

Úlohy s ohraničeniami

OBSAH PREDNÁŠKY

- Hlavné aplikačné oblasti
- Základné definície
- Metódy riešenia
- Logické programovanie ohraničení
- Jazyk ECLiPSe

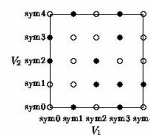
Hlavné aplikačné oblasti

- Rozvrhovanie
 - časové
 - výrobné,
 - záťaž procesorov,
 - rozvrh hodín a pod.
 - priestorové (rozmiestňovanie skladov, pobočiek, strojov ...)
- Plánovanie
- Rôzne optimalizačné úlohy
 - investičný,
 - distribučný,
 - dopravný a pod.
- Konfiguračný návrh

Základné definície

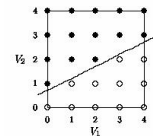
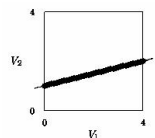
- Úloha s ohraničeniami (CSP) je vo všeobecnosti definovaná pomocou 3 množín:
 - Množina **premenných** $V = \{V_1, V_2, \dots, V_n\}$
 - Množina **domén** $D = \{D_1, D_2, \dots, D_n\}$
 - Množina **ohraničení** $C = \{C^1, \dots, C^n, C^{1,2}, \dots, C^{n-1,n}, \dots, C^{1,2,\dots,n}\}$
- Domény môžu byť:
 - **Konečné** (boolovské, ohraničené celočíselné, vymenované)
 - **Nekonečné** (neohraničené celočíselné, racionálne, reálne)
- Ohraničenia môžu byť:
 - **Unárne** (C^i), **binárne** ($C^{i,j}$), ..., **n-árne** ($C^{1,2,\dots,n}$)
 - **Positívne** alebo **negatívne** definované

Definície ohraničení



- Ohraničenia definované vymenovaním povolených (zakázaných) kombinácií hodnôt, napr. $C^{1,2} = \{(sym0, sym2), (sym0, sym3), (sym1, sym0), \dots\}$

- Ohraničenia definované predpisom
 - Napr. nerovnosť pre celočíselné premenné $C^{1,2}: V_2 > 0.5V_1 + 0.75$

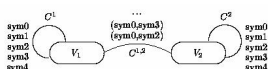
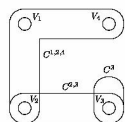
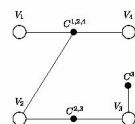


- alebo rovnosť pre reálne premenné $C^{1,2}: V_2 = 0.25V_1 + 1$

Grafická reprezentácia úloh s ohraničeniami

Sieť ohraničení

- Neorientovaný graf s dvoma typmi uzlov (pre premenné a pre ohraničenia), hrany sú možné len medzi uzlami rôznych typov
- Hypergraf s jedným typom uzlov (premenne) a hyperhranami (ohraničenia)
- Jednoduchšie znázornenie pre binárne CSP



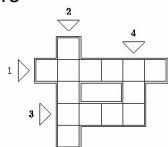
Čo znamená riešiť úlohu s ohraničeniami?

- **Priestor prehľadávania P** je definovaný ako kartézsky súčin domén všetkých premenných, t.j. $D_1 \times D_2 \times \dots \times D_n$
- **Riešiť úlohu s ohraničeniami** znamená jednu z nasledujúcich úloh:
 - Dokázať existenciu alebo neexistenciu riešenia
 - Zistiť počet riešení
 - Nájsť jedno riešenie
 - Nájsť všetky riešenia
- Ak je definovaná **funkcia nákladov** $f_c(P) \rightarrow \mathbb{R}$ nad P, potom je možné riešiť aj optimalizačnú verziu CSP, t.j.
 - Nájsť optimálne riešenie (ktoré minimalizuje f_c)

Príklad 1. (doplňovačka)

- Nasledovné *slová* sú k dispozícii pre vloženie do danej doplňovačky:

- 6 písmen: žirafa, európa
- 5 písmen: puška, kurín
- 4 písmen: auto, íver
- 3 písmen: pór, eso

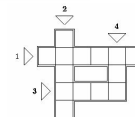
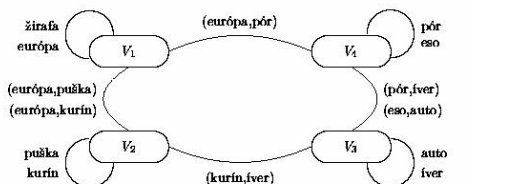


- Transformácia na úlohu s ohraničeniami:

- $V = \{V_1, V_2, V_3, V_4\}$
- $D_i = \{\text{žirafa, európa, puška, kurín, auto, íver, pór, eso}\} \quad \forall i = 1..4$
- $C = \{C^1, C^2, C^3, C^4, C^{1,2}, C^{1,4}, C^{2,3}, C^{3,4}\}$
- Napr. $C^1 = \{(\text{žirafa}), (\text{európa})\}$, $C^{1,2} = \{(\text{žirafa, žirafa}), \dots, (\text{eso}), \dots, (\text{európa, puška}), \dots\}$

Rozvrhovanie a logistika (Úlohy s ohraničeniami) Ján Paralič (www.tuke.sk/paralicj) 7

Sieť ohraničení pre príklad 1. (redukovaná verzia)

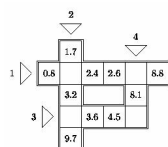


Rozvrhovanie a logistika (Úlohy s ohraničeniami) Ján Paralič (www.tuke.sk/paralicj) 8

Príklad 2. (numerická doplňovačka)

- Doplňte 4 kladné *reálne čísla* do voľných políčok tak, aby

- Súčet čísiel v každom riadku a stĺpci bol 20
- Čísla sa zvyšujú v smere zľava doprava
- Čísla sa zvyšujú v smere zhora nadol



- Transformácia na úlohu s ohraničeniami:

- $V = \{V_{12}, V_{14}, V_{32}, V_{34}\}$
- $D_i = \mathbb{R} \quad \forall i$
- $C = \{C^{12}, C^{14}, C^{32}, C^{34}, C^{12,14}, C^{12,32}, C^{32,34}, C^{14,34}\}$
- Napr. $C^{12} = \{V^{12} > 1.7, V^{12} < 2.4\}$, $C^{12,14} = \{V^{12} < V^{14} \cdot 0.8 + V^{12} + 2.4 + 2.6 + V^{14} + 8.8 = 20\}$, etc.

Rozvrhovanie a logistika (Úlohy s ohraničeniami) Ján Paralič (www.tuke.sk/paralicj) 9

Prístupy k riešeniu úloh s ohraničeniami nad konečnými doménami

- Redukčné algoritmy** (sa snažia zredukovať P, t.j. domény premenných a/alebo ohraničenia tak aby nájdenie riešenia bolo čo najjednoduchšie)
 - Zabezpečenie silnej k-konzistencie je výpočtovo náročné, ale následné nájdenie riešenia je veľmi jednoduché
- Prehľadavacie algoritmy** (prehľadávajú P za účelom nájdenia riešení)
 - Jednoduché algoritmy, ale často musia prehľadávať obrovské časti P
- Kombinované algoritmy** (kombinujú prístupy 1 a 2 v snahe využiť výhody oboch prístupov)
 - Pred každým krokom prehľadávania sa aplikuje (jednoduchší) redukčný algoritmus

Rozvrhovanie a logistika (Úlohy s ohraničeniami) Ján Paralič (www.tuke.sk/paralicj) 10

Redukčné algoritmy 1.

- k-konzistencia** ($k \in \langle 1, n \rangle$) siete ohraničení vlastne znamená konzistenciu všetkých podmnožín množiny premenných V s kardinalitou k
- Formálna definícia k-konzistencie:

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.
- Silná k-konzistencia** znamená, že sieť ohraničení je j-konzistentná pre všetky j z intervalu $\langle 1, k \rangle$

Rozvrhovanie a logistika (Úlohy s ohraničeniami) Ján Paralič (www.tuke.sk/paralicj) 11

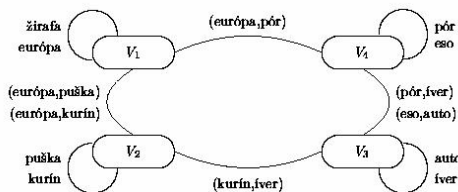
Redukčné algoritmy 2.

- 1-konzistenciu (uzlovú konzistenciu, NC) je veľmi jednoduché zabezpečiť, stačí porovnať každú doménu D_i s unárnym ohraničením C^i a odstrániť všetky nekonzistentné hodnoty z D_i
- 2-konzistencia (hranová konzistencia, AC) znamená, že hodnoty v ľubovoľnej doméne D_i sa porovnávajú s hodnotami inej premennej D_j a ak pre ľubovoľnú hodnotu v_m^i z D_i neexistuje konzistentná hodnota z D_j (s ohľadom na C^{ij}), táto hodnota v_m^i sa odstráni z domény D_i
- Existujú rôzne verzie algoritmu AC s rôznou časovou a priestorovou zložitosťou (AC2 .. AC7)
- Zložitost' algoritmov pre zabezpečenie k-konzistencie $k > 2$ je exponenciálna, preto sa používajú iba veľmi zriedkavo (niekedy sa používa 3-konzistencia nazývaná tiež konzistencia po ceste, PC)

Rozvrhovanie a logistika (Úlohy s ohraničeniami) Ján Paralič (www.tuke.sk/paralicj) 12

Redukčné algoritmy 3.

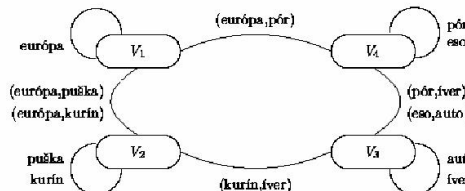
- Aplikujeme teraz algoritmus AC na úlohu z príkladu 1 pri danom poradí premenných: $V_1, V_2, V_4,$ a V_3



- Začneme teda premennou V_1 . V jej doméne pre hodnotu žirafa neexistuje hodnota v doméne premennej V_2 , ani V_4 , ktorá by bola konzistentná s príslušným binárnym ohraničením C^{12} , resp. C^{14} . Túto hodnotu môžeme preto vyradiť

Redukčné algoritmy 4.

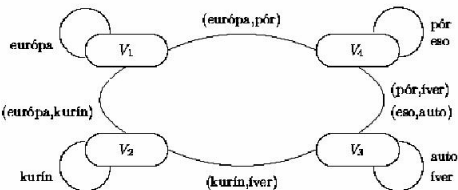
- Pokračujeme teraz premennou V_2 , v doméne ktorej zase hodnota puška nemá kompatibilnú hodnotu v doméne V_3 , ktorá by spĺňala ohraničenie C^{23}



- Hodnotu puška teda môžeme z domény D_2 odstrániť. V dôsledku toho sa nám zjednoduší aj ohraničenie C^{23} , v ktorom vypadne prípustná dvojica hodnôt (európa,puška).

Redukčné algoritmy 5.

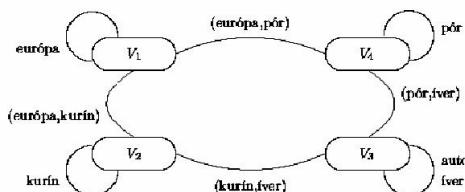
- Ďalšou premennou v poradí je V_4 . V jej doméne existuje hodnota eso, pre ktorú nemáme kompatibilnú hodnotu premennej V_3 vzhľadom na ohraničenie C^{34} .



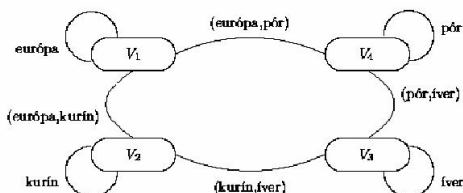
- Hodnotu eso môžeme preto z domény D_4 odstrániť. V dôsledku toho sa upraví aj ohraničenie C^{34} , z ktorého vypadne povolená dvojica (eso,auto).

Redukčné algoritmy 6.

- A nakoniec ostala premenná V_3 . V jej doméne môžeme analogicky odstrániť hodnotu auto vzhľadom na ohraničenie C^{34} .



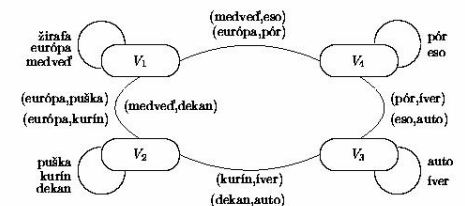
Redukčné algoritmy 7.



- Výsledkom aplikácie redukčného algoritmu AC bola redukcia P na jediný bod, ktorý zároveň predstavuje jediné riešenie danej úlohy s ohraničeniami (doplňovačka).

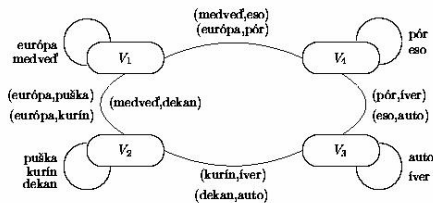
Redukčné algoritmy 8.

- Aplikujeme teraz algoritmus AC na úlohu z príkladu 1 modifikovanú takým spôsobom, že pridáme 2 nové možné slová, a síce medveď a dekan
- Poradie premenných sa nezmení: $V_1, V_2, V_4,$ a V_3



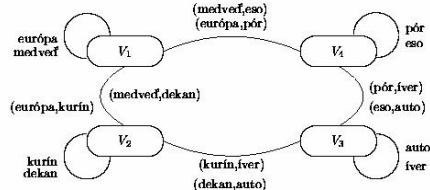
Redukčné algoritmy 9.

- V doméne premennej V_1 , môžeme z rovnakých dôvodov ako predtým (nekompatibilita s V_2 a V_4 , vzhľadom na ohraničenia C^{12} , resp. C^{14}) odstrániť hodnotu žirafa.



Redukčné algoritmy 10.

- Podobne v doméne premennej V_2 , hodnota puška opäť nemá kompatibilnú hodnotu v doméne premennej V_3 , ktorá by spĺňala ohraničenie C^{23} . Môžeme ju teda odstrániť a upraviť podľa toho aj ohraničenie C^{12} vypustením povolenej kombinácie (európa, puška).

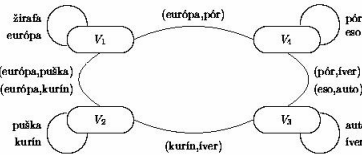
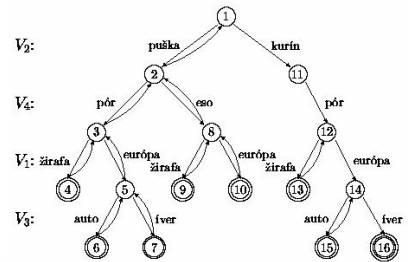


- Žiadna ďalšia redukcia použitím algoritmu AC nie je možná
- Pre CSP s n premennými vo všeobecnosti iba prísna n -konzistencia zabezpečí že následné prehľadávanie nemôže skončiť v slepej uličke!

Prehľadávacie algoritmy 1.

- Základným je algoritmus **spätného navracania** (BT), ktorý vlastne realizuje slepé prehľadávanie P do hĺbky
 - Premenným sa priradujú hodnoty sekvenčne
 - Každé ohraničenie sa testuje hneď ako je to možné (t.j. keď akonáhle sú priradené hodnoty všetkým premenným, nad ktorými je ohraničenie definované)
 - Ak je ohraničenie porušené, BT sa vráti k premennej, ktorej bola priradená hodnota naposledy a pokúsi sa jej priradiť novú hodnotu z jej domény
 - Zvykne dochádzať k „trashing“-u (opakovanie tej istej príčiny neúspechu v rôznych častiach priestoru prehľadávania)
- Lepšie (ale výpočtovo náročnejšie) prehľadávacie algoritmy sú spätné navracanie riadené závislosťami, algoritmus spätneho skoku a mnohé ďalšie

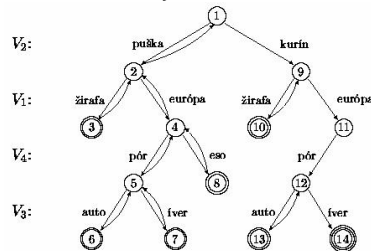
- Strom prehľadávania generovaný algoritmom BT pre dané poradie premenných (V_2, V_4, V_1, V_3) a usporiadanie hodnôt (viď redukovaná sieť ohraničení dole)



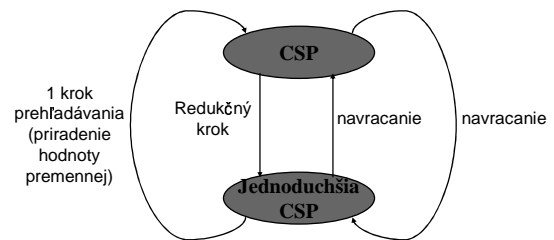
- Trashing možno pozorovať v uzloch 6 a 15: tá istá príčina neúspechu – $C^{3,4}$ nekompatibilita slov pór a auto

Prehľadávacie algoritmy 3.

- Efektívnosť procesu prehľadávania je daná nielen použitým algoritmom prehľadávania ale aj:
 - Usporiadaním premenných** (viď. napr. strom prehľadávania algoritmu BT pre príklad 1 so zmeneným poradím premenných na V_2, V_4, V_1, V_3)
 - Poradie priradzovania hodnôt z domény** (napr. ak zmeníme poradie hodnôt v D_1 a D_3 ušetríme ďalšie dva uzly v strome prehľadávania, a síce uzly s číslami 10 a 13)



Kombinované algoritmy 1.



Logické programovanie

- Deklaratívne programovanie
- Logické premenné
- Unifikácia => syntaktická manipulácia s neinterpretovanými štruktúrami
- Prehľadávajúca stratégia BT => „generuj a testuj“

Spĺňanie ohraňení

- Deklaratívna formulácia úlohy ako:
 - Premenné s počiatočnými doménami
 - Ohraňenia
- Redukčné algoritmy (určitý stupeň konzistencie)

Logické programovanie ohraňení (CLP)

- Deklaratívne programovanie
- Premenné s atribútmi => rozšírená unifikácia
- Sémantické objekty (kvôli špecifickým typom domén)
- Kombinovaná stratégia prehľadávania => „ohraňič a generuj“
- Integrované optimalizačné algoritmy
- 2 rôzne technológie

Rozvrhovanie a logistika (Úlohy s ohraňeniami) Ján Paralič (www.tuke.sk/paralic) 25

Logické programovanie ohraňení

Konečné domény

- Neúplné redukčné algoritmy
- Samostatný algoritmus pre každý typ ohraňení (lokálna propagácia)
- Optimalizácia metódou „vetvenia a medzí“

Nekonečné domény

- Úplné redukčné algoritmy
- Jeden algoritmus pre celú množinu ohraňení
- Optimalizácia použitím simplexového algoritmu

Rozvrhovanie a logistika (Úlohy s ohraňeniami) Ján Paralič (www.tuke.sk/paralic) 26

Jazyk ECLiPSe

- ECLiPSe (ECLiPSe Common Logic Programming System) je logický programovací jazyk (rozšírený Prolog) ktorého cieľom je poskytovať platformu pre integráciu rôznych rozšírení logického programovania, najmä logické programovanie ohraňení (CLP)
- Je postavený na inkrementálnom kompilátore, ktorý kompiluje zdrojový kód v ECLiPSe do kódu podobného WAM a emulátore tohoto abstraktného kódu
- Ďalšie informácie: <http://www.icparc.ic.ac.uk/eclipse>
- Existuje aj diskusná skupina používateľov ECLiPSe (eclipse-users@icparc.ic.ac.uk). Žiadosti o zaradenie, resp. Vyraďenie z diskusnej skupiny adresujte na majordomo@icparc.ic.ac.uk
- Existujú 2 spôsoby spúšťania programov v ECLiPSe:
 - TkEclipse poskytuje interaktívne grafické používateľské rozhranie ku kompilátoru a celému systému ECLiPSe.
 - DosEclipse poskytuje tradičný príkazový riadok
- Manuály sú prístupné v HTML formáte
- ECLiPSe vo verzii 5.3 si môžete stiahnuť pre vlastnú potrebu z <http://neuron.tuke.sk/~paralic/eclipse5.3>

27

Príklad 1 – riešenie v jazyku ECLiPSe

```

:- lib(fd).

doplnovacka(Zoznam) :-
    % PREMENNÉ
    Zoznam = [V1,V2,V3,V4],
    % DOMAINS
    V1 :: [žirafa,európa],
    V2 :: [puška,kurín],
    V3 :: [auto,íver],
    V4 :: [por,esol],
    % OHRANICENIA
    V1 #=> európa #<=> (V2 #=> puška #\ V2 #=> kurín), %C12
    V2 #=> kurín #<=> V3 #=> íver, %C23
    V1 #=> európa #<=> V4 #=> por, %C14
    ((V3 #=> íver #<=> V4 #=> por) #\
    (V3 #=> auto #<=> V4 #=> eso)), %C34
    % PREHLADAVANIE
    labeling(Zoznam).
  
```

Rozvrhovanie a logistika (Úlohy s ohraňeniami) Ján Paralič (www.tuke.sk/paralic) 28

Príklad 2 – riešenie v jazyku ECLiPSe

```

:- lib(clpr).

numericka_doplnovacka(Zoznam) :-
    % PREMENNÉ
    Zoznam = [V12,V14,V23,V34],
    % OHRANICENIA
    {
        1.7 < V12, V12 < 2.4, % C12
        2.6 < V14, V14 < 8.1, % C14
        3.2 < V23, V23 < 3.6, % C23
        8.1 < V34, % C34
        0.8 + V12 + 2.4 + 2.6 + V14 + 8.8 = 20, % C12,14
        1.7 + V12 + 3.2 + V23 + 9.7 = 20, % C12,23
        V23 + 3.6 + 4.5 + V34 = 20, % C23,34
        V14 + 8.1 + V34 = 20 % C14,34
    }.
  
```

Rozvrhovanie a logistika (Úlohy s ohraňeniami) Ján Paralič (www.tuke.sk/paralic) 29

Príklad 2 – výsledky

```

?- numericka_doplnovacka(Zoznam).
Zoznam = [V12, V14, V23, V34]
Yes (0.00s cpu)

% Linear constraints:
{
    V12 < 2.2,
    V14 = 5.3999999999999986 - V12,
    V23 = 5.4 - V12,
    V34 = 6.5 + V12,
    V12 > 1.8000000000000003
}

?- numericka_doplnovacka([V12,
    V14, V23, V34]), V12 = 2.0.
V14 = 3.3999999999999986
V23 = 3.4000000000000004
V34 = 8.5
V12 = 2.0
Yes (0.00s cpu)
  
```

To znamená, že existuje nekonečne veľa riešení, V12 je nezávislá premenná a musí byť z intervalu (1.8, 2.2). Nech napr. V12 = 2.0

Rozvrhovanie a logistika (Úlohy s ohraňeniami) Ján Paralič (www.tuke.sk/paralic) 30

Príklad 2 – riešenie s absolútnou presnosťou

```
:- lib(clpq).

Numericka_doplnovacka(Zoznam) :-
    % PREMENNE
    Zoznam = [V12,V14,V23,V34],
    % OHRANICENIA
    {
        4/5 < V12, V12 < 12/5, 17/10 < V12, V12 < 16/5, % C12
        13/5 < V14, V14 < 44/5, V14 < 81/10, % C14
        V23 > 16/5, V23 < 18/5, V23 < 97/10, % C23
        V34 > 9/2, V34 > 81/10, % C34
        4/5 + V12 + 12/5 + 13/5 + V14 + 44/5 = 20, % C12,14
        17/10 + V12 + 16/5 + V23 + 97/10 = 20, % C12,23
        V23 + 18/5 + 9/2 + V34 = 20, % C23,34
        V14 + 81/10 + V34 = 20 % C14,34
    }.

```

Rozvrhovanie a logistika (Úlohy s ohraničeniami) Ján Paralič (www.tuke.sk/paralic) 31

Príklad 2 – výsledky pre racionálne domény

```
?- numericka_doplnovacka(Zoznam). To znamená, že existuje nekonečne
Zoznam = [V12, V14, V23, V34] veľa riešení, V12 je nezávislá
Yes (0.00s cpu) premenná a musí byť z intervalu
(9/5, 11/5), nech napr. V12 = 2

% Linear constraints:
{
    V12 < 11 / 5, % - numericka_doplnovacka([2, V14,
    V14 = 27 / 5 - V12, V23, V34]).
    V23 = 27 / 5 - V12, V14 = 17 / 5
    V34 = 13 / 2 + V12, V23 = 17 / 5
    V12 > 9 / 5, V34 = 17 / 2
}
Yes (0.01s cpu)

```

Rozvrhovanie a logistika (Úlohy s ohraničeniami) Ján Paralič (www.tuke.sk/paralic) 32

Príklad 2 – optimalizačná verzia (1)

```
:- lib(clpr).

numericka_doplnovacka(Zoznam, Vazeny_sucet) :-
    % PREMENNE
    Zoznam = [V12,V14,V23,V34],
    % KRITERIALNA FUNKCIA
    Vazeny_sucet = 5*V12 + 3*V14 + 2*V2 + V34,
    % OHRANICENIA
    {
        1.7 < V12, V12 < 2.4, % C12
        2.6 < V14, V14 < 8.1, % C14
        3.2 < V23, V23 < 3.6, % C23
        8.1 < V34, % C34
        0.8 + V12 + 2.4 + 2.6 + V14 + 8.8 = 20, % C12,14
        1.7 + V12 + 3.2 + V23 + 9.7 = 20, % C12,23
        V23 + 3.6 + 4.5 + V34 = 20, % C23,34
        V14 + 8.1 + V34 = 20 % C14,34
    },
    % OPTIMALIZACIA
    maximize(Vazeny_sucet).

```

Rozvrhovanie a logistika (Úlohy s ohraničeniami) Ján Paralič (www.tuke.sk/paralic) 33

Príklad 2 – optimalizačná verzia (2)

```
?- numericka_doplnovacka(L,Suma). Namiesto volania predikátu:
No (0.00s cpu) maximize(Vazeny_sucet)

použijeme volanie:
sup(Vazeny_sucet, Supremum)
a výsledok bude:

?- numericka_doplnovacka(L, Sup).
L = [V12, V14, V23, V34]
Sup = 35.699999999999996
Yes (0.01s cpu)

% Linear constraints:
{
    V14 = 5.3999999999999996 - V12,
    V23 = 5.4 - V12,
    V34 = 6.5 + V12,
    V12 <= 2.2,
    V12 >= 1.8000000000000003
}

```

Rozvrhovanie a logistika (Úlohy s ohraničeniami) Ján Paralič (www.tuke.sk/paralic) 34

Príklad 3 – Investičná úloha

Spoločnosť by chcela zrealizovať päť rôznych investičných akcií (1 až 5) s danými nákladmi a očakávanými ziskami (viď. tabuľka) v priebehu nasledujúceho roka. Ale existuje limit investícií pre nasledujúci rok vo výške 130 peňažných jednotiek. Ktoré investičné akcie majú byť vybrané, aby sa maximalizoval ročný zisk?

Investičná akcia	1	2	3	4	5
Náklady	40	70	30	50	20
Očakávaný ročný zisk	4	5	2	3	1

Rozvrhovanie a logistika (Úlohy s ohraničeniami) Ján Paralič (www.tuke.sk/paralic) 35

Príklad 3 – riešenie v jazyku ECLiPSe

```
:- lib(fd).

investicny(Investicie) :-
    % PREMENNE
    Investicie = [I1, I2, I3, I4, I5],
    % DOMENY
    Investicie :: {0,1},
    % OHRANICENIE
    40*I1 + 70*I2 + 30*I3 + 50*I4 + 20*I5 #<= 130,
    % KRITERIALNA FUNKCIA
    4*I1 + 5*I2 + 2*I3 + 3*I4 + I5 #=: Zisk,
    % OPTIMALIZACIA
    min_max( (labeling(Investicie), write(Investicie), write(' '), ~Zisk)

```

Program je možné napísať aj všeobecne tak, aby riešil ľubovoľnú investičnú úlohu veľkosti N.

Rozvrhovanie a logistika (Úlohy s ohraničeniami) Ján Paralič (www.tuke.sk/paralic) 36

Príklad 3 – výsledky

```
?- investicny(Investicie).  
Investicie = [1, 1, 0, 0, 1]  
Yes (0.02s cpu)  
  
[0, 0, 0, 0, 0] Found a solution with cost 0  
[0, 0, 0, 0, 1] Found a solution with cost -1  
[0, 0, 0, 1, 0] Found a solution with cost -3  
[0, 0, 0, 1, 1] Found a solution with cost -4  
[0, 0, 1, 1, 0] Found a solution with cost -5  
[0, 0, 1, 1, 1] Found a solution with cost -6  
[0, 1, 0, 1, 0] Found a solution with cost -8  
[1, 0, 1, 1, 0] Found a solution with cost -9  
[1, 1, 0, 0, 1] Found a solution with cost -10
```