

Prohledávání a pohled dopředu

Přehled prohledávacích algoritmů pro CSP

● Rozšiřování částečného konzistentního přiřazení

- stromové prohledávání
 - backtracking
 - pohled dopředu (propagace omezení)
 - pohled zpět (inteligentní vracení)
 - neúplná stromová prohledávání

Přehled prohledávacích algoritmů pro CSP

● Rozšiřování částečného konzistentního přiřazení

- stromové prohledávání
 - backtracking
 - pohled dopředu (propagace omezení)
 - pohled zpět (inteligentní vracení)
 - neúplná stromová prohledávání

● Procházení úplných nekonzistentních přiřazení

- generuj a testuj
- lokální prohledávání

Přehled prohledávacích algoritmů pro CSP

● Rozšiřování částečného konzistentního přiřazení

- stromové prohledávání
 - backtracking
 - pohled dopředu (propagace omezení)
 - pohled zpět (inteligentní vracení)
 - neúplná stromová prohledávání

● Procházení úplných nekonzistentních přiřazení

- generuj a testuj
- lokální prohledávání

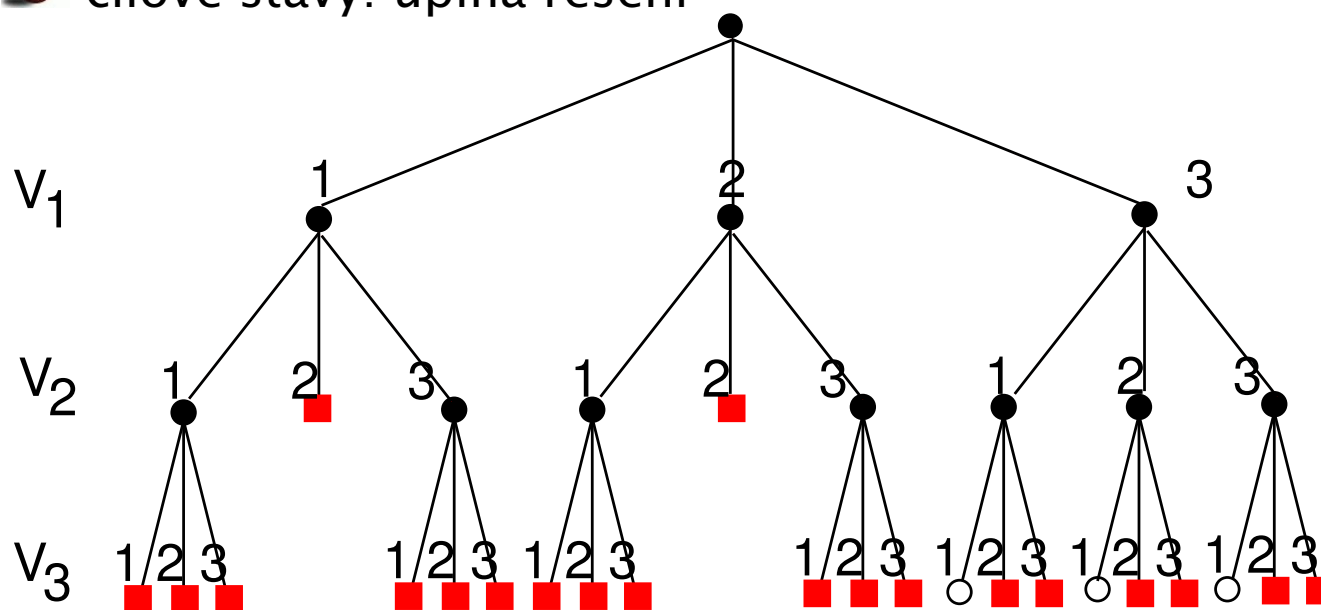
● Kombinování úplných nekonzistentních přiřazení

- *population-based search*

Prohledávací algoritmy pro CSP

Prohledávací algoritmy prochází (traversálně) **graf stavového prostoru**

- uzly grafu (stavy): konzistentní částečné instanciaci
- iniciální stav: prázdné přiřazení
- hrany grafu: operátory, které rozšíří částečné řešení $[x_1/a_1, \dots, x_j/a_j]$ o přiřazení jedné proměnné, které není v konfliktu s dřívějšími přiřazeními, tj. vznikne $[x_1/a_1, \dots, x_j/a_j, x_{j+1}/a_{j+1}]$
- cílové stavy: úplná řešení



- červené čtverečky: chybný pokus o instanciaci, řešení neexistuje
- nevyplněná kolečka: nalezeno řešení (cílové stavy)
- černá kolečka: vnitřní uzel, máme pouze částečné přiřazení

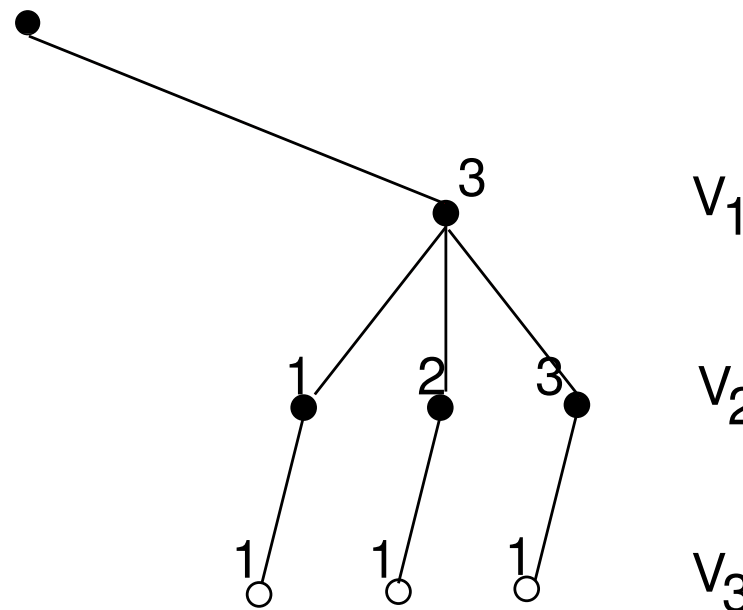
Prohledávací algoritmy pro CSP: vlastnosti

Bod volby: z uzlu grafu vede více hran

• máme na výběr, kterou hodnotu přiřadíme proměnné

CSP má **řešení bez navracení** vzhledem k uspořádání d , jestliže všechny listy v jeho grafu stavového prostoru reprezentují řešení.

• v grafu nejsou žádné slepé větve



Backtracking (BT)

- **Prohledávání stavového prostoru do hloubky**
- Dvě fáze backtrackingu
 - **dopředná fáze**: proměnné jsou postupně vybírány, rozšiřuje se částečné řešení přiřazením konzistentní hodnoty (pokud existuje) pro další proměnnou
 - **zpětná fáze**: pokud neexistuje konzistentní hodnota pro aktuální proměnnou algoritmus se vrací k předchozí přiřazené hodnotě

Backtracking (BT)

- **Prohledávání stavového prostoru do hloubky**
- Dvě fáze backtrackingu
 - **dopředná fáze**: proměnné jsou postupně vybírány, rozšiřuje se částečné řešení přiřazením konzistentní hodnoty (pokud existuje) pro další proměnnou
 - **zpětná fáze**: pokud neexistuje konzistentní hodnota pro aktuální proměnnou algoritmus se vrací k předchozí přiřazené hodnotě
- Proměnné dělíme na
 - **minulé** – proměnné, které už byly vybrány (a mají přiřazenu hodnotu)
 - **aktuální** – proměnná, která je právě vybrána a je jí přiřazována hodnota
 - **budoucí** – proměnné, které budou vybrány v budoucnosti

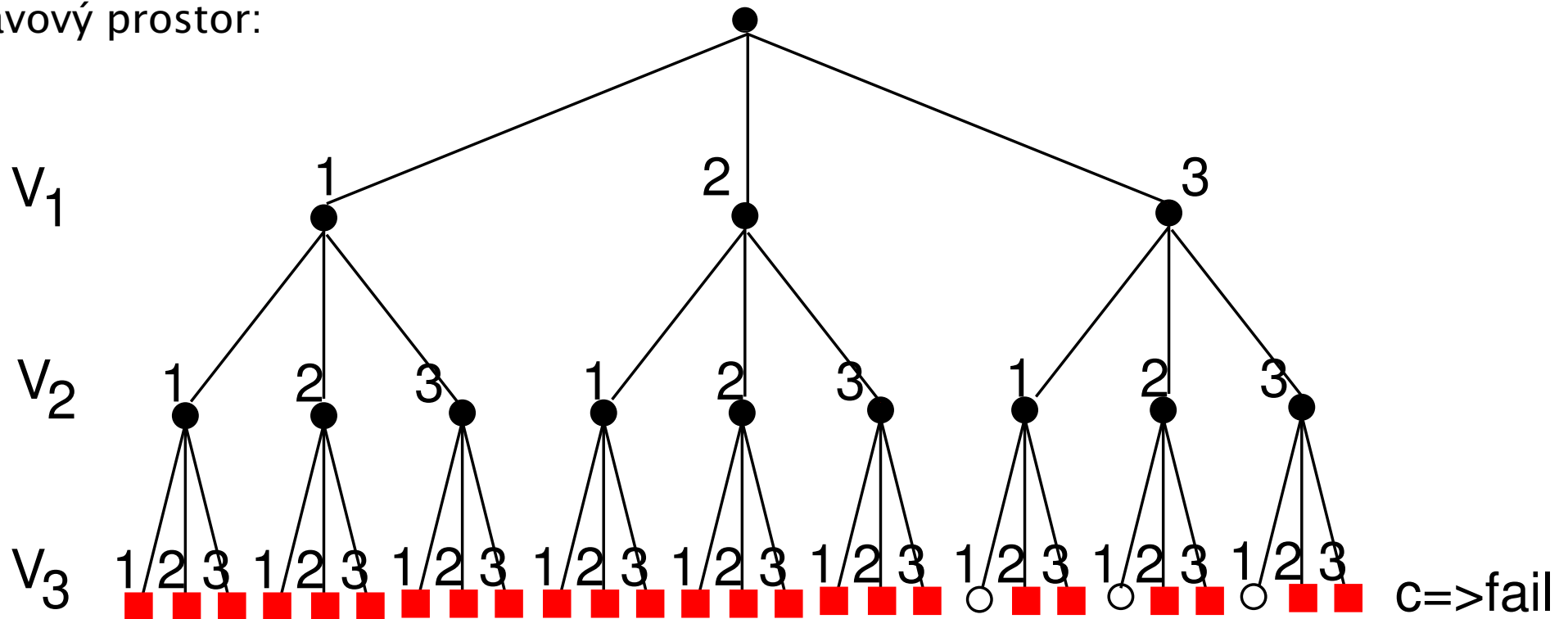
Příklad: backtracking

- Příklad: $V_1, V_2, V_3 \in \{1, 2, 3\}$, $c : V_1 = 3 \times V_3$
- Stavový prostor:

Příklad: backtracking

● Příklad: $V_1, V_2, V_3 \in \{1, 2, 3\}$, $c : V_1 = 3 \times V_3$

● Stavový prostor:



Příklady vizualizací viz <http://www.fi.muni.cz/~hanka/vis>

● zpětná vazba k této bakalářská práci?

● viz kontakt na stránkách

Algoritmus backtrackingu

```
procedure Backtracking( $(X, D, C)$ )  
   $i := 1$  (inicializace čítače proměnných)  
   $D'_i := D_i$  (kopírování domény)  
  while  $1 \leq i \leq n$   
    přiřazení  $x_i := \text{Select-Value}$   
    if  $x_i$  is null (žádná hodnota nebyla vrácena)  
       $i := i - 1$  (zpětná fáze)  
    else  $i := i + 1$  (dopředná fáze)  
       $D'_i := D_i$   
  if  $i = 0$  return „nekonzistentní“  
  else return přiřazené hodnoty  $\{x_1, \dots, x_n\}$   
end Backtracking
```

- Algoritmus vrací pouze první řešení

- Série domén D'_i : $\forall i : D'_i \subseteq D_i$.

Select-Value pracuje nad (promazává) D'_i

Hodnoty v D'_i ještě netestovány vzhledem k aktuálnímu částečnému přiřazení

Uspořádání hodnot pro backtracking

- procedure Select-Value
 - while D'_i is not empty
 - vyber a smaž libovolný $a \in D'_i$
 - if Consistent($\vec{a}_{i-1}, x_i = a$) return a
 - return null
- Backtracking ověřuje v každém kroku konzistenci podmínek vedoucích z minulých proměnných do aktuální proměnné
- Backtracking tedy zajišťuje konzistenci podmínek
 - na všech minulých proměnných
 - na podmínkách mezi minulými proměnnými a aktuální proměnnou

Problémy a vylepšení backtrackingu

- **Thrashing**: opakované objevování stejných nekonzistencí a částečných úspěchů při prohledávání

Problémy a vylepšení backtrackingu

- **Thrashing**: opakované objevování stejných nekonzistencí a částečných úspěchů při prohledávání
- **Algoritmy pohledu dopředu** kontrolují podmínky dopředu
 - backtracking odhalí nesplnění podmínky teprve když přiřadí hodnoty jejím proměnným
příklad: $A, B, C, D, E \in 1..10, A=3 * E$
konzistenčními algoritmy lze předem upravit domény A a E

Problémy a vylepšení backtrackingu

- **Thrashing**: opakované objevování stejných nekonzistencí a částečných úspěchů při prohledávání
- **Algoritmy pohledu dopředu** kontrolují podmínky dopředu
 - backtracking odhalí nesplnění podmínky teprve když přiřadí hodnoty jejím proměnným
příklad: $A, B, C, D, E \in 1..10, A=3 * E$
konzistenčními algoritmy lze předem upravit domény A a E
- **Backjumping** se vrací k původci chyby
 - příklad: $A, B, C, D, E \in 1..10, A > E$
backtracking vyzkouší všechny možnosti pro B, C, D než odhalí $A \neq 1$
hned po prvním neúspěšném přiřazení E se lze vrátit k přiřazování A

Problémy a vylepšení backtrackingu

- **Thrashing**: opakované objevování stejných nekonzistencí a částečných úspěchů při prohledávání
- **Algoritmy pohledu dopředu** kontrolují podmínky dopředu
 - backtracking odhalí nesplnění podmínky teprve když přiřadí hodnoty jejím proměnným
příklad: $A, B, C, D, E \in 1..10, A=3 * E$
konzistenčními algoritmy lze předem upravit domény A a E
- **Backjumping** se vrací k původci chyby
 - příklad: $A, B, C, D, E \in 1..10, A > E$
backtracking vyzkouší všechny možnosti pro B, C, D než odhalí $A \neq 1$
hned po prvním neúspěšném přiřazení E se lze vrátit k přiřazování A
- **Dynamický backtracking**: změna uspořádání minulých proměnných
- **Neúplná prohledávání**: hledání pouze v některých částech stavového prostoru

Algoritmy pohledu dopředu (*look-ahead*) LA

- Používají propagace omezení
- Vyvolány před přiřazováním hodnoty další proměnné
- Snaží se zjistit, jak rozhodnutí o výběru proměnných a hodnot ovlivní budoucí prohledávání
- Po provedení propagace omezení lze mnohem lépe
 - rozhodnout, **kteřou proměnnou přiřadit**
 - většinou lepší přiřadit proměnné, které nejvíce omezují zbytek stavového prostoru
 - příklad: $A, B, C, D, E \in 1..10$, $D=A*B*C*E$, $E>5$, lépe je začít s E

Algoritmy pohledu dopředu (*look-ahead*) LA

- Používají propagace omezení
- Vyvolány před přiřazováním hodnoty další proměnné
- Snaží se zjistit, jak rozhodnutí o výběru proměnných a hodnot ovlivní budoucí prohledávání
- Po provedení propagace omezení lze mnohem lépe
 - rozhodnout, **kteřou proměnnou přiřadit**
 - většinou lepší přiřadit proměnné, které nejvíce omezují zbytek stavového prostoru
 - příklad: $A, B, C, D, E \in 1..10$, $D=A*B*C*E$, $E>5$, lépe je začít s E
 - rozhodnout, **kteřou hodnotu přiřadit proměnné**
 - snaha o výběr hodnoty, která umožní nejvíce voleb v budoucích přiřazeních
 - příklad: $A, B, C \in 1..10$, $A*B<10$, $B=C*2$, pro A je lepší vybrat 1
- Vylepšení složitosti nejhoršího případu oproti naivnímu backtrackingu zřídka. Cílem je snaha o efektivní využití algoritmů propagace omezení.

Pohled dopředu pro výběr hodnoty

procedure Look-Ahead((X, D, C))

rozdíly od backtrackingu

$i := 1$ (inicializace čítače proměnných)

$D'_i := D_i$ pro $1 \leq i \leq n$ (kopírování domény)

while $1 \leq i \leq n$

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

if x_i is null (žádná hodnota nebyla vrácena)

$i := i - 1$ (zpětná fáze)

nastav všechny $D'_k, k > i$ na jejich hodnotu před poslední instanciací x_i

else $i := i + 1$ (dopředná fáze)

if $i = 0$ return „nekonzistentní“

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

end Look-Ahead

● Select-Value-XXX se liší dle typu LA algoritmu

● Ukládáno n kopií každé D'

● pro každou úroveň ve stavovém prostoru jedna kopie

Kontrola dopředu (*forward checking*) FC

```
● procedure Select-Value-Forward-Checking
  while  $D'_i$  is not empty
    vyber a smaž libovolný  $a \in D'_i$ 
    for  $\forall k, i < k \leq n$ 
      for  $\forall b \in D'_k$ 
        if not Consistent( $\vec{a}_{i-1}, x_i = a, x_k = b$ )
          smaž  $b$  z  $D'_k$ 
    if  $\exists k, i < k \leq n: D'_k$  is empty ( $x_i = a$  vede do slepé větve, nevybírej  $a$ )
      nastav všechny  $D'_k, i < k \leq n$  na hodnotu před výběrem  $a$ 
    else return  $a$ 
  return null
```

● FC je rozšíření backtrackingu

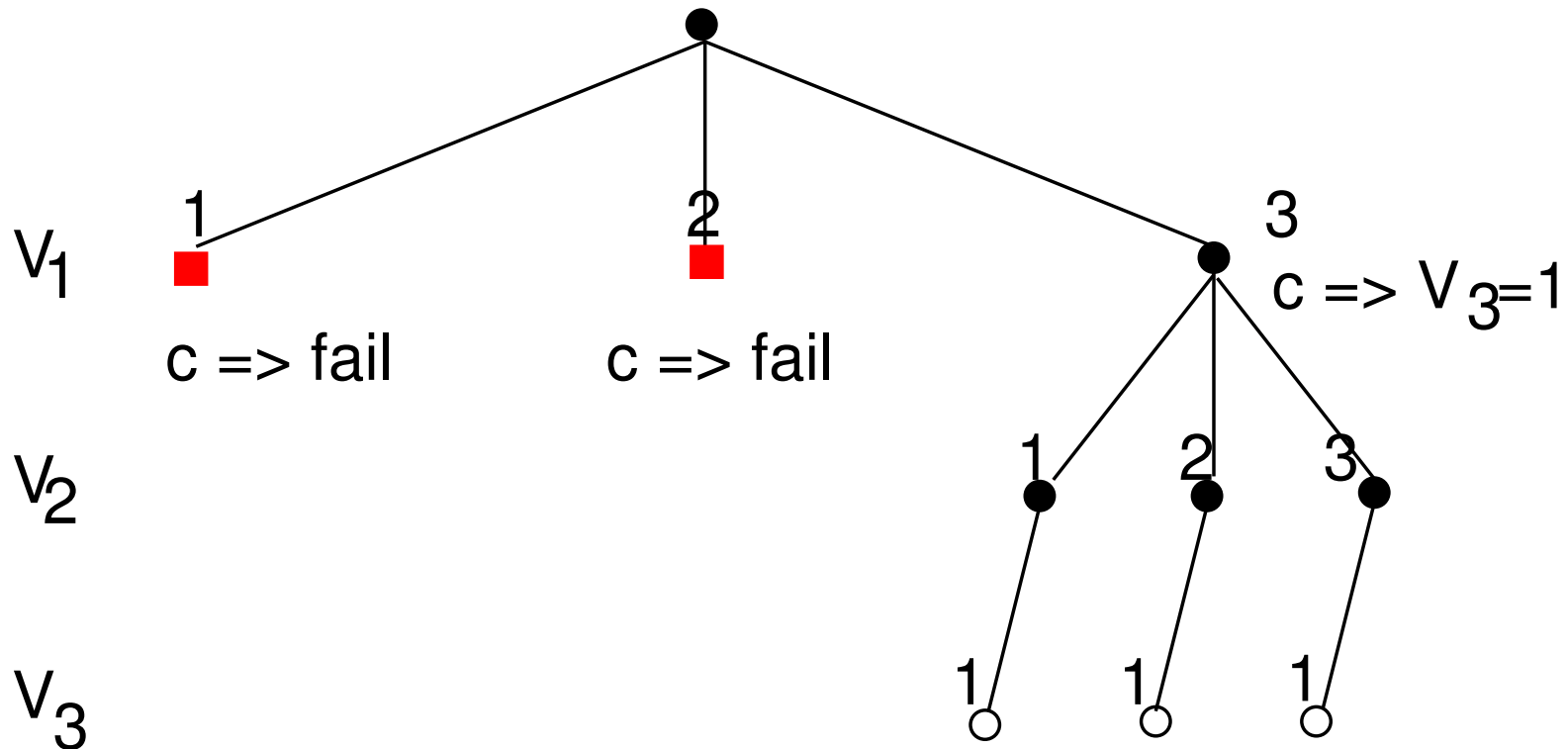
● FC navíc zajišťuje konzistenci mezi aktuální proměnnou a budoucími proměnnými, které jsou s ní spojeny dosud nesplněnými podmínkami

● Hrany z aktuální proměnné do minulých proměnných není nutno testovat

Příklad: kontrola dopředu

● Příklad: $V_1, V_2, V_3 \in \{1, 2, 3\}$, $c : V_1 = 3 \times V_3$

● Stavový prostor:



Opravdový úplný pohled dopředu (RFLA)

procedure Select-Value-Look-Ahead

while D'_i is not empty

vyber a smaž libovolný $a \in D'_i$

repeat Deleted := false

for $\forall j, i < j \leq n$

for $\forall k, i < k \leq n$

for $\forall b \in D'_j$

if neexistuje $c \in D'_k$ taková, že

Consistent($\vec{a}_{i-1}, x_i = a, x_j = b, x_k = c$)

smaž b z D'_j

Deleted := true

until Deleted = false

(Real Full) Look-Ahead, RFLA n. LA
n. Maintaining Arc-Consistency MAC

Implementace AC-1 algoritmu

Lze nahradit silnějšími AC algoritmy

Opravdový úplný pohled dopředu (RFLA)

procedure Select-Value-Look-Ahead

while D'_i is not empty

vyber a smaž libovolný $a \in D'_i$

repeat Deleted := false

for $\forall j, i < j \leq n$

for $\forall k, i < k \leq n$

for $\forall b \in D'_j$

if neexistuje $c \in D'_k$ taková, že

Consistent($\vec{a}_{i-1}, x_i = a, x_j = b, x_k = c$)

smaž b z D'_j

Deleted := true

until Deleted = false

if $\exists k, i < k \leq n$ takové, že $D_k = \emptyset$ (nevybírej a)

nastav všechny $D'_j, i < j \leq n$ na hodnotu před výběrem a

else return a

return null

(Real Full) Look-Ahead, RFLA n. LA

n. Maintaining Arc-Consistency MAC

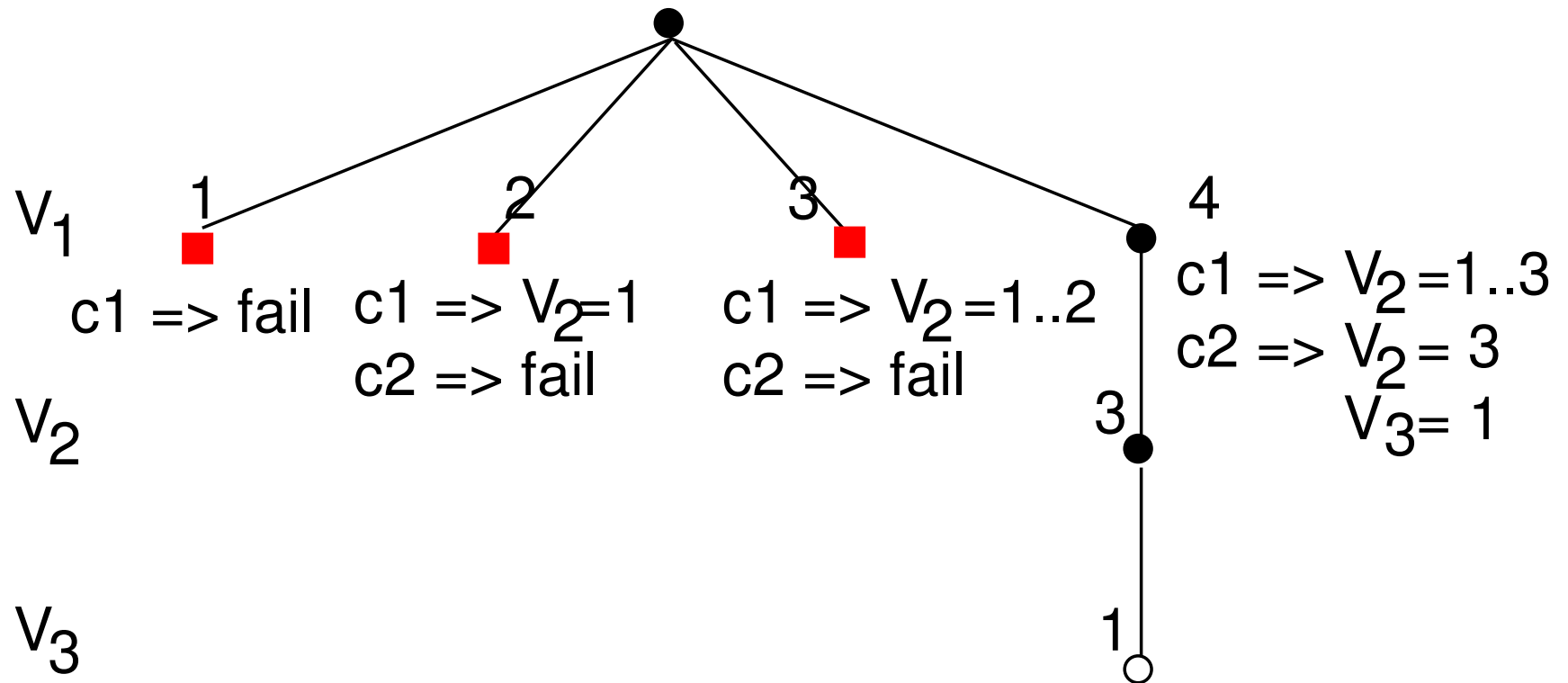
Implementace AC-1 algoritmu

Lze nahradit silnějšími AC algoritmy

Příklad: pohled dopředu

● Příklad: $V_1, V_2, V_3 \in \{1, 2, 3\}$, $c1 : V_1 > V_2$, $c2 : V_2 = 3 \times V_3$

● Stavový prostor:



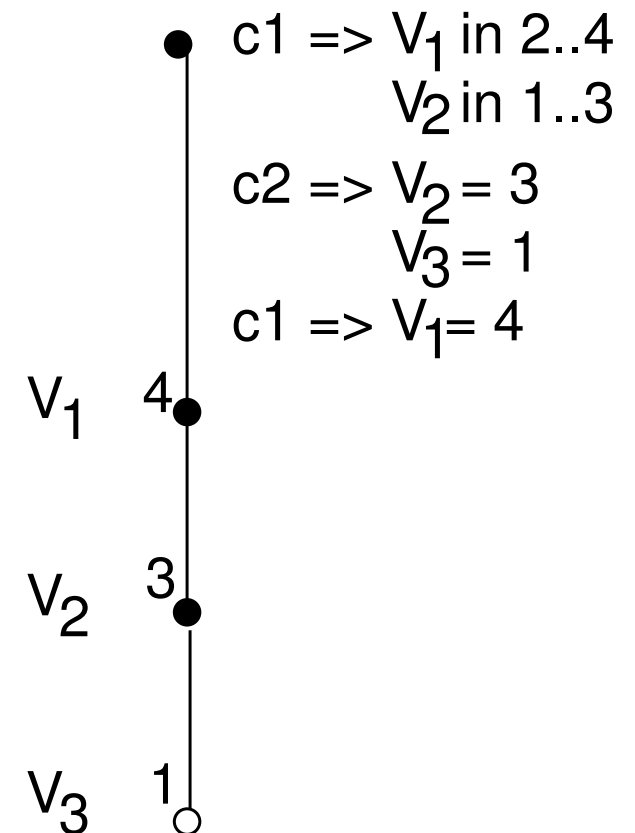
Prohledávání s iniciální konzistencí

Prohledávací algoritmus často aplikován až po **zajištění iniciální konzistence**

- typicky: iniciální konzistence a pak kontrola dopředu nebo
iniciální konzistence a pak pohled dopředu

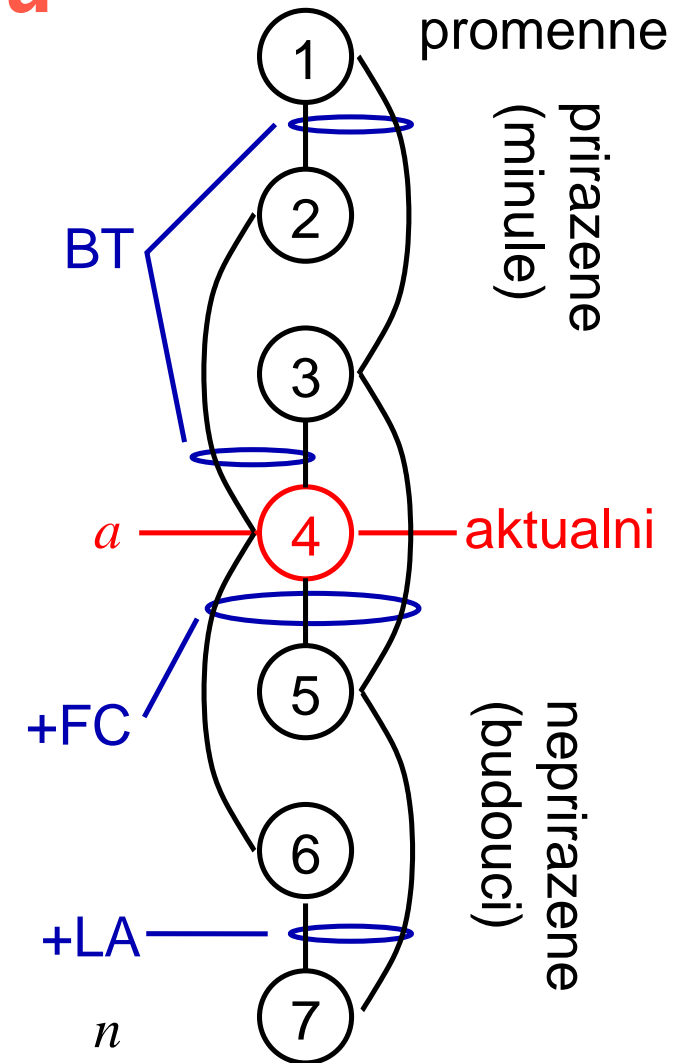
Příklad:

- pohled dopředu + iniciální konzistence
- V_1, V_2, V_3 in $1 \dots 4$
 $c1 : V_1 > V_2, \quad c2 : V_2 = 3 \times V_3$



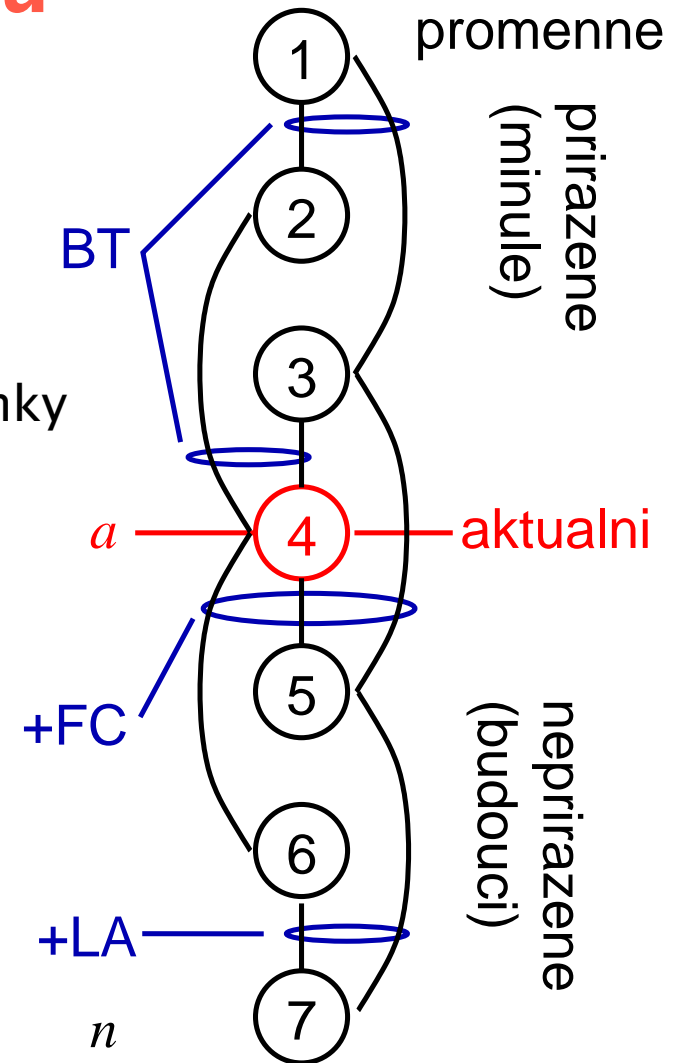
Srovnání algoritmů

- **Backtracking (BT)** kontroluje v kroku a podmínky $c(V_1, V_a), \dots, c(V_{a-1}, V_a)$ z minulých proměnných do aktuální proměnné



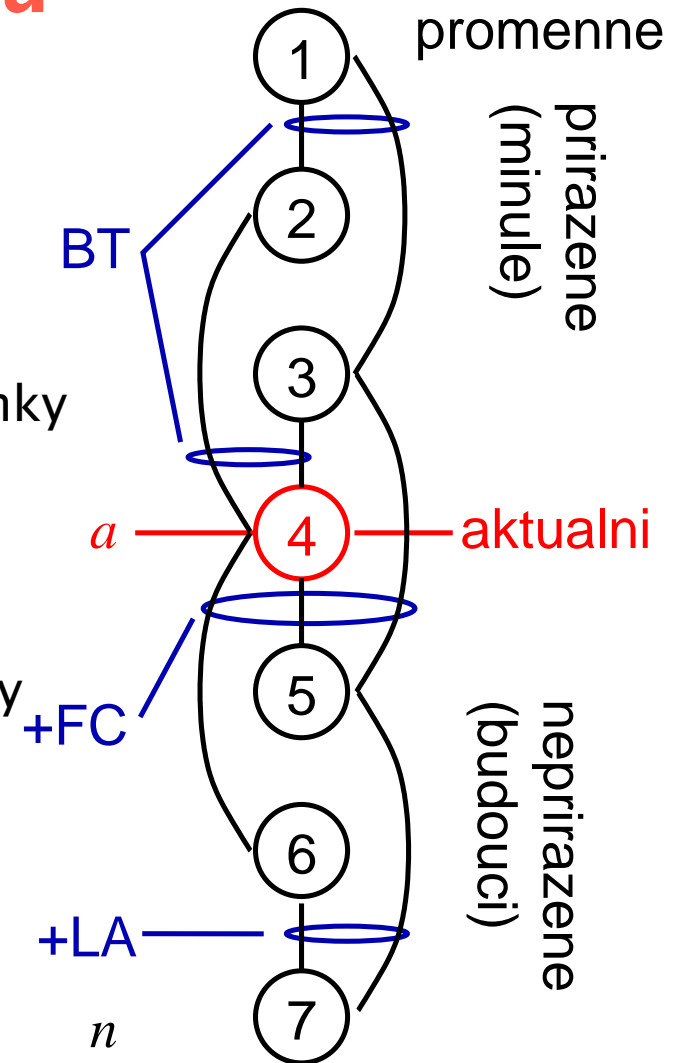
Srovnání algoritmů

- **Backtracking (BT)** kontroluje v kroku a podmínky $c(V_1, V_a), \dots, c(V_{a-1}, V_a)$ z minulých proměnných do aktuální proměnné
- **Kontrola dopředu (FC)** kontroluje v kroku a podmínky $c(V_{a+1}, V_a), \dots, c(V_n, V_a)$ z budoucích proměnných do aktuální proměnné



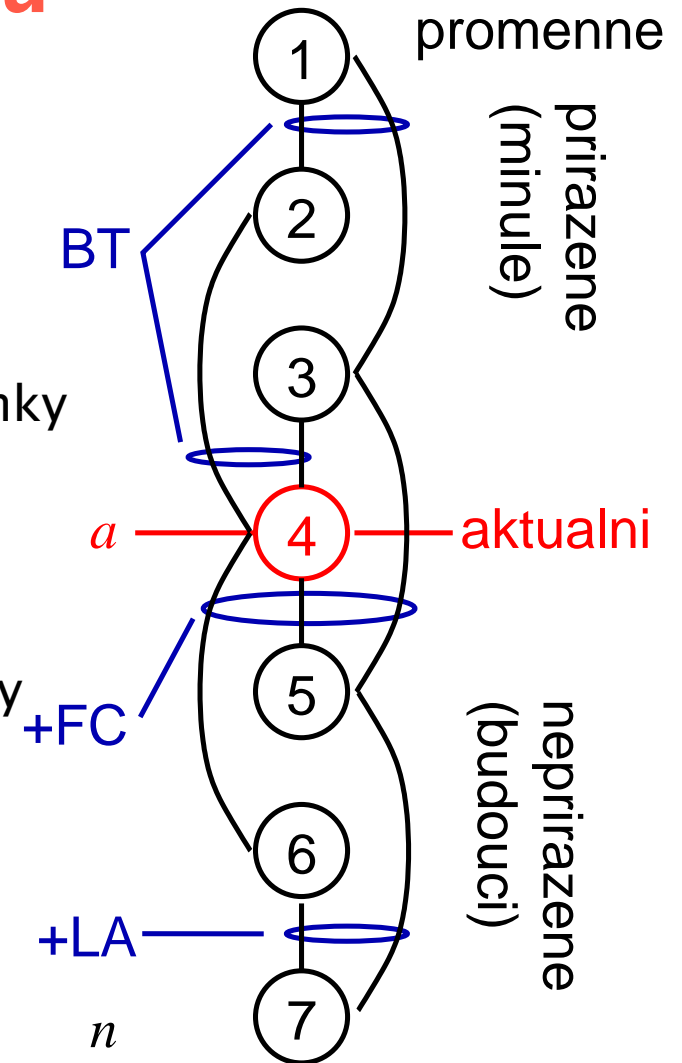
Srovnání algoritmů

- **Backtracking (BT)** kontroluje v kroku a podmínky $c(V_1, V_a), \dots, c(V_{a-1}, V_a)$ z minulých proměnných do aktuální proměnné
- **Kontrola dopředu (FC)** kontroluje v kroku a podmínky $c(V_{a+1}, V_a), \dots, c(V_n, V_a)$ z budoucích proměnných do aktuální proměnné
- **Pohled dopředu (LA)** kontroluje v kroku a podmínky $\forall l(a \leq l \leq n), \forall k(a \leq k \leq n), k \neq l : c(V_k, V_l)$ z budoucích proměnných do aktuální proměnné a mezi budoucími proměnnými



Srovnání algoritmů

- **Backtracking (BT)** kontroluje v kroku a podmínky $c(V_1, V_a), \dots, c(V_{a-1}, V_a)$ z minulých proměnných do aktuální proměnné
- **Kontrola dopředu (FC)** kontroluje v kroku a podmínky $c(V_{a+1}, V_a), \dots, c(V_n, V_a)$ z budoucích proměnných do aktuální proměnné
- **Pohled dopředu (LA)** kontroluje v kroku a podmínky $\forall l(a \leq l \leq n), \forall k(a \leq k \leq n), k \neq l: c(V_k, V_l)$ z budoucích proměnných do aktuální proměnné a mezi budoucími proměnnými



- **Cvičení:** porovnejte jednotlivé algoritmy pro příklad $V_1, V_2, V_3 \in \{1, 2, 3, 4\}$, $V_1 = V_2 * 2, V_3 = V_2 + 1$, nakreslete odpovídající stavový prostor a u každého stavu uveďte, ke kterým propagacím omezení dochází.

Shrnutí: pohledu dopředu pro výběr hodnoty

● **Kontrola dopředu (*forward checking*) FC**

- FC zajišťuje konzistenci mezi aktuální proměnnou a budoucími proměnnými, které jsou s ní spojeny dosud nesplněnými podmínkami

● **Opravdový úplný pohled dopředu (*real full look-ahead*) RFLA**

- často se odkazuje: LA algoritmus **RFLA=LA** nebo **Maintaining Arc-Consistency MAC**
- LA je rozšíření FC, LA zajišťuje hranovou konzistenci
- LA navíc ověřuje i konzistenci všech hran mezi budoucími proměnnými

Shrnutí: pohledu dopředu pro výběr hodnoty

● **Kontrola dopředu (*forward checking*) FC**

- FC zajišťuje konzistenci mezi aktuální proměnnou a budoucími proměnnými, které jsou s ní spojeny dosud nesplněnými podmínkami

● **Opravdový úplný pohled dopředu (*real full look-ahead*) RFLA**

- často se odkazuje: LA algoritmus **RFLA=LA** nebo **Maintaining Arc-Consistency MAC**
- LA je rozšíření FC, LA zajišťuje hranovou konzistenci
- LA navíc ověřuje i konzistenci všech hran mezi budoucími proměnnými

● **Úplný pohled dopředu (*full look-ahead*) FLA**

- pouze jeden průchod `repeat until` cyklem algoritmu

Shrnutí: pohledu dopředu pro výběr hodnoty

● **Kontrola dopředu (*forward checking*) FC**

- FC zajišťuje konzistenci mezi aktuální proměnnou a budoucími proměnnými, které jsou s ní spojeny dosud nesplněnými podmínkami

● **Opravdový úplný pohled dopředu (*real full look-ahead*) RFLA**

- často se odkazuje: LA algoritmus **RFLA=LA** nebo **Maintaining Arc-Consistency MAC**
- LA je rozšíření FC, LA zajišťuje hranovou konzistenci
- LA navíc ověřuje i konzistenci všech hran mezi budoucími proměnnými

● **Úplný pohled dopředu (*full look-ahead*) FLA**

- pouze jeden průchod `repeat until` cyklem algoritmu

● **Částečný pohled dopředu (*partial look-ahead*) PLA**

- pouze jeden průchod `repeat until` cyklem algoritmu
- nahrazuje `for $\forall k, k \neq j, i < k \leq n$` výrazem `for $\forall k, j < k \leq n$`

tj. budoucí proměnné porovnávány pouze s těmi, které je následují (DAC)

Výběr hodnoty

Jakým způsobem vybírat ze zbývajících hodnot v doméně proměnné?

- Triviální výběr hodnoty

- doména procházena ve vzrůstajícím pořadí
- doména procházena v klesajícím pořadí

- Obecný princip výběru hodnoty: **první úspěch** (*succeed first*)

- volíme pořadí tak, abychom výběr nemuseli opakovat
- ?- $A, B, C \in \{1, 2\}, A = B + C$

Výběr hodnoty

Jakým způsobem vybírat ze zbývajících hodnot v doméně proměnné?

- Triviální výběr hodnoty

- doména procházena ve vzrůstajícím pořadí
- doména procházena v klesajícím pořadí

- Obecný princip výběru hodnoty: **první úspěch (*succeed first*)**

- volíme pořadí tak, abychom výběr nemuseli opakovat
- ?- $A, B, C \in \{1, 2\}, A = B + C$
optimální výběr $A = 2, B = 1, C = 1$ je bez backtrackingu

Pohled dopředu pro dynamický výběr hodnoty

- Výběr hodnoty v doméně proměnné
 - zatím jsme vybírali první konzistentní hodnotu - jednalo se o **statický výběr hodnoty**
 - místo toho zkusíme proměnné přiřadit každou z hodnot v doméně a vyzkoušíme efekt použití FC nebo jiného LA algoritmu

Pohled dopředu pro dynamický výběr hodnoty

● Výběr hodnoty v doméně proměnné

- zatím jsme vybírali první konzistentní hodnotu - jednalo se o **statický výběr hodnoty**
- místo toho zkusíme proměnné přiřadit každou z hodnot v doméně a vyzkoušíme efekt použití FC nebo jiného LA algoritmu

● **Minimální konflikt**

- výběr hodnoty, která smaže nejmenší počet hodnot z domén budoucích proměnných
- experimentálně: tato heuristika vykazuje velmi dobré výsledky

Pohled dopředu pro dynamický výběr hodnoty

● Výběr hodnoty v doméně proměnné

- zatím jsme vybírali první konzistentní hodnotu - jednalo se o **statický výběr hodnoty**
- místo toho zkusíme proměnné přiřadit každou z hodnot v doméně a vyzkoušíme efekt použití FC nebo jiného LA algoritmu

● **Minimální konflikt**

- výběr hodnoty, která smaže nejmenší počet hodnot z domén budoucích proměnných
- experimentálně: tato heuristika vykazuje velmi dobré výsledky

● **Maximální velikost domény**

- výběr hodnoty, která způsobí vytvoření největší minimální velikost domény mezi všemi budoucími proměnnými
- intuice: proměnné, které mají malé domény, způsobí pravděpodobněji nekonzistenci

Výběr proměnných: statický

Uspořádání (výběr) proměnných má velký vliv na velikost stavového prostoru

- **Statické uspořádání proměnných:** výběr proměnných dán předem
 - triviálně: výběr nejlevější proměnné (tj. proměnné s nejmenším indexem)
 - empirické i teoretické studie ukázaly, že některá statická uspořádání vedou obecně ke zmenšení velikost stavového prostoru

Výběr proměnných: statický

Uspořádání (výběr) proměnných má velký vliv na velikost stavového prostoru

● **Statické uspořádání proměnných:** výběr proměnných dán předem

- triviálně: výběr nejlevější proměnné (tj. proměnné s nejmenším indexem)
- empirické i teoretické studie ukázaly, že některá statická uspořádání vedou obecně ke zmenšení velikost stavového prostoru

● **Maximální kardinalita**

- první proměnnou (uzel grafu) vybereme náhodně
- každý uzel je spojen s maximálním počtem už uspořádaných (= dříve vybraných) vrcholů
- proměnné vybíráme v pořadí dle tohoto počtu vrcholů

Výběr proměnných: statický

Uspořádání (výběr) proměnných má velký vliv na velikost stavového prostoru

● **Statické uspořádání proměnných:** výběr proměnných dán předem

- triviálně: výběr nejlevější proměnné (tj. proměnné s nejmenším indexem)
- empirické i teoretické studie ukázaly, že některá statická uspořádání vedou obecně ke zmenšení velikost stavového prostoru

● **Maximální kardinalita**

- první proměnnou (uzel grafu) vybereme náhodně
- každý uzel je spojen s maximálním počtem už uspořádaných (= dříve vybraných) vrcholů
- proměnné vybíráme v pořadí dle tohoto počtu vrcholů

● **Minimální šířka**

- proměnné uspořádány tak, aby byla minimalizována šířka grafu (minimalizován počet zpětných hran)
- odzadu vybíráme proměnné s nejmenším počtem hran v aktualizovaném grafu (algoritmus viz dříve)

Výběr proměnných: dynamický

- **Dynamické uspořádání proměnných:** výběr proměnných počítán až v průběhu prohledávání
 - Ize využít (výpočtů) algoritmů pohledu dopředu

Výběr proměnných: dynamický

- **Dynamické uspořádání proměnných:** výběr proměnných počítán až v průběhu prohledávání
 - lze využít (výpočtů) algoritmů pohledu dopředu
- **First-fail**
 - velmi často používaná metoda (vhodná jako default)
 - výběr proměnné, která nejvíce omezí zbytek stavového prostoru
 - vybereme proměnnou **s nejmenší doménou**
 - později už by mohlo být těžší pro tuto proměnnou nalézt správnou hodnotu
 - kombinace first-fail a maximální kardinality
 - $A, B, C \in \{1, 2, 3\}, A < 3, A = B + C$ nejlépe je začít s výběrem A

Pohled dopředu pro dynamický výběr proměnné

procedure Var-Ord-Look-Ahead((X, D, C))

rozdíly od Look-Ahead

$i := 1$ (inicializace čítače proměnných)
 $D'_i := D_i$ pro $1 \leq i \leq n$ (kopírování domény)
 $s := \min_{1 \leq j \leq n} \|D'_j\|$ (nalezni proměnnou s nejmenší doménou)
 $x_1 := x_s$ (přeskládej proměnné tak, aby x_s byla první)
while $1 \leq i \leq n$
 přiřazení $x_i := \text{Select-Value-XXX}$
 if x_i is null (žádná hodnota nebyla vrácena)
 $i := i - 1$ (zpětná fáze)
 nastav všechny $D'_k, k > i$ na jejich hodnotu před poslední instanciací x_i
 else if $i < n$
 $s := \min_{i < j \leq n} \|D'_j\|$ (nalezni budoucí proměnnou s nejmenší doménou)
 $x_{i+1} := x_s$ (přeskládej proměnné tak, aby x_s následovala x_i)
 $i := i + 1$ (dopředná fáze)
if $i = 0$ return „nekonzistentní“
else return přiřazené hodnoty $\{x_1, \dots, x_n\}$
end Var-Ord-Look-Ahead

Silnější pohled dopředu

- Algoritmy pohledu dopředu zatím uvažovaly pouze hranovou konzistenci
 - vede k mazání hodnot z domény
- Po každém přiřazení hodnoty proměnné lze vyžadovat dosažení vyššího stupně konzistence
 - navíc jsou přidávána i nová omezení
- Kdy má cenu vyšší úrovně konzistence uvažovat?
 - vhodné pro propagaci nad podmnožinou proměnných
 - globální podmínky
 - nebo při rozhodování o výběru hodnoty
 - nepřidáváme nová omezení

Doplňkové materiály

Course on Constraint Programming and Scheduling (10 hodin)

- bakalářský kurz vyučovaný v angličtině na Hochschule Konstanz, Technik, Wirtschaft, und Gestaltung (Německo) v květnu 2009
- <http://www.fi.muni.cz/~hanka/konstanz09/>
- propagace, prohledávání, modelování, příklady
- omezující podmínky
- použití omezujících podmínek pro rozvrhování

Část přednášky *Constraint Programming: Search*

- algoritmy v rekurzivním pseudokódu
- příklady prohledávání stavového prostoru včetně řešení