2 Problémy s ohraničeniami

Ako ilustračného reprezentanta kombinatorického problému s ohraničeniami nad konečnými doménami použijeme známy hlavolam Sudoku, ktorého jedna inštancia je zobrazená na Obrázku 1. Hlavolam pozostáva z mriežky vytvorenej deviatimi riadkami, deviatimi stĺpcami a deviatimi 3x3 štvorcami. Spolu obsahuje 81 pozícií (pozícia 11 reprezentuje ľavú hornú a 99 zase pravú dolnú pozíciu – prvá cifra značí riadok a druhá zase stĺpec), pričom niektoré pozície sú obsadené zatiaľ čo iné sú voľné. Cieľom je na každú voľnú pozíciu priradiť jednu číslicu z {1, 2, 3, 4, 5, 6, 7, 8, 9} tak, aby v žiadnom riadku, stĺpci ani štvorci neboli dve rovnaké číslice.

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

Obrázok 1. Ilustračný problém s ohraničeniami nad konečnými doménami.

Z formálneho hľadiska je problém s ohraničeniami vo všeobecnosti definovaný pomocou troch množín:

- množina premenných $V = \{V_1, V_2, ..., V_n\}$
- množina domén $D = \{D_1, D_2, ..., D_n\}$
- množina ohraničení $C = \{C_1, ..., C_n, C_{1,2}, ..., C_{n-1,n}, ..., C_{1,2,...,n}\}$.

Premenné reprezentujú tie udalosti, stavy atď., hodnoty ktorých sú relevantné pre riešenie konkrétneho problému. V príklade na Obrázku 1 je najprirodzenejšou reprezentáciou taká, keď každá pozícia mriežky je reprezentovaná samostatnou premennou. Výsledkom bude súbor 81 premenných, kde V_{11} reprezentuje pozíciu 11 a V_{99} zase pozíciu 99.

Každá doména $D_i \in D$ zodpovedá premennej $V_i \in V$ a vlastne definuje možné hodnoty, ktoré premenná V_i môže nadobúdať. V našom prípade sú použité konečné domény enumeračného typu. Všetky domény sú rovnaké – množina deviatich číslic 1 až 9.

Nad premennými z množiny V je definovaná množina ohraničení. Indexy jednotlivých ohraničení označujú árnosť týchto ohraničení a premenné, nad ktorými tieto ohraničenia sú definované. Tak napr. ohraničenie $C_{i,j}$ je binárnym ohraničením nad premennými V_i a V_j.

Definované ohraničenia stanovujú prípustnosť hodnôt premenných v rámci nejakého kontextu. Týmto kontextom sú hodnoty iných premenných (počet týchto premenných závisí

od árnosti ohraničení). Napríklad, ak nejaké ohraničenie je definované nad podmnožinou premenných, potom toto ohraničenie pre každú premennú z tejto podmnožiny stanovuje prípustnosť priradenia hodnoty z jej domény v kontexte tvorenom hodnotami ostatných premenných danej podmnožiny. Jedinou výnimkou sú unárne ohraničenia, pri ktorých nie je potrebné brať do úvahy hodnoty iných premenných.

Pre skupinu n premenných je teda možné definovať unárne, binárne, ... až n-árne ohraničenia. Celkový počet ohraničení, ktoré je možné definovať pre n premenných je

$$\sum_{i=1}^{n} \binom{n}{i} \tag{1}$$

pričom i označuje árnosť ohraničení. Je teda zrejmé, že je možné definovať až n unárnych ohraničení a jedno n-árne ohraničenie. Maximálny počet binárnych ohraničení je daný vzťahom $(n^2 - n)/2$. Okrem n-árneho ohraničenia všetky ostatné sú definované iba nad nejakou podmnožinou premenných (premenné z tejto podmnožiny sa označujú ako relevantné voči príslušnému ohraničeniu). V našom prípade teda je možných až 81 unárnych ohraničení, 3240 binárnych, 85320 ternárnych ohraničení, atď.

Všetky ohraničenia sú definované nad podmnožinami premenných originálnej množiny a obmedzujú kombinácie hodnôt, ktoré premenné daných podmnožín môžu nadobudnúť. Extrémnymi prípadmi sú ohraničenia, ktoré nepripúšťajú žiadne alebo naopak pripúšťajú všetky možné kombinácie hodnôt premenných, nad ktorými sú tieto ohraničenia definované. V prvom prípade nie je možné nájsť také riešenie, pre ktoré by takéto extrémne ohraničenie bolo splnené. V druhom prípade je také ohraničenie v množine ohraničení zaradené iba formálne a možno ho z tejto množiny vypustiť.

Pre reprezentáciu nejakého ohraničenia je možné použiť vymenovanie jednotlivých kombinácií hodnôt (či už povolených alebo zakázaných) alebo ohraničenie možno zadať nejakým predpisom, ktorý pre každú kombináciu hodnôt premenných umožňuje rozhodnúť, či daná kombinácia porušuje dané ohraničenie alebo nie. To, ktorý tvar je využitý, závisí od konkrétnej úlohy a spôsobu práce s ohraničeniami. Náš demonštračný príklad priamo zo svojej definície ponúka 27 rôznych 9-árnych ohraničení (jedno pre každý z deviatich riadkov, stĺpcov a štvorcov). Aj keď je možné takéto ohraničenie reprezentovať vymenovaním prípustných kombinácií hodnôt relevantných premenných, bolo by to nepraktické pre počet týchto kombinácií (najmenší počet by bol pre ôsmy stĺpec – 24 prípustných možností, najväčší pre tretí riadok – 40320 prípustných možností).

V praktických úlohách sa vyskytujú ohraničenia rôznych árností. Pretože však unárne ohraničenia je možné zohľadniť príslušnou redukciou domén jednotlivých premenných (resp. naopak je ich možné rozšíriť na binárne ohraničenia) a ohraničenia vyšších árností môžu byť redukované na binárne ohraničenia (aj keď niekedy za cenu zavedenia nových pomocných premenných) [1] [39], väčšina prác sa zaoberá práve binárnymi problémami s ohraničeniami – problémami, v ktorých všetky ohraničenia sú definované najviac nad dvoma premennými.

Použitý ilustračný problém používa špeciálny typ ohraničení – „alldiff“ ohraničenia (všetky premenné v rámci takéhoto ohraničenia musia mať navzájom rôzne hodnoty). Pri tomto type sa 9-árne ohraničenie dá nahradiť 84 ternárnymi alebo 36 binárnymi ohraničeniami rovnakého typu (po jednom pre každý výber troch alebo dvoch premenných z daných deviatich relevantných premenných).

Pri reprezentácii predpisom je možné ohraničenie použiť iba na testovanie, či nejaká kombinácia hodnôt premenných je z hľadiska daného ohraničenia prípustná alebo nie. Príkladom môže byť ternárne ohraničenie: „tri premenné z toho istého riadku (resp. stĺpca či štvorca) musia mať navzájom rôzne hodnoty“. Takýmto ohraničením môže byť napr. $C_{12,13,22}$. Reprezentácia vymenovaním umožňuje nielen iba testovať ale aj použiť ohraničenie generatívnym spôsobom – priamo získať kombinácie hodnôt relevantných premenných, ktoré sú validné z pohľadu daného ohraničenia. Príkladom môže byť binárne ohraničenie $C_{21,31}$ pri vymenovaní majúce tvar množiny {(1,2),(1,3), ..., (9,7), (9,8)} so 72 dvojicami.

Okrem ohraničení na rôznosť hodnôt kombinácií premenných problém ohraničuje priamo niektoré premenné. Toto reprezentujú unárne ohraničenia dané vymenovaním prípustných hodnôt napr. C_{13}={8}.

Keď úloha pre niektoré premenné resp. kombinácie premenných nedefinuje ohraničenia priamo, je možné takéto ohraničenia formálne zaviesť. Aby takéto ohraničenie nemenilo definíciu úlohy, povoľuje všetky možné kombinácie hodnôt relevantných premenných. Tak je možné zaviesť napr. $C_{31,42}$={(1,1), ..., (9,9)} s 81 dvojicami hodnôt, C_{11}={1, ..., 9} s deviatimi hodnotami a $C_{31,32,42}$ so 729 trojicami hodnôt.

Keďže je potrebné splniť všetky ohraničenia (teda aj unárne), je možné znížiť komplexnosť problému redukciou domén jednotlivých premenných. Po tejto redukcii premennej V_i bude zodpovedať redukovaná doména, ktorá vznikne z pôvodnej domény vypustením všetkých prvkov nespĺňajúcich unárne ohraničenie viažúce sa na premennú V_i.

Táto redukcia sa môže rozšíriť aj na definície ohraničení. Ak binárne ohraničenie povoľuje nejakú takú kombináciu hodnôt, ktorá obsahuje hodnotu, porušujúcu unárne ohraničenie viažúce sa k príslušnej premennej, potom je možné toto binárne ohraničenie „sprísniť“ a danú kombináciu hodnôt nepovoliť. Takýmto spôsobom dochádza k zvyšovaniu prísnosti ohraničení (danej pomerom počtu kombinácií hodnôt povolených daným ohraničením voči všetkým možným kombináciám hodnôt).

Napríklad keďže unárne ohraničenie C_{13} obsahuje iba jednu hodnotu 8 zatiaľ čo binárne ohraničenie $C_{12,13}$ obsahuje 72 dvojíc, je možné $C_{12,13}$ sprísniť iba na osem prípustných dvojíc $C_{12,13}$ = {(1,8), (2, 8), ..., (7,8), (9,8)}. Podobne je možné sprísniť na osem dvojíc aj $C_{12,22}$. A následne ternárne ohraničenie $C_{12,13,22}$ možno na základe $C_{12,13}$ a $C_{12,22}$ sprísniť na sedem prípustných trojíc (ak by bolo reprezentované vymenovaním a nie predpisom).

Obidva uvedené prípady (vymenovanie a predpis) ilustrujú explicitný tvar ohraničenia. Ohraničenie však môže byť vyjadrené aj implicitným spôsobom – samotné ohraničenie nie je priamo uvedené ale je reprezentované iba prostredníctvom iných ohraničení, z ktorých môže byť indukované. Príkladom je $C_{41,32}$. Keďže zahŕňa dve pozície v rozličných stĺpcoch, riadkoch aj štvorcoch, nijako neohraničuje hodnoty na daných dvoch pozíciách. Avšak napr. ohraničenia C_{22}, $C_{22,32}$, C_{15}, $C_{14,15}$ indukujú neprípustnosť dvojíc (7,...) a (...,5) pre $C_{41,32}$ ktoré tým pádom by malo povoľovať iba 64 dvojíc. Takúto indukciu je možné (aspoň čiastočne) urobiť pri reprezentácii problému, ale zvyčajne sa to nerobí a prenecháva sa to príslušným metódam pre riešenie úloh s ohraničeniami.

Ilustračný príklad bude teda reprezentovaný tak, že unárne ohraničenia budú dané podľa Obrázku 2, binárne ohraničenia budú reprezentované vymenovaním povolených dvojíc, pričom budú sprísnené v závislosti od obsahu unárnych ohraničení, a ternárne a viacárne ohraničenia budú dané predpisom.

Domény premenných vlastne definujú priestor. Je to vlastne priestor všetkých potenciálnych riešení (kandidátov riešení) – všetkých možných kombinácií hodnôt z domén premenných. Ak každá z domén bude reprezentovaná ako jedna súradnica tohto priestoru, potom každý bod tohto priestoru bude reprezentovať jednu kombináciu hodnôt premenných. Tento priestor takto vznikne ako kartézsky súčin domén premenných – ako množina všetkých kombinácií hodnôt jednotlivých premenných.

3 Algoritmické spĺňanie ohraničení

Táto trieda metód bola v minulosti označovaná rôznymi názvami napr. *konzistentné značkovanie* (consistent labeling) či *spĺňajúce priradenia* (satisficing assignment). Napokon sa pre ňu vžilo označenie *spĺňanie ohraničení* (constraint satisfaction).

Jednou z prvých aplikačných domén sa stala interaktívna grafika, kde prvým systémom bol Sutherlandov Sketchpad [35] začiatkom šesťdesiatych rokov. Jeho pokračovateľom je Borningov ThingLab [5]. Tieto systémy dovoľovali používateľovi kresliť a manipulovať s geometrickými obrázkami, podliehajúcimi ohraničeniam, po displeji počítača.

Prvé práce spojené s oblasťou algoritmov momentálne zahŕňaných pod týmto spoločným názvom siahajú do polovice sedemdesiatych rokov. Popudom k rozvoju v tejto oblasti sa stali algoritmy používané pre interpretáciu trojrozmerných scén na základe dvojrozmerných obrazov (bolo potrebné zohľadňovať ohraničenia pri interpretácii čiarových obrazov – napr. trojrozmerná interpretácia nejakej čiary ako hrany musí byť rovnaká na oboch jej koncoch). Vtedajší študent MIT AI Lab (Laboratória pre umelú inteligenciu na Massachussetskom technologickom inštitúte) David Waltz si uvedomoval kombinatorickú explóziu znižujúcu použiteľnosť vtedy známych interpretačných metód a preto pre jej zmenšenie vytvoril filtračný algoritmus na odstránenie nekonzistentných interpretácií [38].

Ďalší ranný vývoj v tejto oblasti je spojený s takými menami ako Eugene C. Freuder, John Gaschnig, Robert M. Haralick, Alan K. Mackworth a Ugo Montanari [23] [13] [17]. Došlo k abstrahovaniu vlastností algoritmov, k ich oddeleniu od aplikácií a k formovaniu oblasti do takého tvaru, v akom je známa dnes. Popisy základných algoritmov pre spĺňanie ohraničení je možné nájsť napr. v prehľadových publikáciách [22] a [10]. Z on-line dostupných rozsiahlejších publikácií sú k dispozícii [36] (v angličtine) a [25] (v slovenčine).

Jednou z tried algoritmov sú *konzistenčné algoritmy* (consistency enforcing algorithms). Podstatou týchto algoritmov je orezanie priestoru kandidátov riešení na nejaký jeho podpriestor. Ak úloha má jedno alebo viac riešení, tak tento výsledný podpriestor v ideálnom prípade bude obsahovať iba také body, ktorých každá súradnica je súčasťou aspoň jedného riešenia (ak úloha má iba jedno riešenie, tak tento podpriestor bude obsahovať iba jeden bod). Ak však úloha nemá riešenie ktoré by splnilo všetky definované ohraničenia, tak priestor kandidátov je možné redukovať na prázdny podpriestor.

Samotná redukcia je založená na filtrácii. Pri filtrácii sa vyhľadávajú také hodnoty z domén premenných, ktoré nie sú súčasťou žiadneho riešenia. V doménach premenných sú ponechávané iba hodnoty, ktoré sú konzistentné s definovanými ohraničeniami.

V procese filtrácie môže dochádzať nielen k redukcii domén ale aj k redukcii ohraničení, keď ohraničenie je sprísňované – nejaká kombinácia hodnôt premenných (ktoré sú relevantné pre dané ohraničenie) je označená ako neprípustná. Obidva tieto typy redukcie sú navzájom

previazané. Zakázanie nejakej kombinácie hodnôt môže viesť k zbytočnosti niektorých hodnôt v doménach premenných alebo naopak zakázanie nejakej hodnoty v doméne nejakej premennej implikuje neprípustnosť všetkých kombinácií hodnôt v ktorých daná hodnota vystupuje.

Podobne môže dochádzať k redukcii medzi ohraničeniami rôznych árností. Zakázanie nejakej kombinácie i premenných (sprísnenie i-árneho ohraničenia) môže mať za následok sprísnenie (i+j)-árneho ohraničenia (vypustenie všetkých kombinácií obsahujúcich ako svoju súčasť danú kombináciu i premenných). Alebo sprísnenie (i+j)-árneho ohraničenia môže podnietiť vypustenie všetkých tých kombinácií z i-árneho ohraničenia, ktoré už nie sú konzistentné s ohraničením s vyššou árnosťou.

Uvedená filtrácia môže mať reťazový charakter. Vypustenie nejakej hodnoty z domény nejakej premennej môže prostredníctvom sprísnenia nejakého ohraničenia spôsobiť redukciu domény inej premennej. Podobne sprísnenie jedného ohraničenia môže prostredníctvom redukcie domény nejakej premennej mať za následok sprísnenie iného ohraničenia definovaného nad danou premennou. Toto reťazenie môže nastať podobným spôsobom aj medzi ohraničeniami rôznych árností. Takýto reťazový proces filtrácie sa označuje ako *šírenie ohraničení* (constraint propagation).

Výhodou konzistenčných algoritmov je, že ak robia „fair“ šírenie ohraničení (t.j. každé ohraničenie, ktorého premenné sa zmenili, je znovu aktivované), tak takéto šírenie vedie k jedinečnému stavu siete ohraničení – a teda nezáleží na poradí, v ktorom sú jednotlivé ohraničenia uvažované [15].

Konzistenčné algoritmy pracujú s pojmom *konzistencie*. Jedna z používaných podôb tohto pojmu je (i,j)-konzistencia, kde v úlohe hodnôt i a j môžu vystupovať prirodzené čísla z intervalu <1, n-1>, pričom ich súčet je menší alebo rovný n. Voľne by sa dalo povedať, že sa jedná vlastne o konzistenciu podmnožín premenných z množiny všetkých premenných V a hodnota i vlastne udáva kardinalitu týchto podmnožín. Úlohou konzistenčných algoritmov je túto konzistenciu zabezpečiť. Konzistenčné algoritmy vlastne zabezpečia, že nejaká subsieť siete ohraničení je konzistentná voči svojmu okoliu. Pritom pod týmto okolím sa chápu nejaké premenné, ktoré sa nevyskytujú v danej subsieti – teda takýchto okolí pre nejakú subsieť môže existovať vo všeobecnosti viac.

Formálne je možné povedať, že sieť ohraničení je *(i,j)-konzistentná*, ak platí:

> Nech ľubovoľným i premenným sú priradené také hodnoty z ich domén, aby všetky ohraničenia definované nad touto i-ticou premenných boli splnené. Pre ľubovoľných ďalších j premenných je možné vybrať také hodnoty z ich domén, že všetky ohraničenia definované nad vzniknutou (i+j)-ticou premenných sú splnené.

Ohraničenie definované nad i premennými je (i,j)-konzistentné, ak jeho explicitné vyjadrenie nepripúšťa žiadnu takú kombináciu i hodnôt, ktorá nie je prípustná indukovanou podobou daného ohraničenia (v príklade na Obrázku 1 napr. explicitná podoba $C_{11,12}$ obsahuje 72 prípustných dvojíc, avšak indukovaná podoba tohto ohraničenia obsahuje prípustných dvojíc menej – napr. vďaka ohraničeniam $C_{11,13}$ a $C_{12,13}$ nie sú prípustné dvojice, obsahujúce hodnotu 8). Inak povedané, každá kombinácia hodnôt povolená explicitnou podobou ohraničenia sa dá konzistentne rozšíriť o hodnoty ďalších j premenných.

Takto definovaná (i,j)-konzistencia teda vlastne znamená, že sieť ohraničení je práve vtedy (i,j)-konzistentná, ak neexistuje žiadne také ohraničenie definované nad skupinou i premenných, ktoré by nebolo (i,j)-konzistentné. Sieť ohraničení je prísne (i,j)-konzistentná, ak je súčasne (k,j)-konzistentná pre všetky hodnoty k, pre ktoré je splnená nerovnosť $1 \leq k \leq i$ (prísna konzistencia nejakého stupňa znamená konzistenciu daného stupňa ako aj všetkých nižších stupňov).

Použitie konzistenčných algoritmov je potom možné chápať ako transformáciu siete ohraničení na ekvivalentnú ale explicitnejšiu sieť, pričom existujúce ohraničenia sú použité pre odvodenie ďalších a ich pridanie do siete. Zabezpečenie (i,j)-konzistencie potom má podobu sprísňovania i-árnych ohraničení tak, aby boli konzistentné až s (i+j)-árnymi ohraničeniami. Ak nejaké ohraničenie nie je explicitne dané, tak je ho možné dodefinovať – bude jednoducho reprezentovať ohraničenie, pripúšťajúce všetky kombinácie hodnôt z domén príslušných premenných.

Pojem (1,1)-konzistencie znamená, že v doméne ľubovoľnej premennej V_i sa nachádzajú iba také hodnoty, ktoré ak spĺňajú ohraničenie C_i, tak sú zároveň konzistentné s doménou ľubovoľnej inej premennej V_j, $j \neq i$. To znamená, že pre každé $h_i \in D_i$ existuje taká hodnota $h_j \in D_j$, že dvojica (h_i,h_j) neporušuje ohraničenia definované nad premennými V_i a V_j (napr. $C_{i,j}$ a C_j). Takýto typ konzistencie v sieti ohraničení zahŕňa vždy dva uzly reprezentujúce dve premenné a preto sa nazýva *hranovou* konzistenciou a označuje sa AC (arc consistency).

Použitý ilustračný príklad na Obrázku 1 nie je hranovo konzistentný. Dôvodom je napr. to, že C_{31} nie je konzistentné s $C_{31,38}$, $C_{31,51}$ a $C_{31,61}$ – obsahuje totiž aj hodnoty 1, 7 a 9, pričom však tieto hodnoty nemôžu byť konzistentne rozšírené o hodnoty premenných V_{38}, V_{51} alebo V_{61} bez porušenia príslušného binárneho ohraničenia.

Podobne (2,1)-konzistencia znamená, že ak z domén ľubovoľných dvoch rôznych premenných V_i a V_j je možné vybrať takú dvojicu hodnôt, ktorá neporušuje žiadne ohraničenie, tak táto dvojica je konzistentná s doménou ľubovoľnej inej premennej V_k, pričom $k \neq i$ a $k \neq j$. To znamená, že pre každú dvojicu (h_i,h_j) ($h_i \in D_i$ a $h_j \in D_j$) neporušujúcu žiadne z trojice ohraničení $C_{i,j}$, C_i a C_j existuje nejaká hodnota $h_k \in D_k$ taká, že trojica (h_i,h_j,h_k) neporušuje žiadne z ohraničení $C_{i,j,k}$, $C_{i,k}$, $C_{j,k}$ a C_k.

Táto konzistencia vlastne zaručuje, že ľubovoľná kombinácia dvoch hodnôt z domén dvoch rôznych premenných, povolená priamym ohraničením definovaným nad týmito dvoma premennými, je súčasne povolená všetkými možnými cestami dĺžky 2 medzi týmito dvoma premennými. Preto sa nazýva konzistenciou *po ceste* a označuje PC (path consistency).

Použitý ilustračný príklad nie je konzistentný po ceste. Dôvodom je napr. to, že $C_{12,22}$ nie je konzistentné s $C_{12,22,13}$ a $C_{12,13}$ – obsahuje totiž aj dvojicu (8,5), pričom však táto nemôže byť konzistentne rozšírená o hodnotu premennej V_{13} bez porušenia príslušného ternárneho ohraničenia.

Podobne (1,2)-konzistencia znamená, že v doméne ľubovoľnej premennej V_i sa nachádzajú iba také hodnoty, ktoré ak spĺňajú ohraničenie C_i, tak sú zároveň konzistentné s doménami ľubovoľných iných dvoch premenných V_j, $j \neq i$ a V_k, $k \neq i$ a $k \neq j$. To znamená, že pre každé $h_i \in D_i$ neporušujúce ohraničenie C_i existujú také hodnoty $h_j \in D_j$ a $h_k \in D_k$, že trojica (h_i,h_j,h_k) neporušuje žiadne z ohraničení $C_{i,j,k}$, $C_{i,j}$, $C_{i,k}$, $C_{j,k}$, C_j a C_k.

Táto konzistencia vlastne zaručuje, že ľubovoľná hodnota z domény premennej umožní indukovať také ohraničenie nad dvojicou ďalších dvoch premenných, ktoré je povolené priamym ohraničením definovaným nad týmito dvoma premennými. Preto sa nazýva konzistenciou *inverznou po ceste* a označuje PIC (path inverse consistency).

Použitý ilustračný príklad nie je inverzne konzistentný po ceste. Dôvodom je napr. to, že C_{11} nie je konzistentné s $C_{11,13}$ a $C_{11,12,13}$ – obsahuje totiž aj hodnotu 8, pričom však táto hodnota nemôžu byť konzistentne rozšírené o hodnoty premenných V_{12} a V_{13} bez porušenia príslušného ternárneho ohraničenia.

Analogickým spôsobom je možné definovať konzistencie vyšších stupňov. Keďže (i,j)-konzistencia znamená, že i-tice hodnôt je možné konzistentne rozšíriť o ďalších j hodnôt, dosahovanie takejto konzistencie vlastne znamená nutnosť úpravy (sprísňovania) i-árnych ohraničení. Dôležité je uvedomiť si, že definícia konzistencie vyššieho stupňa nevyžaduje konzistenciu stupňa nižšieho a naopak. Platnosť resp. neplatnosť (i,j)-konzistencie vo všeobecnosti nič nehovorí o platnosti alebo neplatnosti (k,l)-konzistencie pre $i \neq k$ alebo $j \neq l$. A teda (i,j)-konzistencia neovplyvňuje priamo domény premenných v prípade $i > 1$. Preto po dosiahnutí vyššieho stupňa konzistencie musí nasledovať zabezpečenie aj nižších stupňov, aby sa sprísnenie ohraničení vyššej árnosti rozšírilo aj na ohraničenia nižších árností a cez unárne ohraničenia ovplyvnilo domény premenných.

Z praktických dôvodov sa konzistencie vyšších stupňov nepoužívajú kvôli vysokým výpočtovým nárokom na ich zabezpečenie (majú exponenciálnu zložitosť). My sa budeme venovať iba tým najjednoduchším, ktoré ovplyvňujú priamo unárne ohraničenia premenných a tým vlastne aj priamo ich domény. Cenou za dosiahnutie iba nízkeho stupňa konzistencie je nie vždy dostatočný stupeň redukcie ohraničení. Vo všeobecnosti totiž garanciu, že v doménach premenných ostanú iba hodnoty participujúce aspoň na jednom z riešení, môže poskytnúť iba. prísna (i,j)-konzistencia v prípade, že $i + j = n$.

Pretože existujú rôzne typy konzistencie, musia existovať aj rôzne navzájom odlišné algoritmy, ktoré umožňujú tieto konzistencie dosahovať. Vo všeobecnosti, od toho aký typ konzistencie daný algoritmus zabezpečuje, závisí aj zložitosť (a tým aj praktická použiteľnosť) tohto algoritmu.

Pre zabezpečenie hranovej konzistencie je potrebné porovnávať navzájom domény dvojíc premenných. Ak jedna doména obsahuje takú hodnotu, pre ktorú nie je možné vybrať z domény druhej premennej žiadnu hodnotu bez toho, aby binárne ohraničenie definované nad týmito dvoma premennými bolo porušené, tak je nutné danú hodnotu z domény prvej premennej vypustiť.

Naviac je potrebné zohľadniť fakt, že aj keď v určitom momente je nejaká premenná V_i hranovo konzistentná voči premennej V_j, toto sa môže neskôr zmeniť po redukcii domény D_j prislúchajúcej premennej V_j – a teda vlastne redukcia domény premennej V_j zapríčiňuje nutnosť návratu ku kontrole premennej V_i.

Algoritmus pre zabezpečenie hranovej konzistencie môže mať viacero podôb s rôznou zložitosťou a priestorovými nárokmi, danými jednak spôsobom implementácie a jednak mierou redundantných testov. V literatúre je možné nájsť celý rad rôznych implementácií: AC1 opakovane revidujúcu premenné ak nastala nejaká zmena v unárnych ohraničeniach, AC2 realizujúcu iba jednu úplnú iteráciu cez všetky premenné [23] (je ekvivalentný Waltzovmu filtračnému algoritmu [38]), AC3 používajúcu dynamickú frontu uzlov

čakajúcich na kontrolu [23], AC4 založenú na pojme podpory hodnôt premenných hodnotami iných premenných [27], AC5 v generickom tvare umožňujúcu špecializáciu okrem iného aj na AC3 a AC4 [37], AC6 narábajúcu s nutnou podporou hodnôt premenných [3], AC6+ [4] a AC7 [14] založené na symetrii ohraničení a AC8 dekomponujúcu problém do série menších problémov [20].

Principiálny algoritmus AC pre zabezpečenie hranovej konzistencie môže mať podobu podľa Programu 1 (fragment reálneho kódu v jazyku Common Lisp).

```
(defun vyhovuju-2-hodnoty-ohraniceniu (prem1 hodn1 prem2 hodn2)
  (member (cons hodn1 hodn2)
          (binarne-ohranicenie prem1 prem2) :test #'equal))

(defun vyhovuje-hodnota-premennej (premenna1 hodnota1 premenna2)
  (some #'(lambda (hodnota2)
            (vyhovuju-2-hodnoty-ohraniceniu
               premenna1 hodnota1 premenna2 hodnota2))
        (unarne-ohranicenie premenna2)))

(defun vyhovuje-hodnota-premennym (premenna hodnota)
  (every #'(lambda (premenna2)
             (vyhovuje-hodnota-premennej premenna hodnota premenna2))
         (zavisle-premenne premenna)))

(defun AC (&optional (zoznam *zoznam-premennych*))
  (unless (null zoznam)
    (let ((premenna (first zoznam)) zmena)
      (dolist (hodnota (unarne-ohranicenie premenna))
        (unless (vyhovuje-hodnota-premennym premenna hodnota)
          (redukcia-unarneho-ohranicenia premenna hodnota)
          (setf zmena t)))
      (if (not zmena)
          (AC (rest zoznam))
          (AC (union (rest zoznam) (zavisle-premenne premenna)))))))
```

Program 1. Fragment implementácie AC algoritmu.

Algoritmus pracuje so zoznamom premenných, ktorých unárne ohraničenia je potrebné skontrolovať (a prípadne sprísniť). Tento zoznam na začiatku obsahuje zoznam všetkých premenných. Postupne sa z neho odoberajú premenné a kontroluje sa ich konzistencia. Ak sa zmení unárne ohraničenie niektorej premennej, do tohto zoznamu sú pridané (ak už z neho boli odstránené) všetky tie premenné, ktoré na danej premennej závisia prostredníctvom binárnych ohraničení. Algoritmus končí až sa zoznam vyprázdni.

Kontrola hodnoty unárneho ohraničenia nejakej premennej V_i sa deje voči všetkým premenným, ktoré sú závislé na V_i prostredníctvom binárnych ohraničení (iba premenné viazané binárnym ohraničením s V_i môžu byť ovplyvnené zmenou C_i). Akonáhle sa zistí nekonzistencia hodnoty voči niektorej z týchto premenných, kontrola končí neúspechom, pričom kontrola danej hodnoty voči zostávajúcim závislým premenným sa vynechá.

Pri kontrole hodnoty unárneho ohraničenia nejakej premennej V_i voči inej premennej V_j sa kontroluje iba dovtedy, pokiaľ sa nezistí konzistencia. Akonáhle sa zistí, kontrola končí úspechom a kontrola danej hodnoty z C_i voči ostatným hodnotám z C_j sa vynechá.

Dve hodnoty (jedna z C_i a druhá z C_j) sú konzistentné vtedy, ak sa daná kombinácia nachádza v zozname povolených dvojíc binárneho ohraničenia $C_{i,j}$.

Aj PIC algoritmus môže mať viacero podôb s rôznou zložitosťou a priestorovými nárokmi. V literatúre je možné nájsť napr. PIC1 pamätajúci si iba vypúšťané hodnoty ale s veľkou časovou zložitosťou a PIC2 s optimálnou časovou zložitosťou [7]. Principiálna podoba algoritmu PIC je veľmi podobná uvedenému AC algoritmu. Rozdiel je iba v tom, že funkcia `vyhovuje-hodnota-premennym` v tomto prípade kontroluje danú hodnotu voči nie jednej ale dvom premenným, pričom daná hodnota musí vyhovovať nielen každej z kontrolovaných premenných osobitne (teda je potrebné dvojnásobné volanie funkcie `vyhovuje-hodnota-premennej`) ale aj obom súčasne a teda pribudnú dve nové funkcie `vyhovuje-hodnota-dvom-premennym` (musí byť nájdená aspoň jedna kombinácia hodnôt tých dvoch premenných podporujúca kontrolovanú hodnotu) a `vyhovuju-3-hodnoty-ohraniceniu` (kontrola voči relevantnému 3-árnemu ohraničeniu).

Existujú algoritmy, ktoré umožňujú dosiahnuť väčšiu redukciu ohraničení ako ponúka nejaký stupeň konzistencie, avšak menšiu ako ponúka vyšší stupeň konzistencie (je to vlastne spájanie algoritmov do kooperačných schém) [2]. Príkladom je *redukovaná konzistencia po ceste* RPC (reduced path consistency) alebo alternatívy k-RPC a max-RPC [9]. RPC algoritmus pracuje v móde ako AC (porovnáva dvojice hodnôt premenných) a len v situácii, keď hodnota premennej V_i je konzistentná iba s jednou hodnotou z unárneho ohraničenia inej premennej V_j, dochádza k prepnutiu do módu PC a daná dvojica hodnôt je kontrolovaná, či môže byť konzistentne rozšírená o nejakú hodnotu tretej ľubovoľnej premennej V_k. Ak nejaká premenná toto rozšírenie neumožňuje, na rozdiel od PC algoritmu (ktorý by sprísnil zodpovedajúce binárne ohraničenie $C_{i,j}$) dochádza k redukcii unárneho ohraničenia C_i vypustením pôvodne kontrolovanej hodnoty.

Na rozdiel od RPC, variant k-RPC sa prepína do módu PC keď hodnota premennej V_i je konzistentná iba s k alebo menej hodnotami z unárneho ohraničenia inej premennej V_j (musí byť možnosť konzistentného rozšírenia na tri hodnoty aspoň pre jednu z hodnôt premennej V_j). Alternatíva max-RPC sa prepína do módu PC pre všetky hodnoty unárneho ohraničenia premennej V_j, s ktorými je konzistentná skúmaná hodnota premennej V_i.

Algoritmus by sa podobal algoritmu AC. Rozdiel by bol v tom, že v tomto prípade funkcia `vyhovuje-hodnota-premennej` by nekontrolovala iba či testovaná hodnota je podporovaná nejakou hodnotou druhej premennej, ale by aj určovala koľkými hodnotami je podporovaná. Pri počte rôznom od 1 by sa okamžite rozhodlo o úspechu alebo neúspechu. V prípade podpory iba jednou premennou by bola volaná nová funkcia `vyhovuju-dve-hodnoty-premennym` (dvojica hodnôt musí byť konzistentná s každou premennou zo zoznamu všetkých ďalších premenných), následne ďalšia nová funkcia `vyhovuju-dve-`

`hodnoty-premennej` (dvojica hodnôt musí byť konzistentná aspoň s jednou hodnotou tretej premennej) a už spomínaná `vyhovuju-3-hodnoty-ohraniceniu`.

Okrem spomínaných algoritmov je možné použiť aj tzv. singletonový konzistenčný algoritmus, ktorý pracuje systémom pokus-omyl [8]. Validitu každej hodnoty v unárnom ohraničení premennej kontroluje tak, že dané unárne ohraničenie zúži na práve túto hodnotu a aplikuje nejaký redukčný algoritmus. Ak dôjde v procese redukcií k vyprázdneniu nejakého ohraničenia (signalizujúcemu neexistenciu riešenia), kontrolovanú hodnotu vypúšťa, inak ju ponecháva. V závislosti na tom, aký algoritmus je použitý na redukciu, je možné získať napr. SAC (singleton arc consistency), SPIC (singleton path inverse consistency) či SRPC (singleton restricted path consistency), atď.

Principiálna podoba tohto algoritmu má tvar podľa Programu 2 (opäť fragment reálneho kódu v jazyku Common Lisp).

```
(defun singleton-test-hodnoty (prem hodnota redukcny-algoritmus)
  (let ((validita-hodnoty t))
    (ulozenie-stavu :unarne-ohranicenia)
    (nastavenie-unarneho-ohranicenia prem hodnota)
    (funcall redukcny-algoritmus)
    (when (vyprazdnenie-ohranicenia)
      (setf validita-hodnoty nil))
    (obnovenie-stavu :unarne-ohranicenia)
    validita-hodnoty))

(defun singleton (algoritmus)
  (do ((zmena t)) ((not zmena))
    (setf zmena nil)
    (dolist (premenna *zoznam-premennych*)
      (dolist (hodnota (unarne-ohranicenie premenna))
        (unless (singleton-test-hodnoty premenna hodnota algoritmus)
          (redukcia-unarneho-ohranicenia premenna hodnota)
          (setf zmena t))))))

(defun SAC () (singleton #'AC))
(defun SPIC () (singleton #'PIC))
(defun SRPC () (singleton #'RPC))
```

Program 2. Fragment implementácie všeobecného singletonového algoritmu a jeho využitie pre implementáciu algoritmov SAC, SPIC a SRPC.

Je potrebné však povedať, že singletonový algoritmus realizuje mnohonásobné opakovanie redukčného algoritmu (pre každú potenciálnu hodnotu každej premennej), čo sa

výrazne prejaví na výpočtovej náročnosti. Preto je vhodné pred jeho začiatkom znížiť počet hodnôt premenných, na ktoré bude musieť byť aplikovaný (napr. použitím AC).

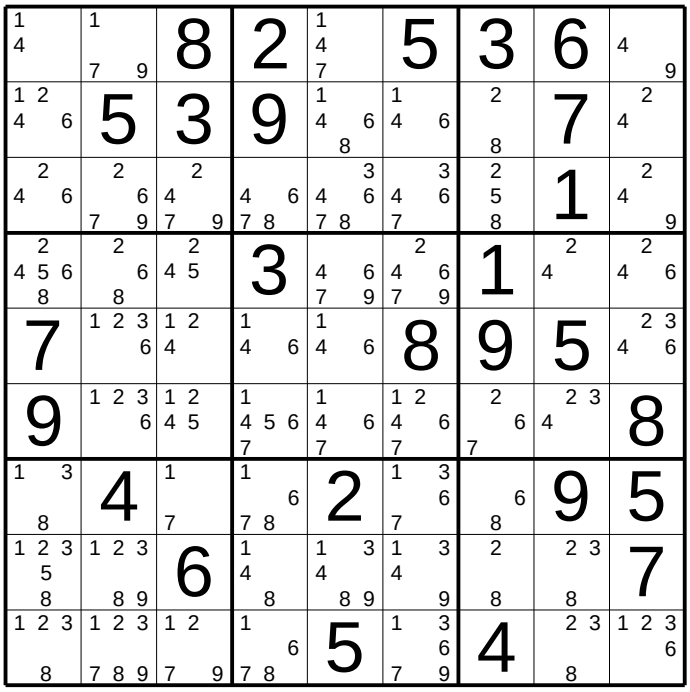
Ukážme si ako je možné použiť popísané algoritmy pre riešenie ukážkového príkladu podľa Obrázku 1. Ten istý prípad je znázornený aj na Obrázku 2, pričom na jednotlivých pozíciách mriežky sú zobrazené unárne ohraničenia jednotlivých premenných (na obsadenej pozícii existuje iba jedna možnosť, na neobsadenej zatiaľ prichádzajú do úvahy všetky možnosti).

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

Obrázok 2. Ilustračný problém s unárnymi ohraničeniami premenných.

Ak sa v takejto situácii použije AC algoritmus, tak napr. pre C_{17} prebehne redukcia. Postupne budú vypustené hodnoty: 1 pre chýbajúcu podporu v C_{38}, 2 kvôli C_{14}, 4 pre C_{97}, 5 pre C_{16}, 6 pre C_{18}, 7 pre C_{28}, 8 pre C_{13} a 9 pre C_{57}. Takýmto spôsobom bola C_{16} zredukovaná na jednu hodnotu a následne bolo možné hodnotu 3 vypustiť z ohraničení všetkých premenných reprezentujúcim ten istý riadok, stĺpec a štvorec ako prislúcha C_{16}. Výsledkom bolo orezanie unárnych ohraničení podľa Obrázku 3.

V situácii podľa Obrázku 3 už použitie AC neprináša ďalšiu redukciu ohraničení. Ak sa použije napr. RPC, tak redukcia unárnych ohraničení môže pokračovať. Pri skúmaní hodnoty 8 z C_{77} voči C_{27} sa zistila podpora iba hodnotou 2. Keďže podpora bola iba jednou hodnotou, prepol sa mód a začalo sa skúmať, či kombinácia 8 z C_{77} a 2 z C_{27} je konzistentná voči unárnemu ohraničeniu každej zo zvyšných premenných. Keďže z C_{87} nebolo možné danú kombináciu hodnôt validne doplniť žiadnou hodnotou, tak hodnota 8 bola vypustená z C_{77}. Ďalšími redukciami sú napr. vypustenie 2 z C_{67} (rovnaký dôvod ako bol pre vypustenie 8 z C_{77}), 6 z C_{67} (v konflikte s aktuálnym C_{77}), atď. Postupnou redukciou sa dospeje do situácie na Obrázku 4.

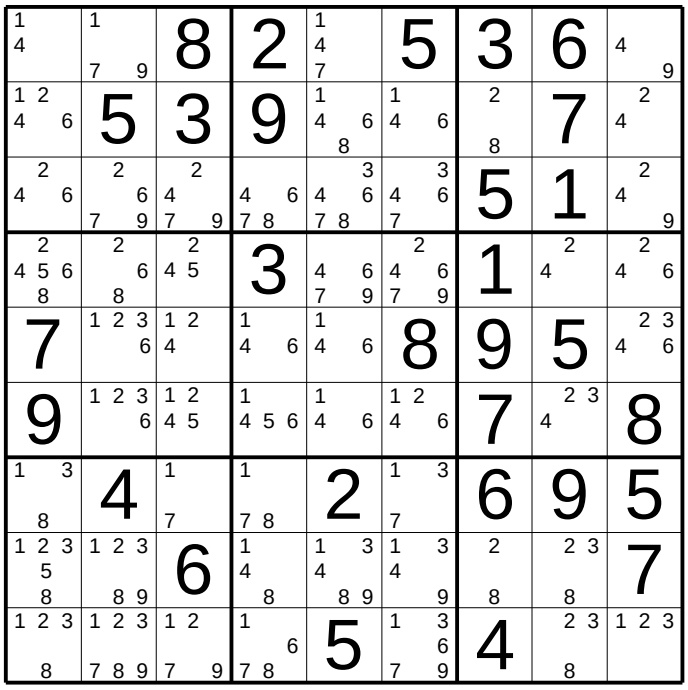


Obrázok 3. Ilustračný problém s unárnymi ohraničeniami premenných po aplikovaní algoritmu AC.

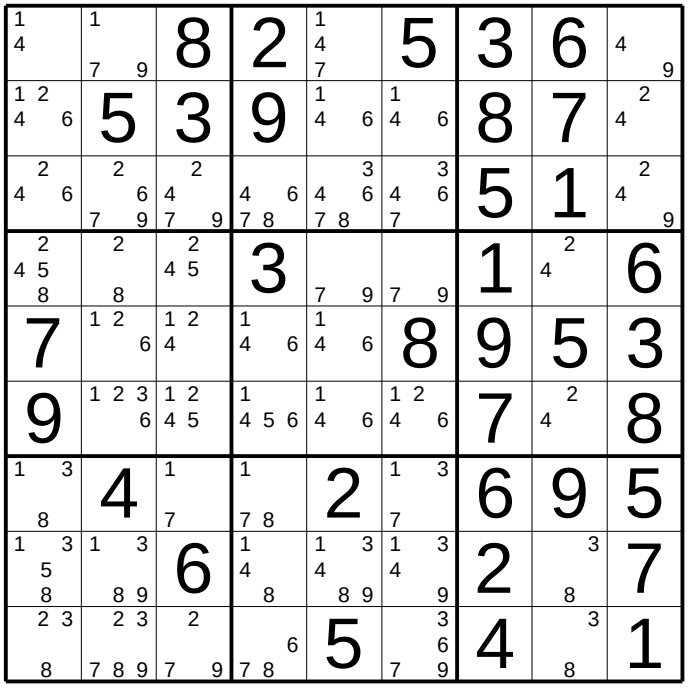
Ak by sa však v Obrázku 3 namiesto RPC použilo PIC, tak napr. pri skúmaní C_{37} sa každá hodnota z tohto ohraničenia bude kontrolovať voči dvojiciam premenných aby sa overilo, či je s unárnymi ohraničeniami týchto premenných konzistentná. Hodnota 2 bude vypustená, pretože pri porovnaní voči C_{27} a C_{87} nebolo možné vybrať takú kombináciu rôznych hodnôt tak, aby obe boli rôzne od hodnoty 2. Z analogického dôvodu bolo potrebné vypustiť aj hodnotu 8 z C_{37}.

Pokračovať v redukcii unárnych ohraničení v situácii ilustrovanej Obrázkom 4 možno napr. prostredníctvom algoritmu max-RPC. Pri kontrole validity hodnoty 4 z C_{49} voči C_{39} je síce táto hodnota podporovaná dvomi hodnotami z C_{39}, avšak ani v jednom prípade nie je možné rozšírenie o tretiu hodnotu buď z C_{19} alebo C_{29} (v závislosti na použitej hodnote z C_{39}). Toto vedie k vypusteniu hodnoty 4 z C_{49}. Z rovnakého dôvodu dôjde k vypusteniu hodnoty 4 aj z ohraničenia C_{59}.

Redukcia ohraničení by mala inú podobu, ak by v situácii na Obrázku 4 bol použitý algoritmus SAC. Napríklad je možné začať skúmať hodnotu 2 v C_{27}. Ak by sa C_{27} zredukovalo iba na túto hodnotu, tak pre zabezpečenie hranovej konzistencie je nutné redukovať C_{29} na hodnotu 4 (teraz hodnota 2 nemá oporu v C_{27}) a C_{39} na hodnotu 9 (hodnota 2 nemá oporu v C_{27} a hodnota 4 zase v C_{29}). Následkom úprav C_{29} a C_{39} bude C_{19} redukovaná na prázdnu množinu – a to je signál, že pôvodný predpoklad (redukcia C_{27} na hodnotu 2) bol chybný a teda hodnota 2 je odstránená z C_{27}. Opakovaným použitím SAC sa získa situácia podľa Obrázku 5.



Obrázok 4. Ilustračný problém s unárnymi ohraničeniami premenných po aplikovaní algoritmov AC a RPC.



Obrázok 5. Ilustračný problém s unárnymi ohraničeniami premenných po aplikovaní algoritmov AC, RPC a SAC.

Ak sa v situácii podľa Obrázku 5 použije SPIC, tak možno napr. redukovať C_{15} na hodnotu 4. Následné skúmanie hodnoty 4 v C_{25} spôsobí jej vypustenie (nebude možné ju konzistentne rozšíriť o ďalšie dve hodnoty ak jedna z nich bude braná z C_{15}). Podobný osud postihne aj hodnotu 4 v C_{55} a C_{65}. Ak potom sa bude uvažovať ľubovoľná hodnota z aktuálneho C_{25}, tak sa nebude dať doplniť žiadnou kombináciou hodnôt z C_{55} a C_{65} – a teda C_{25} sa vyprázdni. Znamená to, že prvotné zúženie C_{15} na hodnotu 4 nie je možné a táto hodnota je vypustená z C_{15}. Z úplne rovnakého dôvodu je nutné vypustiť z C_{15} aj hodnotu 1.

Pre porovnanie výkonnosti jednotlivých algoritmov boli použité problémy z [16]. V ponuke bolo 100 úloh v štyroch kategóriách obtiažnosti. Ako doplnok bolo použitých aj 10 úloh z [21], zaraďovaných medzi najťažšie doposiaľ objavené inštancie daného hlavolamu. Na základe priemerného stupňa orezania unárnych ohraničení (a tým pádom aj domén premenných) boli jednotlivé algoritmy zoradené takto: AC < RPC = PIC < max-RPC < SAC < SRPC = SPIC < Smax-RPC. V tomto prípade teda AC dosiahol najmenší stupeň redukcie a Smax-RPC zase najväčší. Na daných úlohách RPC mal rovnakú výkonnosť ako PIC (a teda aj SRPC ako SPIC). Toto zodpovedá zoradeniu algoritmov napr. podľa [9]. Rozdiel bol iba v tom, že PIC dosahuje minimálne také orezanie ako RPC (teda môže aj väčšie, nielen rovnaké ako to bolo v našom prípade).

Na súbore 100 úloh z [16] dokázali jednotlivé algoritmy riešiť problém nasledovne: AC úplne vyriešil 23 úloh, RPC a PIC vyriešili po 46 úloh, max-RPC 56 úloh, SAC bol úspešný v 97 prípadoch a SRPC, SPIC a Smax-RPC mali sto percentnú úspešnosť. Pri použití [21] boli všetky uvedené algoritmy neúspešné – hoci bol dosiahnutý určitý stupeň redukcie, ich schopnosti redukovať ohraničenia nestačili ani na jednu z úloh. Prvým úspešným algoritmom bol až DSAC (double singleton arc consistency), algoritmus analogický k SAC ibaže redukujúci binárne ohraničenia. Tento už vyriešil všetkých desať úloh (a taktiež všetkých sto z predchádzajúceho súboru).

4 Stochastické prehľadávanie v akcii

Existuje pomerne veľký počet algoritmov založených na stochastických princípoch, ktoré je možné použiť pre riešenie úloh s ohraničeniami. Ako príklad použijeme *evolučné algoritmy* (evolutionary algorithms). Tieto algoritmy tvoria širokú skupinu prístupov a smerov, ktorých jednotiacim prvkom je využívanie princípov prirodzeného výberu a génovej dedičnosti (od snahy presne kopírovať tieto mechanizmy cez využívanie ich zjednodušených podôb až po iba vzdialenú inšpiráciu týmito princípmi).

Hlavnými impulzmi (aj keď to neboli prvé impulzy v tejto oblasti) pre rozvoj týchto algoritmov sa na prelome šesťdesiatych a sedemdesiatych rokov stali práce I. Rechenberga [30] a H.P. Schwefela [32] v oblasti riešenia optimalizačných úloh z oblasti hydrodynamiky, práce L.J. Fogela, A.J. Owensa a M.J. Walsha [12] v oblasti riešenia predikčných úloh a práce J.H. Hollanda [18] z oblasti skúmania fenoménu adaptácie v prírode a jeho prenosu do umelých systémov. Tieto práce sa stali hybným momentom pre vznik troch samostatných smerov, ktoré sa spočiatku vyvíjali izolovane, neskôr však dochádza medzi nimi ku kontaktu a vzájomnému prelínaniu ideí. Následkom toho je vznik algoritmov spájajúcich prvky typické pre rôzne smery. Popisy základných prvkov týchto algoritmov je možné nájsť on-line napr. v prehľadovej publikácii [24].