dann heißt $V_c$ Cycle-Cut-Set, falls $G_T$ ein Wald ist.

# Weitere Analyseverfahren

## Constraint-Netz-Reformulierung: Cycle-Cut-Set-Analyse

Illustration:




Sei $d_c = \max\{|D_i| \mid x_i \text{ aus } \mathcal{C}_c\}$. Dann lässt sich das Constraint-Netz $\mathcal{C}$ in $\mathcal{O}(d_c^m \cdot (n-m) \cdot d_T^2)$ Schritten lösen.

Warum lässt sich nicht jede Lösung von $\mathcal{C}_c$ zu einer globalen Lösung ausbauen?

Bemerkungen:

- Idee der Cycle-Cut-Set-Analyse: Hat man die Variablen $V_c$ eines Cycle-Cut-Set derart instanziiert, dass alle Constraints des mit $G[V_c]$ assoziierten Constraint-Netzes $\mathcal{C}_c$ erfüllt sind, so kann man effizient testen, ob sich diese Teillösung zu einer vollständigen Lösung ausbauen lässt.
- Bezeichne $\mathcal{C}_T$ das mit $G_T$ assoziierte Constraint-Netz, und sei $n = |V|, m = |V_c|$ und $d_T = \max\{|D_i| \mid x_i$ aus $\mathcal{C}_T\}$. Dann gelingt die Lösung von $\mathcal{C}_T$ in $\mathcal{O}((n-m) \cdot d_T^2)$ Schritten. Warum?
- Heuristik: Je kleiner der Cycle-Cut-Set, desto effizienter ist die Lösungsfindung.
- Die Unterscheidung von $d_T$ und $d_c$ zeigt, dass nicht allein die Größe eines Cycle-Cut-Sets interessant ist.
- Es existiert kein effizienter Algorithmus zur Bestimmung eines minimalen Cycle-Cut-Sets.

# Weitere Analyseverfahren

## Constraint-Netz-Reformulierung: Cycle-Cut-Set-Analyse

Variante [Dechter/Pearl 1988]:

1. Es werden eine Reihe von Cycle-Cut-Sets $V_c$ konstruiert. Hierbei ist die Vorgabe eines Maximums für $|V_c|$ ist sinnvoll.
2. Die resultierenden Constraint-Netze $\mathcal{C}_T$ werden bzgl. der Anzahl ihrer Lösungen untersucht.
3. Heuristik: Je mehr Lösungen ein Constraint-Netz $\mathcal{C}_T$ hat, um so einfacher ist es zu lösen, auch wenn die Variablen des Cycle-Cut-Sets vorbelegt sind.

   ➜ Wähle Cycle-Cut-Set, der zu dem Constraint-Netz $\mathcal{C}_T$ mit der größten Anzahl von Lösungen führt.