

Pohled zpět

Přehled algoritmů pohledu zpět *look-back*

● Algoritmy skoku zpět (*backjumping*)

- při zpětném průchodu se nevracíme pouze jeden krok jako algoritmus backtrackingu
- snažíme se vracet co nejdále až ke zdroji chyby

● Algoritmy učení (*constraint recording, no-good learning*)

- *no-good* = chybné částečné přiřazení
- přidáme nová omezení, která zakazují nalezená chybná přiřazení

● Dynamický backtracking

- při skoku zpět se snažíme nezapomínat už udělanou práci
- měníme hodnoty pouze u minulých proměnných s konfliktem (ostatní neměníme)
- realizace: změním uspořádání proměnných

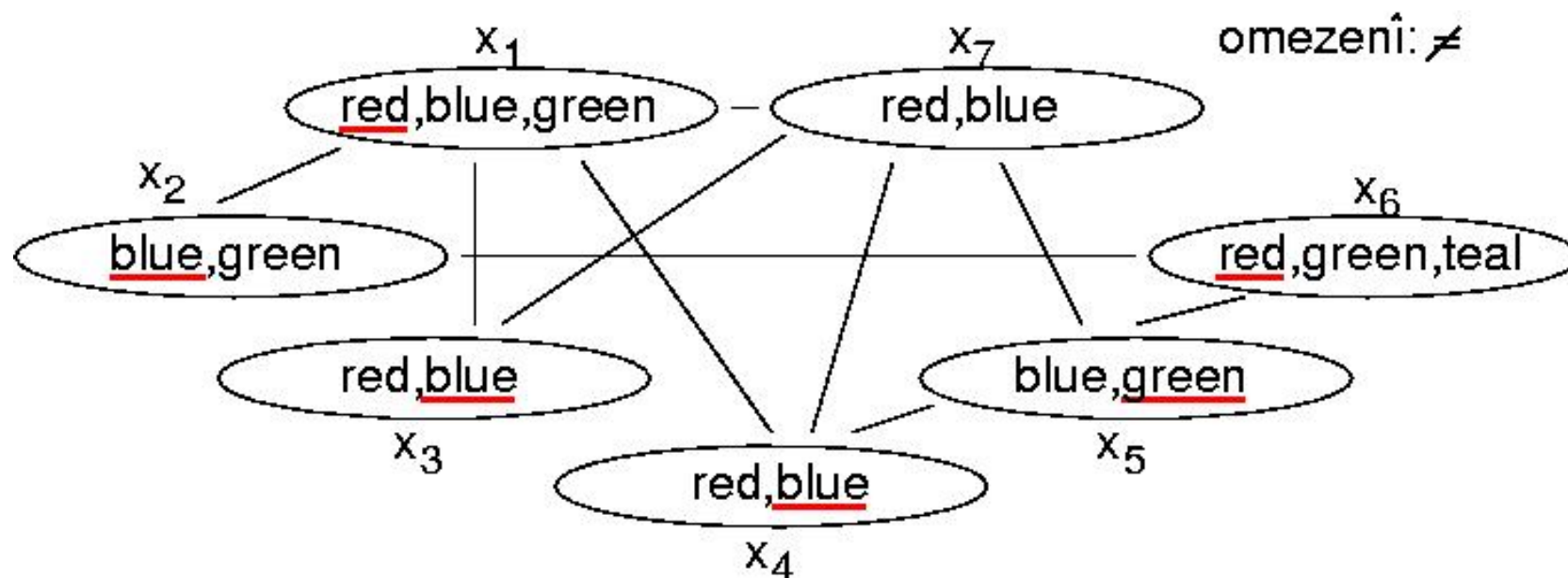
● Backmarking

- pamatuje si, kde testy na konzistenci neuspěly
- eliminuje opakování dříve provedených konzistenčních testů

Konfliktní množina

pevné uspořádání proměnných (x_1, \dots, x_n)

- Mějme částečné konzistentní přiřazení $\vec{a} = (a_{i_1}, \dots, a_{i_k})$ libovolné podmnožiny proměnných a uvažujme dosud nepřiřazenou proměnnou x . Jestliže neexistuje hodnota b z domény x tak, aby $(\vec{a}, x = b)$ bylo konzistentní, říkáme, že \vec{a} je **konfliktní množina** x a nebo, že \vec{a} je **v konfliktu s** x .

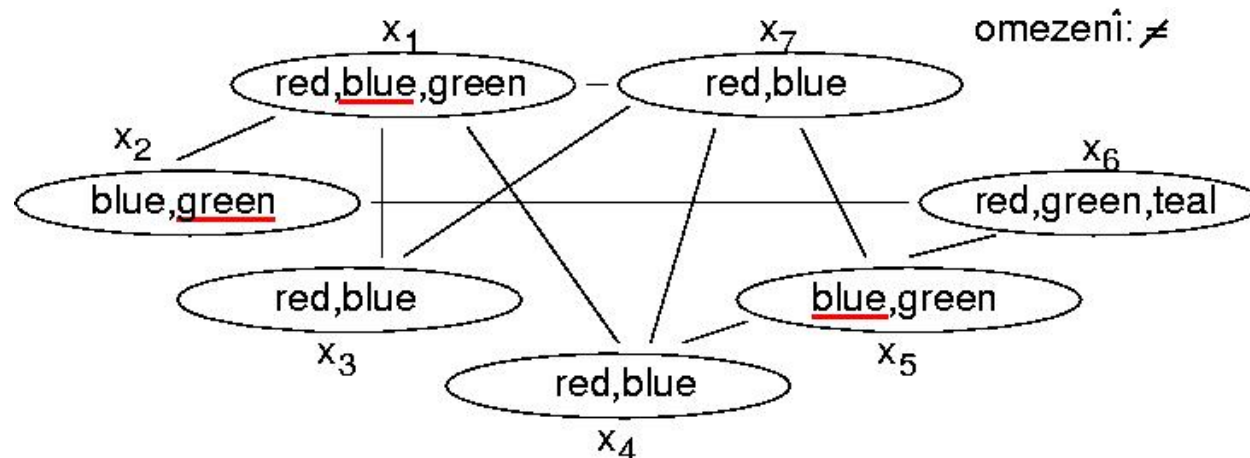


- vyznačené přiřazení je konfliktní množina vzhledem k x_7
- Pokud \vec{a} neobsahuje j -tici ($j < k$), která je v konfliktu s x , pak nazýváme \vec{a} **minimální konfliktní množina**. $\{x_1/red, x_3/blue\}$: min.konf.množina vzhledem k x_7

Chybné přiřazení

- Mějme částečné konzistentní přiřazení $\vec{a}_{i-1} = (a_1, \dots, a_{i-1})$. Jestliže je \vec{a}_{i-1} v konfliktu s x_i , pak ho nazýváme **list slepé větve**.
 - přiřazení v předchozím příkladu je list slepé větve
- Částečné přiřazení \vec{a} , které se nevyskytuje v žádném řešení CSP, se nazývá **chybné přiřazení (no-good)**.

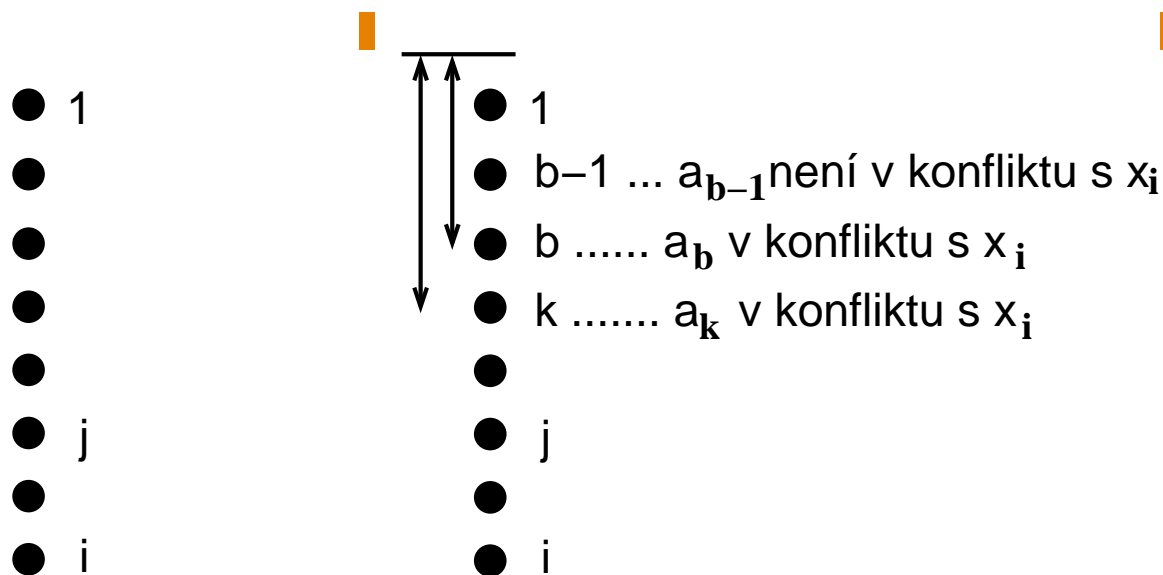
- konfliktní množiny jsou chybná přiřazení



- $[x_1/\text{blue}, x_2/\text{green}, x_5/\text{blue}]$ je chybné přiřazení, ale nepatří do žádné konfliktní množiny
- konfliktní množina = chybné přiřazení vzhledem k jediné proměnné
- **Minimální chybná přiřazení** neobsahují chybná přiřazení méně proměnných.

Konfliktní proměnná

- Mějme list slepé větve \vec{a}_{i-1} . Proměnná x_j ($j < i - 1$) je **bezpečná**, pokud je \vec{a}_j chybné přiřazení.
 - také říkáme, že skok z x_i na x_j je bezpečný nevynecháme žádné řešení
- Mějme list slepé větve \vec{a}_{i-1} . Proměnná x_b je **konfliktní proměnná (culprit)** vzhledem k \vec{a}_{i-1} , jestliže $b = \min\{k < i \mid \vec{a}_k \text{ v konfliktu s } x_i\}$.

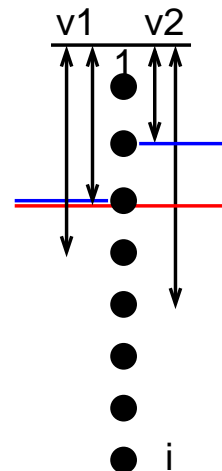


● **Tvrzení:** Skok ke konfliktní proměnné je bezpečný.

Gaschnigův skok zpět

● *Gaschnig's backjumping (GBJ)*

- když nedokážeme přiřadit hodnotu proměnné x_i ,
vracíme se zpět (skáčeme zpět) ke konfliktní proměnné
- určení konfliktní proměnné
 - pro každou hodnotu $v_i \in x_i$ nalezneme **proměnnou s nejmenším indexem**,
která je s x_i/v_i v konfliktu
(přiřazení této proměnné musíme určitě změnit, aby šla hodnota v_i použít)
 - **konfliktní proměnná** je ta z nich, která má největší index
(když její hodnotu změníme, můžeme zrušit konflikt s odpovídající hodnotou)



Algoritmus GBJ

Každá proměnná x_i uchovává v $latest_i$ index konfliktní proměnné

procedure GBJ((X, D, C))

rozdíly od backtrackingu

$i := 1$ (inicializace čítače proměnných)

$D'_i := D_i$ (kopírování domény)

$latest_i := 0$ (inicializace čítače na konfliktní proměnnou)

while $1 \leq i \leq n$

přiřazení $x_i := \text{Select-Value-GBJ}$

if x_i is null (žádná hodnota nebyla vrácena)

$i := latest_i$ (skok zpět)

else $i := i + 1$ (dopředná fáze)

$D'_i := D_i$

$latest_i := 0$

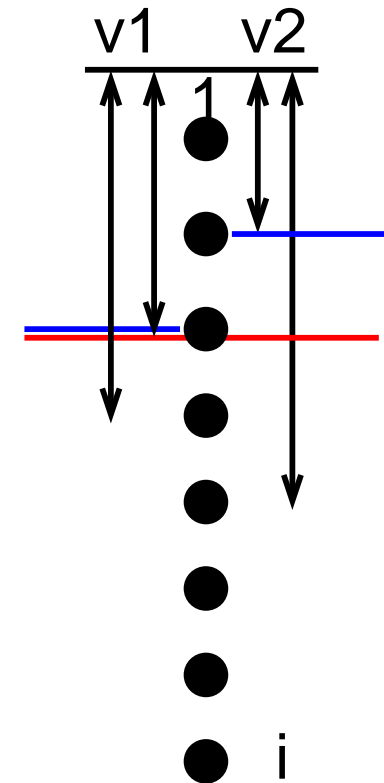
if $i = 0$ return „nekonzistentní“

else return přiřazené hodnoty $\{x_1, \dots, x_n\}$

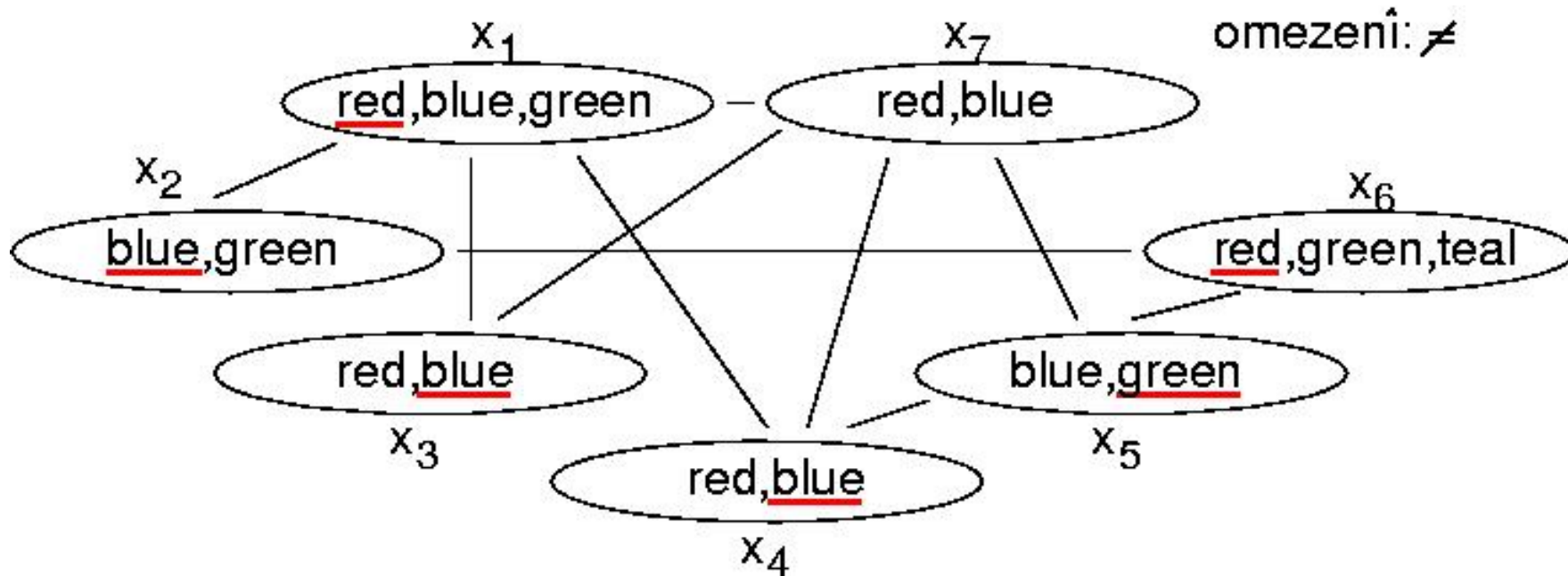
end GBJ

GBJ: výběr hodnoty

```
procedure Select-Value-GBJ
while  $D'_i$  is not empty
  vyber a smaž libovolný  $v \in D'_i$ 
  consistent := true
   $k := 1$ 
  while  $k < i \wedge$  consistent
    if  $k > latest_i$ 
       $latest_i := k$ 
    if not Consistent( $\vec{v}_k, x_i = v$ )
      consistent := false
    else  $k := k+1$ 
  if consistent
    return  $v$ 
return null          (v doméně  $x_i$  neexistuje konzistentní hodnota)
```



GBJ: příklad



- vyznačené přiřazení zároveň říká, že předchozí hodnoty už jsme vyzkoušeli (a vyřadili)
- $x_7 = \text{red}$ vyřadí x_1 (tj. $latest_7 = 1$), $x_7 = \text{blue}$ vyřadí $x_3 \Rightarrow$ celkem $latest_7 = 3$
- vracíme se ke konfliktní proměnné x_3 , ta už má ale prázdnou doménu
- $latest_3 = 2$, vracíme se tedy k x_2
 - $x_3 = \text{red}$ dala $latest_3$ na 1, $x_3 = \text{blue}$ dala $latest_3$ na 2
- lépe (až k x_1) se ale vrátit neumíme, GBJ se v tomto okamžiku vrací k předchozí proměnné

Konflikty řízený skok zpět

Conflict-directed backjumping CBJ

- Při skoku zpět na proměnnou x_j z listu slepé větve nemusíme nalézt v doméně x_j žádné hodnoty pro přiřazení. \vec{a}_{j-1} se pak nazývá **vnitřní slepá větev**.■
- CBJ skáče zpět v listu slepé větve i ve vnitřní slepé větvi
 - udržujeme J_i **množinu skoků zpět** pro každou proměnnou pomocí nesplněných omezení
 - proměnná $x_j (j < i)$ je **bližší** k x_i než $x_k (k < i)$, jestliže $j > k$, naopak x_k je **vzdálenější**
 - omezení R je **vzdálenější než** omezení S , jestliže je nejbližší proměnná v rozsahu R vzdálenější než nejbližší proměnná v rozsahu S (pokud totožné proměnné, rozhodují další proměnné). Pro uspořádání (x_1, x_2, \dots) :
 - rozsah $R_1: (x_3, x_5, x_7)$, rozsah $S_1: (x_2, x_6, x_8, x_9)$ R_1 je vzdálenější než S_1 od x_{10} ■
 - rozsah $R_2: (x_3, x_5, x_8, x_9)$, rozsah $S_2: (x_2, x_6, x_8, x_9)$ R_2 je vzdálenější než S_2 od x_{10} ■
 - mezi nesplněnými omezeními vybereme to nejvzdálenější (ostatní omezení neumožní nejdelší možný skok zpět)
 - skočíme zpět na nejbližší proměnnou v tomto omezení (je bezpečné změnit proměnnou, která byla nejpozději přiřazená)

CBJ: výběr hodnoty

```
procedure Select-Value-CBJ
```

```
while  $D'_i$  is not empty
```

```
    vyber a smaž libovolný  $v \in D'_i$ 
```

```
    consistent := true
```

```
     $k := 1$ 
```

```
    while  $k < i \wedge$  consistent
```

```
        if Consistent( $\vec{v}_k, x_i = v$ )
```

```
             $k := k + 1$ 
```

```
        else  $R :=$  nejvzdálenější nesplněné omezení
```

```
            přidej proměnné v rozsahu  $R$  vyjma  $x_i$  do  $J_i$ 
```

```
            consistent := false
```

```
    if consistent
```

```
        return  $v$ 
```

```
return null
```

(v doméně x_i neexistuje konzistentní hodnota)

Algoritmus CBJ

procedure CBJ((X, D, C))

$i := 1$

$D'_i := D_i$

$J_i := \emptyset$

while $1 \leq i \leq n$

přiřazení $x_i := \text{Select-Value-}$ CBJ

if x_i is null

$ip := i$

$i := \text{index poslední proměnné v } J_i$

$J_i := J_i \cup J_{ip} - \{x_i\}$

(takto upravená J_i se použije ve vnitřní slepé větvi)

else $i := i + 1$

$D'_i := D_i$

$J_i := \emptyset$

if $i = 0$ return „nekonzistentní“

else return přiřazené hodnoty $\{x_1, \dots, x_n\}$

end CBJ

rozdíly od backtrackingu

(inicializace čítače proměnných)

(kopírování domény)

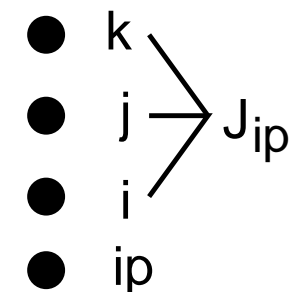
(inicializace množiny skoků zpět)

(žádná hodnota nebyla vrácena)

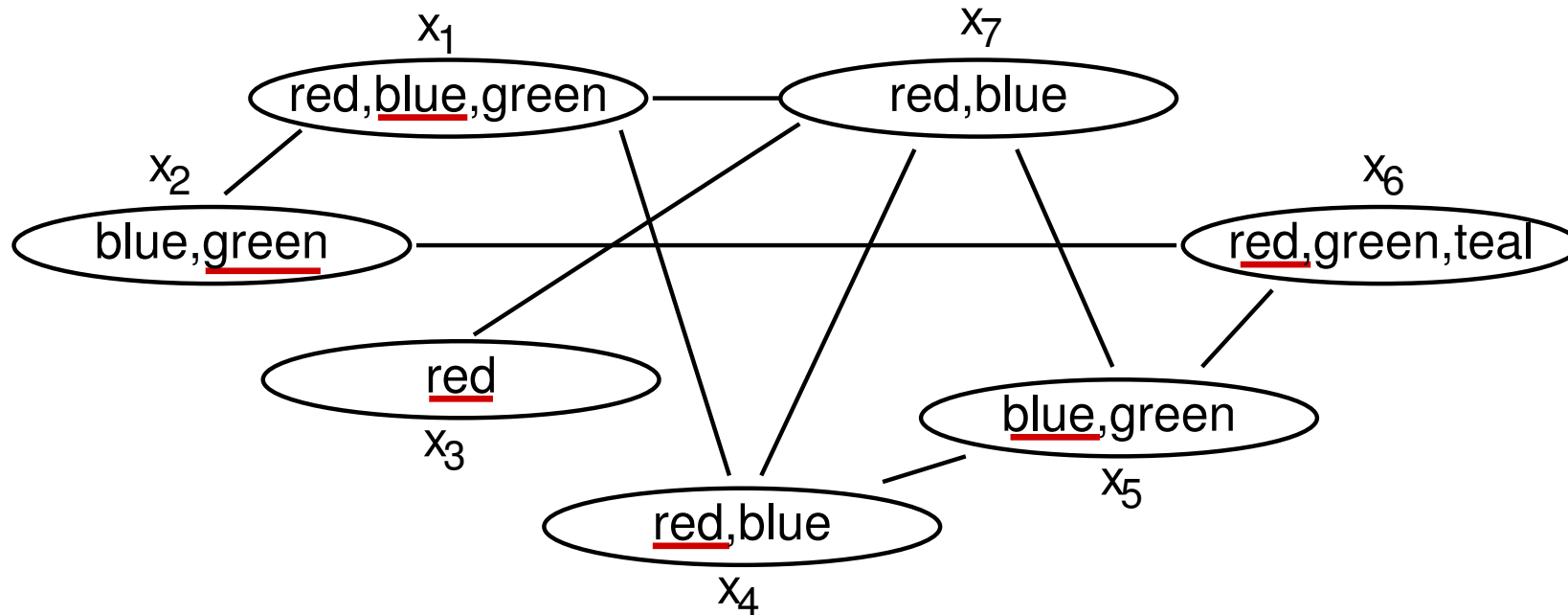
(skok zpět)

(spoj množiny skoku zpět)

(dopředná fáze)



CBJ: příklad



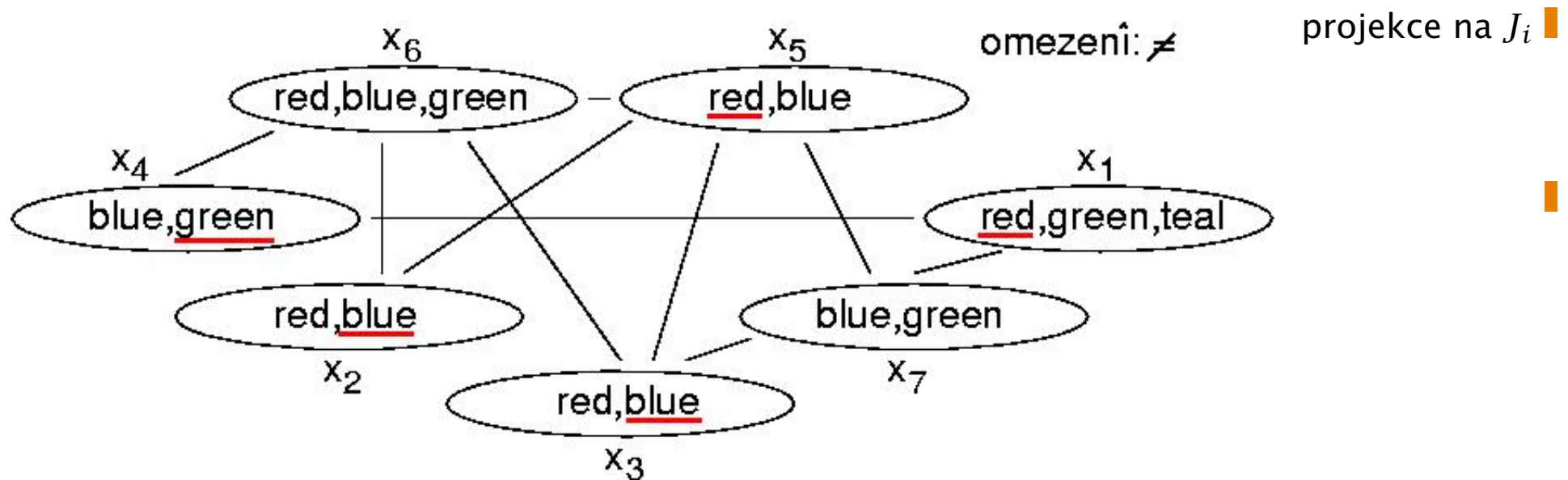
- před vstupem do Select-Value-CBJ pro proměnnou x_7 je $J_7 = \emptyset$
- $x_3 \neq x_7$ je nejvzdálenější omezení, které vyřadilo $x_7 = \text{red}$, tedy $J_7 = \{x_3\}$
- $x_1 \neq x_7$ je nejvzdálenější omezení, které vyřadilo $x_7 = \text{blue}$, tedy $J_7 = \{x_1, x_3\}$
- skáče zpět na poslední proměnnou v J_7 , tedy na x_3
- do J_3 , která byla zatím prázdná, přidáme x_1
- doména x_3 je prázdná, v J_3 je jediná proměnná x_1 a na tu se vracíme

Algoritmy učení

- Množiny skoků zpět jsou chybná přiřazení vypočítaná během prohledávání
- Tato chybná přiřazení se mohou vyskytovat i později v jiných cestách stromu prohledávání a jsou znovu počítána
- Přidáme chybná přiřazení jako nová omezení při detekci slepé větve
 - nemá cenu uchovávat celou slepou větev \vec{a}_i v proměnné x_{i+1}
na toto přiřazení už později nenarazíme
 - uchováme chybná přiřazení na podmnožině proměnných $\{x_1, \dots, x_i\}$
- Prořezávání stavového prostoru
 - čím menší chybná přiřazení se podaří uchovat, tím rychlejší bude prohledávání
- Zvětšování množiny omezení
 - cena za prořezávání stavového prostoru nesmí přerůst cenu za zvětšování množiny omezení

Učení skoku zpět (*jumpback learning*)

- Využijeme chybná přiřazení, která jsme se naučili v CBJ
- Algoritmus učení skoku zpět \equiv CBJ + přidávání nových omezení
- Po dosažení listu slepé větve \vec{a}_{i-1} přidáme omezení zakazující $\vec{a}_{i-1}[J_i]$



- po dosažení listu slepé větve v x_6 je $J_6 = \{x_2, x_4, x_5\}$
 \Leftarrow red vyřadila $x_6 \neq x_5$, blue vyřadila $x_6 \neq x_2$, green vyřadila $x_6 \neq x_4$
- $\vec{a}_5[J_6]$ tedy zakazuje přiřazení [$\langle x_2, \text{blue} \rangle$, $\langle x_4, \text{green} \rangle$, $\langle x_5, \text{red} \rangle$]
- později při procházení stromu lze např. ze znalosti $x_2 = \text{blue}$, $x_4 = \text{green}$ odvodit $x_5 \neq \text{red}$