Príloha E

Príloha E

Článok v zborníku z konferencie:

Paralič, J. – Sabol, T. – Mach, M. – Hatala, M. – Džbor, M.: *Intelligent Development Tool for Constraint Satisfaction.* In: Proc. of the Scientific Conference on Artificial Intelligence in Industry. 22-24 apríl, 1998, High Tatras. KKUI FEI TU Košice. 231–236.

Príloha D

Príloha D

Technický report:

Mach, M.: *Configuration design in CLIPS - room allocation problem.* ENCODE Project Report TR-Encode-TUKE-4-95. Katedra kybernetiky a umelej inteligencie, Technická univerzita v Košiciach, 1995, 45 s.

Príloha C

Príloha C

Príspevok na konferencii:

Mach, M.: *Design Problems with Soft Linear Constraints.* In: Proc. of the 2nd On-line World Conference on Soft Computing in Engineering Design and Manufacturing. 23-27 jún, 1997, Cranfield, Bath, Bombay, East Lansing, Nagoya.
http://www.zen.co.uk/cim.inst/wsc2/conf_hall.html
http://www.bath.ac.uk/Departments/Eng/wsc2/home.html

Príloha B

Príloha B

Technický report:

Mach, M.: *Evolučné algoritmy pre CSP s konečnými doménami.* Technický report. Katedra kybernetiky a umelej inteligencie, Technická univerzita v Košiciach, 2000, 31 s.

Príloha A

Príloha A

Fragment pripravovanej monografie:

Mach, M. – Paralič, J.: *Úlohy s ohraničeniami: od teórie k programovaniu* (pracovný názov). Elfa, Košice, 2000 (monografia pripravovaná do tlače).

Por. číslo	*Zaradenie v texte*	*Publikácia*
[14]	–	Mach, M.: A Generic Knowledge Based Problem Solving Method for Parameter Design. In: Proc. of the 12th Int. Conference on Process Control and Simulation ASRTP'96. Volume I. 10-13 september, 1996, Košice. BERG TU Košice. 449–454.
[15]	–	Naď, P. – Drozdová, J. – Straka, P. – Mach, M.: Uplatnenie metódy genetických algoritmov pri formulovaní kŕmnych dávok prežúvavcov. In: Zborník referátov zo seminára "Problémy bielkovinovej výživy zvierat". 11 október, 1996, Nitra. Výskumný ústav živočíšnej výživy. 143–145.
[16]	–	Hatala, M. – Paralič, J. – Mach, M. – Sabol, T.: Intelligent Development of Large-Scale Knowledge Based Systems. In: Proc. of the Conference on Intelligent Technologies. Vol-I. November, 1996, Herľany. Elfa s.r.o. Košice. 21–28.
[17]	–	Sabol, T. – Mach, M. – Hatala, M. – Paralič, J.: ENCODE - prostredie pre riešenie konfiguračných úloh. AT&P Journal, vol. II, 1995, 4. 19–22.
[18]	–	Naď, P. – Drozdová, J. – Mach, M.: Počítačový program pre zostavovanie kŕmnych dávok prežúvavcov. In: Zborník z konferencie "Zásadné otázky rozvoja vedy a techniky v SR. Výsledky vedy, výskumu a vývoja – NOVTECH'95" - 2. diel. 18-19 október, 1995, Žilina. Dom techniky ZSVTS. 147–149.
[19]	–	Sabol, T. – Mach, M. – Hatala, M. – Paralič, J.: Knowledge Based Design in CLIPS. In: Proc. of the 6th Int. DAAAM Symposium. 26-28 október, 1995, Krakow. DAAAM Int. 293–294.
[20]	príloha D	Mach, M.: Configuration design in CLIPS - room allocation problem. ENCODE Project Report TR-Encode-TUKE-4-95. Katedra kybernetiky a umelej inteligencie, Technická univerzita v Košiciach, 1995, 45 s.

Por. číslo	*Zaradenie v texte*	*Publikácia*
[7]	–	Paralič, J. – Sabol, T. – Mach, M. – Hatala, M. – Džbor, M.: Constraint Satisfaction Engine for the Encode Workbench. In: Proc. of the 8th Int. DAAAM Symposium. 23-25 október, 1997, Dubrovník. DAAAM Int. Viedeň. 255–256.
[8]	príloha C	Mach, M.: Design Problems with Soft Linear Constraints. In: Proc. of the 2nd On-line World Conference on Soft Computing in Engineering Design and Manufacturing. 23-27 jún, 1997, Cranfield, Bath, Bombay, East Lansing, Nagoya. http://www.zen.co.uk/cim.inst/wsc2/conf_hall.html http://www.bath.ac.uk/Departments/Eng/wsc2/home.html
[9]	–	Mach, M.: Experimental analysis of CSE. ENCODE Project Report, TR-Encode-TUKE-2-97. Katedra kybernetiky a umelej inteligencie, Technická univerzita v Košiciach, 1997, 105 s.
[10]	–	Paralič, J. – Sabol, T. – Mach, M. – Hatala, M.: Configuration Design as Constraint Satisfaction. In: Proc. of the Int. Computer Science Conference microCAD'96. Section H: Information Technology in Mechanical Engineering. 29 február, 1996, Miskolc-Egyetemváros. 64–68.
[11]	–	Mach, M.: Using genetic algorithms for problems with soft constraints. In: Proc. of the 3rd Int. Workshop on Artificial Intelligence Techniques AIT'96. 9-11 september, 1996, Brno. PC DIR. 173–174.
[12]	–	Naď, P. – Drozdová, J. – Straka, P. – Mach, M.: Optimalizačný program pre výživu hospodárskych zvierat zostavený na princípe genetických algoritmov. In: Zborník zo seminára "Výskum a vývoj pre prax NOVTECH'96" - 1. diel. 26-28 november, 1996, Žilina. Dom techniky ZSVTS. 56–60.
[13]	–	Paralič, J. – Hatala, M. – Sabol, T. – Mach, M.: ENCODE - ENvironment for COnfiguration DEsign. In: Proc. of the 12th Int. Conference on Process Control and Simulation ASRTP'96. Volume I. 10-13 september, 1996, Košice. BERG TU Košice. 455–460.

Zoznam prác autora súvisiacich s problematikou riešenia úloh s ohraničeniami

Por. číslo	*Zaradenie v texte*	*Publikácia*
[1]	príloha A	Mach, M. – Paralič, J.: Úlohy s ohraničeniami: od teórie k programovaniu (pracovný názov). Elfa, Košice, 2000 (monografia pripravovaná do tlače).
[2]	príloha B	Mach, M.: Evolučné algoritmy pre CSP s konečnými doménami. Technický report. Katedra kybernetiky a umelej inteligencie, Technická univerzita v Košiciach, 2000, 31 s.
[3]	–	Mach M.: Design Problems with Soft Linear Constraints. In: Chawdhry, P.K. – Roy, R. – Pant, R.K. (eds.): Soft Computing in Engineering Design and Manufacturing. Springer Verlag, London, 1998, ISBN 3-540-76214-0. 221–229.
[4]	–	Sabol, T. – Hatala, M. – Mach, M. – Paralič, J.: Environment for Configuration Design. In: Proc. of the I. Internal Scientific Conference of the Faculty of Electrical Engineering and Informatics. 5 február, 1998, Košice. FEI TU Košice. 17–18.
[5]	príloha E	Paralič, J. – Sabol, T. – Mach, M. – Hatala, M. – Džbor, M.: Intelligent Development Tool for Constraint Satisfaction. In: Proc. of the Scientific Conference on Artificial Intelligence in Industry. 22-24 apríl, 1998, High Tatras. KKUI FEI TU Košice. 231–236.
[6]	–	Mach, M.: Linear Soft Constraint Satisfaction. In: Proc. of the 3rd Int. Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets MENDEL'97. 25-27 jún, 1997, Brno. PC DIR Brno. 228–233.

Snahy menovaných v oblasti výskumu zastrešovala v rokoch 1994 až 1997 grantová úloha "Metódy reprezentácie a riešenia úloh, definovaných systémom ohraničení", získaná z VGA MŠ SR doc. Ing. Júliusom Csontó, CSc. pod číslom 1/1686/94.

Súbežne v rokoch 1995 až 1998 bola katedra zapojená do medzinárodného projektu COPERNICUS CIPA-CT94-0149 "ENCODE - ENvironment for COnfiguration DEsign". Cieľom tej časti projektu, za ktorú bola zodpovedná katedra, bolo navrhnúť a implementovať systém pre riešenie ohraničení, ktorý by bol vhodný pre riešenie rôznych návrhových problémov. Vďaka úsiliu riešiteľov vznikol systém CSE[26] (constraint satisfaction engine) založený na systéme SCREAMER – nedeterministickej nadstavbe jazyka Common Lisp. Z vyššie menovaných sa na úspešnej realizácii tohto projektu podieľali doc. Ing. Tomáš Sabol, CSc., Ing. Ján Paralič, PhD. a autor predkladaného textu.

Okrem týchto troch pracovníkov katedry sa na projekte podieľal ešte aj Ing. Marek Hatala, CSc., bývalý externý ašpirant doc. Ing. Júliusa Csontó, CSc. Svoje úsilie sústreďoval na vizualizáciu činnosti algoritmov, kombinujúcich prehľadávanie priestoru riešení so zabezpečovaním určitého stupňa konzistencie. Po skončení projektu sa však už problematike ohraničení viac nevenoval.

Na prvý VEGA grant nadviazala v rokoch 1998 až 2000 ďalšia grantová úloha číslo 1/5032/98 "Integrácia prostriedkov inteligentných technológií", opäť získaná z VGA MŠ SR doc. Ing. Júliusom Csontó, CSc. Ako jedna z uvažovaných inteligentných technológií bolo v tomto projekte zahrnuté aj riešenie úloh s ohraničeniami.

Oproti stavu v polovici deväťdesiatych rokov došlo na katedre v súčasnosti k útlmu výskumných aktivít v oblasti riešenia úloh s ohraničeniami. Jediným pracovníkom katedry, ktorý sa v súčasnosti v rámci výskumnej činnosti zaoberá teoretickými otázkami tejto oblasti, je autor predkladanej práce. Dôvodom tohto poklesu záujmu nie je neperspektívnosť danej oblasti, ale objavenie sa mnohých iných zaujímavých oblastí, ktoré pritiahli pozornosť ostatných.

Študenti majú možnosť stretnúť sa s ohraničeniami v priamej výuke v rámci niekoľkých ponúkaných predmetov. Sú to "Seminár z jazykov pre umelú inteligenciu" (systém ECL^iPS^e a jeho použitie), "Rozvrhovanie a logistika" (aplikácia metód riešenia úloh s ohraničeniami), a čiastočne aj predmety "Evolučné algoritmy" (časť predmetu venovaná riešeniu úloh s ohraničeniami pomocou evolučných algoritmov) a "Projektovanie znalostných systémov" (časť predmetu venovaná znalostnému modelovaniu metód riešenia úloh s ohraničeniami). Prvé dva predmety sú zabezpečované Ing. Jánom Paraličom, PhD., zatiaľ čo druhé dva predmety zabezpečuje autor predkladanej práce.

Aj naďalej sa niektorí študenti stretávajú s problematikou ohraničení počas riešenia diplomových prác. Tieto práce sú v súčasnosti zamerané buď na evolučné algoritmy alebo na riešenie úloh časového rozvrhovania.

[26]Jeden z textov, v ktorých bol systém CSE prezentovaný, je umiestnený v prílohe E.

parametrického návrhu. V súčasnosti sa už problematike práce s ohraničeniami nevenuje.

Ing. Ján Paralič, PhD. sa začal venovať problematike s ohraničeniami po nástupe na doktorandské štúdium u doc.Ing. Júliusa Csontó, CSc. Spočiatku sa venoval rozšíreniu Warenovho abstraktného stroja pre implementáciu ohraničení na doméne booleovských premenných. Veľký vplyv na neho mal študijný pobyt vo Výskumnom ústave počítačového priemyslu (ECRC) v Mníchove, kde sa v začiatkoch éry logického programovania ohraničení zrodil jazyk $CHIP$. Počas svojho pobytu sa podieľal na implementácii systému ECL^iPS^e.

Po svojom návrate sa venoval rozširovaniu možností logického programovania ohraničení pre riešenie úloh rozvrhovania. Svoje úsilie zameriaval najmä na riešenie disjunktívnych ohraničení a optimálne riešenie úloh s ohraničeniami. V rámci svojej dizertačnej práce okrem iného implementoval niektoré metódy pre riešenie disjunktívnych ohraničení v prostredí logického programovania ohraničení a venoval sa návrhu logaritmických metód určovania hraníc pre účely urýchľovania dôkazu neexistencie lepších riešení pri riešení optimalizačných úloh.

Okrem teoretickej práce sa venoval aj praktickému využívaniu metód pre riešenie ohraničení. Z tejto oblasti je najvýznačnejšou jeho práca rozvrhovania brám zavážaných do narážacích pecí za účelom ohrevu na požadovanú teplotu pred valcovaním na teplej širokopásovej valcovni vo VSŽ, a.s. v Košiciach.

Menovaný sa dodnes venuje problematike logického programovania ohraničení a jeho využitiu pre riešenie úloh rozvrhovania, prevažne však iba v rámci výukového procesu.

Autor predloženej práce sa prvý krát s problematikou ohraničení stretol v praktickej rovine v roku 1989 pri vytváraní komerčného dedikovaného expertného systému KRAVEX, zameraného na diagnostiku ochorení hovädzieho dobytka a zostavovanie optimálnych kŕmnych dávok. Okrem autora, v tom čase externého ašpiranta doc. Ing. Júliusa Csontó, CSc., sa na implementácii tohto expertného systému podieľal aj doc. Ing. Tomáš Sabol, CSc.

V teoretickej rovine sa touto problematikou začal zaoberať až v polovici deväťdesiatych rokov v súvislosti so svojím výskumom v oblasti aplikovania evolučných algoritmov. Venoval sa najmä problematike mäkkých ohraničení, či už v rámci preurčených úloh s ohraničeniami alebo optimalizačných úloh s ohraničeniami[24].

Neskôr sa venoval problematike ohraničení aj z hľadiska využití znalostných technológií, špeciálne modelovania znalostí. Z tohto obdobia pochádza návrh metódy CRO pre riešenie preurčených optimalizačných problémov[25].

Možnosti využitia technológie modelovania znalostí a evolučného počítania sprístupňoval a aj naďalej sprístupňuje študentom v rámci inžinierskeho štúdia. Oblasť teórie riešenia úloh s ohraničeniami pomocou evolučných algoritmov je dodnes jednou z jeho odborných záujmových oblastí.

[24]Jedna z prác, viažúcich sa k tejto problematike, je prezentovaná v prílohe C.

[25]Táto metóda bola prezentovaná aj v práci, ktorá tvorí prílohu D.

Ohraničenia a Katedra kybernetiky a umelej inteligencie FEI TU v Košiciach

Problematika ohraničení sa v implicitnej podobe objavovala v riešení výskumných úloh a výuke od samého začiatku existencie katedry. V mnohých oblastiach, ktoré súviseli so zameraním katedry, vystupovali ohraničenia – avšak pre ich riešenie sa využívali výhradne metódy a postupy, špecifické pre tú-ktorú oblasť, nevhodné pre riešenie ohraničení všeobecného tvaru. Takýto stav pretrvával až do konca osemdesiatych rokov.

Ohraničenia v explicitnej problémovo nezávislej podobe tvorili jedno z ohnísk záujmu katedry počas celých deväťdesiatych rokov. Ich participácia na aktivitách katedry bola úzko spojená s menom doc. Ing. Júliusa Csontó, CSc. a niektorých z jeho ašpirantov resp. doktorandov.

Na prelome osemdesiatych a deväťdesiatych rokov sa pod jeho vedením vytvorila skupina pracovníkov, zaoberajúcich sa logickým programovaním. V tom období zažívala mohutný rozmach oblasť logického programovania ohraničení, ktorá sa snažila využiť deklaratívnosť reprezentácie úloh, ponúkanú logickým programovaním, a rozšíriť ju o možnosť práce s ohraničeniami. Keďže tieto oblasti mali k sebe blízko, nebol prekvapením posun záujmu k problematike riešenia ohraničení. Tento krok k ohraničeniam vykonal sám hlavný predstaviteľ spomínanej skupiny.

Vďaka tomu sa na pôde katedry začala rozvíjať oblasť logického programovania ohraničení. Začali experimenty s programovým systémom TRILOGY a neskôr aj so systémom ECL^iPS^e. Tieto systémy sa používali nielen pre potreby riešenia výskumných úloh, ale možnosť oboznámiť sa s nimi mali aj študenti - ako v priamej výuke tak aj prostredníctvom riešenia diplomových úloh. Systém ECL^iPS^e je súčasťou ponuky katedry v rámci inžinierskeho štúdia dodnes.

Doc. Ing. Július Csontó, CSc. sa v rámci tejto oblasti venoval predovšetkým využitiu technológie logického programovania ohraničení pre riešenie úloh časového a priestorového návrhu. Jedným z výsledkov jeho činnosti bolo vytvorenie experimentálneho znalostného systému, založeného na modifikovaných Allenových časových reláciách, určeného pre riešenie úloh rozvrhovania s časovými ohraničeniami. Problematike tvorby časových rozvrhov ostal verný dodnes.

Doc. Ing. Tomáš Sabol, CSc., bývalý interný ašpirant doc. Ing. Júliusa Csontó, CSc., sa v prvej polovici deväťdesiatych rokov venoval problematike konzistentného značkovania grafov ohraničení pomocou metód, založených na technológii celulárnych automatov. Najvýznačnejším teoretickým výsledkom jeho činnosti bol návrh paralelného relaxačného algoritmu pre riešenie systému ohraničení, založeného na využívaní lokálnych heuristík. Tento algoritmus bol publikovaný v roku 1994.

Okrem teoretických prác sa spočiatku venoval využívaniu metód pre riešenie ohraničení pre návrh viachodnotových logík, neskôr ich využitiu pre účely

Prílohy

[64] Waltz, D.: Understanding line drawings of scenes with shadows. In: Winston, P.H. (ed.): The psychology of computer vision. McGraw Hill, New York, 1975, 19–91.

[65] Weigel, R. – Bliek, Ch. – Faltings, B.V.: On reformulation of constraint satisfaction problems (extended version). In: Proc. of the Conference SARA'98. http://liawww.epfl.ch/~weigel/papers.html

[66] Zdráhal, Z. – Motta, E. – Stutt, A.: Optimising revision procedures for configuration design. Technical Report TR-112, Open University, Milton Keynes, UK, 1994.

[51] Ryan, C.: Shades – a polygenic inheritance scheme. In: Proc. of the 3rd Int. Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets MENDEL'97. PC-DIR, Brno, 1997, 140–147.

[52] Sabol, T.: Parallel solution of a system of declarative constraints using local heuristics. Trans. of the Technical University of Košice, vol. 4, 1994, no. 3-4, 249–254.

[53] Sadeh, N. – Fox, M.S.: Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. Artificial Intelligence, vol. 86, 1996, 1–41.

[54] Schmotzer, M.: Analýza a návrh nových prostriedkov pre riešenie úloh časového rozvrhovania v prostredí logického programovania ohraničení. Diplomová práca, Katedra kybernetiky a umelej inteligencie, Technická univerzita Košice, 1997.

[55] Schreiber, G. et al.: Knowledge engineering and management. The CommonKADS methodology. MIT Press, Cambridge, 2000.

[56] Schwefel, H.P.: Kybernetische Evolution als Strategie der experimentallen Forschung in der Stroemungstechnik. Diplomová práca, Technická univerzita, Berlín, 1965.

[57] Steels, L.: Components of expertise. The AI Magazine, vol. 11, 1990, no. 2, 29–49.

[58] Sutherland, I.: Sketchpad: a man-machine graphical communication system. In: Proc. of the IFIP Spring Joint Computer Conference, 1963.

[59] Tansley, D.S.W. – Hayball, C.C.: Knowledge-based systems analysis and design. A KADS Developer's Handbook. Prentice Hall, London, 1993.

[60] Tsang, E.: Foundations of Constraint Satisfaction. Academic Press, San Diego, 1993.

[61] Tsang, E.: No more "partial" and "full looking ahead". Artificial Intelligence, vol. 98, 1998, 351–361.

[62] Verfaillie, G. – Schiex, T.: Solution reuse in dynamic constraint satisfaction problems. In: Proc. of the 12th National Conference on Artificial Intelligence AAAI'94, Seatle, Washington, 1994, 307–312.

[63] Wallace, R.J.: Directed arc consistency preprocessing. In: Meyer, M. (ed.): Constraint Processing (LNCS 923). Springer Verlag, Berlin, 1995, 121–137.

[38] Mackworth, A.K.: Consistency in networks of relations. Artificial Intelligence, vol. 8, 1977, 99–118.

[39] McAllester, D.: Constraint satisfaction search. Lecture Notes for 6.824, Artificial Intelligence. 1992. http://www.ai.mit.edu/people/dam/notes/csp.ps

[40] Michalewicz, Z.: Genetic algorithms + data structures = evolution programs. 3rd edition. Springer Verlag, New York, 1996.

[41] Michalewicz, Z. – Janikow, C.Z.: Handling constraints in genetic algorithms. In: Proc. of the 4th Int. Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, 1991, 151–157.

[42] Mittal, S. – Falkenhaimer, B.: Dynamic constraint satisfaction problems. In: Proc. ot the 8th National Conference on Artificial Intelligence AAAI'90, Boston, Massachusetts, 1990, 25–32.

[43] Mudambi, S. – Prestwich, S.: Improved branch and bound in constraint logic programming. In: Proc. of the 1st Int. Conference on Principles and Practice of Constraint Programming CP'95, Cassis, France, September 1995, 533–548.

[44] Newell, A.: The knowledge level. Artificial Intelligence, vol. 18, 1982, 87–127.

[45] Ošmera, P. – Kvasnička, V. – Pospíchal, J.: Genetic algorithms with diploid chromosomes. In: Proc. of the 3rd Int. Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks, Rough Sets MENDEL'97. PC-DIR, Brno, 1997, 111–116.

[46] Pánik, M.: Rozšírenie expertného systému CLIPS o mechanizmus spĺňania ohraničení. Diplomová práca, Katedra kybernetiky a umelej inteligencie, Technická univerzita Košice, 1997.

[47] Paralič, J.: Riešenie úloh rozvrhovania logickým programovaním ohraničení. Dizertačná práca, Katedra kybernetiky a umelej inteligencie, Technická univerzita v Košiciach, máj 1997.

[48] Popper, M. – Kelemen, J.: Expertné systémy. Alfa, Bratislava, 1989.

[49] Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart, 1973.

[50] Richardson, J.T. et al: Some guidelines for genetic algorithms with penalty functions. In: Proc. of the 3rd Int. Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, 1989, 191–197.

[25] Gent, I.P. et al.: How not to do it. Report 97.27, School of computer studies, University of Leeds, 1997. http://www.scs.leeds.ac.uk/bms/papers.html

[26] Goldberg, D.E.: Genetic algorithms in search, optimization, and machine learning. Addison Wesley, Reading, MA, 1989.

[27] Gomes, C.P. – Selman, B. – Kautz, H.: Boosting combinatorial search through randomization. In: Proc. of the 15th National Conference on Artificial Intelligence AAAI'98, Madison, WI, 1998.
http://simon.cs.cornell.edu/home/selman/papers-ftp/papers.html

[28] Grefenstette, J.J.: Genetic algorithms for changing environments. In: Männer, R. – Manderick, B. (eds.): Parallel Problem Solving from Nature. North-Holland, 1992, 137–144. ftp://ftp.aic.nrl.navy.mil/pub/papers/1992/

[29] Haralick, R.M. – Shapiro, L.G.: The consistent labeling problem: part I. IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 1, 1979, no. 2. 173 – 184.

[30] Hayes, B.: Can't get no satisfaction. American Scientist, vol. 85, 1997, no. 2, 108–112.

[31] Hayes-Roth, F. – Waterman, D.A. – Lenat, D.B.: Building expert systems. Addison-Wesley, Reading, MA, 1983.

[32] Hernandez, L.F.G. – Corne, D.W.: Evolutionary divide and conquer for the set covering problem. In: Fogarty, T.C. (ed.): Evolutionary Computing (LNCS 1143). Springer Verlag, Heidelberg, 1996, 198–208.

[33] Holland, J.H.: Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, 1975.

[34] Kondrak, G. – van Beek, P.: A theoretical evaluation of selected backtracking algorithms. Artificial Intelligence, vol. 89, 1997, 365–387.

[35] Kowalczyk, R.: Constraint consistent genetic algorithms. In: Proc. of the 1997 IEEE Int. Conference on Evolutionary Computation ICEC'97. Indianapolis, IN, April 1997, 343–348.

[36] Kumar, V.: Algorithms for constraint satisfaction problems: a survey. The AI Magazine, vol. 13, 1992, no. 1. 32–44.
http://www-users.cs.umn.edu/~kumar/

[37] Larrosa, J. – Meseguer, P. – Schiex, T.: Maintaining reversible DAC for Max-CSP. Artificial Intelligence, vol. 107, 1999, no. 1, 149–163.

[11] Chandrasekaran, B. – Johnson, T.R. – Smith, J.W.: Task-structure analysis for knowledge modeling. Communications of the ACM, vol. 35, 1992, no. 9, 124–137.

[12] Clancey, W.J.: Heuristic classification. Artificial Intelligence, vol. 27, 1985, 289–350.

[13] Davis, L. (ed.): Genetic algorithms and simulated annealing. Pitman, London, 1987.

[14] Davis, L. (ed.): Handbook of genetic algorithms. Van Nostrand Reinhold, New York, 1991

[15] Debruyne, R. – Bessiere, Ch.: Some practicable filtering techniques for the constraint satisfaction problem. In: Proc. of IJCAI'97, Nagoya, Japan. 412–417. http://www.lirmm.fr/˜bessiere/publis.html

[16] Dechter R.: Constraint networks. In: Encyclopaedia of Artificial Intelligence, 2nd ed. Wiley and Sons, 1992, 276–285.
ftp://ftp.ics.uci.edu/pub/CSP-repository/papers/survey.ps

[17] DeJong, K.A.: The analysis of the behavior of a class of genetic adaptive systems. PhD Theses, University of Michigan, Ann Arbor, 1975.

[18] Eiben, A. – Raué, P-E. – Ruttkay, Z.: GA-easy and GA-hard constraint satisfaction problems. In: Meyer, M. (ed.): Constraint Processing (LNCS 923). Springer Verlag, Berlin, 1995, 267–283.

[19] Eiben, A. – van der Hauw, J.K. – van Hemert, J.I.: Graph coloring with adaptive evolutionary algorithms. Journal of Heuristics, vol. 4, 1998, 25–46. http://www.wi.leidenuniv.nl/˜gusz/

[20] Fogel, L.J. – Owens, A.J. – Walsh, M.J.: Artificial intelligence through simulated evolution. John Wiley, New York, 1966.

[21] Freuder, E.C.: Synthesizing constraint expressions. Communications of the ACM, vol. 21, 1978, no. 11, 958–966.

[22] Freuder, E.C.: The many path to satisfaction. In: Meyer, M. (ed.): Constraint Processing (LNCS 923). Springer Verlag, Heidelberg, 1995, 103–119.

[23] Freuder, E.C. – Wallace, R.J.: Partial constraint satisfaction. Artificial Intelligence, vol. 58, 1992, 21–70.

[24] Frost, D. – Dechter, R.: In search of the best constraint satisfaction search. In: Proc. of the 12th National Conference on Artificial Intelligence AAAI'94, Seatle, Wa, 1994, 301–306.

Literatúra

[1] Allen, J.A. – Minton, S.: Selecting the right heuristic algorithm: runtime performance predictors. In: McCalla, G. (ed.): Advances in Artificial Intelligence (LNAI 1081). Springer Verlag, Heidelberg, 1996, 41–53.

[2] Bacchus, F. – van Beek, P.: On the conversion between non-binary and binary constraint satisfaction problems. In: Proc. of the 15th National Conference on Artificial Intelligence AAAI'98, Madison, WI, 1998. http://www.cs.ualberta.ca/˜vanbeek/papers.html

[3] Barták, R.: Constraint programming: in pursuit of the holy grail. In: Proc. of the Conference WDS'99, Praha, 1999. http://kti.mff.cuni.cz/˜bartak/downloads/WDS99.pdf

[4] Bäck, T. – Schutz, M. – Khuri, S.: A comparative study of a penalty function, a rapid heuristic, and stochastic operators with the set-covering problem. In: Alliot, J.M. et al. (eds.): Artificial Intelligence (LNCS 1063). Springer Verlag, Heidelberg, 1995, 320–332.

[5] Bessiere, Ch.: Arc consistency in dynamic constraint satisfaction problems. In: Proc. of the 9th National Conference on Artificial Intelligence AAAI'91, MIT Press, 1991, 221–226.

[6] Bessiere, Ch. – Regin, J-Ch.: MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In: Freuder, E.C.: (ed.) Principles and Practice of Constraint Programming (LNCS 1118). Springer Verlag, Heidelberg, 1996, 61–75. http://www.litmm.fr/˜bessiere/publis.html

[7] Borning, A.: The programming language aspects of ThingLab, a constraint oriented simulation laboratory. ACM Transactions on Programming Languages and Systems, vol. 3, 1981, no. 4, 252–387.

[8] Brailsford, S.C. – Potts, C.N. – Smith, B.M.: Constraint satisfaction problems: algorithms and applications. European Journal of Operational Research, vol. 119, 1999, 557–581.

[9] Camponogara, E. – Talukdar, S.N.: A genetic algorithm for constrained and multiobjective optimization. In: Proc. of the 3rd Nordic Workshop on Genetic Algorithms and their Applications 3NWGA. Helsinki, Finland, August 1997, 49–61.

[10] Chandrasekaran, B.: Design problem solving: a task analysis. The AI Magazine, vol. 11, 1990, no. 4, 59–71.

6 Záver

Metódy, prezentované v tomto texte, počas doby svojej existencie jasne preukázali opodstatnenosť svojho použitia pre riešenie rôznych typov problémov s ohraničeniami.

Je ťažké predpovedať budúcnosť týchto metód a ďalší vývoj v oblasti nástrojov pre riešenie úloh s ohraničeniami definovanými nad konečnými doménami. Je veľmi pravdepodobné, že vzhľadom na svoj aplikačný potenciál a úspešnosť pri doterajšom riešení praktických problémov, tieto metódy ani v budúcnosti nebudú patriť na perifériu záujmu odborníkov z relevantných oblastí.

V rámci jednotlivých prístupov zrejme aj naďalej bude prebiehať honba za stále výkonnejšími variantmi algoritmov umožňujúcimi posunúť hranice oblasti prakticky riešiteľných úloh o nejaký kus ďalej.

Jedným z možných smerov vývoja je silnejšie presadzovanie "znalostného princípu" spôsobujúce odklon od orientácie na riešenie všeobecných problémov s ohraničeniami. Toto by malo za následok vznik nových metód, ktoré vďaka začleneniu znalostí o charakteristikách nejakej užšej triedy úloh s ohraničeniami budú schopné riešiť problémy danej triedy efektívnejším spôsobom. Príkladom môže slúžiť napr. [53], kde znalosti o doménovej oblasti umožnili vytvorenie účinných heuristík pre výber premenných a hodnôt.

Štúdium interakcie rozličných metód je príkladom ďalšieho možného smerovania vývoja v danej oblasti. Prirodzeným vyústením môže byť vznik hybridných metód kombinujúcich prvky, ktoré sú dnes typické pre rozličné prístupy. V malej miere sa tento smer stáva už prítomnosťou. Doplnenie riadeného množstva stochastičnosti do prehľadávacích metód [27], znalostná inicializácia prvotnej populácie evolučného algoritmu či využitie princípu konzistenčných algoritmov pre zefektívnenie evolučného prehľadávania [35] sú jednými z príkladov takéhoto vzájomného obohacovania.

Jednou z veľkých otázok je určenie, kedy by mala byť použitá tá-ktorá metóda. V súčasnosti sa užívateľ rozhoduje podľa všeobecného názoru na túto otázku (conventional wisdom) ktorý pre daný konkrétny problém môže byť škodlivým, podľa svojej intuície, subjektívnej obľuby nejakej metódy alebo jednoducho použije slepo ten nástroj ktorý má práve k dispozícii. Budúcnosť by mala patriť voľbe metódy podľa charakteristík problému, alebo ešte lepšie, zostaveniu vhodnej metódy z preddefinovanej množiny prvkov, šitej na mieru riešenému problému.

porovnávané navzájom tie prístupy, ktoré sú prezentované v tomto texte. Priame porovnanie je spomenuté v [8] kde je prezentovaný záver, že metódy spĺňania ohraničení poskytujú kvalitnejšie riešenie než evolučné algoritmy, avšak iba pri akceptovaní ich väčších nárokov na dobu výpočtov.

Zaujímavú informáciu poskytuje aj to, aká je rozmernosť problémov používaných pre empirickú analýzu jednotlivých metód. Napr. pri uvažovaní úloh s doménami obsahujúcimi tri[23] hodnoty, metódy spĺňania ohraničení používajú testovacie problémy až do veľkosti približne 3^{350} bodov v priestore prehľadávania. Naproti tomu veľké percento prác zameraných na použitie evolučných algoritmov pre riešenie úloh s ohraničeniami pracuje s priestorom prehľadávania obsahujúcim 3^{1000} bodov a viac.

Je dôležité si uvedomiť, že rozličné typy metód sú založené na rozličných princípoch a to môže spôsobovať ich rozličnú vhodnosť pre riešenie nejakého konkrétneho problému. Napr. v [18] je prezentovaný evolučný algoritmus ktorý dokázal vyriešiť problém n-dám pre hodnotu n rovnú až 10000 čo vysoko prevyšuje rozmernosť tohto problému vyriešeného niektorou z metód spĺňania ohraničení. Naproti tomu pri riešení "zebra" problému (reprezentovaného pomocou 25 premenných pričom každej je potrebné priradiť jednu z piatich možných hodnôt) ten istý variant evolučného algoritmu dosiahol menšiu úspešnosť než 50%, pričom tento problém je riešiteľný metódami spĺňania ohraničení.

Kritickým faktorom v tomto prípade je množstvo propagácií ktoré umožňujú ohraničenia. Ak podstata problému je taká, že pomerne malým množstvom úsilia je možné dosiahnuť veľké orezanie priestoru prehľadávania, tak metódy spĺňania ohraničení sú vhodnejšie než evolučné algoritmy pracujúce stále nad tým istým (pôvodným) priestorom prehľadávania. Úlohy ľahké pre nejaký typ metód môžu byť náročné pre inú skupinu metód a naopak.

Pri použití znalostného prístupu, na rozdiel od ostatných dvoch prezentovaných prístupov, značne narastajú časové nároky na formulovanie problému pretože súčasťou tejto formulácie je aj proces získavania potrebných znalostí. Podobne narastá aj úsilie, ktoré je potrebné vynaložiť pri implementácii. Pri dostupnosti vhodných znalostí je však možné riešiť úlohy veľmi efektívne, pretože je možné k jednotlivým ohraničeniam (a ich porušeniam) pristupovať individuálnym spôsobom so zohľadnením ich podstaty a väzieb medzi ohraničeniami. Tento prístup umožňuje riešiť aj veľmi zložité problémy, neriešiteľné alebo iba obtiažne riešiteľné inými metódami.

[23]Pri väčších doménach sa pochopiteľne používajú menšie počty premenných. Používajú sa úlohy až do 20 hodnôt v doménach čo však v kombinácii s v tomto prípade typickým počtom premenných menším ako 50 poskytuje menší priestor prehľadávania než uvádzané hodnoty pre ternárne domény.

vzdali svojej univerzálnosti v prospech zaradenia prvkov charakteristických iba pre užšiu skupinu problémov – či už sa jednalo o menšie zúženie aplikačnej oblasti napr. zohľadnením epistatickej podstaty problémov s ohraničeniami (danej veľkým množstvom väzieb medzi hľadanými hodnotami premenných, sprostredkovaných definovanými ohraničeniami) alebo o väčšie zúženie aplikačnej oblasti reprezentované napr. zohľadňovaním štruktúry siete ohraničení.

Dobré výsledky boli dosiahnuté aj použitím adaptívnych verzií evolučných algoritmov, ktoré sa v určitej miere dokázali prispôsobiť konkrétnemu riešenému problému.

Keďže pri znalostných prístupoch jednotlivé metódy (modely uvažovania) reprezentujú šablóny, ktoré je potrebné naplniť doménovými znalosťami viazanými na konkrétny riešený prípad, tak tieto metódy sú také dobré ako kvalitnými znalosťami boli doplnené. To znamená, že ich nie je možné porovnať z napr. hľadiska rýchlosti riešenia.

Možným kritériom by bolo množstvo a typ požadovaných znalostí ktoré je potrebné získať pre riešenie problému. Napr. metóda prezentovaná v prílohe D vyžaduje získanie znalostí o ohraničeniach, o prípustných opravách odstraňujúcich porušenia jednotlivých ohraničení a o možných prípadoch uvoľňovania ohraničení. Výkonnosť metódy bude vyššia, ak budú prítomné aj znalosti pre explicitnú inicializáciu – tieto však nie sú nutné pre použitie metódy, pretože inicializácia môže byť realizovaná napr. náhodným spôsobom.

Keďže však nejaký typ znalostí pri jednom probléme je dostupný pomerne ľahko zatiaľ čo pre iný problém nie je dostupný vôbec, tak nie je možné porovnanie, založené na typoch vyžadovaných znalostí, realizovať vo všeobecnej rovine odhliadnuc od konkrétneho problému. Pre každý problém je totiž najvhodnejšia tá metóda, ktorá na jednej strane vyžaduje iba také znalosti ktoré sú pre riešenie daného problému dostupné a na druhej strane vie zužitkovať čo najviac z tých znalostí ktoré sú pre riešenie daného problému k dispozícii.

Zaujímavým spôsobom pristupuje k výberu vhodnej metódy pre riešenie konkrétneho problému [1]. Je možné kandidátske metódy nechať určitú dobu riešiť daný problém, pričom sa sleduje nejaká výkonnostná charakteristika poskytujúca predikciu toho, ktorá metóda je ako vhodná pre daný problém. Keďže daná charakteristika musí zohľadňovať podstatu testovaných metód, tak nemôže byť úplne univerzálnou. V práci je prezentovaná napr. charakteristika vhodná pre odhad výkonnosti kombinovaných algoritmov pre spĺňanie ohraničení.

V niektorých prácach je možné objaviť porovnanie nejakého prístupu voči známym heuristickým metódam používaným pre riešenie nejakého typu problémov s ohraničeniami. Príkladom môže byť problém vyfarbovania grafov, kde evolučný algoritmus podľa [19] bol síce pre grafy s malým počtom uzlov horší ako DSatur heuristika avšak pre veľké grafy bol lepšou alternatívou.

Porovnanie prezentovaných prístupov navzájom nie je jednoduché. Literatúra obsahuje iba veľmi málo prác venujúcim sa tejto otázke a keď áno, tak nie sú

ritmov. Prehľadávacie algoritmy sú dopĺňané redukčnou zložkou kvôli zmenšeniu hodnoty d – veľkosti domén ktoré je potrebné prehľadať.

Ani podrobný teoretický rozbor vlastností dvoch metód nemusí jednoznačne stanoviť preferenciu jednej metódy voči druhej – pre nejaký typ problémov môže byť jedna metóda lepšia než druhá, zatiaľ čo pre iný typ problémov ich úlohy môžu byť vymenené. Napr. metóda poskytujúca väčšie orezanie priestoru prehľadávania môže byť výpočtovo zložitejšia – a teda výsledný efekt použitia takejto metódy závisí od konkrétneho problému, či zisk v orezávaní preváži stratu spôsobenú zložitosťou metódy alebo nie. Dôsledkom toho je, že pri vzájomnom porovnaní viacerých metód vznikajú rôzne parciálne usporiadania v závislosti na použitom kritériu(-ách) [15] [34] [61].

Okrem hľadania najlepšieho algoritmu môže existovať aj snaha nájsť "nie zlý" algoritmus (obzvlášť v podmienkach ak všeobecne najlepší ani neexistuje). To je algoritmus ktorý síce nie je nutne najlepší pre každý problém avšak oproti najlepšiemu algoritmu pre ten-ktorý problém jeho výkonnosť nie je omnoho horšia pre žiadny typ problému. Avšak aj obsadenie takejto úlohy sa môže časom meniť. V komunite odborníkov, zaoberajúcich sa oblasťou metód spĺňania ohraničení, dlhú dobu prevládal názor, že použitie algoritmu doprednej kontroly nie je zlá voľba, či už samotného alebo kombinovaného s nejakým variantom spätného skoku a/alebo heuristikou pre výber premennej. Toto však v nie veľmi dávnej minulosti bolo niektorými autormi spochybnené v prospech algoritmu udržiavania hranovej konzistencie [6].

V oblasti použitia evolučných algoritmov pre riešenie úloh s ohraničeniami, tak ako to je zvykom v oblasti evolučným algoritmov všeobecne, našla uplatnenie predovšetkým empirická analýza. Konkrétne výsledky jednotlivých porovnaní je možné nájsť v prácach, ktorých zoznam je súčasťou textu uvedeného v prílohe B. Väčšinou však tieto analýzy zahŕňajú iba porovnanie variantov prezentovaných v danej práci navzájom alebo ich porovnanie voči štandardnej podobe evolučného algoritmu. Iba veľmi málo prác sa venuje porovnaniu variantov určených pre riešenie úloh s ohraničeniami ktoré by boli vyvinuté rôznymi autormi.

Všetky tieto práce signalizujú chabú výkonnosť evolučného algoritmu základného (štandardne používaného) tvaru. Nie je to prekvapujúce ak si uvedomíme že evolučný algoritmus:

1. je typickým predstaviteľom *slabých* (weak) algoritmov – algoritmov schopných riešiť veľmi široký okruh úloh, platiacich však za to nízkou výkonnosťou pri riešení týchto úloh

2. je použiteľnejší pre riešenie úloh kde postačuje nájdenie dostatočne dobrého riešenia a nie je nutné hľadať riešenie najlepšie.

V súlade s týmto chápaním sú jednoducho interpretovateľné výsledky analýz, ktoré favorizujú tie varianty evolučných algoritmov, ktoré sa v nejakej miere

"Testovanie evolučných algoritmov pre CSP" v prílohe B). Je snaha pri použití nejakého problému ako testovacieho vybrať jeho parametre tak aby úloha bola "ťažká". Z tohto dôvodu sa používa nastavenie parametrov úlohy z oblasti *fázového prechodu* (phase transition) medzi oblasťou, kde ohraničenia ponechávajú dostatok priestoru pre existenciu riešenia, a oblasťou, kde ohraničenia preurčujú problém [30], alebo z oblasti *50% bodu* (označovaného niekedy aj ako cross point) keď polovica z generovaných príkladov má riešenie zatiaľ čo ostatné generované príklady riešenie nemajú [24].

Nevýhodou takéhoto prístupu je, že dosiahnuté výsledky nepovedia nič o chovaní metódy pri riešení príkladu s nejakou inou štruktúrou ohraničení. Navyše, pri porovnávaní výsledkov dosiahnutých rôznymi autormi, je potrebné uvážiť aj možné skreslenie zapríčinené rôznymi (rôzne efektívnymi) spôsobmi implementácie toho istého algoritmu rôznymi autormi. Popis mnohých ďalších faktorov relevantných pre empirickú analýzu možno nájsť v [25].

Čo sa týka metód spĺňania ohraničení, vyhodnotenia jednotlivých metód je možné nájsť v prácach ktorých zoznam je uvedený na konci textu prezentovaného v prílohe A.

Samotné redukčné algoritmy sa samostatne veľmi nepoužívajú. Je to dané ich časovou zložitosťou (veď už pre zabezpečenie 3-konzistencie je v najhoršom prípade časová zložitosť existujúcich metód závislá na počte premenných kvadratickým spôsobom a pre riešenie reálnych úloh je často potrebné zabezpečiť ešte vyšší stupeň konzistencie). Svoje použitie nachádzajú najmä v spojitosti s inými typmi algoritmov.

Základom prakticky používaných metód pre spĺňanie ohraničení býva prehľadávanie. Vo všeobecnosti každý problém môže byť riešený prostredníctvom štandardného algoritmu navracania. Problémom pri jeho použití je však to, že tento algoritmus nie je príliš efektívny – jeho časová zložitosť pre najhorší možný prípad[22] (keď riešenie neexistuje – aby toto daný algoritmus zistil tak musí vygenerovať maximálny strom prehľadávania) je $O(md^n)$ kde m je počet ohraničení, n počet premenných a d veľkosť domén.

Z dôvodu tejto zložitosti pri samostatnom použití prehľadávacích algoritmov je algoritmus navracania nahradzovaný skokovými a/alebo pamäťovými variantmi prehľadávacích algoritmov. Použitie heuristík pre usporiadanie premenných a domén je vedené snahou orientovať hľadanie do sľubných oblastí priestoru prehľadávania a tým sa vyhnúť uvádzanému najhoršiemu prípadu čo možno najviac.

Z vyššie uvedenej časovej zložitosti je zrejmá aj podstata hybridných algo-

[22]V najhoršom možnom prípade je potrebné vygenerovať všetky body priestoru prehľadávania pretože nedochádza k žiadnemu orezávaniu tohto priestoru počas prehľadávania. Súčasne pri kontrole jednotlivých ohraničení v najhoršom možnom prípade je porušené vždy to ohraničenie, ktoré je kontrolované ako posledné – a teda pri kontrole každého skúmaného potenciálneho riešenia je potrebné preskúmať správnosť všetkých ohraničení.

5 Ktorá je tá pravá?

Pri existencii veľkého množstva metód pre riešenie toho istého problému je prirodzenou otázkou tá z nadpisu kapitoly – ktorá z tohto množstva metód je tou vhodnou práve pre daný problém? Je nutné hneď na úvod povedať, že neexistuje žiadne globálne porovnanie všetkých metód. To, čo je možné v literatúre nájsť, sú iba parciálne porovnania zahŕňajúce iba malý počet metód[21]. Iba niekoľko málo prác sa venovalo rozsiahlejšiemu porovnaniu – aj tu sa však zvyčajne počet porovnávaných metód dal vyjadriť jednociferným číslom.

Ešte horšia situácia než vo výbere vhodnej metódy spomedzi metód určitého typu je v porovnávaní rôznych prístupov navzájom. Dôvodom je aj obtiažnosť voľby porovnávacieho kritéria, pretože metódy založené na rôznych prístupoch riešia problém rozdielnymi spôsobmi a teda jedna charakteristika nemusí byť aplikovateľná na všetkých účastníkov porovnania. Zdá sa síce, že vhodnou vlastnosťou by mohol byť čas potrebný k nájdeniu riešenia (resp. fakt či v danom časovom horizonte bolo toto riešenie nájdené) avšak toto môže byť skreslené rôznou úrovňou implementácie porovnávaných metód. Navyše existuje veľa charakteristík, ktoré nie sú relevantné z teoretického hľadiska, avšak sú veľmi dôležité pre praktické použitie jednotlivých metód, ako napr. jednoduchosť a rýchlosť implementácie, jednoduchosť vysvetlenia metódy, ap.

Vo všeobecnosti je možné metódy porovnať prostredníctvom dvoch rôznych prístupov:

- teoreticky
- empiricky.

Teoretická analýza sa z prístupov, prezentovaných v tomto texte, využíva iba pri metódach spĺňania ohraničení. Dôvodom je to, že sa jedná o triedu úplných algoritmov, ktoré na rozdiel od ostatných dvoch prezentovaných prístupov garantujú nájdenie riešenia ak toto riešenie existuje.

Najčastejším typom teoretickej analýzy je určovanie zložitosti (časovej a priestorovej) vo forme závislosti na takých charakteristikách problému ako sú počet premenných, počet ohraničení a maximálna veľkosť domén. Zvyčajne sa pritom uvažuje najhorší možný prípad a iba v málo prípadoch sa uvažuje štatisticky priemerný prípad. Nevýhodou takéhoto typu analýzy je, že nepovie nič o typickom chovaní metódy ani o jej chovaní pre konkrétny prípad.

Toto je možné zistiť prostredníctvom empirickej analýzy – realizáciou experimentov založených na riešení konkrétnych prípadov. Pre tieto testy sa používajú buď náhodné problémy alebo typické úlohy slúžiace ako etalóny (viď kap.

[21] Väčšinou si totiž pri prezentácii nejakej novej metódy autor pre porovnanie vybral jednu alebo iba niekoľko málo iných metód – toto umožňuje demonštrovať prevahu prezentovanej metódy iba voči malému počtu iných metód a iba pre istú triedu úloh.

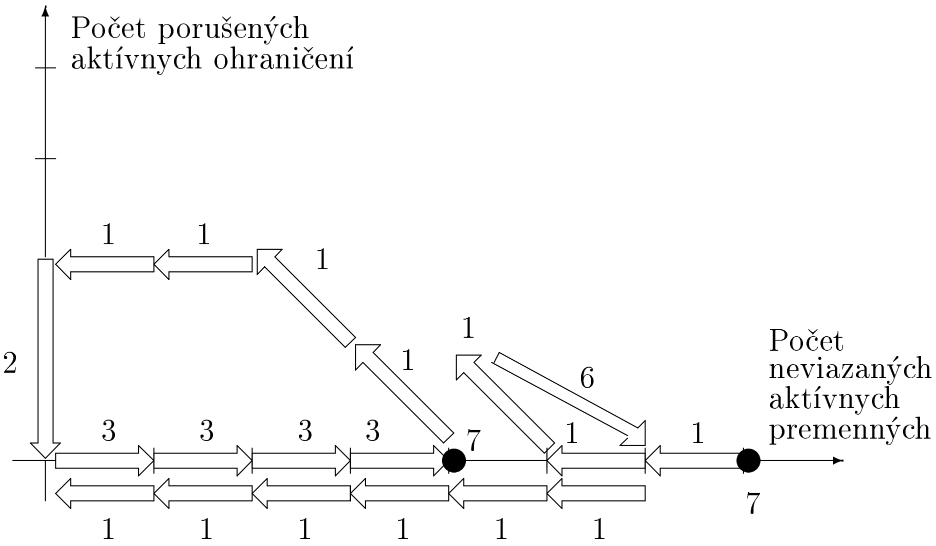
časť trajektórie rovnobežná s vodorovnou osou začínajúca v počiatku a končiaca v bode na vodorovnej osi.

Následne trajektória pokračuje časťou reprezentujúcou priradenie vhodných hodnôt všetkým dosiaľ neviazaným premenným. Najprv je potrebné zohľadniť predchádzajúce priradenia hodnôt úpravou domén aktivovaných premenných (krok 7). Porušenia ohraničení, vzniknuté pri priraďovaní hodnôt týmto novým premenným, sú riešené korekciami. Tieto korekcie však menia hodnoty iba v rámci tej skupiny premenných, ktorá je aktuálne riešená – nedochádza k návratu k skupine premenných ktorým boli hodnoty priraďované v predchádzajúcej etape.

Tu uvedený postup je vhodný pre riešenie statických problémov s ohraničeniami. Znalostný prístup však nemá žiadny problém ani s inými typmi úloh, pretože môže používať ľubovoľný typ riešiaceho kroku. Stačí ak pre jeho realizáciu sú k dispozícii vhodné znalosti. Metóda CRO uvedená v prílohe D je určená pre riešenie optimalizačného problému ktorý nemusí mať riešenie spĺňajúce všetky ohraničenia.

napr. náhodným výberom, avšak je zrejmé, že to pravé urýchlenie riešenia sa získa práve použitím doménovo závislých heuristík.

Z hľadiska znázornenia procesu riešenia úlohy v stavovom priestore neexistuje žiadna typická trajektória reprezentujúca znalostný prístup. Príčinou je to, že v tomto prístupe je možné použiť vo všeobecnosti všetky typy riešiacich krokov (samozrejme ak sa podarí pre ich realizáciu získať potrebné znalosti) a tie usporiadať podľa rôznych stratégií [20] čím sa môžu získať trajektórie rôznych tvarov. Ukážka jednej možnej trajektórie je na obr. 18.



Obr. 18: (Ne)typická trajektória pre znalostný prístup.

Reprezentuje stratégiu "hlavných komponentov" ktorá rozlišuje dôležitejšie a menej dôležité premenné – táto dôležitosť premenných však nie je daná ich významom ale vplyvom na postup riešenia úlohy. Dôležitejšie premenné sú tie, pre ktoré platí, že úloha býva jednoduchšie riešiteľná ak týmto premenným sa priradí hodnota už v rannom štádiu riešenia. Zvyčajne sú to tie premenné, ktoré svojimi hodnotami dokážu determinovať hodnoty ostatných premenných (alebo aspoň obmedziť ich domény na iba niekoľko málo možností).

Preto bola najprv vytypovaná skupina takýchto premenných a iba tie boli ponechané ako aktívne. Po vhodnom priradení hodnôt týmto premenným (čo nemusí byť priamočiary proces) končí prvá časť trajektórie v počiatku. V tomto okamihu je možné sekvenciou aktivačných krokov (krok 3) aktivovať aj ostatné premenné a tak ich zapojiť do hľadania výsledného riešenia. Tomuto zodpovedá

[20] O tom, ktorá stratégia bude použitá pre riešenie konkrétneho problému, môže byť tiež rozhodnuté na základe doménových znalostí.

Jedným z výsledkov neskoršieho vývoja v tejto oblasti sú viacvrstvové znalostné modely. Príkladom môže byť štvorvrstvový expertízny model z KADS metodológie [59] alebo trojvrstvový znalostný model podľa jej nasledovníka – metodológie CommonKADS [55]. Tieto modely reprezentujú prostriedok uľahčujúci ako samotné získavanie znalostí tak aj prenos týchto znalostí pre riešenie iných úloh.

Vrstvy týchto modelov je možné v podstate rozdeliť do dvoch skupín. Spodná doménová vrstva obsahuje konkrétne znalosti týkajúce sa riešeného prípadu – základné fakty, pravidlá, relácie, objekty a štruktúry. Ostatné vrstvy okrem iného obsahujú napr. inferenčnú úroveň obsahujúcu znalosti o inferenciách a znalostných objektoch nad ktorými tieto inferencie sú realizované. Inferenčná vrstva tak reprezentuje akúsi šablónu uvažovania, ktorá nie je závislá na konkrétnom príklade, a teda môže byť znovupoužitá aj pre iné problémy. Príkladom je v znalostnej komunite známa heuristická klasifikácia. Inými príkladmi sú modely pre diagnostiku, monitorovanie, navrhovanie, plánovanie ap.

Doménová vrstva reprezentuje použitie takejto šablóny pre riešenie konkrétneho problému. Jej úlohou je definovať obsah jednotlivých objektov, použitých na všeobecnej inferenčnej úrovni – znalostí pre realizáciu jednotlivých inferenčných krokov (napr. systém pravidiel pre realizáciu unifikačného kroku), znalostných štruktúr nad ktorými je realizovaná inferencia (napr. pozorované príznaky pri klasifikácii) a podporné znalosti (napr. model systému pre diagnostikovanie poruchy podľa modelu).

Výhodou takéhoto prístupu je, že ak sa návrhárovi podarí identifikovať typ úlohy ktorú je potrebné riešiť, potom voľba nejakého modelu pre riešenie takéhoto typu úloh mu jasne špecifikuje, ktoré znalosti z doménovej oblasti je potrebné získať.

Pri použití princípu znalostného modelovania pre riešenie úloh s ohraničeniami rolu znalostných štruktúr, nad ktorými je realizovaná inferencia, hrajú premenné a ich domény. Výsledkom nejakého inferenčného kroku môže byť napr. naviazanie premennej na nejakú hodnotu, uvoľnenie premennej, alebo zmena domény premennej. Iným výsledkom môže byť samotná premenná alebo hodnota ako výsledok selekcie. Jednotlivé inferencie pokrývajú jednak všetky tie riešiace kroky, ktoré sú prípustné pre daný model, a jednak pomocné činnosti (napr. už spomínaná selekcia). Definované ohraničenia zase môžu vystupovať v úlohe podporných znalostí. Inferenčné kroky môžu byť usporiadané v nejakej štruktúre podľa zvolenej stratégie riešenia. Ukážkou môže byť napr. stratégia CRO (complete-revise-optimize) uvedená v prílohe D.

Aby bolo možné nejaký znalostný model použiť, je potrebné doň vložiť konkrétne znalosti. Tak napr. pre realizáciu priraďovacieho kroku je potrebné mať znalosť pre výber premennej, ktorej sa bude priradzovať hodnota, a znalosť pre výber hodnoty z jej domény. Pre uvoľňovací krok je potrebné mať zase znalosti pre výber premennej resp. viacerých premenných na uvoľnenie. Niektoré z týchto znalostí je samozrejme možné nahradiť nejakou všeobecne použiteľnou metódou,

2. opravné procedúry

3. operátory zachovávajúce validitu.

Pri použití prvého spôsobu každý jedinec pred určením jeho vhodnosti je dekódovaný do podoby neporušujúcej žiadne ohraničenie. Podobným spôsobom pracujú opravné procedúry. Ak sa v populácii vyskytne jedinec porušujúci nejaké ohraničenie, tak je opravený – niektorá jeho časť je zmenená s cieľom dosiahnutia jeho validnosti. V poslednom prípade zase ak pre generovanie nového jedinca sa použije materiál jedincov spĺňajúcich všetky ohraničenia, tak potom ich bude spĺňať aj novovygenerovaný potomok. Tieto posledne menované tri prístupy sú značne efektívne pretože algoritmus vlastne prehľadáva iba tú časť priestoru prehľadávania v ktorej každý bod spĺňa všetky ohraničenia. Na druhej strane zostavenie vhodného dekódera, opravnej procedúry alebo operátora zachovávajúceho validitu nie je triviálnou záležitosťou. Navyše, typicky to je záležitosť ktorá je doménovo závislá.

4.3 Znalostný prístup

Podstatou tohto prístupu k riešeniu úloh je orientácia na konkrétne znalosti o tom-ktorom riešenom probléme. Keďže takýmto spôsobom je možné zohľadniť špecifiká riešeného problému a nielen jeho všeobecné charakteristiky, spoločné pre celú skupinu úloh nejakého typu, tak je možné očakávať vytvorenie vysoko efektívneho postupu pre nájdenie riešenia tohto problému. Nevýhodou takejto orientácie na konkrétne znalosti však je námaha, spojená so získavaním potrebných znalostí, a najmä ich neprenositeľnosť pre riešenie iného problému (resp. ich prenositeľnosť pre riešenie iba malého okruhu veľmi podobných úloh).

Korene znalostných prístupov siahajú do druhej polovice šesťdesiatych rokov keď sa stali základom pre tvorbu systémov označovaných ako *expertné* [31], [48]. V počiatočnom období sa ako vedúci princíp používal "prenos znalostí" (knowledge transfer) ktorý neumožňoval diferenciáciu medzi znalosťami a nepodporoval ich organizáciu pomocou nejakých štruktúr (ak sa neráta osobitné uloženie všetkých problémovo špecifických znalostí v síce oddelenej ale ďalej už neštruktúrovanej báze znalostí).

Zmena nastala v osemdesiatych rokoch a bola inicializovaná prácou Allena Newella [44]. Táto zmena súvisela s prechodom k princípu "modelovania znalostí" s oddelením obsahu znalostí od ich reprezentácie. Postupne začali vznikať znalostné modely rozlišujúce rozličné typy znalostí, pričom každý z týchto typov bol charakterizovaný nie spôsobom reprezentácie alebo zdrojom pôvodu, ale úlohou ktorú hral v uvažovanej doménovej oblasti. Vznik týchto prvých modelov je spojený s takými menami ako William J. Clancey, Balakrishnan Chandrasekaran a Luc Steels [10], [11], [12], [57].

spôsobom jedinec bude reprezentovať vždy iba aktívne premenné. Ak sa nejaká premenná stane aktívnou, tak jedinec bude rozšírený o túto premennú a naopak, ak nejaká premenná sa stane neaktívnou, tak reprezentácia jedinca sa môže skrátiť o túto premennú (inak povedané, evolučný algoritmus s premenlivou dĺžkou reprezentácie vlastne vo svojom repertoári bude vlastniť ďalšie dva riešiace kroky – aktiváciu a deaktiváciu). Keďže v tomto prípade rôzne jedince môžu mať rôzne veľkosti, tak v populácie je možné súčasne skúmať rôzne delenia premenných na aktívne a neaktívne.

Pri preurčených úlohách nie je nutná žiadna úprava. Jednoducho sa ako riešenie poskytne najvhodnejší nájdený jedinec. Podmienkou pre to je to aby vhodnosť jedincov nebola založená na dvojhodnotovom posudzovaní "spĺňa všetko/porušuje niečo" ale aby zohľadňovala počty porušených ohraničení resp. dôležitosť týchto ohraničení či ich umiestnenie v hierarchii ohraničení.

O málo zložitejšie je riešenie optimalizačných problémov s ohraničeniami. Tu k vhodnosti založenej na spôsobe spĺňania ohraničení pribúda ešte optimalizačná funkcia. Toto je možné vyriešiť dvoma základnými spôsobmi:

1. penalizačné funkcie
2. mnohokriteriálna optimalizácia.

Prvý spôsob je založený na myšlienke zahrnutia optimalizačnej funkcie do celkovej vhodnosti jedincov [4], [50]. Základom vhodnosti jedinca v tomto prípade zvyčajne býva hodnota optimalizačnej funkcie dosiahnutá daným jedincom. Táto hodnota je súčasne penalizovaná za porušenie každého ohraničenia. Pritom vhodným nastavením váh jednotlivých penalizácií je možné získať rozličné stratégie, napr. keď aj najhoršie riešenie spĺňajúce všetky ohraničenia je vhodnejšie než riešenie porušujúce nejaké ohraničenie, alebo prípad mäkkých ohraničení keď hľadané globálne riešenie leží v oblasti zakázanej nejakým ohraničením. Konkrétny príklad takéhoto riešenia pre prípad mäkkých ohraničení je prezentovaný v prílohe C.

Mnohokriteriálna optimalizácia pracuje súčasne s viacerými vhodnosťami jedinca. Optimalizačná funkcia reprezentuje jednu z nich, zatiaľ čo druhá vhodnosť (resp. ostatné vhodnosti) vyjadruje úspešnosť jedinca pri spĺňaní jednotlivých ohraničení. Typicky sa v tomto prípade využíva pojem paretovej dominancie [9].

Niektoré optimalizačné problémy sa vyznačujú tým že nájdenie nejakého riešenia spĺňajúceho všetky ohraničenia nie je veľmi problematické – teda ak by sa v takýchto problémoch neuvažovala podmienka optimalizácie tak problém by bol jednoduchý. V takýchto úlohách je možné pre hľadanie optimálneho riešenia využiť ďalšie princípy [32], [40], [41]:

1. dekódery

Opäť je potrebné na začiatku vygenerovať počiatočnú populáciu. Generovanie nejakého jedinca tejto počiatočnej populácie v tomto prípade znamená priradiť niektorým premenným nejaké hodnoty z ich domén. Táto časť trajektórie, reprezentovaná postupnosťou niekoľkých priraďovacích krokov, končí vždy v nejakom bode na vodorovnej osi. Počet krokov môže byť rovnaký pre každého jedinca alebo sa tieto počty môžu líšiť.

Druhá časť trajektórie reprezentuje zmeny jedinca. Akonáhle pri týchto zmenách sa trajektória dostane do bodu neležiaceho na vodorovnej osi, tak nasledovným krokom je korigovaný počet porušení ohraničení na nulu. Pri týchto zmenách sa mení počet neviazaných premenných pričom sa môže nielen zmenšovať ale aj zväčšovať. Uvoľňovací krok je teda možné použiť v dvoch situáciách – ako reprezentanta zmeny (v tomto prípade je znázornený ako vodorovná šípka na úrovni vodorovnej osi) alebo ako opravný prostriedok (v tomto prípade začína nad vodorovnou osou a končí na nej).

Jedna zmena môže byť značne komplexná (môže zahŕňať naviazanie nejakých premenných, uvoľnenie iných a zmenu hodnôt ďalších) – a teda môže byť reprezentovaná jedným korekčným krokom, jedným priraďovacím krokom, jedným uvoľňovacím krokom, alebo môže byť zložená z niekoľkých takýchto krokov. Navyše, môže byť doplnená opravným uvoľňovacím krokom.

Ak sa uvažuje trajektória reprezentujúca v každej generácii najvhodnejšieho jedinca, tak v prípade úspešného nájdenia riešenia neporušujúceho žiadne ohraničenie končí daná trajektória v počiatku. V opačnom prípade končí v inom bode na vodorovnej osi.

Varianty evolučných algoritmov prezentované v prílohe B sú tam rozoberané s ohľadom na riešenie statických problémov s ohraničeniami. To však neznamená, že by evolučné algoritmy boli vhodné iba pre riešenie statických úloh – je ich možné použiť pre všetky typy problémov s ohraničeniami.

Evolučné algoritmy vždy sústreďujú svoje úsilie na tú časť priestoru prehľadávania, ktorá sa javí najperspektívnejšia, pričom táto perspektívnosť je reprezentovaná vhodnosťou bodov nachádzajúcich sa v danej oblasti. To znamená, že pri zmene vhodnosti jednotlivých bodov algoritmus je schopný preniesť svoju pozornosť na inú oblasť – bez ohľadu na to či v minulosti túto oblasť už prehľadával alebo nie. Následkom toho je možné riešiť úlohy s dynamickými zmenami ohraničení[19]. Jedinou podmienkou je aby tie ohraničenia, ktoré sa môžu meniť, boli zahrnuté v definícii vhodnosti jedincov (a nie napr. v špeciálnom tvare reprezentácie jedincov).

Pri výskyte dynamických zmien premenných je možné využiť rovnakú transformáciu na statický problém ako pri metódach spĺňania ohraničení. Inou možnosťou je použiť variant algoritmu s premenlivou dĺžkou jedincov pričom takýmto

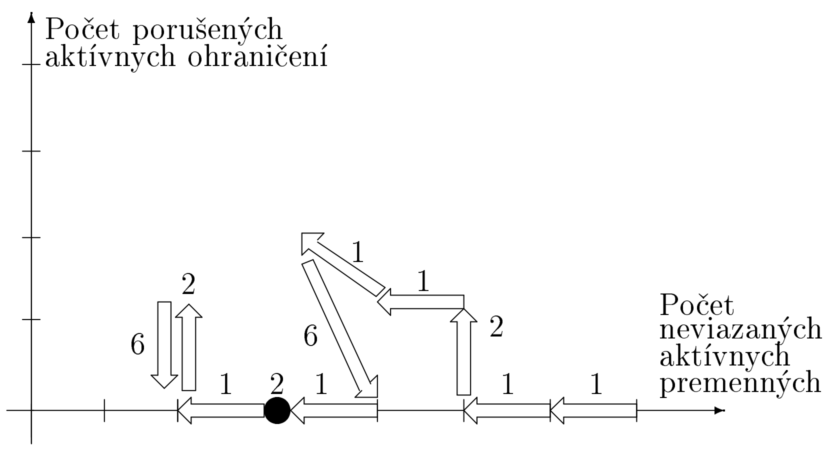
[19] Pri častých a "drastických" zmenách ohraničení môže byť potrebné využiť variant evolučného algoritmu špecializovaného na riešenie nestacionárnych úloh napr. podľa [26], [28], [45], [51].

Ak sa uvažuje trajektória reprezentujúca v každej generácii najvhodnejšieho jedinca, tak v prípade úspešného nájdenia riešenia neporušujúceho žiadne ohraničenie končí daná trajektória v počiatku. V opačnom prípade končí v inom bode na zvislej osi.

Inou možnou reprezentáciou je reprezentácia, keď jedinec reprezentuje parciálne riešenie – nejakú kombináciu hodnôt iba určitej podmnožiny premenných. V tomto prípade takýto jedinec vlastne reprezentuje podpriestor priestoru prehľadávania P. V prípade, že podmnožina premenných s priradenou hodnotou je rozšírená na všetky premenné, tak reprezentovaný podpriestor je redukovaný na jeden bod priestoru prehľadávania P. Keďže nový jedinec vzniká úpravou materiálu existujúcich jedincov, tak pri takomto spôsobe reprezentácie vlastne algoritmus reprezentuje hierarchické prehľadávanie – na základe nejakej skupiny podpriestorov priestoru P je generovaný nový podpriestor v tom istom priestore.

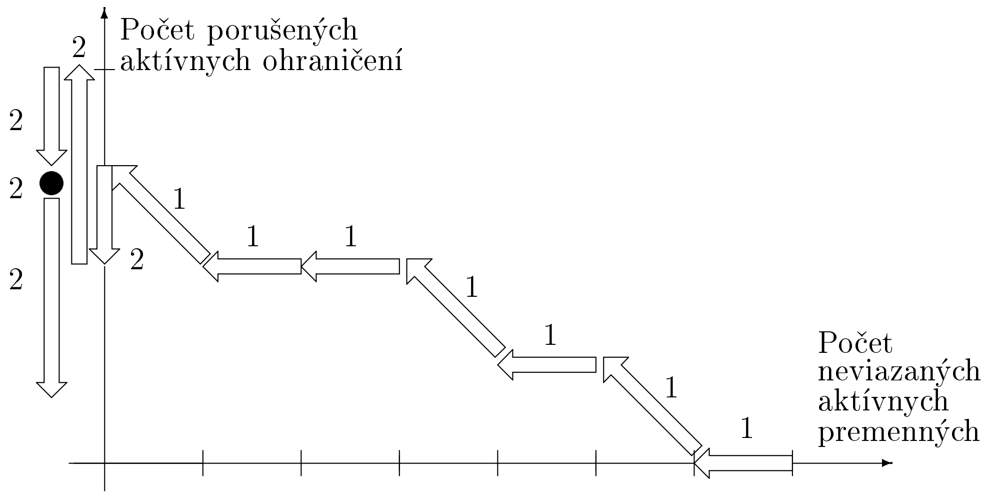
V prípade takejto parciálnej reprezentácie je možné zabezpečiť, že v populácii sa vyskytnú iba jedince ktoré neporušujú žiadne ohraničenie – v prípade porušenia nejakého ohraničenia sa jednoducho premenná, zodpovedná za toto porušenie, uvoľní. Preferencia jedincov je potom založená na možnosti redukcie reprezentovaných podpriestorov pri spĺňaní definovaných ohraničení. Vo všeobecnosti ten jedinec, ktorý môže byť rozšírený o viac hodnôt a viacerými spôsobmi, je považovaný za vhodnejší než ten, ktorého rozšíriteľnosť je menšia.

Z hľadiska stavového priestoru parciálna reprezentácia vyžaduje používanie troch typov krokov: priradenie (krok 1), korekciu (krok 2) a uvoľnenie (krok 6). Pri vytváraní nového jedinca totiž môže dôjsť k zmene hodnoty priradenej nejakej premennej, k priradeniu hodnoty voľnej premennej a taktiež k uvoľneniu premennej keď táto premenná sa stáva neviazanou premennou. Ukážka typickej trajektórie v stavovom priestore je na obr. 17.



Obr. 17: Typická trajektória pre evolučné algoritmy (parciálna reprezentácia).

jedinca tejto počiatočnej populácie vlastne znamená priradiť každej premennej nejakú hodnotu z jej domény. Počas samotného algoritmu je jedinec transformovaný na iného jedinca zmenou hodnôt niektorých premenných (tieto hodnoty môžu byť doplnené z materiálu iných existujúcich jedincov alebo môžu byť jednoducho nahradené nejakými hodnotami z domén príslušných premenných). Tomuto zodpovedá trajektória[18] ktorej tvar je na obr. 16.



Obr. 16: Typická trajektória pre evolučné algoritmy (úplná reprezentácia).

Prvá časť trajektórie pozostáva zo sekvencie priraďovacích krokov. Výsledkom je jedinec v počiatočnej populácii obsahujúci hodnotu pre každú premennú, pričom môže porušovať nejaké ohraničenie – prvá časť trajektórie teda končí vždy v nejakom bode na zvislej osi.

Pri prechode z generácie na generáciu sa jedinec môže meniť ale aj nemusí. Závisí to od toho, či použitý variant evolučného algoritmu umožňuje jedincom zotrvanie v populácii alebo vynucuje ich náhradu a či novovygenerovaný jedinec je rôzny od už existujúceho jedinca alebo nie. Počet korekčných krokov z ktorých je zložená druhá časť trajektórie je vo všeobecnosti rôzny pre rôzne jedince populácie. Extrémne prípady sú ak sa nejaký jedinec nemení počas celého behu algoritmu alebo naopak ak sa mení v každej generácii.

Uvedená trajektória obsahuje aj korekčný krok vedúci k zhoršeniu vhodnosti jedinca. Toto je možné iba v prípade ak použitá verzia evolučného algoritmu pripúšťa náhradu existujúceho jedinca novým jedincom ktorý je horší.

[18] Trajektória znázorňuje zmeny iba jedného jedinca z populácie. V skutočnosti by bolo potrebné v stavovom priestore S znázorniť toľko trajektórií z koľkých jedincov pozostáva populácia.

ďalšími generáciami. Pritom prechod od jednej generácie ku generácii nasledujúcej vo všeobecnosti znamená zmenu jedincov v populácii. Niektoré jedince sú z populácie vypúšťané a uvoľnené pozície sú dopĺňané novými jedincami.

Nové jedince sú generované z materiálu jedincov existujúcich v populácii v tejktorej generácii. Pre vytvorenie jedného nového jedinca je možné využiť jedného alebo viac existujúcich jedincov, pričom táto tvorba je viac alebo menej inšpirovaná princípom génovej dedičnosti. Výber existujúcich jedincov, z ktorých budú vytvárané nové jedince, ako aj výber jedincov, z ktorých bude vytvorená nasledujúca generácia, je realizovaný na pravdepodobnostnom základe pričom niektoré jedince sú preferované na úkor iných jedincov. Táto preferencia vo väčšej alebo menšej miere odráža princíp prirodzeného výberu pôsobiaceho pri evolučnom spôsobe vývoja. Implementovaná je prostredníctvom tzv. *funkcie vhodnosti* (fitness function) udávajúcej šancu jedinca byť vybraným.

Keďže pri riešení konkrétnych úloh každý jedinec predstavuje jedno riešenie, tak uvedený populačný princíp umožňuje prehľadávať priestor prehľadávania P paralelným spôsobom, pričom toto prehľadávanie je kompromisom medzi extenzívnym skúmaním stále nových a nových častí priestoru prehľadávania (reprezentovaným vytváraním nových jedincov) a medzi využívaním získanej informácie pre sústreďovanie prehľadávania iba do najsľubnejších oblastí skúmaného priestoru (zastúpeným výberom jedincov).

V závislosti od spôsobu reprezentácie riešení problému prostredníctvom jedincov sú evolučné algoritmy schopné implementovať ako bodové prehľadávanie priestoru potenciálnych riešení P tak aj hierarchické prehľadávania tohto priestoru[17].

Jednou z možných reprezentácií je tá keď jedinec reprezentuje úplné riešenie – nejakú kombináciu hodnôt všetkých premenných. V tomto prípade takýto jedinec vlastne reprezentuje jeden bod priestoru prehľadávania P. Keďže nový jedinec vzniká úpravou materiálu existujúcich jedincov, tak pri takomto spôsobe reprezentácie vlastne algoritmus reprezentuje bodové prehľadávanie – na základe nejakej skupiny bodov priestore P je generovaný nový bod v tom istom priestore.

V prípade úplnej reprezentácie sa v populácii môžu vyskytovať ako jedince spĺňajúce všetky ohraničenia tak aj jedince porušujúce jedno alebo niekoľko z týchto ohraničení. Preferencia jedincov je potom založená na ich schopnosti splniť definované ohraničenia. Vo všeobecnosti ten jedinec, ktorý spĺňa viac dôležitejších ohraničení, je považovaný za vhodnejší než ten, ktorý spĺňa menej menej dôležitých ohraničení.

Z hľadiska stavového priestoru sa základnými krokmi javia priradenie (krok 1) a korekcia (krok 2). Na začiatku ešte pred samotnou realizáciou jadra evolučného algoritmu je potrebné vygenerovať počiatočnú populáciu. Generovanie nejakého

[17] Je potrebné povedať že hierarchické prehľadávanie sa používa iba ojedinele, v drvivej väčšine sa využíva bodový spôsob prehľadávania.

odznovu ale aj pokračovať v riešení problému.

Túto optimalizačnú techniku je možné využiť aj pre riešenie preurčených problémov. Vytvorí sa problém ktorého ohraničeniami budú všetky tvrdé ohraničenia pôvodného problému – tie ktoré nutne musia byť splnené. Nesplnenie ostatných ohraničení sa premietne do zostavenia optimalizačnej funkcie – jej hodnota bude tým väčšia, čím bude porušených viac ohraničení, čím budú porušené dôležitejšie ohraničenia resp. čím viac budú ohraničenia porušené. Bude sa teda jednať o minimalizačný problém pričom dolná medza pre hodnotu optimalizačnej funkcie bude 0 a horná zase bude daná porušením všetkých mäkkých ohraničení pôvodného problému.

4.2 Evolučné algoritmy

Evolučné algoritmy tvoria pomerne 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 [49] a H.P. Schwefela [56] v oblasti riešenia optimalizačných úloh z oblasti hydrodynamiky, práce L.J. Fogela, A.J. Owensa a M.J. Walsha [20] v oblasti riešenia predikčných úloh, a práce J.H. Hollanda [33] z oblasti skúmania fenoménu adaptácie v prírode a jeho prenosu do umelých systémov. Tieto práce sa stali hybným momentom pre vznik *evolučných stratégií* (evolution strategies), *evolučného programovania* (evolutionary programming) a *genetických algoritmov* (genetic algorithms).

Tieto tri smery sa vyvíjali spočiatku izolovane až do začiatku deväťdesiatych rokov, kedy medzi nimi došlo ku kontaktu. Odvtedy dochádza aj k vzniku algoritmov spájajúcich prvky typické pre rôzne smery. Z pôvodných smerov posledne spomínaný nadobudol spomedzi nich najväčšie rozšírenie a stal sa dominantným. S jeho ďalším rozvojom[16] sú spojené také mená ako Lawrence Davis, Kenneth A. DeJong, David E. Goldberg, John J. Grefenstette, Darrell Whitley [13], [14], [17]. Sú spájané ako s rozvojom teórie tak aj s aplikovaním týchto algoritmov na najrozličnejšie problémy. Špeciálne s ich využívaním pre riešenie úloh s ohraničeniami je spájané meno Zbigniewa Michalewicza [40].

Triedu evolučných algoritmov je možné stručne charakterizovať ako populačné stochastické algoritmy. Typicky pracujú nad populáciou jedincov. Táto populácia počas práce algoritmu je podrobovaná zmenám – prechádza postupne ďalšími a

[16]Pre zakladateľa smeru Johna H. Hollanda boli genetické algoritmy (ním nazývané ako *adaptívne a reprodukčné plány*) iba prostriedkom pre modelovanie adaptačných procesov. Súčasný tvar genetického algoritmu ako nástroja pre riešenie úloh sa objavil napr. v práci [26] ktorá sa stala "bibliou" komunity priaznivcov genetických algoritmov.

ných nad premennými, ktoré sa stávajú aktívnymi, alebo z deaktivácie ohraničení definovaných nad nejakou premennou, ktorá sa stala neaktívnou. V tomto prípade je možné tento problém pretransformovať na statický tak, že domény jednotlivých premenných budú doplnené o špeciálnu hodnotu znamenajúcu neaktívnosť premennej. Ohraničenie definované nad premennou ktorej bola priradená táto špeciálna hodnota je považované za splnené bez ohľadu na to aké hodnoty sú priradené ostatným relevantným premenným.

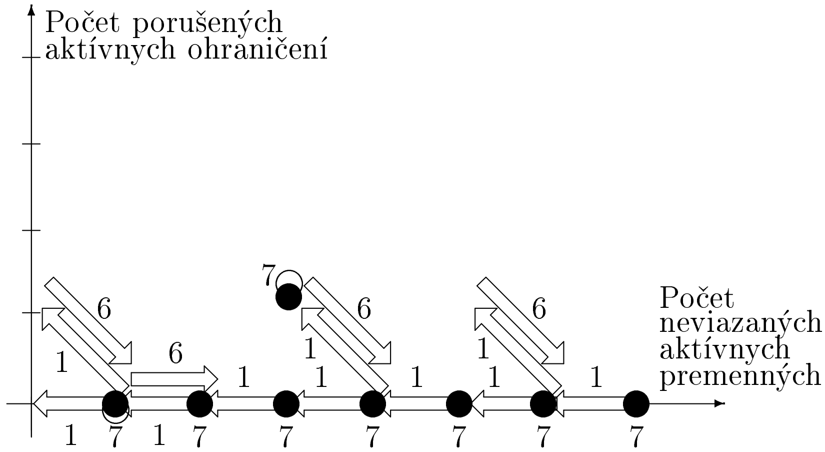
Pri dynamických zmenách ohraničení závisí postup od toho či ohraničenie bolo sprísnené alebo uvoľnené. Sprísnenie ohraničenia je možné "beztrestne" realizovať počas vykonávania algoritmu keďže nemá vplyv na výsledky redukcie/prehľadávania časti priestoru P pred zmenou ohraničenia (pretože riešenie v tejto časti priestoru P nebolo nájdené pred sprísnením ohraničenia, tak tam nemôže byť nájdené ani po jeho sprísnení). Naproti tomu uvoľnenie ohraničenia signalizuje, že vykonaná redukcia/prehľadávanie časti priestoru P už nie je platné – riešenie sa môže vyskytnúť aj v tej časti priestoru P kde sa predtým nevyskytlo. V tomto prípade zmena ohraničenia spôsobuje vytvorenie nového statického problému ktorý je potrebné riešiť.

Pri preurčených problémoch riešenie takejto úlohy končí neúspechom. V tomto prípade je možné uvoľnením (neuvažovaním) nejakého ohraničenia definovať nový statický problém. Takýmto spôsobom sa vytvárajú nové a nové problémy s rôznymi kombináciami uvoľnených ohraničení dovtedy, kým sa nájde jeden s existujúcim riešením [46]. Je zrejmé že tento postup je použiteľný skôr v prípade existencie hierarchie ohraničení než v prípade ohraničení rovnakej dôležitosti. Hierarchia ohraničení umožní stanoviť ktorú skupinu ohraničení je vhodné skúsiť uvoľniť v tom ktorom kroku (najprv sa zvyčajne neuvažujú najmenej významné ohraničenia).

V prípade, že sa požaduje nie ľubovoľné riešenie, ale je zadaná aj optimalizačná funkcia podľa ktorej sa posudzuje kvalita jednotlivých riešení, pristupuje k prehľadávaniu ešte aj optimalizácia. Základnou myšlienkou využívanou v tomto prípade je hľadať riešenie obvyklým spôsobom, po jeho nájdení vytvoriť nový statický problém pridaním nového ohraničenia ktoré zabezpečí, že riešenie musí byť lepšie ako doteraz najlepšie nájdené riešenie, a opäť spustiť hľadanie.

Pri znalosti dolnej a hornej medze pre hodnotu optimalizačnej funkcie (pri minimalizácii napr. horná hranica môže byť daná doteraz najlepším riešením) je možné použiť rozličné stratégie pre definovanie hodnoty pridávaného ohraničenia napr. zmenou o nejaký inkrement [43] alebo delením intervalu na polovicu [54] [47]. Je dôležité si uvedomiť, že celkový čas potrebný na nájdenie riešenia môže výrazne ovplyvniť voľba týchto hraníc na začiatku hľadania riešenia. Efektívna voľba týchto hraníc je silne závislá od konkrétnych znalostí o riešenej úlohe. Pre rozsiahle úlohy je obvykle nevyhnutné venovať určitý čas na výpočet čo možno najlepšieho odhadu pre dolnú a hornú hranicu.

Keďže nachádzaním lepších a lepších riešení sa mení iba dodatočné ohraničenie tým že sa sprísňuje, tak po jeho zmene je možné nielen začať riešiť problém



Obr. 15: Typická trajektória pre kombinované algoritmy.

priraďovacieho kroku alebo po aplikovaní kroku redukčného. V prípade priraďovacieho kroku je nutnosť navracania signalizovaná porušením nejakého ohraničenia. V prípade redukčného kroku to zase signalizuje vyprázdnenie domény niektorej z neviazaných premenných. V oboch prípadoch prvé použitie uvoľňovacieho kroku končí na vodorovnej osi.

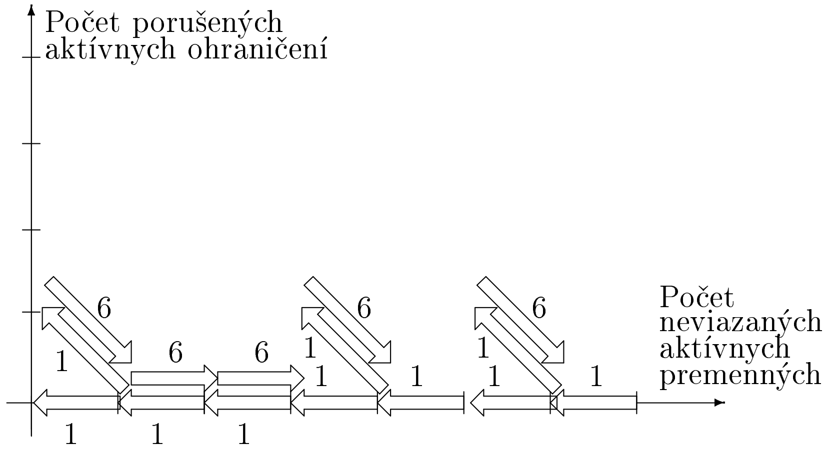
Ak úloha má riešenie, tak trajektória opäť končí v počiatku – a ako posledný krok je vždy aplikovaný priraďovací krok. Ak úloha riešenie nemá, tak potom trajektória končí v tom istom bode, v ktorom začala. Posledným krokom v tomto prípade je uvoľňovací krok - nemá zmysel po uvoľnení prvej premennej odstraňovať redukciu spôsobenú predspracovaním.

Podoby algoritmov pre spĺňanie ohraničení uvedené v Prílohe A sú primárne určené pre riešenie statických problémov s ohraničeniami. Sú totiž založené na predpoklade nemennosti množiny premenných a ohraničení. Pri svojej činnosti buď nájdu riešenie spĺňajúce všetky ohraničenia alebo končia neúspechom. A konečne, uvažujú iba jediný typ vlastností skúmaných riešení – ich schopnosť splniť definované ohraničenia.

Napriek tomu ich je možné do určitej miery využiť aj pre riešenie iných typov problémov s ohraničeniami – dynamických, preurčených aj optimalizačných[15]. Typicky sa takéto problémy transformujú na postupnosti statických úloh riešiteľných týmito metódami.

Pomerne jednoduchá situácia nastáva pri dynamických zmenách premenných, ak zmeny jednotlivých ohraničení pozostávajú iba z aktivácie ohraničení definova-

[15]Je však potrebné upozorniť že pre riešenie týchto špecifických typov problémov s ohraničeniami existujú aj špecializované varianty spomínaných metód napr. [5] [42] [62] pre dynamické problémy, [23] [37] [63] pre preurčené problémy a [60] pre optimalizačné problémy. Tieto varianty algoritmov nie sú náplňou predkladaného textu.



Obr. 14: Typická trajektória pre prehľadávacie algoritmy.

úloh s ohraničeniami umožňujúce viac obmedziť oblasti priestoru P ktoré je potrebné prehľadať.

Samozrejme je možné kombinovať redukčné algoritmy s prehľadávacími tak, že sa využijú prednosti jedného typu na vyváženie nevýhod druhého typu algoritmov a naopak. Pri takejto hybridizácii problém výpočtovej zložitosti redukčných algoritmov je obchádzaný tým, že sa požaduje zabezpečenie nižšieho stupňa konzistencie, než ako by to bolo nutné v prípade samotných redukčných algoritmov. Princíp takéhoto spojenia je jednoduchý – naďalej sa prehľadáva priestor P. V okamihu, keď to je možné, je tento priestor redukovaný pomocou redukčných algoritmov čím sa zmenšuje priestor, ktorý musí byť naďalej prehľadaný pomocou prehľadávacích algoritmov.

Z hľadiska stavového priestoru hybridné algoritmy používajú širšiu paletu základných typov riešiacich krokov: priradenie (krok 1), uvoľnenie (krok 6) a redukciu (krok 7). Typický tvar výslednej trajektórie je na obr. 15.

Počet redukčných krokov závisí od toho, v ktorých fázach prehľadávania sa šírenie ohraničení používa. Je ho možné použiť pred začiatkom samotného prehľadávania (toto je označované ako predspracovanie) alebo aj počas prehľadávania medzi jednotlivými priraďovacími krokmi (táto redukcia je postavená na tom, že priradenie hodnoty nejakej premennej je vlastne to isté ako zúženie domény tejto premennej iba na jednu hodnotu).

Niekedy (pri navracaní k predchádzajúcej premennej) je potrebné aplikovať redukčný krok v opačnom zmysle – je potrebné obnoviť stav ohraničení a jednotlivých domén do tvaru pred redukčným krokom (na obrázku je takýto spätný redukčný krok znázornený ako nevyplnený krúžok).

Slepý koniec s následným návratom k jednej z už naviazaných premenných môže byť detekovaný pri takomto type algoritmu v dvoch prípadoch: po aplikácii

ným priraďovaním hodnôt jednotlivým premenným) až na jeden bod alebo pri detekovaní porušenia nejakého ohraničenia návrat na najbližšiu úroveň s ešte nepreskúmanou alternatívou[14].

Zmenšovanie priestoru prehľadávania pri použití prehľadávacích algoritmov má iný charakter ako v prípade redukčných algoritmov (aj keď v prípade binárnych premenných to nie je viditeľné navonok). Zatiaľ čo pri redukčných algoritmoch bol prítomný princíp "vypustiť všetky kombinácie hodnôt premenných obsahujúce hodnotu, ktorá je vypúšťaná z domény nejakej premennej" tak teraz to je "ponechať všetky kombinácie hodnôt premenných obsahujúce hodnotu priraďovanú nejakej premennej".

Z hľadiska stavového priestoru podobne ako algoritmy šírenia ohraničení aj uvedené prehľadávacie algoritmy využívajú iba dva typy riešiacich krokov. V tomto prípade to sú priradenie (krok 1) a uvoľnenie (krok 6). Keďže je možné aj také priradenie, ktoré spôsobí porušenie nejakého ohraničenia, tak priraďovací krok môže byť znázorňovaný dvojako. Naproti tomu je uvoľňovaná vždy tá premenná ako prvá, ktorej viazanie na konkrétnu hodnotu z jej domény spôsobilo všetky porušenia ohraničení – a teda uvoľňovací krok vždy končí na vodorovnej osi. Pri algoritme navracania je dĺžka kroku (vo vodorovnom smere) vždy jednotková. Pri skokových algoritmoch táto dĺžka môže byť väčšia pretože je možné uvoľniť naraz viacero premenných (a teda jeden uvoľňovací krok nie je inverziou jedného priraďovacieho).

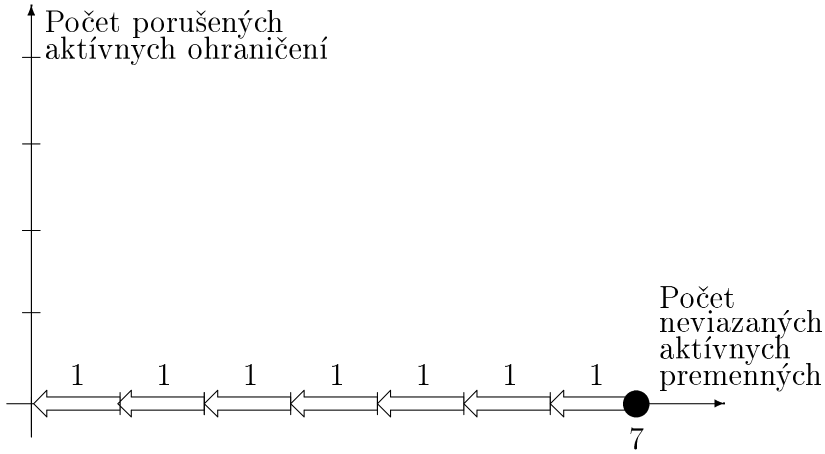
Tieto dva typy riešiacich krokov sa striedajú podľa toho, či momentálnej etape hľadania riešenia zodpovedá zužovanie priestoru prehľadávania alebo návrat k alternatívnemu rozhodnutiu. Ich počet nemôže byť vopred presne daný. Počet priraďovacích krokov je minimálne n (je potrebné priradiť hodnoty všetkým premenným) a maximálne $\parallel D_1 \parallel \times \ldots \times \parallel D_n \parallel$ (v najhoršom prípade je potrebné vyskúšať všetky možné kombinácie hodnôt z domén premenných). V ideálnom prípade nie je nutné realizovať žiadny uvoľňovací krok. Celkový počet uvoľňovacích krokov nepresiahne počet priraďovacích krokov (v prípade, že úloha nemá riešenie, je potrebné uvoľniť všetky realizované priradenia a teda počet priraďovacích krokov je rovnaký ako počet uvoľňovacích krokov).

Takýmto spôsobom vzniká tvar trajektórie v stavovom priestore typický pre prehľadávacie algoritmy, ktorý je znázornený na obr. 14.

Ak má úloha riešenie, tak trajektória končí v počiatku. Ak však úloha riešenie nemá, tak potom trajektória končí v tom istom bode, v ktorom začala (žiadna premenná nemá priradenú hodnotu a teda žiadne ohraničenie nemôže byť porušené).

Prehľadávacie algoritmy je možné použiť samostatne aj pre riešenie všeobecného problému s ohraničeniami. Ibaže v tomto prípade sa typicky nepoužíva základný algoritmus navracania ale nejaké jeho varianty zohľadňujúce podstatu

[14]Situácia, keď sa nejakej premennej nepodarilo konzistentne priradiť žiadnu hodnotu z jej domény a je teda nutné inicializovať navracanie, sa označuje ako *slepý koniec* (deadend).



Obr. 13: Typická trajektória pre redukčné algoritmy.

Táto trajektória je zložená iba z dvoch riešiacich krokov: redukcie (krok 7) a priradenia (krok 1).

Prvý typ kroku zodpovedá procesu šírenia ohraničení pri konzistenčných algoritmoch alebo substitúcii hodnôt pri substitučných algoritmoch keď dochádza k úpravám jednotlivých ohraničení a domén. Počet krokov takéhoto typu je premenlivý a závisí od konkrétneho problému.

Druhý typ krokov slúži pre vygenerovanie riešenia. Počet priraďovacích krokov je n – je potrebné priradiť hodnotu každej premennej. Pritom priraďovanie hodnoty nejakej premennej V_i sa nedeje ako výber ľubovoľnej hodnoty z jej domény D_i ale pre tento výber sa použijú iba tie hodnoty z domény D_i, ktoré sú konzistentné s hodnotami už priradenými ostatným premenným. Tým sa napríklad pri existencii viacerých disjunktných riešení zabráni tomu, aby sa nejakej premennej priradila hodnota patriaca k jednému riešeniu a nejakej inej premennej zase hodnota z druhého riešenia.

V prípade neexistencie riešenia aplikovaním redukčného kroku sa získa prázdny podpriestor (doména niektorej premennej sa vyprázdni) a teda nie je možné aplikovať priraďovací krok.

Aj keď použitie redukčných algoritmov samotných zaručuje nájdenie riešenia, uvedené použitie pre všeobecný problém s ohraničeniami je skôr teoretické ako praktické. Dôvodom je veľká zložitosť týchto algoritmov obmedzujúca ich praktickú použiteľnosť. Iná situácia je pri niektorých špeciálnych tvaroch úloh keď je postačujúce zabezpečiť iba nejakú nižšiu úroveň konzistencie – vtedy sú tieto algoritmy využiteľné naznačeným spôsobom.

Podstatou prehľadávacích algoritmov je stratégia hierarchického prehľadávania priestoru prehľadávania P. Základom všetkých týchto algoritmov je klasický algoritmus navracania – postupné zmenšovanie priestoru prehľadávania (sekvenč-

Jednotlivé algoritmy je možné podľa spôsobu manipulácie s priestorom prehľadávania P rozdeliť do dvoch základných tried:

- redukčné algoritmy
- prehľadávacie algoritmy.

Podstatou redukčných algoritmov je orezanie priestoru P na nejaký jeho podpriestor. Ak úloha má jedno alebo viac riešení tak tento výsledný podpriestor v ideálnom prípade bude obsahovať iba jeden bod – jedno z možných riešení. Ak však úloha nemá riešenie ktoré by splnilo všetky definované ohraničenia, tak priestor P je možné redukovať na prázdny podpriestor.

Samotná redukcia môže byť založená na dvoch princípoch – filtrácii a substitúcii. 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.

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. Takýto reťazový proces filtrácie sa označuje ako *šírenie ohraničení* (constraint propagation).

Naproti tomu substitúcia je založená na existencii viacerých riešení ktoré zdieľajú hodnoty nejakej skupiny premenných. Potom z jedného riešenia je možné získať riešenie iné tak, že pre tie premenné, v ktorých sa tieto dve riešenia líšia, bude skupina hodnôt vyskytujúca sa v jednom riešení nahradená skupinou hodnôt vyskytujúcou sa v druhom riešení. V takomto prípade nie je potrebné uvažovať všetky riešenia ale iba jedno – pomocou nájdených substitúcií je možné ostatné riešenia spätne vytvoriť z tohto jedného. Substitučné algoritmy sa teda snažia redukovať domény premenných o tie hodnoty, ktoré vystupujú v alternatívnych riešeniach.

Filtračný princíp je základom konzistenčných algoritmov (consistency enforcing algorithms) a substitučný princíp zase tvorí jadro substitučných algoritmov. Oba tieto typy vytvárajú spolu triedu redukčných algoritmov.

Z hľadiska znázornenia procesu riešenia úlohy v stavovom priestore je pre samostatné použitie redukčných algoritmov charakteristický tvar trajektórie podľa obr. 13.

4 Metódy riešenia problémov s ohraničeniami

V tejto kapitole budú uvedené tri prístupy k riešeniu problémov s ohraničeniami: spĺňanie ohraničení, evolučné algoritmy a znalostný prístup. Ich popis je prezentovaný na pomerne málo detailnej úrovni – cieľom je skôr poskytnúť jednotiaci pohľad na tieto prístupy využitím mechanizmu riešiacich krokov.

Detaily o niektorých metódach patriacich k prezentovaným prístupom je možné nájsť v prílohách. Spĺňaniu ohraničení je venovaná príloha A. Príloha B sa zaoberá použitím evolučných algoritmov pre riešenie úloh s ohraničeniami a príloha C prezentuje jeden variant genetického algoritmu pre riešenie preurčených optimalizačných úloh s mäkkými ohraničeniami. Znalostný prístup je v prílohe D reprezentovaný metódou pre riešenie preurčených optimalizačných úloh.

4.1 Spĺňanie ohraničení

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

Jednou z prvých aplikačných domén sa stala interaktívna grafika, kde prvým systémom bol Sutherlandov Sketchpad [58] začiatkom šesťdesiatych rokov. Jeho pokračovateľom je Borningov ThingLab [7]. 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 [64].

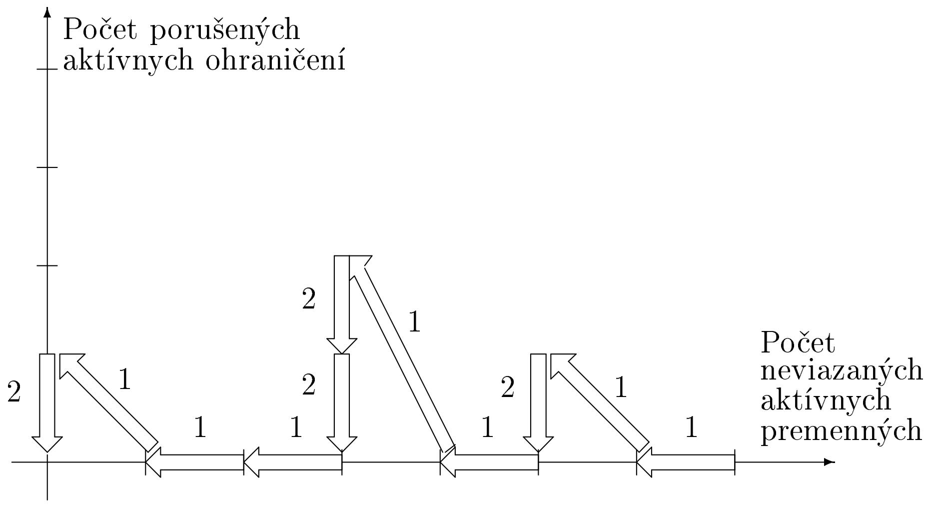
Ďalší ranný vývoj tejto oblasti je spojený s takými menami ako Eugene C. Freuder, John Gaschnig, Robert M. Haralick, Alan K. Mackworth a Ugo Montanari [38], [21], [29]. Došlo k abstrahovaniu vlastností algoritmov, k ich oddeleniu od aplikácií a k formovaniu oblasti do takého tvaru v akom je známa dnes[13].

[13]Popisy základných algoritmov pre spĺňanie ohraničení je možné nájsť napr. v prehľadových publikáciách [16], [36] a [39]. Najobsiahlejšou publikáciou takéhoto typu je [60].

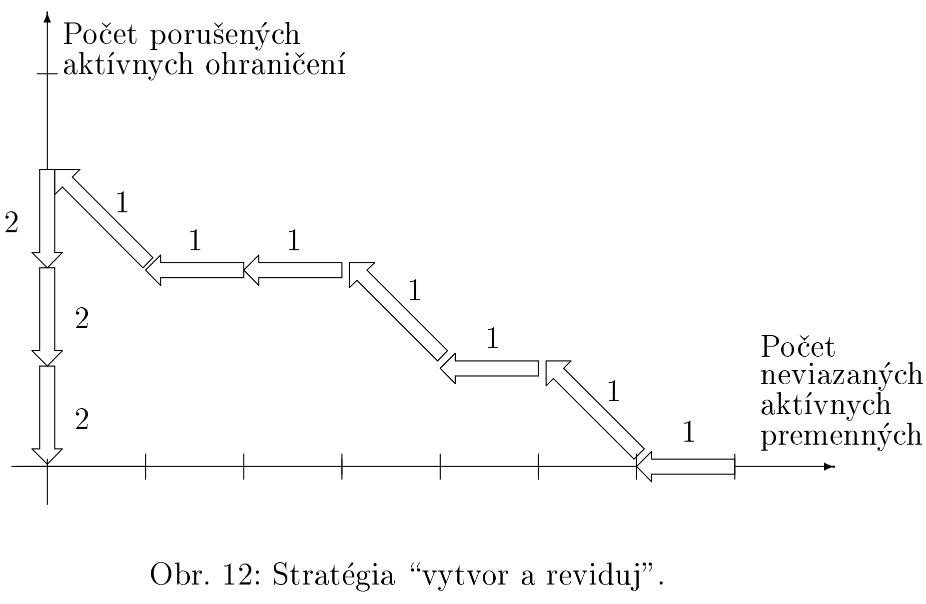
Príkladom prehľadávania je algoritmus *generuj a testuj*, ktorý umožňuje systematickým spôsobom generovať a testovať jednotlivé body priestoru prehľadávania. Ak fázy generovania a testovania sú oddelené, tak sa jedná o bodové prehľadávanie. Ak sa tieto fázy prelínajú (časť testovania je realizovaná ešte pred ukončením generovania), tak ide o príklad hierarchického prehľadávania.

Pre hierarchické prehľadávanie je typickým krokom uvoľňovací krok (návrat pri nájdení nekonzistencie v nejakom podpriestore) zatiaľ čo bodové prehľadávanie je charakterizované korekčným krokom (umožňujúcim zmenu hodnoty jednej alebo viacerých premenných).

To, ktoré riešiace kroky sa použijú, môže závisieť nielen od použitej riešiacej stratégie, ale aj od typu riešeného problému. Pri statickom probléme s ohraničeniami môžu nájsť svoje uplatnenie priraďovací, korekčný, uvoľňovací a redukčný krok. Použitie ostatných krokov nemá v tomto prípade uplatnenie. Naopak pre dynamické problémy sú typickými krokmi aktivačný a deaktivačný krok. Relaxačný krok nájde zase svoje použitie pri riešení preurčených problémov.



Obr. 11: Stratégia "rozširuj a reviduj".



Obr. 12: Stratégia "vytvor a reviduj".

state existujú dva základné tvary trajektórie, ktoré zodpovedajú dvom stratégiám prehľadávania priestoru P:

- *rozširuj a reviduj* (extend and revise)

  Aktívnym premenným sú priraďované hodnoty z ich domén až pokiaľ sa nevyskytne prvé porušenie nejakého aktívneho ohraničenia. Najprv sa koriguje riešenie tak, aby sa toto porušenie ohraničenia odstránilo, a až po jeho odstránení sa pokračuje v ďalšom priraďovaní hodnôt premenným.

- *vytvor a reviduj* (complete and revise)

  Najprv sa priradia hodnoty všetkým premenným bez ohľadu na ohraničenia. Až po naviazaní všetkých aktívnych premenných nasleduje modifikácia riešenia vedúca ku korekcii porušení ohraničení.

Príklady možných trajektórií vytvorených týmito dvoma základnými stratégiami sú znázornené na obr. 11 a obr. 12. Číselné označenie jednotlivých riešiacich krokov zodpovedá vyššie uvedenému deleniu týchto krokov do uvedených tried.

Z pohľadu spôsobu prehľadávania priestoru prehľadávania P (priestoru potenciálnych riešení) uvedené dve stratégie zodpovedajú dvom rôznym spôsobom prehľadávania:

- hierarchickému prehľadávaniu
- bodovému prehľadávaniu.

Postupným sekvenčným priraďovaním hodnôt jednotlivým premenným s ich okamžitým testovaním je vlastne priestor prehľadávania P (definovaný doménami jednotlivých premenných) prehľadávaný hierarchickým spôsobom – priradením hodnoty h nejakej premennej V_i z jej domény D_i sa zúži oblasť momentálneho záujmu z celého priestoru prehľadávania na menší podpriestor, charakterizovaný hodnotou h pre i-tu dimenziu. Ak v tomto podpriestore nebola objavená žiadna nekonzistencia tak sa tento podpriestor ďalej zužuje na ešte menší podpriestor (priradením hodnoty ďalšej premennej). Tento proces cyklicky pokračuje až kým nie je zistená nekonzistencia skúmaného podpriestoru alebo až kým celý priestor prehľadávania nebol redukovaný na jeden bod reprezentujúci hľadané riešenie. Ak je v niektorom podpriestore zistená nekonzistencia, tak tento podpriestor sa celý vylúči z ďalšieho prehľadávania (jeho body sa už detailnejšie skúmať nebudú) a prechádza sa na iný podpriestor.

Pri bodovom prehľadávaní sa konzistencia (platnosť ohraničení) skúma pre jednotlivé body priestoru prehľadávania. Ak skúmaný bod je konzistentný, tak reprezentuje hľadané riešenie. Ak nie je konzistentný, prechádza sa na skúmanie iného bodu, ktorý môže byť z okolia predchádzajúceho bodu alebo aj z celkom inej (vzdialenej) časti priestoru prehľadávania.

5. relaxácia

   Jedno alebo viacero ohraničení sa stáva neaktívnymi (môže to byť sprevádzané aktiváciou nejakých iných neaktívnych ohraničení). Tento krok je možné reprezentovať šípkou paralelnou s osou o_y (ak počet porušení ohraničení klesol, tak šípka je orientovaná nadol) alebo bodom (ak počet porušených ohraničení sa nezmenil).

6. uvoľnenie

   Jedna alebo niekoľko aktívnych premenných (s priradenou hodnotou) sa stáva neviazanými. Toto môže mať za následok pokles celkového počtu porušení ohraničení, ak nejaké aktívne ohraničenie porušené aktuálnym riešením je definované nad premennou ovplyvnenou týmto krokom. V stavovom priestore tomuto kroku zodpovedá šípka vpravo (môže byť rovnobežná s osou o_x ak počet porušení neklesol).

7. redukcia

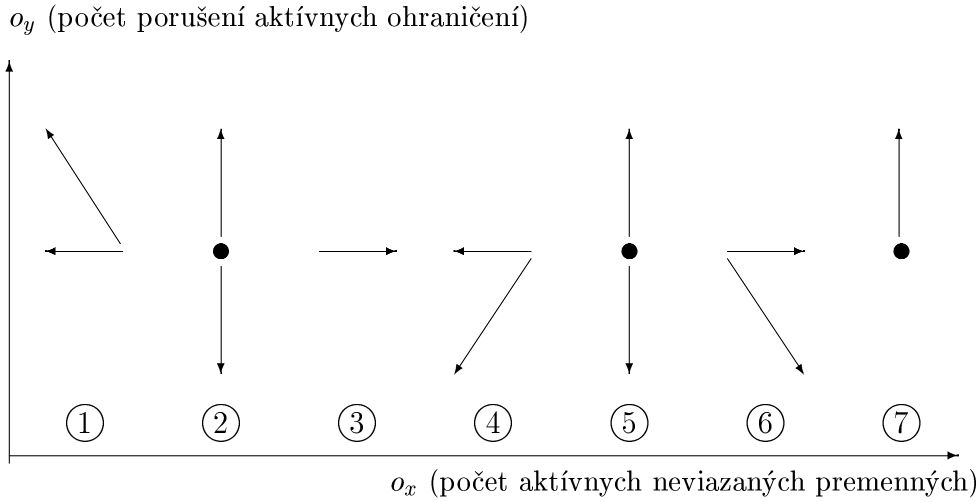
   Jedna alebo niekoľko hodnôt je vypustených z domén aktívnych premenných alebo z aktívnych ohraničení. Ak sa jedná o neviazané premenné resp. o ohraničenia definované nad dosiaľ neviazanými premennými, potom tento krok nemá žiadny vplyv na počet porušení ohraničení ani na počet neviazaných premenných a vtedy je v stavovom priestore znázorňovaný ako bod. Inak môže znamenať zvýšenie počtu porušených ohraničení.

Tieto riešiace kroky slúžia na transformáciu aktuálneho riešenia na riešenie nové. Sekvencia týchto krokov potom umožňuje počiatočné riešenie transformovať na riešenie konečné.

Z obrázku je zrejmé, že niektoré kroky (alebo niektoré ich varianty) majú v stavovom priestore rovnaké znázornenie. Teda z tvaru trajektórie nie je možné vždy presne určiť, z ktorých riešiacich krokov je zložená. Navyše, medzi jednotlivými riešiacimi krokmi existuje aj vzájomná zameniteľnosť. Napr. prípad keď nejaká premenná je uvoľnená (a teda sa stáva neviazanou) s následným priradením inej hodnoty z jej domény môže byť chápaný ako korekcia jej hodnoty.

Základným riešiacim krokom (prítomným vždy) je priradenie. Zvyčajne však nestačí pre nájdenie vyhovujúceho riešenia a preto je tento krok doprevádzaný korekčným krokom alebo uvoľnením premenných. V niektorých typoch problémov s ohraničeniami však ani tieto tri kroky nepostačujú pre nájdenie riešenia a musia byť využité aj ostatné kroky. Napr. pri rôznych transformáciách problému môže dôjsť k zmenám premenných a teda k uplatneniu aktivačného a/alebo deaktivačného kroku. Vo všeobecnosti, čím viac typov riešiacich krokov nejaká riešiaca metóda vie využiť, tým širší je rozsah aplikovateľnosti tejto metódy.

Existuje viacero stratégií pre riešenie problémov s ohraničeniami. Každá stratégia vytvára v stavovom priestore trajektóriu charakteristického tvaru. V pod-



Obr. 10: Základné typy riešiacich krokov.

2. korekcia

   Hodnoty jednej alebo viacerých aktívnych premenných sú modifikované s cieľom obnovenia platnosti nejakého ohraničenia. V S to možno reprezentovať šípkou paralelnou s osou o_y (ak počet porušení ohraničení klesol, tak šípka je orientovaná nadol) alebo bodom (ak počet porušených ohraničení sa nezmenil).

3. aktivácia

   Jedna alebo viac premenných je aktivovaných a zahrnutých do aktuálneho riešenia (teda prechádzajú z množiny V^n do V^a). Môže to byť sprevádzané rozšírením množiny ohraničení C^a o nové ohraničenia, viažúce sa na novouvažované premenné. Pretože novo aktivované premenné sú neviazané, tento krok je v stavovom priestore reprezentovaný šípkou vpravo rovnobežnou s osou o_x.

4. deaktivácia

   Jedna alebo viac premenných je vyradených z aktuálneho riešenia (teda prechádzajú z množiny V^a do V^n). Ohraničenia, ktoré boli definované nad týmito deaktivovanými premennými, sa stávajú neaktívnymi (a teda prechádzajú z množiny C^a do C^n). V stavovom priestore to je reprezentované ako šípka vľavo (paralelná s osou o_x ak žiadne porušené ohraničenie nebolo deaktivované).

3 Riešiace kroky a riešiace stratégie

Riešiť problém s ohraničeniami znamená hľadať vhodné priradenie hodnôt všetkým aktívnym premenným. Toto hľadanie je procesom ktorého stav v nejakom okamihu môže byť charakterizovaný počtom aktívnych *neviazaných* premenných (to sú premenné ktoré ešte nemajú priradenú hodnotu zo svojej domény – niekedy sú označované ako voľné premenné) a počtom porušení aktívnych ohraničení definovaných nad premennými. Pri určovaní aktuálneho stavu riešenia môže dôjsť k situácii keď nejaké aktívne ohraničenie je definované nad neviazanou aktívnou premennou a teda nie je možné overiť splnenie alebo porušenie tohto ohraničenia. Takéto ohraničenie v tomto prípade nie je považované za porušené.

Tento proces hľadania riešenia môže byť reprezentovaný v dvojrozmernom *stavovom priestore* S (state space) [66], kde

os o_x reprezentuje počet aktívnych doposiaľ neviazaných premenných

os o_y reprezentuje počet porušení aktívnych ohraničení.

Z podstaty problémov s ohraničeniami vyplýva, že využívané sú iba nezáporné časti oboch súradných osí.

Proces hľadania riešenia môže byť znázornený v tomto stavovom priestore ako trajektória spájajúca počiatočný bod hľadania riešenia s výsledným stavom. Na začiatku procesu hľadania riešenia žiadna aktívna premenná nemá priradenú hodnotu. Dôsledkom toho je, že žiadne aktívne ohraničenie nie je považované za porušené. Teda počiatočný bod trajektórie (reprezentujúci počiatočné riešenie ktoré je "prázdne") sa nachádza na osi o_x a jeho vzdialenosť od počiatku je rovná počtu aktívnych premenných definovaných na začiatku riešenia úlohy. Pretože konečné riešenie by malo byť kompletné (všetky aktívne premenné sú viazané – majú priradenú nejakú hodnotu zo svojej domény) trajektória končí na osi o_y. Ak konečné riešenie spĺňa všetky aktívne ohraničenia, potom koncový bod trajektórie je umiestnený v počiatku systému súradníc.

Trajektória v stavovom priestore, reprezentujúca hľadanie riešenia, pozostáva z rôznych krokov. Tieto kroky je možné rozdeliť do niekoľkých základných typov (ich možné tvary sú zobrazené na obr. 10):

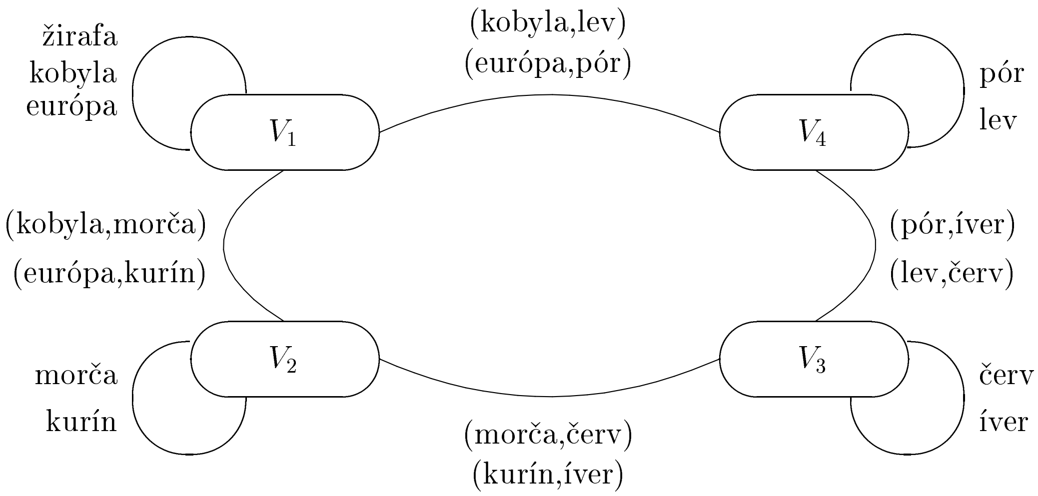
1. priradenie

   Jednej neviazanej aktívnej premennej sa priradí hodnota z jej domény. Toto naviazanie môže spôsobiť aj porušenie jedného alebo viacerých aktívnych ohraničení. Tento krok je v stavovom priestore S reprezentovaný šípkou smerujúcou vľavo (je paralelná s osou o_x ak žiadne ohraničenie nebolo porušené týmto krokom).

syntakticky (napr. slová sú tým podobnejšie, čím viac písmen majú spoločných). K dispozícii je nasledovný výber slov:

6 písmen	:	žirafa, kobyla, európa
5 písmen	:	morča, kurín
4 písmená	:	červ, íver
3 písmená	:	pór, lev

Pre vyjadrenie tohto problému je potrebné opäť využiť štyri premenné a osem (štyri unárne a štyri binárne) ohraničení. Spolu vytvárajú sieť ohraničení podľa obr. 9 kde jednotlivé ohraničenia sú zadané explicitným vymenovaním povolených kombinácií hodnôt. Splnenie všetkých ohraničení je považované za nutnú podmienku nájdenia riešenia.



Obr. 9: Sieť ohraničení pre optimalizačný ilustračný problém.

Pre tento problém existujú dve prijateľné riešenia ktoré spĺňajú všetky ohraničenia. Ktoré z nich je preferovanejšie záleží na optimalizačnej funkcii. Ak sa pod podobnosťou slov bude rozumieť sémantická podoba, tak ako riešenie bude preferovaná kombinácia (“kobyla”, “morča”, “červ”, “lev”). Ak by sa podobnosť chápala v syntaktickom zmysle, tak by bola preferovaná kombinácia (“európa”, “kurín”, “íver”, “pór”).

aktívne všetky ohraničenia. Ak sa nepodarí nájsť také riešenie, ktoré by spĺňalo tieto všetky ohraničenia, tak binárne ohraničenia môžu byť uvoľňované. Snahou bude uvoľniť čo najmenší počet ohraničení, pričom všetky binárne ohraničenia sú si rovnocenné.

2.4 Optimalizačné problémy

Pri optimalizačných problémoch je s jednotlivými potenciálnymi riešeniami asociovaná okrem hodnotiacej funkcie aj optimalizačná funkcia f_o. Jedná sa vlastne o funkciu, ktorá má byť optimalizovaná – hľadá sa poloha jej extrému[9].

Aby potenciálne riešenie bolo považované za skutočné riešenie problému, musí nielen spĺňať všetky ohraničenia v uspokojujúcej miere (čo sú zvyčajne všetky ohraničenia alebo iba nejaká ich časť)[10], ale zároveň nadobúda extrémnu hodnotu optimalizačnej funkcie f_o spomedzi tých riešení, ktoré vyhovujúcim spôsobom spĺňajú ohraničenia.

V prípade existencie mäkkých ohraničení sa preferuje síce ich splnenie avšak ich prípadné nesplnenie (zvyčajne v tvare nie príliš veľkého porušenia[11]) nemusí mať za následok odmietnutie potenciálneho riešenia. Tak sa môže stať že aj v prípade existencie riešenia spĺňajúceho všetky ohraničenia je vybrané iné riešenie ktoré síce nespĺňa všetky ohraničenia ale jeho hodnota optimalizačnej funkcie je dostatočne[12] väčšia ako hodnota prvého riešenia (čím je nesplnenie mäkkých ohraničení väčšie tým musí byť aj väčší rozdiel v hodnotách optimalizačnej funkcie).

Optimalizačná funkcia môže byť definovaná pre všetky body priestoru prehľadávania, alebo iba pre niektoré (v tomto prípade zvyčajne iba pre tie, ktoré vhodným spôsobom spĺňajú ohraničenia a teda ktorých hodnota hodnotiacej funkcie f_h presiahla nejaký prah θ).

Ilustráciou takéhoto typu problémov s ohraničeniami je doplňovačka tvaru podľa obr. 4. Do tejto doplňovačky je teraz potrebné doplniť štyri slová tak, aby všetky riadky a stĺpce boli úplne zaplnené, a pritom je potrebné vyberať slová tak aby boli čo najpodobnejšie. Ich podobnosť pritom môže byť chápaná sémanticky (keď slová sú tým podobnejšie, čím podobnejšie objekty označujú) alebo

[9]Za špeciálny príklad optimalizačného problému s ohraničeniami je možné považovať aj preurčený problém – tvrdé ohraničenia je možné vyjadriť ako definované ohraničenia a zahrnúť ich do určovania hodnoty hodnotiacej funkcie zatiaľ čo mäkké ohraničenia budú zohľadnené pomocou optimalizačnej funkcie.

[10]V závislosti od toho, či je optimalizačný problém kombinovaný s preurčeným problémom alebo nie.

[11]Miera porušenia nejakého ohraničenia je zvyčajne daná konkrétnym riešeným problémom. Pri reálnych doménach a ohraničení v tvare rovnosti môže byť napr. daná ako rozdiel hodnôt pravej a ľavej strany príslušnej rovnosti.

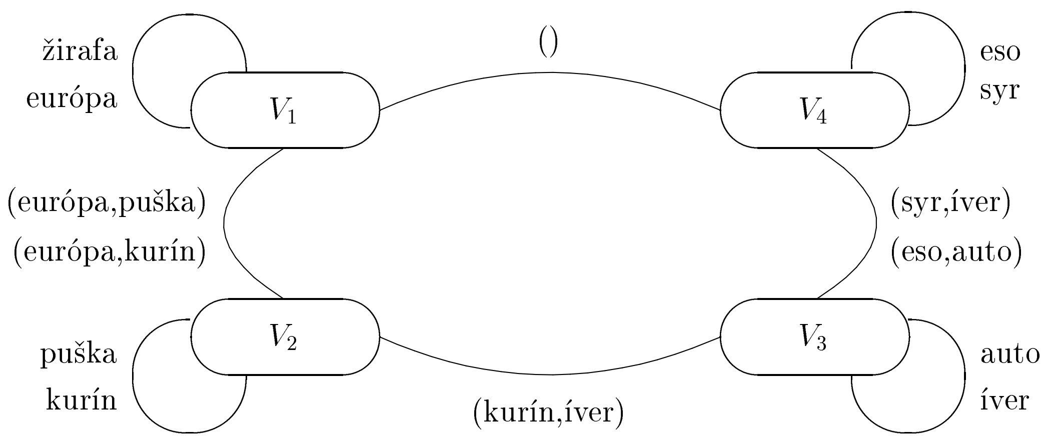
[12]Podľa nejakého kritéria.

bode rozpojiť, takže nejaký riadok a stĺpec sa nebudú viac prekrývať a nebude potrebné zabezpečovať príslušné zdieľanie písmen. V extrémnom prípade tak je možné pracovať so štyrmi nezávislými časťami. K dispozícii je teraz nasledovný pozmenený výber ôsmich slov:

6 písmen	:	žirafa, európa
5 písmen	:	puška, kurín
4 písmená	:	auto, íver
3 písmená	:	eso, syr

Pri transformácii úlohy na problém s ohraničeniami je potrebné použiť štyri premenné s rovnakými osemslovnými enumeračnými doménami. Pretože riadky a stĺpce majú rozdielnu dĺžku tak je pre každú premennú možné zaviesť unárne ohraničenie iba na slová určitého počtu písmen. Keďže je potrebné zaplniť každé políčko a zároveň je možné zapisovať písmená iba do políčok doplňovačky bez skracovania slov, tak tieto unárne ohraničenia sú tvrdými a musia byť nutne splnené. Inak povedané, domény premenných možno redukovať s ohľadom na tieto ohraničenia.

Zdieľanie písmen slovami je opäť vyjadrené pomocou binárnych ohraničení. Keďže teraz toto zdieľanie nie je nutnou podmienkou (aj keď je podmienkou žiadúcou) tak tieto ohraničenia sú mäkké. Spolu pre daný problém je teda možné definovať štyri unárne a štyri binárne ohraničenia. Tieto ohraničenia vytvárajú sieť ohraničení podľa obr. 8. Jednotlivé ohraničenia sú opäť zadané explicitným vymenovaním povolených kombinácií hodnôt.



Obr. 8: Sieť ohraničení pre preurčený ilustračný problém.

Keďže neboli zadané iné údaje, snahou bude nájsť také riešenie, ktoré dosiahne čo najväčší počet správne zdieľaných písmen. Preto na začiatku riešenia budú

kde funkcia $satisfy$ vracia nulu ak kontrolované ohraničenie je porušené kombináciou hodnôt $\overline{x}$ a jednotku ak dané ohraničenie je splnené. V zložitejších prípadoch môže táto funkcia zohľadňovať aj dôležitosť ohraničení

$$f_h(\overline{x}) = \sum_{c \in C^a} w_c \; satisfy(c, \overline{x}) \tag{11}$$

kde hodnota w_c reprezentuje váhu udávajúcu túto dôležitosť[7]. Pre vyjadrenie rôznej dôležitosti môžu byť ohraničenia usporiadané v hierarchii. V tomto prípade je potrebné zabezpečiť mapovanie významu jednotlivých úrovní hierarchie na hodnoty w_c.

Všetky ohraničenia je možné rozdeliť do dvoch skupín na *mäkké* (soft) a *tvrdé* (hard). Pod tvrdým ohraničením sa chápe také ohraničenie, bez splnenia ktorého je potenciálne riešenie neprijateľné – jeho splnenie teda tvorí nutnú podmienku pre prijateľnosť riešenia. Naproti tomu mäkké preferenčné ohraničenie nemusí byť splnené. Potenciálne riešenie môže byť prijateľné aj keď porušuje jedno alebo viac mäkkých ohraničení. Na druhej strane riešenie spĺňajúce nejaké mäkké ohraničenie je žiadanejšie ako to ktoré dané ohraničenie nespĺňa. Tieto dva typy ohraničení je možné navzájom odlíšiť váhami vystupujúcimi v hodnotiacej funkcii. Z toho vyplýva, že ide vlastne o maximalizáciu funkcie f_h, pričom dolná hranica možnej hodnoty prahu θ, ktorý musí byť presiahnutý hodnotou hodnotiacej funkcie, je odvodená od všetkých tvrdých ohraničení.

Riešenie preurčeného problému teda znamená, že potenciálne riešenie môže byť prijateľné aj vtedy, ak porušuje nejaké ohraničenia. Jednoducho tieto ohraničenia budú uvoľnené – presunú sa z množiny aktívnych ohraničení medzi neaktívne. Pritom toto uvoľnenie môže mať za následok aktiváciu alternatívneho (menej prísneho) ohraničenia.

Podobne ako pri statickom probléme[8] existujú iba aktívne premenné a s jednotlivými potenciálnymi riešeniami nie je asociovaná žiadna optimalizačná funkcia. Počas riešenia však bude existovať množina neaktívnych ohraničení C^n (ktorá môže byť na začiatku riešenia prázdna).

Ilustráciou takéhoto typu problémov s ohraničeniami je doplňovačka na obr. 4. Do tejto doplňovačky je potrebné opäť doplniť štyri slová tak, aby všetky riadky a stĺpce boli úplne zaplnené. V prípade potreby, ak to je nevyhnutné kvôli nemožnosti zdieľania písmen viacerými slovami, je možné doplňovačku v príslušnom

[7]Je možné takto zohľadniť aj stupeň porušenia ohraničení (ak niečo také má pre dané ohraničenia zmysel). V tomto prípade hodnota w_c môže byť stupňom porušenia definovaným napr. ako nejaká nezáporná hodnota. Potom nejaká hodnota znamená splnenie daného ohraničenia a čím je toto ohraničenie splnené menej (teda čím je viac porušené) tým je táto hodnota menšia. V extrémnom prípade porušenia je táto hodnota nulová. Je možné súčasne zohľadniť stupeň porušenia ohraničení aj ich dôležitosť.

[8]Je možná aj kombinácia preurčeného problému s problémom dynamickým a/alebo s optimalizačným problémom

aj premenná V_5 (ak premenná V_1 obsahuje štvorpísmenové slovo), premenná V_6 (prípad trojpísmenového slova v premennej V_1) alebo žiadna z nich (šesťpísmenové slovo v premennej V_1). V žiadnom prípade teda nemôžu byť aktívne súčasne premenné V_5 a V_6.

Zdieľanie písmen jednotlivými slovami, vloženými do doplňovačky, je opäť vyjadriteľné binárnymi ohraničeniami. Toto zdieľanie môže byť:

- permanentné – zdieľanie nezávisí od aktuálnej etapy hľadania riešenia (od toho ktorá premenná je práve neaktívna)
- dočasné – existuje iba pre určitý tvar množiny aktívnych premenných, pre nejaké iné rozdelenie premenných na aktívne a neaktívne toto zdieľanie nemusí existovať.

Kvôli dočasnému typu zdieľania môže byť zdieľanie písmen reprezentované viacerými ohraničeniami, pričom aktívne je iba jedno resp. niekoľko z nich. Napríklad piate písmeno v prvom riadku musí byť rovnaké ako prvé písmeno slova umiestneného v stĺpci č. 4 – avšak piate písmeno v prvom riadku môže byť súčasťou slova v premennej V_1, V_5 alebo V_6.

Spolu pre daný problém je teda možné definovať sedem unárnych a desať binárnych ohraničení. Tieto ohraničenia vytvárajú sieť ohraničení podľa obr. 7. Jednotlivé ohraničenia sú opäť zadané explicitným vymenovaním povolených kombinácií hodnôt.

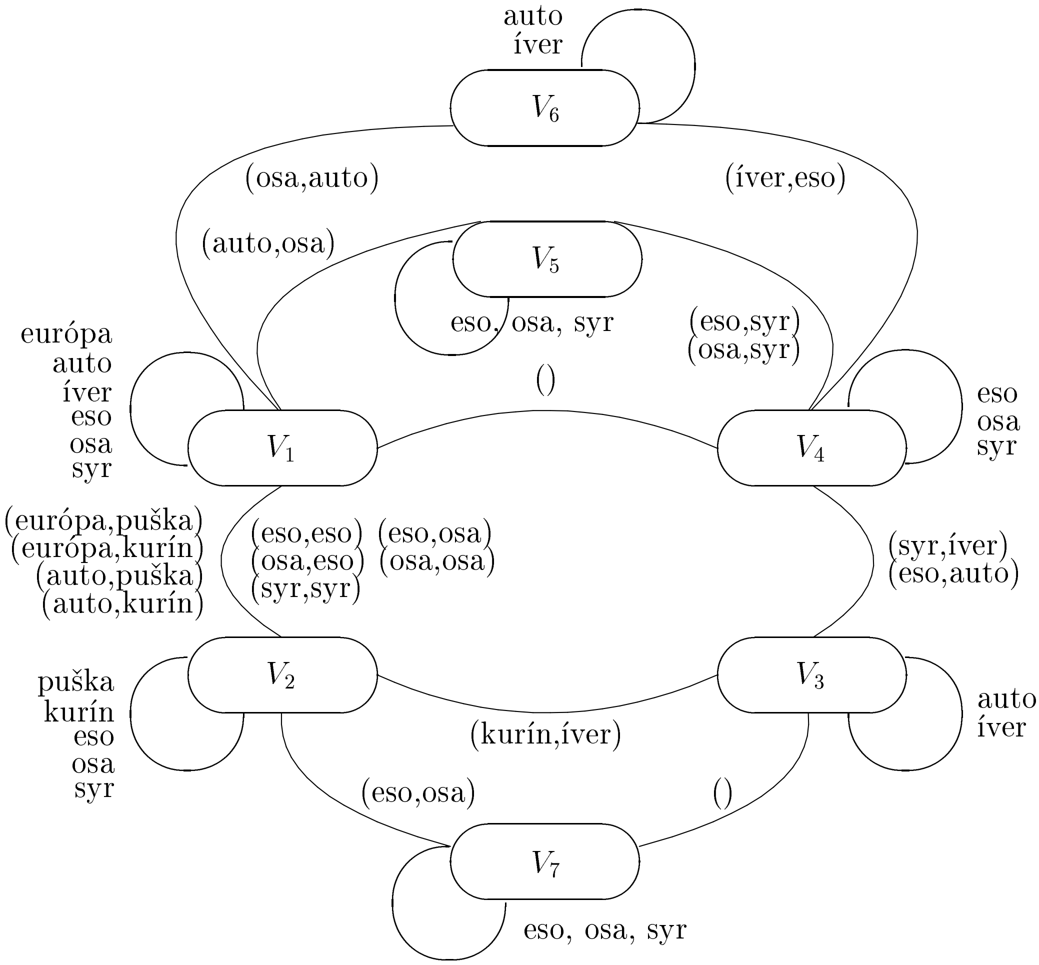
Po zadefinovaní premenných, ich domén a ohraničení definovaných nad nimi je ešte potrebné rozhodnúť, ktoré premenné budú aktívne na začiatku riešenia úlohy. Jednou z možností je voľba minimálnej množiny premenných $V^a = \{V_1, V_2, V_3, V_4\}$. Od toho závisí, ktoré ohraničenia na začiatku riešenia budú považované za aktívne a ktoré za neaktívne. V danom prípade množina neaktívnych ohraničení bude $C^n = \{C^{1,5}, C^{1,6}, C^{2,7}, C^{3,7}, C^{4,5}, C^{4,6}\}$.

2.3 Preurčené problémy

Charakteristickým znakom tejto triedy problémov s ohraničeniami je to, že žiadny bod priestoru prehľadávania nereprezentuje takú kombináciu hodnôt premenných, ktorá by spĺňala všetky aktívne ohraničenia.

Aby potenciálne riešenie bolo považované za prijateľné, musí spĺňať ohraničenia v "uspokojujúcej miere". S jednotlivými potenciálnymi riešeniami je asociovaná hodnotiaca funkcia f_h, ktorá udáva mieru splnenia ohraničení daným riešením. V najjednoduchšom prípade je postavená na počte splnených a porušených aktívnych ohraničení:

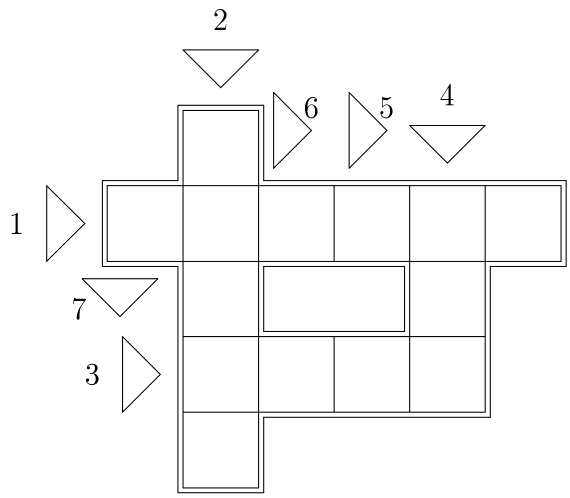
$$f_h(\overline{x}) = \sum_{c \in C^a} satisfy(c, \overline{x}) \tag{10}$$



Obr. 7: Sieť ohraničení pre dynamický ilustračný problém.

spĺňať všetky aktívne ohraničenia. S jednotlivými potenciálnymi riešeniami nie je asociovaná žiadna optimalizačná funkcia.

Ilustráciou takéhoto typu problémov s ohraničeniami je doplňovačka znázornená na obr. 6.



Obr. 6: Ilustračný dynamický problém.

Opäť je potrebné zaplniť riadky a stĺpce doplňovačky vhodnými slovami. Na rozdiel od predchádzajúcej kapitoly je teraz možné do riadku alebo stĺpca vložiť nie iba jedno slovo, ale možno ho vyplniť ľubovoľným počtom slov, pričom posledné písmeno jedného slova musí byť zároveň aj prvým písmenom slova ďalšieho. K dispozícii je teraz zmenený výber slov:

6 písmen	:	európa
5 písmen	:	puška, kurín
4 písmená	:	auto, íver
3 písmená	:	eso, osa, syr

Transformácia tejto úlohy na problém s ohraničeniami je teraz zložitejšia. Keďže k dispozícii sú 3 až 6 písmenové slová, tak počet slov, ktoré je možné umiestniť do doplňovačky (nezávisle na konkrétnych slovách z predloženého zoznamu) kolíše od štyroch do šiestich. Navyše pri umiestňovaní dvoch slov do prvého riadku sú k dispozícii dve možnosti – štvorpísmenové slovo nasledované trojpísmenovým alebo naopak. Preto pre daný problém je teraz potrebné použiť až sedem premenných. Domény týchto premenných sú rovnaké – uvedených osem slov.

Keďže riadok č. 1 je potrebné zaplniť, premenná V_1 bude musieť byť aktívna vždy. V závislosti na tom, aké slovo je priradené tejto premennej, bude aktívna

2.2 Dynamické problémy

Dynamičnosť týchto problémov môže byť spôsobená zmenami množiny premenných alebo ohraničení (resp. môže dochádzať k obom typom zmien súčasne). Oba varianty sú podobné ohľadom práce s ohraničeniami, líšia sa však prístupom k premenným.

Pri dynamických zmenách premenných množina aktívnych premenných nie je nemenná, ale môže sa počas hľadania riešenia meniť. Tieto zmeny závisia napr. na hodnotách priradených ostatným aktívnym premenným. Priradenie nejakej hodnoty i-tej premennej môže mať za následok, že nemá zmysel priradzovať hodnotu j-tej premennej (a teda táto prestáva byť aktívnou) alebo naopak je potrebné priradiť hodnotu aj k-tej premennej (a teda táto premenná sa musí stať aktívnou).

Tento dynamický variant je založený na charakteristikách riešeného problému a zahŕňa aj prípad, ak z dôvodu jednoduchosti sú najprv uvažované iba hlavné otázky, a až po ich vyriešení sa riešenie zameria aj na detaily.

Z tohto dôvodu sú teda uvažované aj neaktívne premenné a teda platí:

$$V^a \cup V^n = V \tag{8}$$

pričom množina V^n je neprázdnou aspoň v jednej fáze hľadania riešenia. Samozrejme, množina V^a vždy obsahuje aspoň jednu premennú.

Pretože ohraničenie je dôležité iba vtedy, ak je definované nad aktívnymi premennými, tak spolu so zmenami množiny aktívnych premenných sa mení aj množina aktívnych ohraničení. Ak je aktivovaná nejaká premenná, tak súčasne musia byť aktivované aj všetky tie ohraničenia, ktoré súvisia s touto premennou a okrem nej sú definované iba nad momentálne aktívnymi premennými.

Inak povedané pre ohraničenia platí podobný vzťah ako pre premenné

$$C^a \cup C^n = C \tag{9}$$

a teda je potrebné uvažovať aj množinu neaktívnych ohraničení.

Na rozdiel od predchodzieho, variant založený iba na dynamických zmenách ohraničení predpokladá zmeny iba pre jednotlivé ohraničenia. V tomto prípade nedochádza k zmene množiny premenných. Zmeny ohraničení bývajú zapríčinené "vonkajším" svetom a môžu mať charakter sprísnenia ohraničení (vrátane dodefinovania nového ohraničenia) alebo uvoľnenia ohraničení (vrátanie zrušenia existujúceho ohraničenia).

Pri tomto variante sa pojem "aktívnosti" uvažuje iba pre ohraničenia zatiaľ čo množina premenných sa počas celého riešenia problému nemení – množina V^n ostáva prázdnou.

Podobne ako pri statickom probléme tak aj pri jeho rozšírení o dynamické zmeny[6] platí, že aby potenciálne riešenie bolo považované za prijateľné, musí

[6]Je samozrejme možná kombinácia dynamického problému s problémom preurčeným a/alebo s optimalizačným problémom (viď nasledujúce kapitoly).

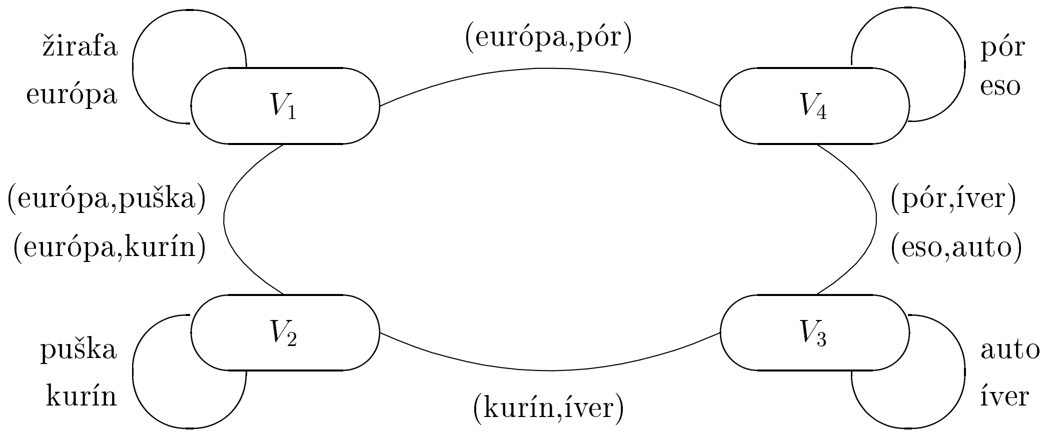
teda tvar $C^1 = \{(\text{“žirafa”}), (\text{“európa”})\}$. Podobným spôsobom je možné definovať unárne ohraničenia pre všetky ostatné premenné.

Jednotlivé slová, umiestnené do doplňovačky, zdieľajú niektoré písmená. Tieto závislosti sa dajú vyjadriť ako viacárne ohraničenia. Napríklad fakt, že druhé písmeno slova v prvom riadku musí byť rovnaké ako druhé písmeno slova umiestneného v stĺpci č. 2 je možné reprezentovať binárnym ohraničením $C^{1,2}$ nad premennými V_1 a V_2. Toto ohraničenie bude povoľovať značný počet rôznych kombinácií a bude mať tvar $\{(\text{“žirafa”},\text{“žirafa”}), \ldots, (\text{“eso”},\text{“eso”}), (\text{“európa”},\text{“puška”}), \ldots\}$.

Celkovo pre daný problém je teda potrebné definovať štyri unárne a štyri binárne ohraničenia[5]. Priestor, ktorý je potrebné prehľadať, je konečný a pozostáva z $8^4 = 4096$ štvoríc slov.

Použitím unárnych ohraničení je možné všetky domény redukovať na dvojprvkové množiny, čím sa získa priestor prehľadávania pozostávajúci zo šestnástich bodov. S ohľadom na to je možné sprísniť aj binárne ohraničenia, napr. keďže teraz v doméne D_1 sa nebude vyskytovať slovo “kurín” tak $C^{1,2}$ nebude povoľovať napr. dvojicu (“kurín”,“auto”). Podobne aj keď ohraničenie $C^{1,2}$ v pôvodnej neredukovanej podobe pripúšťalo aj kombináciu (“žirafa”,“žirafa”), táto v prípade redukovaných domén nie je prípustná.

Uvedené ohraničenia po redukcii vytvárajú sieť ohraničení podľa obr. 5 (jednotlivé ohraničenia sú zadané explicitným vymenovaním povolených hodnôt resp. kombinácií hodnôt).

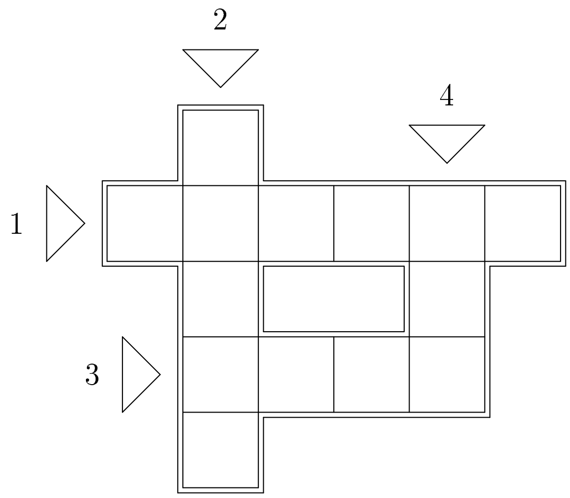


Obr. 5: Sieť ohraničení pre redukovaný statický ilustračný problém.

[5]Ostatné ohraničenia sú reprezentované implicitne – môžu byť indukované z explicitne vyjadrených ohraničení. Takým je napr. ohraničenie $C^{1,3}$ – z explicitne vyjadrených binárnych ohraničení vyplýva, že toto ohraničenie nepovoľuje napr. kombináciu hodnôt (“žirafa”, “íver”) pre premenné V_1 a V_3.

pomerom počtu kombinácií hodnôt povolených daným ohraničením voči všetkým možným kombináciám hodnôt).

Ilustráciou uvedeného typu problémov s ohraničeniami je doplňovačka[4] ktorej tvar je znázornený na obr. 4.



Obr. 4: Ilustračný statický problém.

Do tejto doplňovačky je potrebné doplniť štyri slová tak, aby všetky riadky a stĺpce boli úplne zaplnené. K dispozícii je nasledovný výber slov:

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

Táto úloha môže byť transformovaná na problém s ohraničeniami pomerne priamočiaro. Z formálneho hľadiska jednotlivé riadky a stĺpce môžu byť reprezentované premennými (pre daný problém je teda potrebné použiť štyri premenné – ich indexy zodpovedajú označeniu riadkov a stĺpcov podľa obr. 4). Domény týchto premenných sú rovnaké – uvedených osem slov. Jedná sa teda o problém s konečnými doménami.

Keďže riadok č. 1 má dĺžku 6 znakov, je možné zmysluplne do neho umiestniť iba niektoré zo šesťpísmenových slov. Toto je možné vyjadriť ako unárne ohraničenie definované nad premennou V_1, zodpovedajúcou prvému riadku. Toto ohraničenie bude povoľovať pre premennú V_1 iba dve rôzne hodnoty a bude mať

[4]Táto doplňovačka bude využívaná ako ilustračný príklad v prílohe A.

Podľa typu domén premenných, spôsobu delenia premenných a ohraničení na aktívne a neaktívne, podľa použitej hodnotiacej funkcie a požadovaného prahu θ ako aj podľa optimalizačnej funkcie je možné problémy s ohraničeniami deliť na niekoľko typov. Niektorými z nich (zameranými na nájdenie jedného riešenia) sa zaoberá táto práca a budú popísané v ďalšom podrobnejšie.

2.1 Statické problémy

Pri statických problémoch množiny aktívnych premenných a ohraničení sa nemenia počas hľadania riešenia – ostávajú rovnaké. Z tohto dôvodu zvyčajne nie sú uvažované žiadne neaktívne premenné alebo ohraničenia a teda platí:

$$V^a = V \tag{5}$$
$$C^a = C \tag{6}$$

Aby potenciálne riešenie bolo považované za prijateľné, musí spĺňať všetky aktívne ohraničenia. Množina M vystupujúca v definícii hodnotiacej funkcie podľa (3) má tvar $M = \{0, 1, \ldots, m\}$, kde m je počet všetkých aktívnych ohraničení. Ak hodnotiaca funkcia pre každý bod priestoru prehľadávania stanovuje počet splnených aktívnych ohraničení, tak pre prah θ musí platiť

$$m - 1 \leq \theta < m \tag{7}$$

Pretože potenciálne riešenia sú skúmané iba z hľadiska spĺňania resp. porušovania ohraničení, nie je s nimi asociovaná žiadna optimalizačná funkcia (resp. úlohu optimalizačnej funkcie môže hrať ľubovoľná konštantná funkcia).

Takýto typ problémov je zvyčajne v literatúre považovaný za základný a preto sa často slovo "statický" neuvádza. Podobne sa neuvádzajú pojmy "aktívny" a "neaktívny" a hovorí sa iba o premenných a ohraničeniach hoci sa jedná o aktívne premenné a aktívne ohraničenia.

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 $D_i^r \subset D_i$, 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 viacárne ohraničenie povoľuje nejakú takú kombináciu hodnôt, ktorá obsahuje hodnotu, porušujúcu unárne ohraničenie viažúce sa k príslušnej premennej (a teda táto hodnota sa nevyskytuje v redukovanej doméne danej premennej), potom je možné toto ohraničenie "sprísniť" a danú kombináciu hodnôt nepovoliť. Takýmto spôsobom dochádza aj k redukcii jednotlivých ohraničení – zvyšovaniu ich prísnosti (danej

Naproti tomu optimalizačná funkcia nie je povinnou súčasťou definície problémov s ohraničeniami. Naviac môže byť asociovaná iba s niektorými bodmi priestoru prehľadávania, nemusí byť definovaná nad celým týmto priestorom (avšak vždy musí byť definovaná pre celú tú oblasť, ktorá obsahuje body spĺňajúce všetky ohraničenia. Úlohou optimalizačnej funkcie je kvantitatívne stanoviť vhodnosť kombinácií hodnôt premenných na základe nejakých kritérií (nesúvisiacich s ohraničeniami obsiahnutými v množine C).

Nie vždy pre riešenie problému sú podstatné všetky premenné a všetky ohraničenia. Sú prípady, keď napr. riešenie je tvorené iba podmnožinou premenných alebo spĺňa iba podmnožinu ohraničení. Vo všeobecnosti je možné množinu premenných rozdeliť na dve disjunktné podmnožiny V^a a V^n – *aktívne* premenné a *neaktívne* premenné. Podobne aj ohraničenia môžu byť aktívne a neaktívne – $C^a \cup C^n = C$.

Pod *prijateľným* riešením problému s ohraničeniami je potom možné chápať každé také priradenie hodnôt $\overline{x}$ aktívnym premenným z ich domén, pre ktoré hodnota hodnotiacej funkcie presiahne stanovený prah θ:

$$f_h(\overline{x}) > \theta \tag{4}$$

Pod *optimálnym* riešením problému s ohraničeniami je potom možné chápať také priradenie hodnôt aktívnym premenným z ich domén, pre ktoré hodnota hodnotiacej funkcie presiahne stanovený prah θ a súčasne hodnota optimalizačnej funkcie je taká, že neexistuje nijaké iné priradenie hodnôt aktívnym premenným s väčšou/menšou[3] hodnotou optimalizačnej funkcie a s hodnotou hodnotiacej funkcie tiež presahujúcou daný prah.

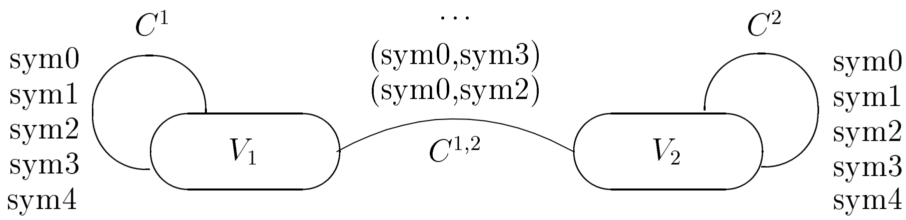
Riešiť problém s ohraničeniami potom zvyčajne znamená jednu z nasledujúcich úloh:

- dokázať existenciu alebo neexistenciu riešenia (decision problem)
- zistiť počet riešení (enumeration problem)
- nájsť jedno riešenie (search problem)
- nájsť všetky riešenia (generation problem).

Najčastejšie je v literatúre venovaná pozornosť úlohe nájdenia jedného riešenia. Je to preto, lebo jednak je to v praxi prevažujúca úloha a jednak ostatné úlohy môžu byť riešené (viac či menej efektívne) využitím metód pre hľadanie jedného riešenia (ak ho je možné nájsť, tak riešenie existuje; opakovaným použitím je možné nájsť všetky riešenia ktoré stačí spočítať ap.).

[3] Závisí to od toho, či je potrebné optimalizačnú funkciu maximalizovať alebo minimalizovať.

Unárne ohraničenie možno reprezentovať hranou oboma koncami viazanou s tým istým uzlom. Binárne ohraničenia sú znázornené ako hrany spájajúce rôzne uzly. Príkladom takéhoto zjednodušeného spôsobu znázornenia binárneho problému s ohraničeniami je sieť ohraničení pre enumeračné premenné V_1 a V_2 z príkladu podľa obr. 1 ktorá je zobrazená na obr. 3.



Obr. 3: Príklad zjednodušeného zobrazenia siete ohraničení pre binárny problém.

Domény premenných vlastne definujú *priestor prehľadávania* P (search space). Je to vlastne priestor všetkých potenciálnych 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. Priestor prehľadávania takto vznikne ako kartézsky súčin domén premenných – teda ako množina všetkých kombinácií hodnôt jednotlivých premenných. V závislosti od použitých domén tento priestor môže obsahovať konečný alebo nekonečný počet bodov.

Každé ohraničenie rozdeľuje priestor prehľadávania na dve oblasti (nemusia byť súvislé). Pre každý bod tohto priestoru je potom možné povedať, či kombinácia hodnôt reprezentovaná daným bodom priestoru spĺňa dané ohraničenie (a patrí do jednej oblasti) alebo ho porušuje (a teda patrí do oblasti druhej).

Nad priestorom prehľadávania P (alebo jeho časťou) môžu by definované rôzne funkcie. V zásade ich je možné rozdeliť do dvoch skupín:

- *hodnotiaca* funkcia f_h
- *optimalizačná* funkcia f_o.

Hodnotiaca funkcia je definovaná podľa

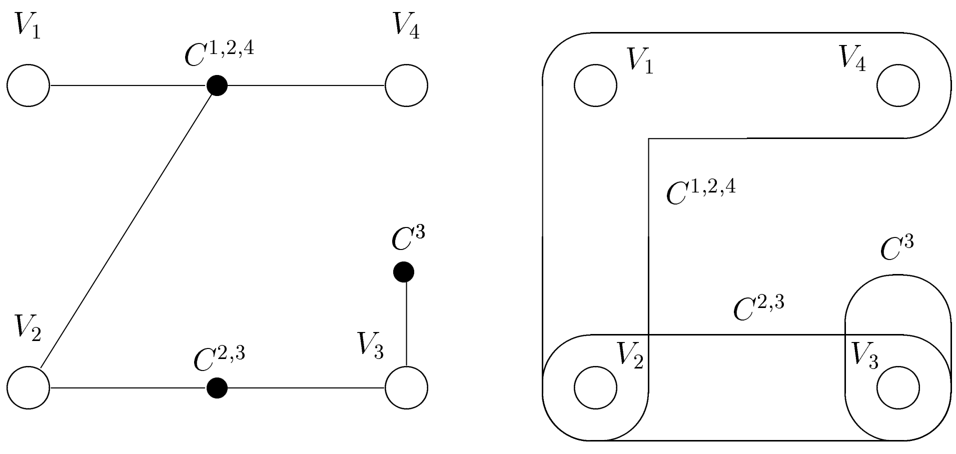
$$f_h \quad : \quad D_1 \times D_2 \times \ldots \times D_n \to M \tag{3}$$

kde M reprezentuje nejakú množinu reálnych alebo celočíselných hodnôt. Pre každý bod priestoru prehľadávania P vyjadruje mieru splnenia jednotlivých ohraničení kombináciou hodnôt, reprezentovanou daným bodom priestoru. Táto funkcia je definovaná nad celým priestorom prehľadávania a to pre každý problém s ohraničeniami.

Všetky tri príklady reprezentujú "pozitívnu" formu – reprezentujú prípustné kombinácie. Vo všeobecnosti ohraničenia môžu byť v pozitívnom alebo negatívnom tvare, reprezentujúc prípustné alebo neprípustné kombinácie hodnôt. Tie isté ohraničenia v "negatívnom" tvare by boli reprezentované ako $\{(sym0, sym0), \ldots\}$ pre enumeračné domény, $V_2 \leq 0.5V_1 + 0.75$ pre celočíselné premenné resp. $V_2 \neq 0.25V_1 + 1$ pre reálne premenné. To, ktorá forma sa použije, zvyčajne závisí od jednoduchosti vyjadrenia.

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 za cenu zavedenia nových pomocných premenných) [2] [65], väčšina prác sa zaoberá práve binárnymi problémami s ohraničeniami – problémami, v ktorých všetky ohraničenia sú definované najviac nad dvoma premennými.

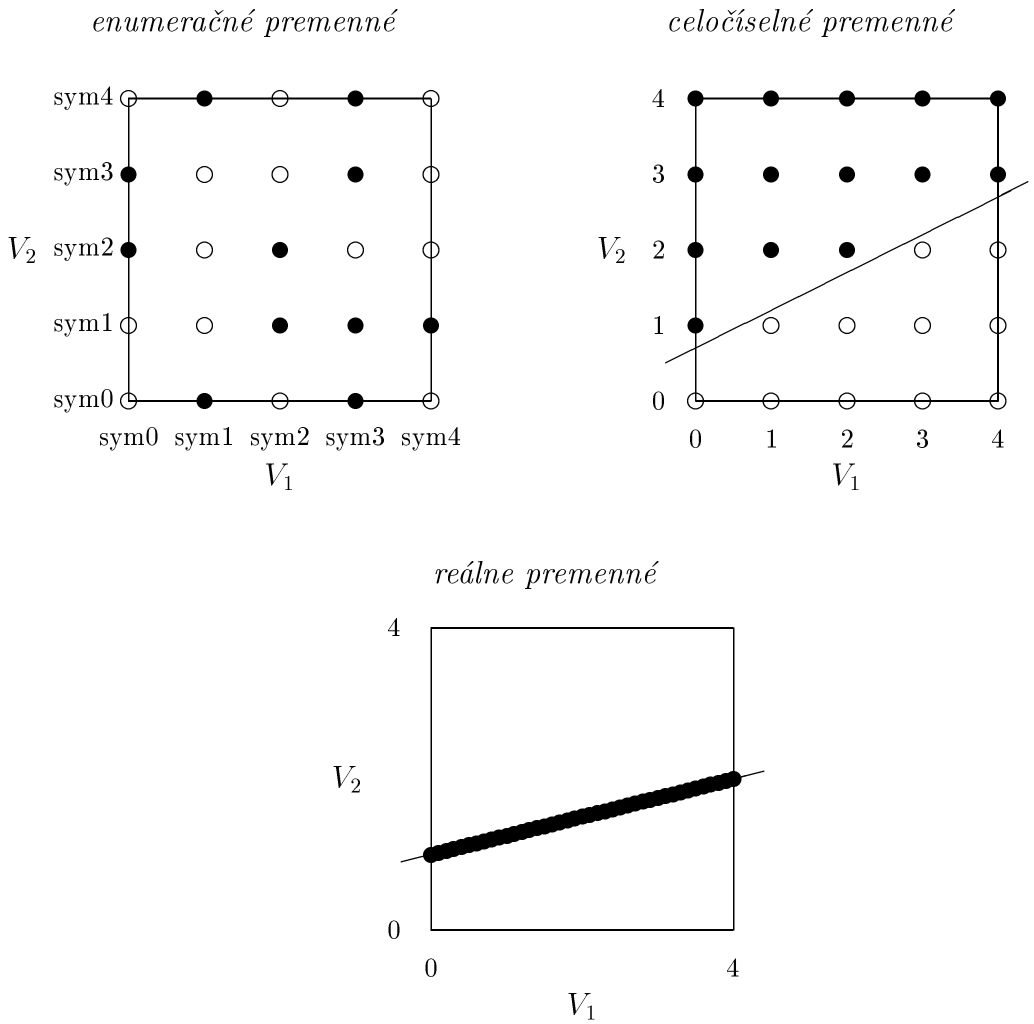
Premenné a ohraničenia je možné graficky znázorniť. Jednou z možností je znázornenie pomocou neorientovaného grafu – *siete ohraničení* (constraint network). Vo všeobecnosti v tejto sieti existujú dva typy uzlov – pre premenné a pre ohraničenia. Hrany sú možné iba medzi uzlami rôzneho typu a vyjadrujú väzbu medzi ohraničeniami a premennými. Inou možnosťou je namiesto grafu použiť hypergraf. V takomto prípade bude existovať iba jeden typ uzlov reprezentujúcich premenné. Ohraničenia budú znázornené ako hyperhrany spájajúce viac premenných súčasne. Na obr. 2 sú ilustrované oba spôsoby znázornenia siete ohraničení pre problém so štyrmi premennými $V_1, \ldots, V_4$ nad ktorými sú definované tri ohraničenia $C^{1,2,4}$, $C^{2,3}$ a C^3.



Obr. 2: Príklady zobrazenia premenných a ohraničení v sieti ohraničení.

Pre binárne problémy je možné použiť jednoduchšie znázornenie. V tomto prípade premenné sú reprezentované uzlami a hrany zase reprezentujú ohraničenia.

Niekoľko príkladov binárnych ohraničení definovaných nad dvoma premennými V_1 a V_2 je uvedených na obr. 1, kde plné krúžky reprezentujú povolené kombinácie hodnôt zatiaľ čo krúžky prázdne zastupujú kombinácie zakázané.



Obr. 1: Príklady binárnych ohraničení.

Zobrazené ohraničenie pre enumeračné premenné je dané vymenovaním

$$\{(sym0, sym2), (sym0, sym3), \ldots\}$$

ohraničenie pre celočíselné premenné zase ako nerovnosť

$$V_2 > 0.5V_1 + 0.75$$

a ohraničenie pre reálne premenné naopak ako rovnosť

$$V_2 = 0.25V_1 + 1.$$

Z hľadiska ohraničení nezávisí na poradí tých premenných, nad ktorými sú definované, a teda platí $C^{i,j} = C^{j,i}$. Inak povedané, to isté ohraničenie je možné označiť toľkými spôsobmi, koľko existuje rôznych permutácií poradia jeho indexov.

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

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

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 $(n^2 - n)/2$. Okrem n-árneho ohraničenia všetky ostatné sú definované iba nad nejakou podmnožinou premenných (premenné z tejto podmnožiny sa označujú ako relevantné voči príslušnému ohraničeniu). Všetky ohraničenia definované nad nejakou premennou V_i vytvárajú množinu ohraničení $C_i^{all} = \{C^i, C^{i,1}, \ldots, C^{1,\ldots,n}\}$. Celkový počet ohraničení ktoré je možné definovať nad nejakou premennou je

$$\sum_{i=1}^{n} \binom{n-1}{i-1}. \tag{2}$$

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 C zaradené iba formálne a možno ho z tejto množiny vypustiť.

Ohraničenia môžu mať rozličný tvar[2]. V zásade je možné použiť vymenovanie jednotlivých kombinácií (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 typu domén premenných a od konkrétnej úlohy. Vo všeobecnosti možno povedať, že napríklad pre enumeračné premenné sa zvyčajne využíva vyjadrenie vymenovaním kombinácií hodnôt a pre reálne premenné sa zase využíva častejšie forma predpisu (napr. v tvare nejakej rovnosti alebo nerovnosti).

[2]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.

2 Problémy s ohraničeniami

Pretože existuje viacero typov problémov s ohraničeniami, je možné nájsť viacej navzájom sa líšiacich definícií tohto typu problémov. Cieľom tejto časti je podať takú definíciu problémov s ohraničeniami, ktorá by pokrývala viacero existujúcich definícií a pre konkrétne triedy problémov s ohraničeniami by na tieto definície mohla byť redukovaná.

Problém s ohraničeniami je vo všeobecnosti definovaný pomocou troch množín:

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

Každá doména $D_i \in D$ zodpovedá premennej $V_i \in V$ a vlastne definuje možné hodnoty, ktoré premenná V_i môže nadobúdať. Tieto domény môžu byť rôznych typov. Najčastejšie sa používa delenie domén na

- *konečné domény* (reprezentantmi môžu byť napríklad booleovské premenné, celočíselné ohraničené premenné alebo premenné s enumeračnými doménami)
- *nekonečné domény* (napríklad celočíselné neohraničené a reálne premenné).

Veľa prístupov, používaných pre riešenie problémov s ohraničeniami, je schopných pracovať iba s jedným typom domén. Príkladom môže byť prístup využívajúci usporiadanie prvkov v doméne, ktorý nie je použiteľný pre tie domény, nad prvkami ktorých nie je definované usporiadanie.

V literatúre je väčšina prác venovaná problémom s konečnými množinami. Naproti tomu pri riešení praktických problémov sa často vyskytujú aj premenné, ktorých domény sú nekonečnými množinami.

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 $C^{i,\ldots,j}$ je definované nad podmnožinou premenných $V^{i,\ldots,j} = \{V_i, \ldots, V_j\} \subset V$, potom toto ohraničenie stanovuje prípustnosť priradenia $V_k = h$ (pre premennú $V_k \in V^{i,\ldots,j}$ a jej hodnotu $h \in D_k$) v kontexte tvorenom hodnotami ostatných premenných z $V^{i,\ldots,j}$. Jedinou výnimkou sú unárne ohraničenia, pri ktorých nie je potrebné brať do úvahy hodnoty iných premenných.

aj ďalšie využiteľné metódy (napr. lokálne heuristiky a celulárne automaty [52]).

Tento predkladaný text sa zameriava na tri rôzne prístupy[1] ktorých využiteľnosť je skúmaná v kontexte rôznych typov problémov s ohraničeniami. Špecificky je pozornosť venovaná metódam spĺňania ohraničení, evolučným algoritmom a znalostným metódam. Tieto prístupy predstavujú pomerne rozmanitú vzorku použiteľných metód pretože zahŕňajú

- ako metódy garantujúce nájdenie riešenia (ak toto existuje) tak aj metódy ktoré toto negarantujú,
- ako metódy pracujúce iba s jedným riešením tak aj metódy s paralelným spôsobom práce,
- ako stochastické metódy tak aj metódy bez akejkoľvek stochastičnosti,
- ako metódy špecializované na riešenie problémov s ohraničeniami tak aj metódy použiteľné aj pre riešenie iných typov úloh,
- ako metódy abstrahujúce od špecifických detailov konkrétnych úloh tak aj metódy stavajúce práve na týchto detailoch.

Predkladaný text je rozdelený do dvoch častí - hlavného textu a príloh. Hlavná časť je venovaná charakteristike jednotlivých typov úloh s ohraničeniami a popisu troch vyššie uvedených prístupov k ich riešeniu. Tento popis je prezentovaný na pomerne všeobecnej úrovni so snahou zachytiť hlavné princípy, na ktorých stoja metódy patriace k prezentovaným prístupom. Ako forma pre tento popis bol zvolený mechanizmus zobrazenia procesu riešenia úloh v stavovom priestore pomocou elementárnych riešiacich krokov. Toto poskytuje jednotiaci pohľad na inak značne rozdielne metódy.

Detailný pohľad na niektoré metódy riešenia problémov s ohraničeniami poskytujú zaradené prílohy. V prílohách sú zaradené práce autora, ktoré už boli publikované alebo sú v súčasnosti pripravované na samostatné publikovanie. Pre ich pochopenie nie je nutné oboznámiť sa s celou hlavnou časťou textu (prílohy nevyžadujú znalosť všeobecného popisu metód uvedeného v hlavnej časti), postačujúcou je znalosť definície problémov s ohraničeniami z kap. 2.

[1] Výber prezentovaných prístupov je založený na skúsenostiach autora s rôznymi metódami pre riešenie úloh s ohraničeniami.

1 Úvod

Pri svojej činnosti je človek obmedzovaný svojím okolím. Toto okolie neustále produkuje ohraničenia ktoré mu vymedzujú oblasť pôsobnosti. Týkajú sa doslova každej oblasti ľudskej prítomnosti, sú všadeprítomné a nie je možné sa im vyhnúť. Nie je možné napr. postaviť strechu skôr ako základy, zabaliť do kufra viac vecí ako sa do neho zmestí, naplánovať schôdzku na dobu keď subjekty budú na rôznych miestach, používať jeden hrniec na varenie dvoch polievok naraz, ap.

Tak je to aj v oblasti technickej praxe. Či sa už jedná o plánovací problém, rozvrhovanie, prideľovanie zdrojov, návrhové úlohy, a či niečo iné, neustále je potrebné zohľadňovať rôzne požiadavky ako napr. dostupnosť zdrojov, časové závislosti, charakteristiky použitých elementov, ap. Na druhej strane reprezentácia problémov pomocou explicitného popisu ohraničení je veľmi prirodzenou a názornou (a pritom pomerne jednoducho a rýchlo zostaviteľnou). Ak si uvedomíme koľko úloh technickej praxe možno formulovať ako úlohy s ohraničeniami, tak je zrejmý dôvod, prečo metódy riešenia úloh s ohraničeniami majú takú dôležitosť.

O význame tejto oblasti pre riešenie praktických problémov svedčí aj úsilie zahrnúť algoritmy pre riešenie úloh s ohraničeniami priamo do výpočtových systémov takým spôsobom, aby užívateľ musel definovať iba problém a nie aj to, ako tento problém riešiť. Toto je označované ako *programovanie ohraničení* (constraint programming) [3] a chápe sa pod ním počítačová implementácia algoritmov pre riešenie úloh s ohraničeniami. V praxi má rôzne podoby, či už formu knižníc, rozšírení programovacích jazykov, doplnkov do existujúcich systémov, alebo formu špecializovaných nástrojov.

Problémy s ohraničeniami je vo všeobecnosti možné rozdeliť do dvoch základných tried – úlohy pracujúce s konečnými alebo nekonečnými doménami. Predkladaný text sa sústreďuje na prvú skupinu. A práve problémy s ohraničeniami ktoré sú definované na konečných doménach majú kombinatorickú povahu. Toto je zdrojom pomerne značných požiadaviek kladených na metódy používané pre riešenie týchto problémov. Veľa typov týchto problémov totiž patrí do kategórie NP-úplných problémov, čo znamená že zložitosť algoritmu, ktorý by garantoval nájdenie riešenia spĺňajúceho všetky ohraničenia, je exponenciálna a teda časové požiadavky daného algoritmu sú vysoké – pre najhorší možný prípad rastú exponenciálne s veľkosťou problému. To však neznamená že by praktické úlohy boli neriešiteľné.

Je ich možné riešiť pomocou pomerne značného počtu prístupov pochádzajúcich z rôznych oblastí. Väčšina z týchto prístupov patrí do oblastí operačného výskumu a umelej inteligencie. Toto ilustruje napr. [22] kde pre riešenie toho istého problému je využitý predikátový počet, výroková logika, celočíselné programovanie, teória automatov, teória grafov, horolezecký algoritmus, neurónová sieť, genetický algoritmus, relačná algebra, syntéza ohraničení, disjunktívna a konjunktívna dekompozícia. Pritom toto vymenovanie nie je kompletné, existujú

Obsah

Ďakujem všetkým mojim kolegom z Katedry kybernetiky a umelej inteligencie FEI TU v Košiciach za cenné rady a pripomienky, podnety a podporu počas môjho pôsobenia na katedre. Obzvlášť ďakujem doc. Ing. Júliusovi Csontó, CSc. za prešliapanie cesty po ktorej som v uplynulom období išiel.

Tento text venujem mojim najbližším za pochopenie a toleranciu k mojej práci, ktorá ich obrala o mnohé spoločné chvíle so mnou.

Práca vznikla za podpory VEGA grantu MŠ SR č. 1/5032/98 "Integrácia prostriedkov inteligentných technológií".

Technická univerzita v Košiciach
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a umelej inteligencie

Vybrané metódy riešenia úloh s ohraničeniami definovanými nad konečnými doménami

Habilitačná práca

Ing. Marián Mach, CSc.

Košice, september 2000