Splňanie ohraničení

1

Farbenie mapy

- Úloha:

  Majme mapu krajín, priraďte každej krajine farbu, tak že:

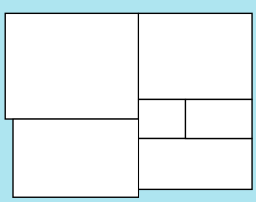
  AK majú dve krajiny spoločnú hranicu,

  TAK musia mať krajiny rôznu farbu
- Je možné použiť iba obmedzené množstvo farieb, snahou je použiť pre úspešné vyriešenie problému minimálne množstvo farieb

2

Príklad - Farbenie mapy

- Pre zjednodušenie uvažujme, že každá krajina má tvar štvoruholníka:

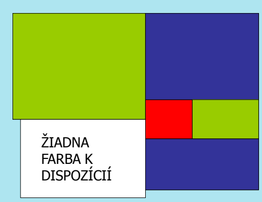


- Koľko farieb je potrebných pre úspešné vyriešenie problému ?

3

Príklad – Farbenie mapy: 3 farby ?

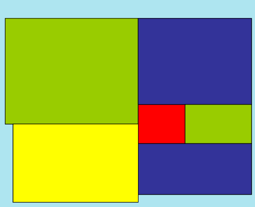
- Pokúsme sa použiť pre vyriešenie problému 3 farby: červenú, modrú a zelenú



- Môžeme si byť istí, že pri použití troch farieb sa problém nedá vyriešiť ?
- Nemôžeme, je potrebné použiť jednu z úplných metód prehľadávania!

4

Príklad – Farbenie mapy: 4 farby



- PRI POUŽITÍ ĎALŠEJ, 4. FARBY SA DÁ PROBLÉM VYRIEŠIŤ

5

Motivácia

- Farbenie mapy je špecifickým problémom
  - Prečo sa ním teda zaoberať ?
- Farbenie mapy je typickým príkladom typu problému s ohraničeniami (problém splňania ohraničení: CSP - Constraint Satisfaction Problem)
- CSP sa vyskytujú v mnohých oblastiach
  - Plánovanie
  - Cestovné poriadky
  - Optimalizačné problémy v oblasti priemyslu

6

príklad: hlavolam sudoku

- http://www.websudoku.com/
- každé Sudoku má jediné riešenie dosiahnuteľné len logickým uvažovaním bez hádania.
- Prázdne miesta treba vyplniť číslami 1 až 9.
- V každom riadku musí byť každé číslo a práve raz.
- V každom stĺpci tiež.
- V každom štvorci 3x3 tiež.

príklad: hlavolam sudoku

				9		3		
	2		6		1	5	9	
3			5		8		2	4
	9					4	8	
6			2		9			1
	5	2					3	
9	7		1		5			3
	3	8	7		4		1	
		5		8				

"Programovanie ohraničení"

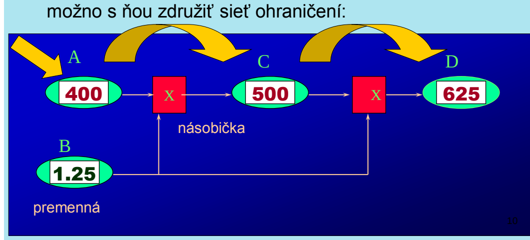
- Odlišná programovacia paradigma
  - "stačí stanoviť, čo je potrebné vyriešiť a systém sám rozhodne, ako daný problém vyriešiť"
  - Kód programu na vyriešenie Sudoku hlavolamu môže mať iba 20 riadkov
  - Napr. ohraničenie pre riadok, že všetky čísla v riadku sú rôzne
  - prevšetky(j in 1..9) všetkyrôzne(prevšetky(i in 1..9) pos[i,j]);
- Veľmi odlišné od tradičných paradigiem:
  - Procedurálne programovanie (Fortran, Pascal, C)
  - Objektovo-orientované programovanie (C++, Java, C#)
  - Funkcionálne programovanie (Lisp, ML, Haskell)

príklad: sústava číselných ohraničení

nech je daná množina rovníc:

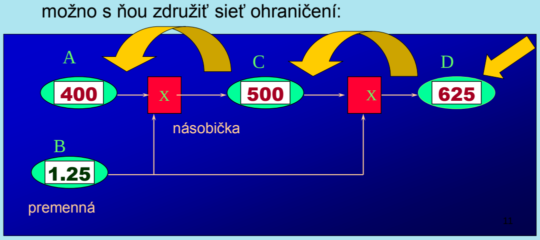
$$\begin{cases} C = A * B \\ D = C * B \\ B = 1.25 \end{cases}$$

možno s ňou združiť sieť ohraničení:



príklad: sústava číselných ohraničení 2

možno s ňou združiť sieť ohraničení:



príklad: kalkulačná tabuľka (spreadsheet)

	A	B	C	D
1	pomer X	1.2		
2	pomer Y	1.1		
4		1. rok	2. rok	3. rok
5	príjem X	3000	= B1 * B5	= B1 * C5
6	príjem Y	5000	= B2 * B6	= B2 * C6
7	výdaje	9000	= B7	= C7
8	celkovo	= B5+B6 -B7	= C5+C6 - C7	=D5+D6 -D7

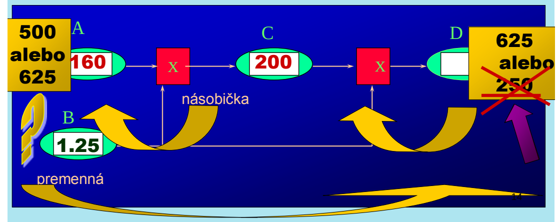
príklad: kalkulačná tabuľka (spreadsheet) 2

modus: čo ak

	A	B	C	D
1	pomer X	1.2		
2	pomer Y	1.1		
4		1. rok	2. rok	3. rok
5	príjem X	3000	3600	4320
6	príjem Y	5000	5500	6050
7	výdaje	9000	9000	9000
8	celkovo	- 1000 -B7	100 - C7	1370 -D7

príklad: sústava číselných ohraničení 3

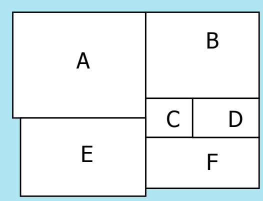
možno s ňou združiť sieť ohraničení:



Od máp k spĺňaniu ohraničení

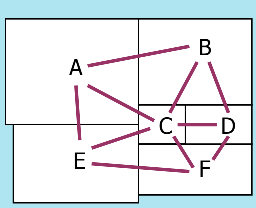
- Upravme problém farbenia máp podľa všeobecných predstáv používaných pri riešení problémov s ohraničeniami:
  - Priradené hodnoty musia spĺňať niektoré z ohraničení

Farbenie mapy – priradenie hodnôt



Farbenie mapy - ohraničenia

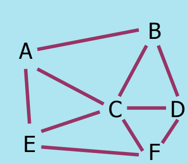
- Grafy opisujú problém všeobecnejšie ako mapy – konvertujme teda mapu na graf
- Hrana vyjadruje skutočnosť, že krajiny majú spoločnú hranicu, čiže musia byť zafarbené odlišnou farbou



Farbenie grafu 3 farbami

- Každému uzlu musí byť priradená hodnota z množiny farieb { r, g , b }
- Napr.: B := b
- Každej hrane medzi uzlami je možné priradiť iba dvojicu z množiny

  { (r,g), (r,b),
  (g, r), (g, b),
  (b, r), (b, g) }



Reprezentácia problému pomocou grafu

- Zovšeobecnime teda problém farbenia troma farbami na problém s ohraničeniami (CSP)
- Každému uzlu musí byť priradená hodnota z pevne danej množiny prípustných hodnôt zvyčajne nazývanej "doména" D
- Každej hrane medzi dvoma uzlami je možné priradiť iba prípustnú dvojicu z množiny prípustných dvojíc hodnôt
  - Hrana sa nazýva ohraničenie
  - Taktiež ide o synonymum pre množinu prípustných dvojíc

Domény CSP

- Domény je možné pokladať za
  - diskrétne konečné
    - zvláštny prípad: boolovské hodnoty
  - diskrétne nekonečné
  - spojité

Ohraničenia: Explicitné vs. implicitné

- Ohraničenie sa vyjadruje (predpokladajme, že x a y majú doménu D={u,v,w})
  - explicitne: množina prípustných dvojíc hodnôt napr $(x,y) \in \{ (u,u), (u,v), (w,u), (v,w) \}$
  - implicitne: napr.:

    $y \neq w$

    ale aj v matematike napr.

    $x + y = 2$

    $x + 2 < y$

Ohraničenia: n-miestne

- 1-miestne (unárne)
  - $x \neq R$
- 2-miestne (binárne)
  - $z_i < z_j \wedge z_i + i \neq z_j - j$
    problém výlučne s binárnymi ohraničeniami sa dá reprezentovať pomocou grafu.
    Hrana medzi dvoma uzlami reprezentuje ohraničenie
- n-miestne, $n \geq 3$
  - všetkyrozdielne $(x_1, ..., x_n)$
    n-miestne ohraničenie sa dá previesť na množinu 2-miestnych ohraničení
    $\{x_1 \neq x_2, x_1 \neq x_3, \ldots\}$

Diskrétna konečná doména

- Predpokladajme, že máme n uzlov a každému uzlu je priradená hodnota z množiny D a že veľkosť množiny D je d
- Počet možných priradení je potom:

  $d * d * d * ... * d = d^n$

- Exponenciálna veľkosť problému $O(d^n)$

Boolovská doména

- Predpokladajme, že máme n uzlov a každému uzlu je priradená hodnota z množiny {T, F}
- Počet možných priradení je potom 2^n

  príklad problému:
  - splniteľnosť formuly v konjuktívnom normálnom tvare, kde každá klauzula má najviac 3 literály – 3SAT
  - $(x_{11} \vee x_{12} \vee x_{13}) \wedge (x_{21} \vee x_{22} \vee x_{23}) \wedge (x_{31} \vee x_{32} \vee x_{33}) \wedge \ldots$
  - kde x_{ij} je boolovská premenná alebo negácia premennej a každá premenná sa môže vyskytovať viackrát vo výraze
  - splniteľnosť boolovskej formuly – či existuje priradenie boolovských hodnôt premenným vo formule také, že formula sa vyhodnotí na T

Boolovská doména: 3SAT

- inštancia príkladu problému:
  - $E = (x_1 \lor \neg x_2 \lor \neg x_3) \land (x_1 \lor x_2 \lor x_4)$
  - premenné: (x_1, x_2, x_3, x_4)
  - riešenie: napr $\{x_1 = T, x_2 = T, x_3 = T, x_4 = T\}$
  - existuje mnoho riešení, napr všetky priradenia s $x_1 = T$

diskrétna nekonečná doména

- diskrétne premenné
  - nekonečné domény (celé čísla, reťazce, atď.)
    - napr. rozvrhovanie úloh: premenné sú začiatočné/koncové dni pre každú úlohu
    - potreba jazyka na vyjadrenie ohraničení, napr. $StartJob_1 + 5 \le StartJob_3$.
    - nekonečne veľa riešení
    - lineárne ohraničenia: riešiteľné
    - nelineárne: neexistuje vešobecný algoritmus

spojitá doména

- spojité premenné
  - napr. zostavenie letového poriadku alebo rozvrhu fakulty.
  - lineárne ohraničenia sú riešiteľné v polynomiálnom čase metódami lineárneho programovania.

jemné ohraničenia

- preferencie (jemné ohraničenia)
  - napr. *modrá* je lepšia než *červená* sa často dajú reprezentovať cenou pre každé priradenie premennej
  - kombinácia optimalizácie s CSP

Riešenie problému spĺňania ohraničení

- Domény pokladajme za konečné
- Predpokladajme, že máme n uzlov a každému uzlu je priradená hodnota z množiny D a že veľkosť množiny D je d
- Počet možných priradení je potom:

$$d * d * d * ... * d = d^n$$

- Exponenciálna veľkosť problému
- pre domény veľkosti $d \Rightarrow O(d^n)$ úplných priradení.

formulovanie problému pomocou stavového priestoru

- Čo je stavový priestor?
- Čo je začiatočný stav?
- Aká je množina operátorov (funkcia generujúca nasledovníkov)?
- Aký je cieľový test?
- Aká je cena cesty?

zvláštnosti

- veľký priestor hľadania
- komutatívnosť
- čo tak použiť hľadanie do šírky alebo hĺbky?
  - pevná hĺbka

31

Čo ešte treba?

- nestačí len funkcia nasledovníka a cieľový test
- ale aj spôsob, ako šíriť ohraničenia na ďalšie kroky, vzniknuvšie v dôsledku nejakého kroku a tiež včasný test zlyhania
- → Explicitné reprezentovanie ohraničení a algoritmy na manipulovanie s ohraničeniami

32

problém spĺňania ohraničení

- množina premenných $V = \{X_1, X_2, \ldots, X_n\}$
- každá premenná X_i má doménu D_i možných hodnôt
  - zvyčajne D_i je diskrétna a konečná
- množina ohraničení $\{C_1, C_2, \ldots, C_p\}$
  - každé ohraničenie C_k obsahuje podmnožinu premenných a určuje prípustné kombinácie hodnôt týchto premenných
- cieľ: priradiť hodnotu každej premennej tak, aby boli všetky ohraničenia splnené
  - priradenie je prvok z $D_1 \times D_2 \times \ldots \times D_n$

33

Spĺňanie ohraničení ako problém hľadania riešenia

- Máme množinu n premenných (uzlov) $V = \{X_1, \ldots, X_n\}$
- Stavy: platné priradenia
  - Platné priradenie : $\{X_{i1} \leftarrow v_{i1}, \ldots, X_{ik} \leftarrow v_{ik}\}$, $0 \leq k \leq n$, také, že hodnoty spĺňajú ohraničenia vzťahujúce sa k uzlom $X_{i1}, \ldots, X_{ik}$
  - Úplné priradenie: $k = n$
- Začiatočný stav: prázdne priradenie {}, $k = 0$
- Operátory:

  $\{X_{i1} \leftarrow v_{i1}, \ldots, X_{ik} \leftarrow v_{ik}\} \rightarrow \{X_{i1} \leftarrow v_{i1}, \ldots, X_{ik} \leftarrow v_{ik}, \mathbf{X_{ik+1} \leftarrow v_{ik+1}}\}$

  priradenie hodnoty premennej, ktorá ju ešte nemá priradenú a takej, ktorá neprotirečí doteraz priradeným premenným
- Cieľový test: $k = n$
- cena riešenia: konštantná rovnaká cena každého kroku (napr 1). v princípe irrelevantná

34

Druhy priradení

- Úplné priradenie:
  - Všetkým uzlom je priradená hodnota
- Neúplné priradenie:
  - Hodnota je priradená všetkým, niektorým alebo žiadnemu uzlu

35

CSP ako problém hľadania - zhrnutie

- začiatočný stav: prázdne priradenie
- stavy: konzistentné priradenia (úplné aj neúplné)
- funkcia nasledovníka: priradenie hodnoty premennej, ktorá ju ešte nemá priradenú a takej, ktorá neprotirečí doteraz priradeným premenným
- cieľový test: priradenie je úplné (všetky premenné majú priradenú hodnotu)
- cena cesty: irrelevantná

- faktor vetvenia *b* na vrchnej úrovni je *nd.*
- $b=(n-l)d$ v hĺbke *l*, takže je $n!d^n$ listov (ale len d^n úplných priradení).

dá sa aj lepšie.......

36

problém splňania ohraničení

- $(\exists x_1)..(\exists x_n)\ (D_1(x_1) \wedge ... \wedge D_n(x_n) \Rightarrow C_1(x_1, ..., x_n) \wedge ... \wedge C_m(x_1, ..., x_n)$

37

riešiť problém splňania ohraničení

- preukázať, či ne/jestuje riešenie
- zistiť počet riešení
- nájsť jedno riešenie
- nájsť všetky riešenia
- nájsť optimálne riešenie (ktoré minimalizuje nejakú oceňovaciu funkciu), ak je taká funkcia definovaná

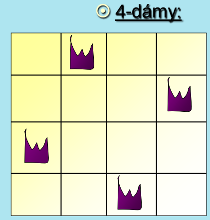
38

Definovanie problému s ohraničeniami

- problém s ohraničeniami (alebo problém splnenia ohraničení) je:
  - ➔ konečná množina premenných: z1, z2, … , zn
  - ➔ pre každú premennú s ňou združená konečná množina (doména) jej možných hodnôt di = {ai1,ai2, … , aini}
  - ➔ pre každú dvojicu premenných zi, zj, i < j, ohraničenie c(zi, zj)
- ◆ príklad: zi < zj ∧ zi + i ≠ zj - j
- ◆ obmedzíme sa na binárne ohraničenia
- riešenie: pre každú premennú zi, hodnota aij z jej domény di taká, že všetky ohraničenia c(zi, zj) sú splnené.

príklad: N-dám:

- dané: šachovnica N x N
- Problém: nájsť (všetky možné) spôsoby rozmiestnenia N dám na šachovnici tak, že sa žiadne 2 dámy nenapádajú.
- 4-dámy:



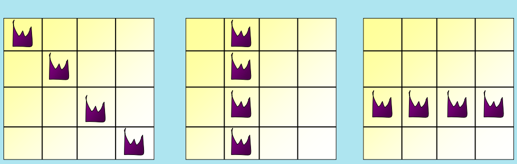
… je riešenie

40

popletených N-dám:

= rovnaká špecifikácia problému až na to, že ľubovoľné 2 dámy sa musia napádať.

- p-4-dámy:



41

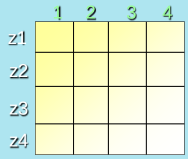
ako formulovať ohraničenia?

- N-dám a p-N-dám sa dajú formulovať ako problémy s ohraničeniami:
  - ➔ záleží na voľbe domén a ohraničení
- jedna možnosť:
  - ➔ pre každú dámu: premenná zi
  - ➔ pre každú zi, doména di je množina všetkých možných pozícií na šachovnici:
    - ◆ di = { (1,1), (1,2), …, (1,N), (2,1), (2,2), … }
  - ➔ c(zi, zj) = ria(zi) ≠ ria(zj) ∧ stl(zi) ≠ stl(zj) ∧ |ria(zi) - ria(zj)| ≠ |stl(zi) - stl(zj)|

42

‘lepšia’ formulácia:

- prijmeme dohodu, že každá dáma bude na zvláštnom riadku:



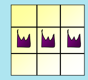
reprezentácia:

➔ pre dámu na riadku i: premenná zi
➔ pre každú zi, doména di = { 1, 2, … , N}
➔ c(zi, zj) = zi ≠ zj ∧ |zi - zj| ≠ |i - j|

43

poznámky k p-N-dám:

- druhý spôsob reprezentácie definuje trochu iný problém:



… toto už nie je riešenie

problém má teraz vždy N+2 riešení !

➔ výhoda pre analýzu zložitosti

výnimka: N = 3:

sú navyše riešenia

44

príklad: 8-dám

- 64 premenných Zij, i = 1 to 8, j = 1 to 8
- doména pre každú premennú {0,1}
- ohraničenia majú tvar:
- riadky a stĺpce
  - Zij = 1 ➔ Zik = 0 for all k = 1 to 8, k≠j
  - Zij = 1 ➔ Zkj = 0 for all k = 1 to 8, k≠l
- uhlopriečky
  - Zij = 1 ➔ Zi+l,k+l = 0 l = 1 to 8, i+l ≤8; k+l≤8 (doprava nahor)
  - Zij = 1 ➔ Zi-l,k+l = 0 l = 1 to 8, i-l ≥1; k+l≤8 (doprava nadol)
  - Zij = 1 ➔ Zi-l,k-l = 0 l = 1 to 8, i-l ≥1; k-l≥1 (doľava a nadol)
  - Zij = 1 ➔ Zi+l,k-l = 0 l = 1 to 8, i+l ≤8 ; k-l≥1 (doľava a nahor)

45

príklad: 8-dám

- 8 premenných Zi, i = 1 to 8
- doména pre každú premennú {1,2,…,8}
- ohraničenia majú tvar:
  - Zi ≠ Zj ak j≠i
  - dámy na tej istej uhlopriečke
    - Zi – Zj ≠ i – j a
    - Zi – Zj ≠ j – i

46

krypto-aritmetický hlavolam

SEND	9567
+MORE	+ 1085
-----------	----------
MONEY	10652

premenné: S E N D M O R Y

domény:
[1..9] pre S and M
[0..9] pre E N D O R Y

47

krypto-aritmetický hlavolam ohraničenia 1

- 1 jednoduché ohraničenie

1000 S + 100 E + 10 N + D +
1000 M + 100 O + 10 R + E
=
10000 M + 1000 O + 100 N + 10 E + Y

alebo

- 5 ohraničujúcich rovností používajúcich prenosy - premenné C1, …, C4 Є [0..1]

D + E = 10 C1 + Y;	SEND
C1 + N + R = 10 C2 + E;	+MORE
C2 + E + O = 10 C3 + N;	-----------
C3 + S + M = 10 C4 + O;	MONEY
C4 = M	

krypto-aritmetický hlavolam ohraničenia 2

•28 ohraničujúcich nerovností

• X≠ Y, X,Y ∊ {**S E N D M O R Y**}

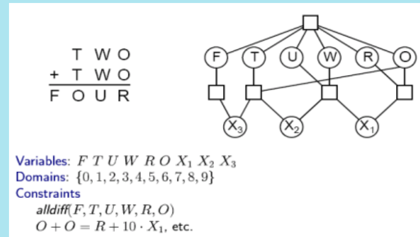
•**alebo**

•jediné ohraničenie

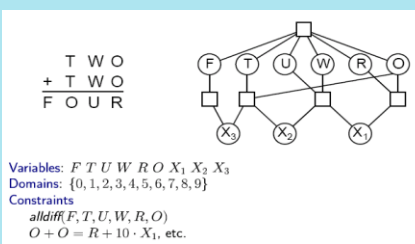
• všetkyrôzne(S, E, N, D, M, O, R, Y)

všetkyrôzne(X1, … , Xn) → kompaktné tvrdenie, že premenné X1, … , Xn všetky majú priradené navzájom rôzne hodnoty
→ globálne ohraničenie (obsahuje n-árne ohraničenie)
→ zvláštne procedúry na prácu s takým ohraničením

krypto-aritmetický hlavolam príklad



krypto-aritmetický hlavolam príklad



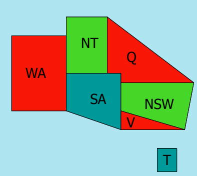
príklad: hlavolam sudoku

- premenné:
- domény:
- ohraničenia:
- cieľ: priraď hodnotu každej premennej tak, aby boli všetky ohraničenia splnené

príklad: hlavolam sudoku

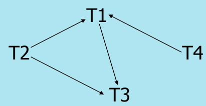
- premenné: $X_{11}, \ldots, X_{99}$
- domény: {1,…,9}
- ohraničenia:
  - riadkové ohraničenie: $X_{11} \neq X_{12}, \ldots, X_{11} \neq X_{19}$
  - stĺpcové ohraničenie: $X_{11} \neq X_{12}, \ldots, X_{11} \neq X_{19}$
  - blokové ohraničenie: $X_{11} \neq X_{12}, \ldots, X_{11} \neq X_{33}$
- cieľ: priraď hodnotu každej premennej tak, aby boli všetky ohraničenia splnené

príklad: ofarbenie mapy



• 7 premenných {WA,NT,SA,Q,NSW,V,T}
• všetky premenné majú tú istú doménu {červená, zelená, modrá}
• nijaké 2 susediace premenné nemajú rovnakú hodnotu:
WA≠NT, WA≠SA, NT≠SA, NT≠Q, SA≠Q, SA≠NSW, SA≠V,Q≠NSW, NSW≠V

príklad: rozvrhovanie úloh

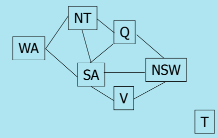


T1 sa musí urobiť počas T3
T2 sa musí dokončiť predtým, než začne T1
T2 sa musí prekrývať s T3
T4 sa musí začať po skončení T1

- Sú tieto ohraničenia vôbec zlučiteľné?
- nájsť časový vzťah medzi každými dvoma úlohami

graf ohraničení

binárne ohraničenia



Aký by bol graf ohraničení pre sudoku?

dve premenné susedia ak sú spojené hranou

Generuj a testuj

- Generuj všetky možné priradenia
- Otestuj každé priradenie, či spĺňa všetky ohraničenia
- Veľmi neefektívny postup !

"Programovanie ohraničení"

- Problém:
  X::{1,2}, Y::{1,2}, Z::{1,2}

  X = Y, X ≠ Z, Y > Z

generovanie a testovanie

X	Y	Z	test
1	1	1	nie
1	1	2	nie
1	2	1	nie
1	2	2	nie
2	1	1	nie
2	1	2	nie
2	2	1	áno

hľadanie s ustupovaním

X	Y	Z	test
1	1	1	nie
		2	nie
	2		nie
2	1		nie
	2	1	áno

Prístupy k riešeniu problémov spĺňania ohraničení

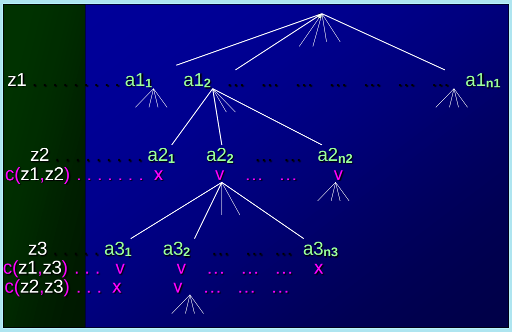
- Redukčné metódy
  - sa snažia zredukovať priestor problému
    - domény premenných a/alebo ohraničenia
  - aby nájdenie riešenia bolo čo najjednoduchšie
  - kľúčový pojem: silná k-konzistencia
    - zabezpečiť ju je výpočtovo náročné,
    - ale potom nájdenie riešenia je veľmi jednoduché
- Prehľadávacie algoritmy
  - prehľadávajú priestor problému, aby našli riešenie
  - jednoduché algoritmy, ale často musia prehľadávať
  - obrovské časti priestoru problému
- Kombinované algoritmy
  - kombinujú redukovanie a prehľadávanie
  - pred každým krokom prehľadávania sa robí redukcia

Metódy prehľadávania

- V prípade nutnosti použitia niektorej z úplných metód prehľadávania je bežným postupom použiť prehľadávanie do hĺbky s neúplným priradením
  - Na začiatku neexistujú žiadne priradenia
  - Generovanie potomkov priradením hodnoty pre ďalší uzol
- Analógia: hľadanie cesty – na začiatku cestu nepoznáme, v každom časovom okamihu k ceste pridávame ďalší segment

reprezentovanie ALEBO stromom:

zvoliť usporiadanie premenných: z1, z2, …, zn



ALEBO strom

- pre každú úroveň (hĺbku) (i) :

→ vytvor hranu pre každé možné priradenie premennej zi

→ prever splnenie všetkých ohraničení c(zj, zi), j < i

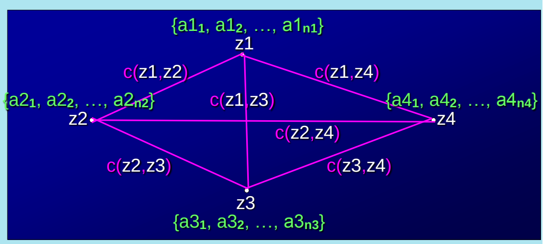
→ ak sú všetky ohraničenia splnené, pokračuj na úroveň i+1

☺ **poznámka**: toto je opis reprezentácie problému pre hľadanie riešenia

→ pri samotnom hľadaní sa buduje iba (malá) časť tohto stromu

62

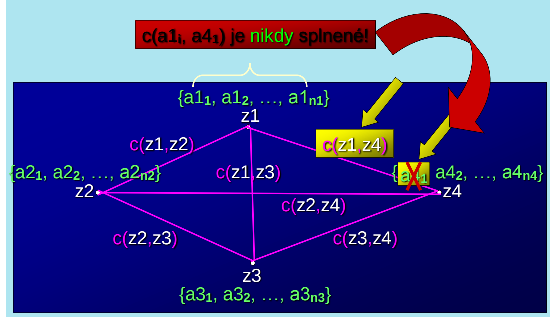
siete ohraničení



+ relaxácia = vyber hranu (ohraničenie) a odstráň nezlučiteľné hodnoty domény

63

Relaxácia:



64

Prehľadávanie s ustupovaním (backtracking, spätný chod)

- Pri prehľadávaní do hĺbky dochádza ku generovaniu všetkých potomkov
  - Vysoké pamäťové nároky
- Prehľadávanie s ustupovaním: počkaj s vytvorením potomka až do doby, kedy ho budeš potrebovať
  - Zapamätaj si iba to, že ho v budúcnosti bude potrebné vytvoriť
- Chronologické vracanie sa

65

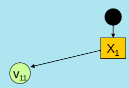
Príklad – Prehľadávanie s ustupovaním (3 uzly)

●

Priradenie = {}

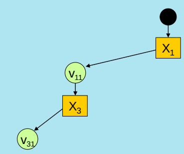
66

Príklad – Prehľadávanie s ustupovaním (3 uzly)



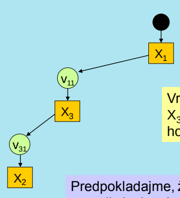
Priradenie = {(X_1,v_{11})}

Príklad – Prehľadávanie s ustupovaním (3 uzly)



Priradenie = {(X_1,v_{11}), (X_3,v_{31})}

Príklad – Prehľadávanie s ustupovaním (3 uzly)

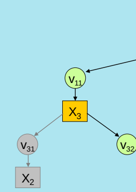


Vrátime sa o úroveň vyššie k uzlu X_3 a vyskúšame priradiť inú hodnotu

Predpokladajme, že žiadna hodnota uzla X_2 nevedie k platnému priradeniu

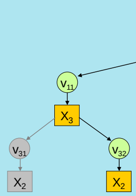
Priradenie = {(X_1,v_{11}), (X_3,v_{31})}

Príklad – Prehľadávanie s ustupovaním (3 uzly)



Priradenie = {(X_1,v_{11}), (X_3,v_{32})}

Príklad – Prehľadávanie s ustupovaním (3 uzly)

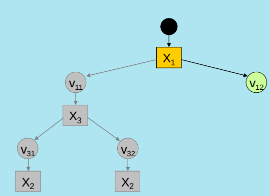


Ustúpenie o úroveň vyššie k uzlu X_3, predpokladajme však, že uzlu X_3 je možné priradiť iba 2 možné hodnoty. Vrátime sa o úroveň vyššie k uzlu X_1.

Znovu predpokladajme, že žiadna hodnota uzla X_2 nevedie k platnému priradeniu

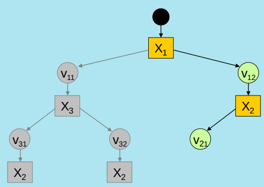
Priradenie = {(X_1,v_{11}), (X_3,v_{32})}

Príklad – Prehľadávanie s ustupovaním (3 uzly)



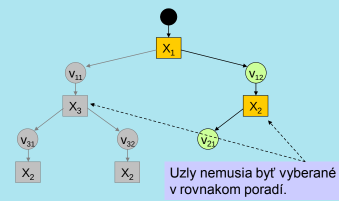
Priradenie = {(X_1,v_{12})}

Príklad – Prehľadávanie s ustupovaním (3 uzly)



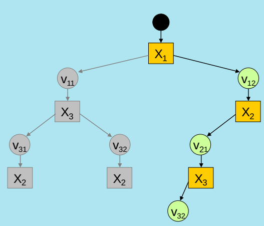
Priradenie = {(X_1,v_{12}), (X_2,v_{21})}

Príklad – Prehľadávanie s ustupovaním (3 uzly)



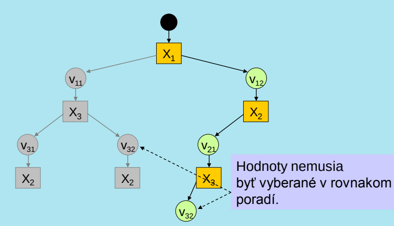
Priradenie = {(X_1,v_{12}), (X_2,v_{21})}

Príklad – Prehľadávanie s ustupovaním (3 uzly)



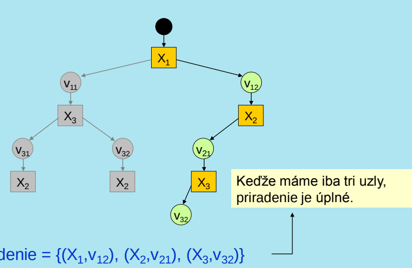
Priradenie = {(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})}

Príklad – Prehľadávanie s ustupovaním (3 uzly)



Priradenie = {(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})}

Príklad – Prehľadávanie s ustupovaním (3 uzly)



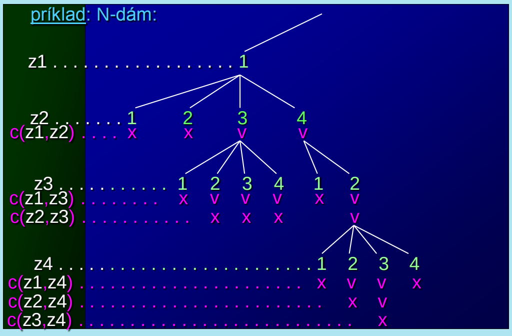
Priradenie = {(X_1,v_{12}), (X_2,v_{21}), (X_3,v_{32})}

Algoritmus - prehľadávanie s ustupovaním

```
CSP-BACKTRACKING(A)
    if priradenie A je úplné then return A
    X ← vyber uzol, ktorý nie je v A, tj premennú s nepriradenou hodnotou
    D ← vyber usporiadanie na doméne X
    for each hodnota v in D do
            if X←v je konzistentné s A then
                pridaj (X←v) do A
                riešenie ← CSP-BACKTRACKING(A)
                if riešenie ≠ neúspech then return riešenie
    return neúspech
```

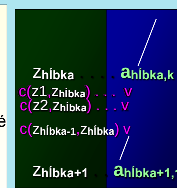
prehľadávanie s ustupovaním

prezeraj ALEBO strom do hĺbky, zľava doprava.



Algoritmus - prehľadávanie s ustupovaním

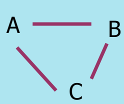
```
Backtr( hĺbka )
  For  k= 1 to n_hĺbka do
    z_hĺbka := a_hĺbka,k ;
    preveruj všetky ohraničenia c(z_i, z_hĺbka)
     s 1 ≤ i < hĺbka  pokiaľ nejaké nesplnené
        If žiadne nesplnené then
        If hĺbka = n then
          return( z1, z2, … , zn)
        Else
             hĺbka := hĺbka + 1;
             Backtr( hĺbka ) ;
             hĺbka := hĺbka - 1;
  End-For
```



Call : Backtr(1)

Je možné zafarbiť trojuholník dvoma farbami ?

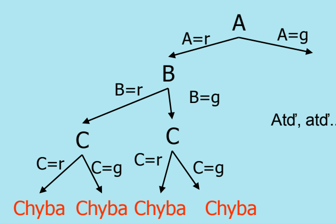
- Je možné priradiť daným uzlom a ohraničeniam hodnoty z množiny {r, g}?



- Je zrejmé, že nie!
- Ako to však dokázať pomocou metód prehľadávania ?

Je možné zafarbiť trojuholník dvoma farbami ?

- Generuj a Testuj": Zafarbi uzly v poradí A, B, C



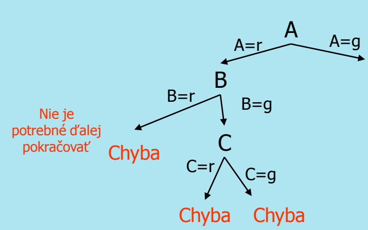
- Všetky pokusy o vygenerovanie vhodného priradenia končia neúspechom

Predčasný neúspech

- Predpokladajme, že neúplné priraďovanie zlyhá, napr. dôjde k porušeniu ohraničenia
- Akékoľvek priradenia hodnôt uzlom s doposiaľ nepriradenou hodnotou, neovplyvnia fakt, že došlo k porušeniu ohraničenia – hľadanie skončí neúspechom
- V prehľadávaní s ustupovaním:
  - len čo dôjde k porušeniu ohraničenia neúplným priradením, je možné orezať prehľadávanie

Je možné zafarbiť trojuholník dvoma farbami ?

- Naivné prehľadávanie s ustupovaním:
  - Orezanie nesprávnych neúplných priradení

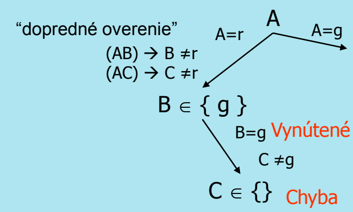


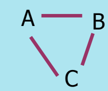
Prehľadávanie s ustupovaním a farbenie grafu

- Naivné prehľadávanie s ustupovaním je niekedy očividne neefektívne
- Po vyfarbení uzla nejakou farbou, nemôžeme už danú farbu priradiť jeho susedom
  - je teda zbytočné vytvárať potomkov tohto uzla

85

Je možné zafarbiť trojuholník dvoma farbami ?





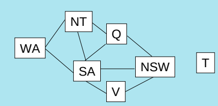
86

Dopredné overenie a problém farbenia grafu

- Po vyfarbení uzla nejakou farbou, nemôžeme už danú farbu priradiť jeho susedom
- Pred priradením hodnoty danému uzlu sa overí, ktoré hodnoty je možné danému uzlu priradiť
- Tým sa niektoré priradenia stávajú vynútenými
- Redukcia faktoru vetvenia a veľkosti stromu

87

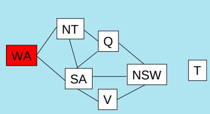
Príklad - Dopredné overenie a farbenie grafu



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

88

Príklad - Dopredné overenie a farbenie grafu

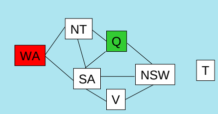


WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB

dopredné overenie odstráni hodnotu R uzlov NT a SA

89

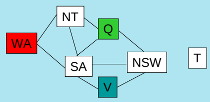
Príklad - Dopredné overenie a farbenie grafu



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	~~G~~B	G	R~~G~~B	RGB	~~G~~B	RGB

90

Príklad - Dopredné overenie a farbenie grafu



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	R~~B~~	B	~~B~~	RGB

91

Príklad - Dopredné overenie a farbenie grafu

Prázdna množina:
Priradenie {(WA ← R), (Q ← G), (V ← B)} nevedie k riešeniu

WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	GB	RGB	RGB	RGB	GB	RGB
R	B	G	RB	RGB	B	RGB
R	B	G	R~~B~~	B	~~B~~	RGB

92

Dopredné overenie

- Po vynútení hodnoty pre daný uzol je potrebné skontrolovať všetkých jeho susedov.
- Všetky hodnoty, ktoré sú s práve vynútenou hodnotou nekonzistentné, sa odstránia z domény.
- Pri odvodzovaní budú použité už iba nové hodnoty, čím sa redukuje počet uzlov, ktoré je potrebné prezrieť.

93

Dopredné overenie

- Znižuje veľkosť domény pre uzly
  - Počet možných alternatív klesá
  - Faktor vetvenia sa znižuje
- Zvláštne prípady:
  - |D| = 1 : uzlu je možné priradiť iba jednu hodnotu
    - Vynútená hodnota
  - |D| = 0 : uzlu nie je možné priradiť žiadnu hodnotu
    - Neúplné priradenie končí neúspechom
      - – orezanie riešenia a spätný chod

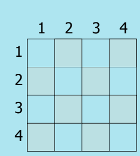
94

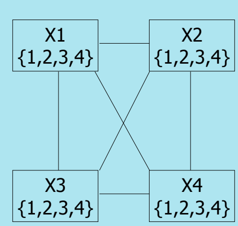
Algoritmus – Modifikované prehľadávanie s ustupovaním

CSP-BACKTRACKING(A, doména)
  if priradenie A je úplné **then return** A
  X ← vyber uzol, ktorý nie je v A
  D ← vyber usporiadanie na doméne X
  for each hodnota v in D **do**
    Pridaj (X←v) do A
    doména ← DOPREDNÉ-OVERENIE (doména, X, v, A)
    if doména pre uzol je prázdna **then return** neúspech
    riešenie ← CSP-BACKTRACKING(A, doména)
    if riešenie ≠ neúspech **then return** riešenie
  return neúspech

95

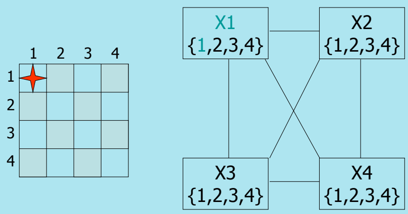
príklad: 4 dámy





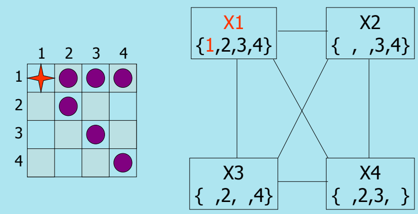
96

príklad: 4 dámy



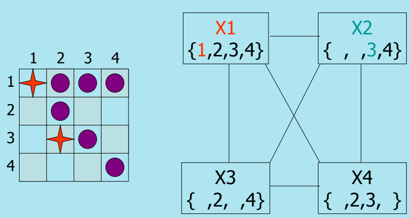
97

príklad: 4 dámy



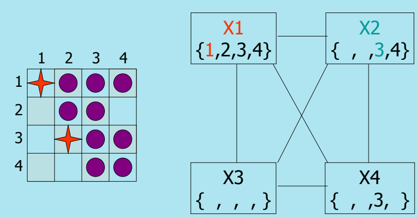
98

príklad: 4 dámy



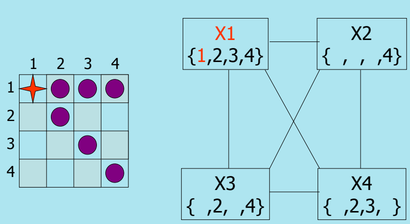
99

príklad: 4 dámy



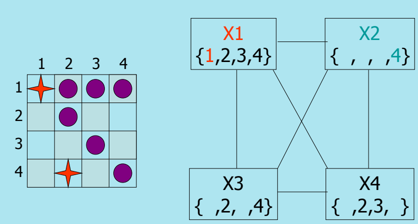
100

príklad: 4 dámy



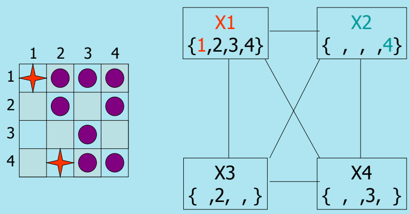
101

príklad: 4 dámy



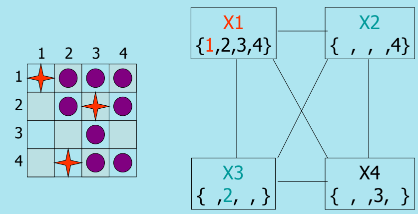
102

príklad: 4 dámy



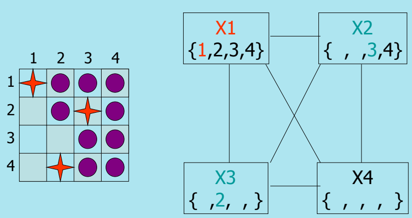
103

príklad: 4 dámy



104

príklad: 4 dámy



105

šírenie ohraničení (constraint propagation) …

… proces určovania, ako možné hodnoty jednej premennej ovplyvnia možné hodnoty iných premenných

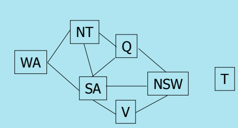
106

dopredné preverovanie (forward checking)

po tom, ako sa premennej X priradí hodnota v, pozri sa na každú nepriradenú premennú Y spojenú s X nejakým ohraničením a zruš z domény premennej Y všetky hodnoty, ktoré nie sú zlučiteľné s v

107

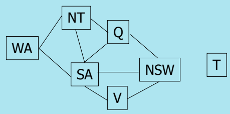
ofarbenie mapy: dopredné preverovanie



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB

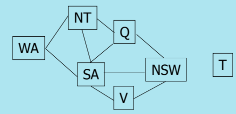
108

ofarbenie mapy: dopredné preverovanie



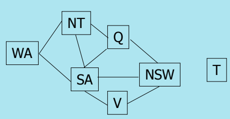
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
1: R	RGB	RGB	RGB	RGB	RGB	RGB

ofarbenie mapy: dopredné preverovanie



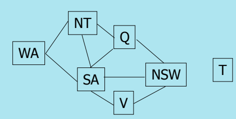
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
1: R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB

ofarbenie mapy: dopredné preverovanie



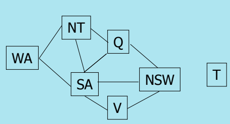
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~R~~GB	2: G	RGB	RGB	~~R~~GB	RGB

ofarbenie mapy: dopredné preverovanie



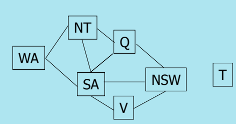
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~RG~~B	2: G	R~~G~~B	RGB	~~RG~~B	RGB

ofarbenie mapy: dopredné preverovanie



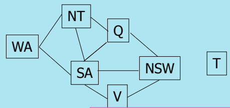
WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~RG~~B	G	R~~G~~B	RGB	~~RG~~B	RGB
R	~~RG~~B	G	R~~G~~B	3:B	~~RG~~B	RGB

ofarbenie mapy: dopredné preverovanie



WA	NT	Q	NSW	V	SA	T
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~RG~~B	G	R~~G~~B	RGB	~~RG~~B	RGB
R	~~RG~~B	G	R~~GB~~	3:B	~~RGB~~	RGB

Iné nekonzistentnosti?

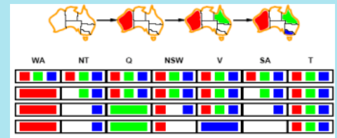


Nemožné priradenia, na ktoré dopredné preverovanie nepríde.

WA	NT	Q				
RGB	RGB	RGB	RGB	RGB	RGB	RGB
R	~~R~~GB	RGB	RGB	RGB	~~R~~GB	RGB
R	~~R~~~~G~~B	G	R~~G~~B	RGB	~~R~~~~G~~B	RGB
R	~~R~~~~G~~B	G	R~~G~~~~B~~	3:B	~~R~~~~G~~~~B~~	RGB

115

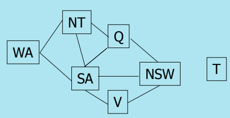
šírenie ohraničení

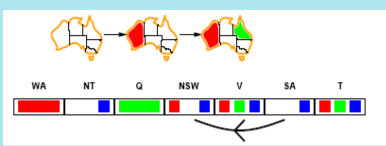


- Riešenie CSP s kombináciou heuristík a dopredného preverovania je efektívnejšie než ľubovoľný z prístupov osamote.
- Dopredné preverovanie šíri informáciu od priradených ku nepriradeným premenným, nevie však odhaliť všetky zlyhania.
  - Šírenie ohraničení opakovane vynucuje dôsledky ohraničení.

116

hranová konzistentnosť

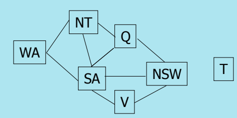


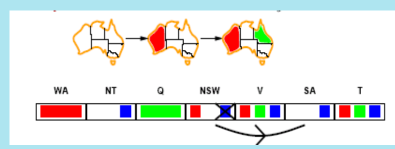


- *X → Y* je konzistentná vtedy a len vtedy ak
  pre každú hodnotu *x* premennej *X* existuje nejaká prípustná hodnota *y* premennej *Y*
- *SA → NSW* je konzistentná vtedy a len vtedy ak
  SA=modrá a *NSW=červená*

117

hranová konzistentnosť



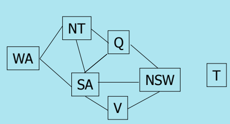


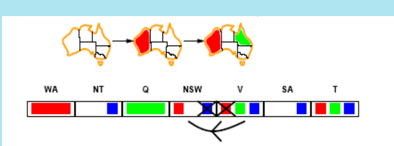
- *X → Y* je konzistentná vtedy a len vtedy ak
  pre každú hodnotu *x* premennej *X* existuje nejaká prípustná hodnota *y* premennej *Y*
- *NSW → SA* je konzistentná vtedy a len vtedy ak
  NSW=*červená* a SA=*modrá*
  NSW=modrá a *SA=???*

Hrana sa stane konzistentnou odstránením *modrej z NSW*

118

hranová konzistentnosť

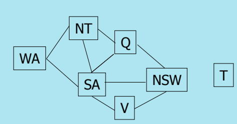


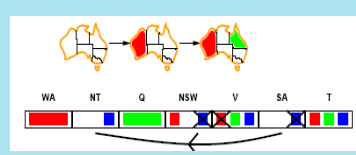


- Znovuprever susedov *NSW*!!
- *V → NSW* je konzistentná vtedy a len vtedy ak
  V=modrá a *NSW=červená*
  V=zelená a *NSW=červená*
  V=červená a *NSW=??*
  - Odstráň *červenú* z *V*

119

hranová konzistentnosť





- *SA → NT* nie je konzistentná hrana
  - a nedá sa skonzistentniť
- preverovanie hranovej konzistentnosti odhalí neúspech skôr než dopredné preverovanie

120

Šírenie ohraničení

- Predpokladajme, že pomocou dopredného overovania (DO) nájdeme pre niektorý z uzlov vynútenú hodnotu
  - Je zbytočné čakať, pokiaľ sa k danému uzlu dopracujeme pomocou prehľadávania s ustupovaním, hodnotu priradíme danému uzlu okamžite.
- Keďže priradením vynútenej hodnoty došlo k zmene, je možné vykonať znova DO
- Proces je príkladom šírenia ohraničení
  - Priradenie hodnoty jednému uzlu sa rozšíri na ostatné uzly cez ohraničenia
  - Šírenie končí v okamihu, keď už nie je možné nič odvodiť

121

šírenie ohraničení

- Skombinovaním hodnôt pre uzly a množiny ohraničení je možné eliminovať ostatné možné hodnoty
- Zvláštne prípady
  - Všetky hodnoty pre uzol sú eliminované
    - Orezanie uzla a ústup späť
  - Všetky hodnoty, okrem jednej boli pre uzol eliminované
    - Ide o vynútenú hodnotu
    - Potrebné znova vykonať rozšírenie ohraničení
- Zníženie faktoru vetvenia, hĺbky a veľkosti stromu hľadania

122

šírenie ohraničení

- Väčšina práce v CSP a programovaní ohraničení pozostáva z:
  - hľadaní výkonnejších foriem šírenia – metód redukujúcich veľkosť domény
  - efektívnejšej implementácií šírenia

123

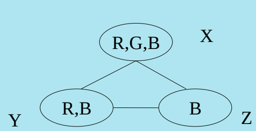
Mechanizmus šírenia ohraničení

- hranová konzistentnosť
  – hrana V_1->V_2 je hranovo konzistentná ak pre všetky x ∈ D_1 existuje y ∈ D_2 také, že (x, y) je ohodnotenie konzistentné s ohraničeniami
- na čo je to dobré?
  – Vymaž spomedzi možných hodnôt pre vrcholy/premenné V_1 a V_2 tie, ktoré kazia (hranovú) konzistentnosť

124

príklad: hranová konzistentnosť

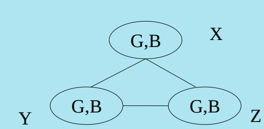
- ofarbenie grafu:
  - premenné: vrcholy X, Y, Z
  - doména: farby (R,G,B)
  - ohraničenie: If hrana(a,b), then farba(a) ≠ farba(b)
- začiatočné priradenia:



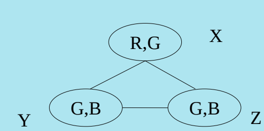
125

medze metódy hranovej konzistentnosti

- neexistuje konzistentné priradenie



- viac konzistentných priradení



127

k-konzistentnosť

- Graf (sieť) ohraničení je k-konzistentný ak pre každú množinu k-1 premenných a každé konzistentné priradenie hodnôt týmto premenným existuje konzistentné priradenie ľubovoľnej k-tej premennej.
- inými slovami: **Nech ľubovoľným *k-1 premenným sú priradené také* hodnoty z ich domén, aby všetky ohraničenia definované nad touto (*k-1)-ticou premenných boli splnené. Pre* ľubovoľnú ďalšiu *k-tu premennú je možné vybrať takú* hodnotu z jej domény, že všetky ohraničenia definované nad vzniknutou *k-ticou premenných sú splnené.***

k-konzistentnosť

- k-konzistentnosť je zovšeobecnením hranovej konzistentnosti:
- k = 2 – hranová konzistentnosť
- k = 1 – vrcholová konzistentnosť
- k = 3 – konzistentnosť po ceste: ľubovoľnú dvojicu susediacich premenných možno rozšíriť na tretiu susediacu premennú

ako zabezpečiť k-konzistentnosť?

- 1-konzistentnosť (NC):
  - stačí porovnať každú doménu Di s unárnym ohraničením Ci a odstrániť všetky nekonzistentné hodnoty z Di
- 2-konzistentnosť (AC):
  - hodnoty ľubovoľnej premennej vi v doméne Di sa porovnávajú s hodnotami inej premennej vj doméne Dj a ak pre ľubovoľnú hodnotu hm,i z Di neexistuje konzistentná hodnota z Dj (s ohľadom na Ci,j), táto hodnota hm,i sa odstráni z domény Di
- k-konzistentnosť, k>2: exponenciálna zložitosť

silná k-konzistentnosť

- Graf ohraničení je silne k-konzistentný ak je k-konzistentný a (k-1)-konzistentný a ... 2-konzistentný a 1-konzistentný t.j. ak je j-konzistentný pre všetky j z intervalu <1,k>
- načo je to dobré?
  - ak máme úlohu s ohraničeniami s n vrcholmi a vieme ju opísať grafom ohraničení, ktorý je silne n-konzistentný, tak riešenie problému sa dá nájsť bez ustupovania
  - ako to?
  - hľadáme riešenie pre nejakú premennú v_1. Len čo ho nájdeme, máme zaručené, že sa dá nájsť pre v_2, pretože graf je 2-konzistentný atď. až po n.

silná k-konzistentnosť

- časová zložitosť je parádna - *O(nd)* namiesto $O(n^2d^3)$ (zložitosť AC3)
- ale! nič nie je zadarmo. ľubovoľný algoritmus na zistenie silnej k-konzistentnosti musí byť exponenciálny v čase.
- možno hľadať „optimum“ koľko nechať na hľadanie a koľko osekávať preverovaním konzistentnosti:
  - začať s preverovaním konzistentnosti, osekávať kým sa dá alebo kým je to ešte ako tak efektívne
  - pokračovať s prehľadávaním (nutne aj s ustupovaním, pokiaľ sme sa nedosekali do silnej n-konzistentnosti; o čo sa ale kvôli zložitosti často ani nepokúšame, najčastejšie hranová konzistentnosť lebo binárny problém)

preverovanie hranovej konzistentnosti

- predspracovanie pre začatím prehľadávania
- prípadne po každom priradení
- znamená viac počítania, ale môže skôr eliminovať nekonzistentné časti stavového priestoru (než prehľadávanie)
- hrany sa musia preverovať systematicky
  - ak premenná stratí jednu hodnotu, musia sa nanovo preveriť všetky hrany do nej smerujúce

Algoritmus AC-3

```
function AC-3(csp) returns CSP s redukovanou doménou
   inputs: csp, binárne CSP s uzlami {X_1, X_2, …, X_N}
           front, front hrán
   for each X_i in {X_1, X_2, …, X_N} do
      for each X_i in {X_1, X_2, …, X_N} do
         if C(X_i, X_j) existuje then front ← (X_i, X_j)
   while front nie je prázdny do
      (X_i, X_j) ← ODSTRÁŇ-PRVÝ(front)
      if NEKONZISTENTNÉ-HODNOTY(X_i, X_j) then
         for each X_k in SUSEDIA[X_i] do
               ZARAĎ-DO-FRONTU((X_k, X_i), front)
```

```
function NEKONZISTENTNÉ-HODNOTY(X_i, X_j)
   returns: true, v prípade ak dôjde k odstráneniu hodnoty

   odstránené ← false
   for each x in DOMÉNA[X_i] do
      if nie je možné priradiť žiadnu hodnotu y z DOMÉNA[X_j],
        tak aby bolo splnené ohraničenie C(X_i, X_j) then
           odstráň x z DOMÉNA[X_i]
           odstránené ← true
   return odstránené
```

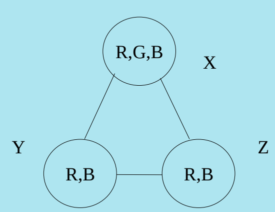
134

zložitosť C-3

- binárny problém s ohraničeniami má nanajvýš n^2 hrán
- každá hrana sa vloží do frontu najviac d razy (najhorší prípad)
  - (X, Y): len d hodnôt vrchola X na zrušenie
- konzistentnosť hrany sa dá preveriť v čase $O(d^2)$
- časová zložitosť je $O(n^2 d^3)$

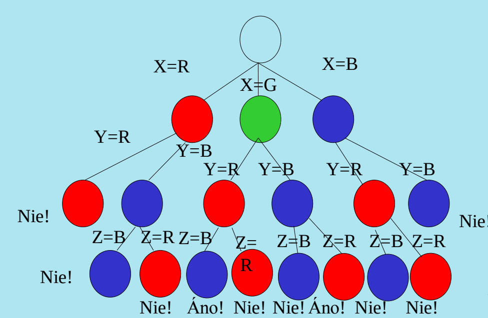
135

graf ohraničení



136

CSP ako hľadanie



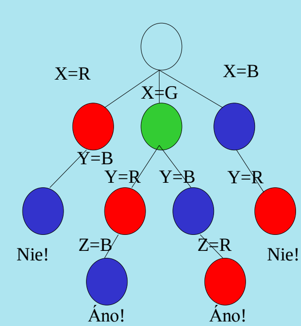
137

súvislosti ustupovania

- CSP ako hľadanie
  - hľadanie do hĺbky
    - Maximálna hĺbka = # počet premenných
  - pokračuje dovtedy, kým nie je čiastočné alebo úplné priradenie
    - ak je konzistentné: return výsledok
    - ak nie je konzistentné (porušuje nejaké ohraničenie) -> vráť sa k predchádzajúcemu priradeniu
  - premárnené úsilie
    - prehľadávajú sa aj vetvy, kde neexistuje priradenie v súlade s ohraničeniami

138

ustupovanie s dopredným preverovaním



139

Heuristické prístupy k riešeniu CSP: dynamické usporadúvanie

- otázka: ktorý uzol rozvíjať ako ďalší?
- doterajší postup:
  - – staticky: ďalší v pevne danom poradí
    - Lexikografické, zľava doprava..
  - – náhodne
- Alternatívny postup:
  - – dynamicky: vybrať “najlepší” v aktuálnom stave

140

otázky voľby

- Ako zvoliť premennú, ktorej sa bude priraďovať?
- Ako zvoliť hodnotu, ktorú priradiť?

141

Dynamické usporadúvanie uzlov

- Doteraz sa vždy vyberal ďalší uzol vetvenia podľa presne stanoveného poradia, čo viedlo k zvýšeniu pamäťových nárokov.
- Výhodnejšie je vyberať ďalší uzol vetvenia heuristicky a dynamicky v závislosti od aktuálneho stavu.

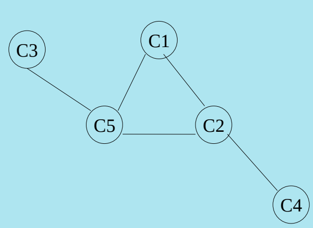
142

Dynamické usporadúvanie

- Heuristiky
  - – najohraničenejšia premenná:
    - zvoľ premennú s najmenšou aktuálnou doménou
  - – najmenej ohraničujúca hodnota:
    - zvoľ hodnotu, ktorá odstraňuje najmenej hodnôt z domén iných premenných

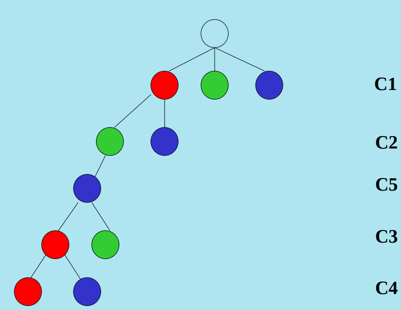
143

Dynamické usporadúvanie



144

Dynamické usporadúvanie

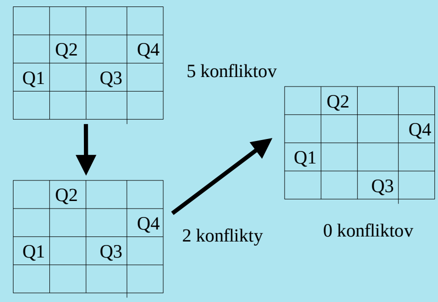


145

inkrementálne opravovanie

- začať so začiatočným úplným priradením
  - hľadať lačno
  - prvé priradenie je pravdepodobne nekonzistentné – t.j. porušuje niektoré ohraničenia
- inkrementálne premeň na platné riešenie
  - použiť heuristiku na zamenenie (iné priradenie) hodnoty ktorá spôsobuje porušenie ohraničenia
  - stratégia “min-konflikt”:
    - zmeň hodnotu tak, aby nastalo čo najmenej porušení ohraničení
    - ak sa nedá určiť inak, rozhoduj náhodne
    - zahrň do horolezeckého algoritmu

inkrementálne opravovanie



Dôležité otázky pri prehľadávaní s ústupom

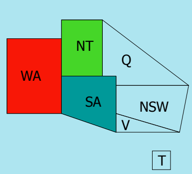
- Ktorý uzol budeme rozvíjať ako nasledujúci?
  - tzv. “Branch Variable” výber uzla na rozvíjanie
- Aké zvoliť usporiadanie potomkov jednotlivých uzlov?
  - tzv. “Branch Value” výber poradia hodnôt na priradenie
- Dobrá voľba môže významne znížiť množstvo potrebných prehľadávaní

Heuristika “Branch Variable“

- “Vyber najviac ohraničený uzol”
  - Vyber uzol s najmenšou doménou
  - Zamerané na zníženie faktoru vetvenia
- “Vyber najviac ohraničujúci uzol”
  - Vyber uzol, ktorého sa týka najviac ohraničení
  - Zamerané na generovanie rozšírení

voľba premennej

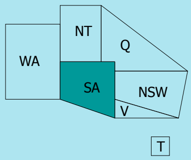
- ofarbenie mapy



voľba premennej

#1: Minimum Remaining Values (aka Most-constrained-variable heuristic or Fail First):
- najohraničenejšia premenná:
  - zvoľ premennú s najmenšou aktuálnou doménou

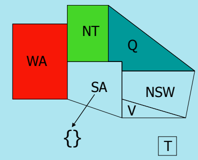
voľba premennej



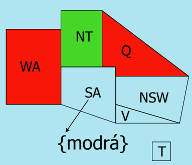
#2: heuristika najväčšieho stupňa vplyvu (degree heuristic):

zvoľ premennú účastnú v najväčšom počte ohraničení iných ešte nepriradených premenných

voľba hodnoty



voľba hodnoty



#3: Least-constraining-value heuristic:

- najmenej ohraničujúca hodnota:
  - zvoľ hodnotu, ktorá odstraňuje najmenej hodnôt z domén iných premenných

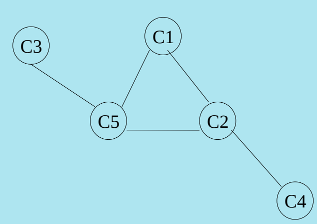
aké heuristiky pre sudoku?

- voľba premennej:
  - najohraničenejšia premenná?
  - heuristika najväčšieho stupňa vplyvu
- voľba hodnoty:
  - najmenej ohraničujúca hodnota?
- iné??

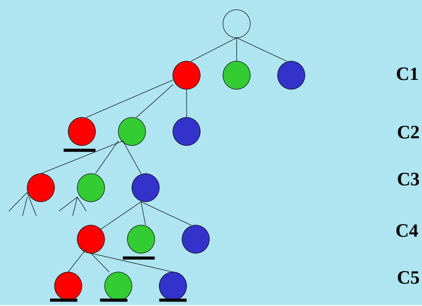
Heuristické prístupy k riešeniu CSP

- vylepšenie ustupovania
  - štandardné ustupovanie, spätný chod
    - vrátiť hľadanie do posledného priradenia
  - spätné skákanie back-jumping
    - vrátiť hľadanie do posledného priradenia, ktoré zmenšilo aktuálnu doménu
    - zmeniť priradenie, ktoré viedlo do slepého konca

Heuristické ustupovanie: spätné skákanie

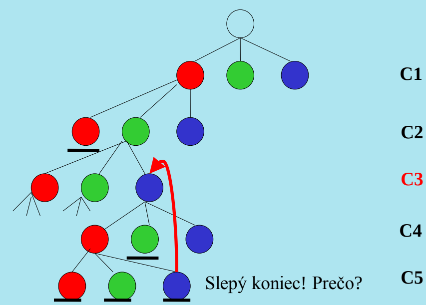


Heuristické ustupovanie: spätné skákanie



158

Heuristické ustupovanie: spätné skákanie



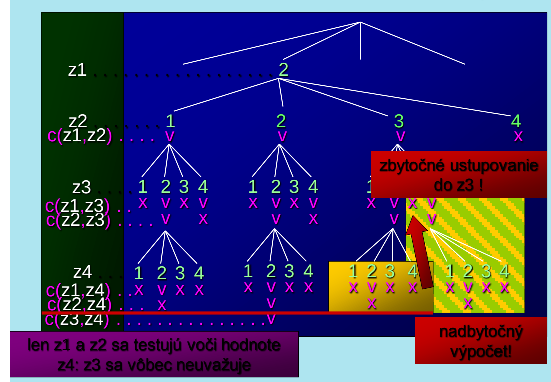
159

spätné skákanie

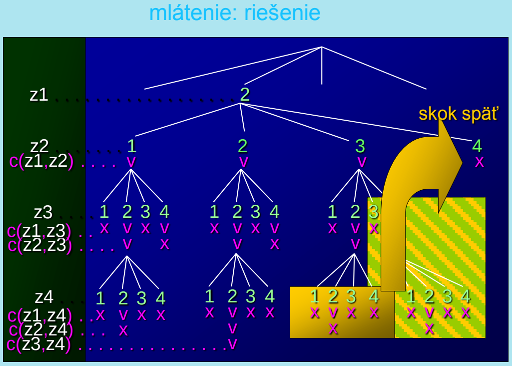
- Predchádzajúce priradenie zúžilo doménu
  - C3 = B
- zmeny v priradení medzi nemôžu ovplyvniť sleposť konca
- vrátiť sa až na C3
  - vyhnúť sa márnej práci – alternatívam na C4
- vo všeobecnosti môže byť dopredné preverovanie efektívnejšie

160

mlátenie (trashing): **príklad**: p-4-dámy:

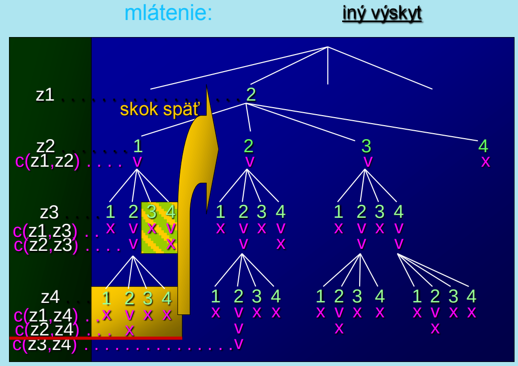


mlátenie: riešenie



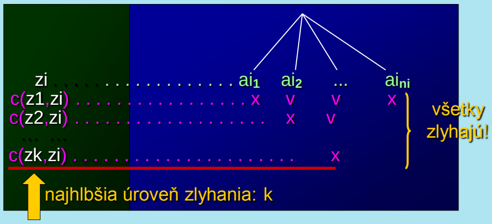
162

mlátenie: **iný výskyt**



163

spätné skákanie (backjumping) (Gaschnig'78)



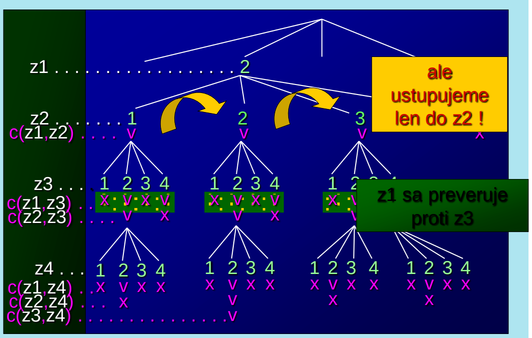
⊙ princíp:

If všetky priradenia premennej zi zlyhajú,
and c(zk, zi) je najlbšie ohraničenie spôsobujúce zlyhanie
Then backjump kvôli zmeneniu priradenia zk

algoritmus spätného skákania

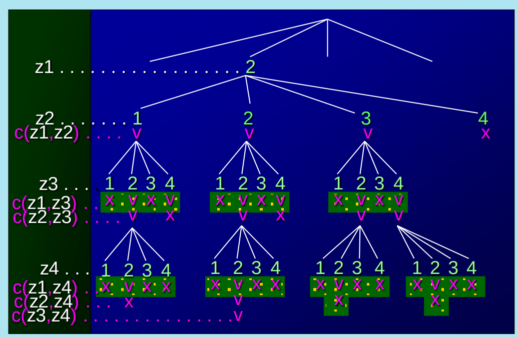
```
BackJ( hĺbka, out: spätnýskok )
  For   k= 1 to n_hĺbka do
    z_hĺbka := a_hĺbka,k ;
    preveruj všetky ohraničenia c(z_i, z_hĺbka)
      s 1 ≤ i < hĺbka pokiaľ nejaké nesplnené
    hĺbkapreverenia_k := najhlbšia preverená úroveň i;
    If žiadne nesplnené then
      If hĺbka = n then
        return( z1, z2, … , zn)
      Else
        hĺbka := hĺbka + 1;
        BackJ( hĺbka, spätnýskok);
        hĺbka := hĺbka - 1;
        If spätnýskok<hĺbka then return;
  End-For
spätnýskok := max(hĺbkapreverenia_k)
```

ďalšie nadbytočné preverovania:



166

nastáva veľmi často:

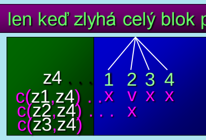


167

zahodenie vs nadbytočné previerky:

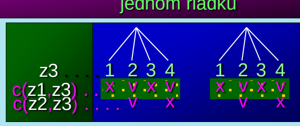
⊙ zahodenie:

len keď zlyhá celý blok previerok



⊙ nadbytočné previerky:

aj pre úspešné previerky a pre previerky len v jednom riadku



168

vyvarovanie sa nadbytočným previerkam:

⊙ 2 prístupy:

1. Tabelácia:

➔ generovanie liem (ukladanie výsledkov previerok do tabuľky)
+ aplikovanie liem (vyhľadávanie v tabuľke pri nových previerkach)

⊙ do istej miery zvyšuje rýchlosť

➔ale nevyvaruje sa naozaj nadbytočným previerkam

⊙ zvyšuje nároky na pamäť

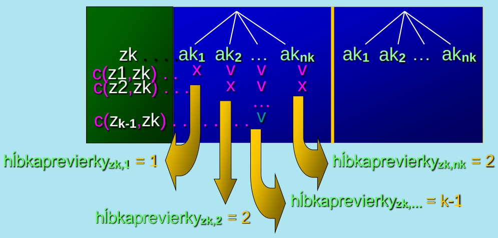
2. Spätné značkovanie (BACKMARKING):

⊙ ~ úplná úspora času (nie sú nadbytočné previerky)

⊙ len obmedzené zvýšenie pamäťovej réžie (2 polia)

spätné značkovanie (backmarking) (Gaschnig '77):

- 2 polia:



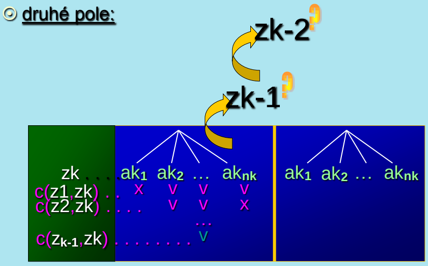
- hĺbkaprevierky(k,l):

= hĺbkaprevierky$_{zk,\,l}$.

170

spätné značkovanie

- druhé pole:



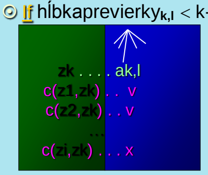
- Backup(k):

= najnižšia úroveň, ku ktorej sme sa vrátili medzi návštevami týchto 2 blokov pre zk.

171

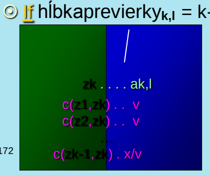
vlastnosť funkcie hĺbkaprevierky$_{k,l}$

- If hĺbkaprevierky$_{k,l}$ < k-1:



výsledok kontroly platnosti poslednej podmienky c(zi,zk) musel byť neúspech, inak by kontrolovanie pokračovalo!

- If hĺbkaprevierky$_{k,l}$ = k-1:

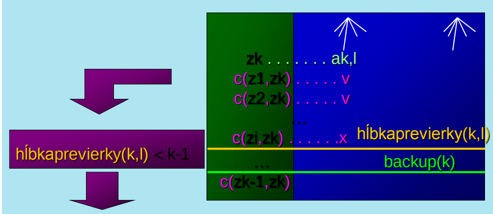


výsledok kontroly platnosti poslednej podmienky c(zk-1,zk) mohol byť úspech alebo neúspech.

172

vlastnosti hĺbkaprevierky vs Backup (1):

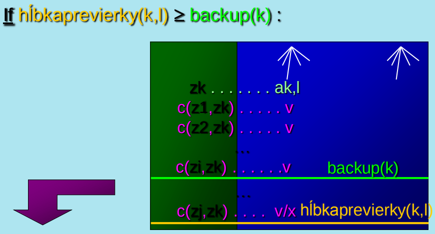
- If hĺbkaprevierky(k,l) < backup(k) :



hodnota ak,l pre premennú zk spôsobila neúspech minule a spôsobí neúspech aj teraz (v tej istej hĺbke)

vlastnosti hĺbkaprevierky versus Backup (2):

- If hĺbkaprevierky(k,l) ≥ backup(k) :



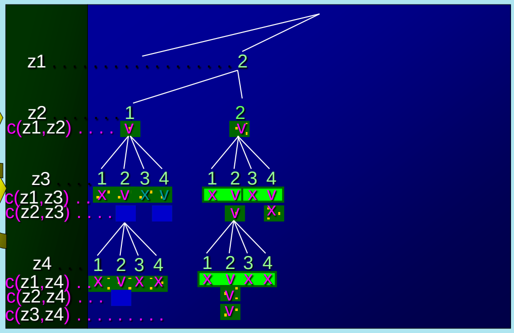
všetky preverovania premenných zm s m nižším ako backup(k) uspeli minule a uspejú znova.

algoritmus spätného značkovania:

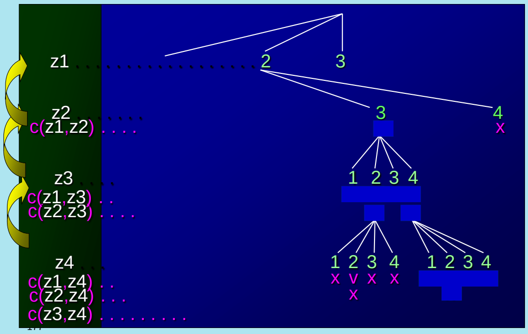
```
BackM( hĺbka, out: hĺbkaprevierky, backup )
For   k= 1 to n_hĺbka do
  z_hĺbka := a_hĺbka,k ;
  If hĺbkaprevierky(hĺbka,k) ≥ backup(hĺbka)
     *použila sa vlastnosť 1*
     preveruj všetky ohraničenia c(z_i, z_hĺbka) s
  backup(hĺbka) ≤ i < hĺbka  pokiaľ nejaké nesplnené;
     *použila sa vlastnosť 2*
    If žiadne nesplnené then
     … … …
End-For
```

175

spätné značkovanie pracuje:



pri omnoho väčšom strome:



Diskusia a výsledky:

Poznámka: začiatočné hodnoty pre hlbkaprevierky(k,l) a backup(k) sú 1

- Experimentálne výsledky:
  - ➔ pre p-4-dámy

	BT	BJ	BM
uzly	29	27	29
previerky	160	139	90

- celkovo:
  - ➔ spätné značkovanie = najlepší algoritmus (nie absolútne)
- kombinovať spätné skákanie a značkovanie? Áno!
  - ➔ ale vylepšenia sa neskombinujú.

178

Inteligentné ustupovanie (Bruynooghe - Pereira '80)

- =~ závislosťou riadené ustupovanie (Dependency-directed Backtracking) (Doyle)
  - ≠ jeden špecifický algoritmus

omnoho viac ide o: generická stratégia, je základom mnohých rôznych algoritmov

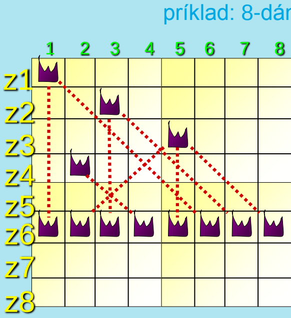
- základná myšlienka:

počas zostrojovania stromu odvoď a zapamätaj ne-dobroty "no-goods" ,
použi ne-dobroty na zlepšenie priebehu spätného chodu.

- ne-dobrota no-good je:

množina priradení hodnôt premenným, ktoré nemôžu byť súčasne časťou nijakého riešenia.

príklad: 8-dám:



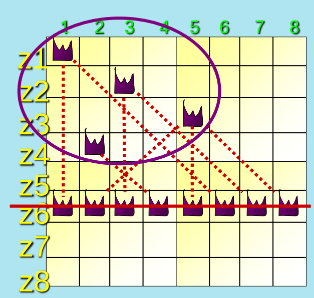
niekoľko jednoduchých ne-dobrôt, porušujúcich 1 ohraničenie:

{z1 = 1, z6 = 1}
{z3 = 5, z6 = 2}
{z2 = 3, z6 = 3}
{z4 = 2, z6 = 4}
{z3 = 5, z6 = 5}
{z1 = 1, z6 = 6}
{z2 = 3, z6 = 7}
{z3 = 5, z6 = 8}

odvodená ne-dobrota: {z1 = 1, z2 = 3, z3 = 5, z4 = 2 }

180

dôvod:

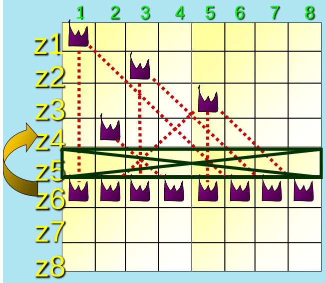


preto, lebo tieto 4 priradenia už nenechávajú žiadnu možnosť umiestnenia z6 !

odvodená ne-dobrota: {z1 = 1, z2 = 3, z3 = 5, z4 = 2 }

181

prípad spätného skoku:



nie je dôvod ustupovania cez hodnoty premennej z5, pretože dôvod zlyhania (ne-dobrota) nezahŕňa z5 !

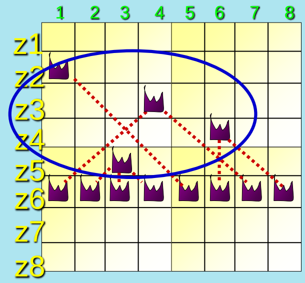
Ustupuj cez najhlbšiu premennú účastnú na dôvode zlyhania (z4).

ALE: Ktoré (základné) ne-dobroty odpamätať? Ktoré ďalšie ne-dobroty odvodzovať?

problém riadenia!

182

iný príklad:

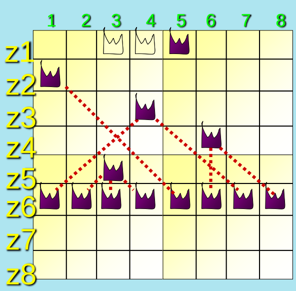


niekoľko jednoduchých ne-dobrôt
{z3 = 4, z6 = 1}
{z5 = 3, z6 = 2}
{z5 = 3, z6 = 3}
{z5 = 3, z6 = 4}
{z2 = 1, z6 = 5}
{z4 = 6, z6 = 6}
{z3 = 4, z6 = 7}
{z4 = 6, z6 = 8}

odvodená ne-dobrota:
{z2 = 1, z3 = 4, z4 = 6, z5 = 3}

183

ošetruje nadbytočnosť ako u spätného značenia:



odvodená ne-dobrota:
{z2=1, z3=4, z4=6, z5=3}

vždy, keď sa táto kombinácia vynorí opäť (pre iné hodnoty z1): nepreveruj, ustupuj!

184

Dynamické preusporiadanie prehľadávania

- = ľubovoľný algoritmus ustupovania
  +
  dynamický výber poradia premenných v strome

Účel:

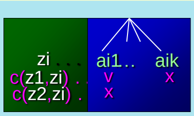
Znížiť veľkosť stromu ako dôsledok "princípu najprv-neúspech"

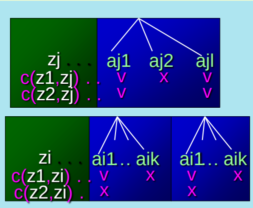
princíp najprv-neúspech:

ak dôsledkom priradenia hodnoty premennej zi je, že dôjde *pravdepodobnejšie* k zlyhaniu než priradením hodnoty premennej zj:
tak: najprv priraď premennej zi.

Účinok?

- 1. predpokladajme, že náš odhad bol správny…





hotovo

… takže získavame v každom prípade

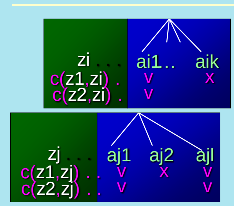
menší strom hľadania !

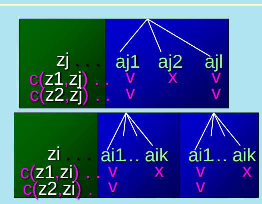
186

Účinok (2)?

- 2. predpokladajme, že náš odhad bol iba čiastočne správny …

Priradenie premennej zi nespôsobuje neúspech, ale má menej úspechov!





… stále máme zisk!

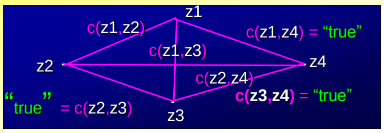
treba naprávať

187

niekoľko všeobecných heuristík prístupu najprv zlyhanie:

➔ Vyber najprv premennú s najmenšou doménou – znižuje faktor vetvenia (menej možností -> menej úspechov)

➔ Zvoľ najprv premennú s najmenším počtom netriviálnych ohraničení:



✚ Heuristika sa zakladá na danom probléme.