FRAMEWORK PRO ŘEŠENÍ VÝPOČETNÍCH ÚLOH S OMEZUJÍCÍMI PODMÍNKAMI

Petr Kahánek
Logis s.r.o., pkahanek@logis.cz
Ostravská Univerzita, Přírodovědecká fakulta, Katedra informatiky, petr.kahanek@osu.cz

ABSTRAKT:
Předmětem tohoto článku jsou knihovny a vývojová prostředí pro řešení výpočetně náročných problémů specifikovaných pomocí omezujících podmínek. Cílem je najít vhodné otevřené prostředí pro řešení nelineárních problémů v jazyce C++.

KLÍČOVÁ SLOVA:
programování s omezujícími podmínkami, backtracking, propagace omezení, strategie větvení, nelineární optimalizace, C++

MOTIVACE

Tento článek vznikl na základě potřeby řešit specifické výpočetní problémy vznikající při rozvrhování výroby oceli, jako např. rozvrhování taveb, alokace materiálu a podobně. Společným rysem těchto problémů je, že jejich řešení podstatně závisí na „zkoušení mnoha možností“, neboli procházení prostoru všech řešení do hloubky. Dále budeme uvažovat problémy popsané pomocí omezujících podmínek [3].

Na potřebu řešit složitý výpočetní problém lze narazit kdekoli, kde se vyvíjený software nějakým způsobem dotýká výrobních postupů či technologie. V současné době je k dispozici celá řada komerčních i freewarových produktů, které nějakým způsobem podporují modelování a řešení takových problémů; v horším případě se jedná o samostatné programy nebo vývojová prostředí postavená na nějakém specifickém jazyce, v lepším případě daný řešič poskytuje knihovnu pro nějaký rozšířený programovací jazyk (např. C++, Javu či C#).

Důležitými otázkami jsou cena a otevřenost systému; komerční nástroje jsou poměrně drahou záležitostí a navíc se u nich nedá očekávat, že by součástí distribuce byly zdrojové kódy - systém tedy do jisté míry je černou skříňkou. Možnost ovlivňovat průběh algoritmu řešícího problém je tak často omezena na modifikaci parametrů nějaké funkce, případně na skládání či řetězení pevně daných, nedělitelných komponent. Vzhledem k vysoké variabilitě praktických úloh se může snadno stát, že nebudeme mít k dispozici řešič „ušitý na míru“ našeho problému a budeme muset sáhnout po něčem „alespoň trochu podobném“, což bude mít pravděpodobně negativní vliv na procesorový čas a paměťové nároky potřebné k nalezení řešení.

Z hlediska vývojaře by bylo ideální mít k dispozici knihovnu, nad kterou budou moci snadno implementovat vlastní řešič, který bude specifický přímo pro nějaký konkrétní problém. Knihovna by přitom měla být:

- psaná dle standardu,
- otevřená,
- parametrizovatelná,
- volně šiřitelná

Zaměříme se na programovací jazyk C++ [7] a platformy Windows a Linux.

PROGRAMOVÁNÍ S OMEZUJÍCÍMI PODMÍNKAMI

Programování s omezujícími podmínkami (Constraint programming, dále jen CP) je metoda v praxi často používaná při řešení různých výpočetních úloh (rozvrhování výroby, alokace materiálu atd.). Úloha je popsána deklarativně pomocí souboru omezení (constraints) nad proměnnými (variables), přičemž proměnná má přiřazenu množinu hodnot, kterých může nabývat (doménu). Řešením CP úlohy je takové ohodnocení proměnných, které splňuje všechny podmínky. Optimálním řešením CP úlohy vzhledem k nějaké účelové funkci je řešení, které tuto účelovou funkci minimalizuje [3][9][11][12].

Při řešení CP se využívá prohledávání prostoru řešení (např. pomocí backtrackingu) v kombinaci s propagací podmínek. Každá podmínka z domén svých proměnných odebírá tzv. nekonzistentní hodnoty (hodnoty, kterých proměnná nemůže nabývat, aniž by porušila podmínku). Propagace je jakási „spolupráce“ podmínek, kdy jejich kombinací můžeme odhalit další nekonzistentní hodnoty [3][11].

Například pro $x,y \in \{0, ..., 10\}$ z podmínek $x < y$ a $y > 5$ propagace vyvodí, že doména proměnné x může být zúžena na $\{7, ..., 10\}$.

Jakmile propagátory už nejsou schopny dále filtrovat domény, je třeba začít s prohledáváním. V každém kroku vybereme jednu proměnnou a omezíme její doménu, přičemž po každém takovém omezení opět spustíme propagaci [3][11][12]. Omezení se podstatně liší u proměnných celočíselných a spojitých. U celočíselných omezení domény vlastně znamená, že již přiřadíme proměnné konkrétní hodnotu, naproti tomu u spojitých proměnných doménu zpravidla rozdělíme na dvě poloviny (vzhledem k reprezentaci reálných čísel v počítači zde vznikají nepříjemné problémy).

Volba proměnné, které budeme doménu omezovat, může být klíčová pro rychlost řešení [3][12]. Je proto vhodné, aby byl způsob, jakým proměnnou vybíráme, v kontextu s daným problémem, respektive aby s výhodou využíval vlastností problému. Patrně nejčastěji se využívá strategie „first-fail“, která říká, že budeme omezovat proměnnou, která má nejmenší doménu [3]. Základní myšlenka v tomto případě je „pokud toto částečné řešení nevede k úplnému řešení, odhalme to co nejdříve“. Podobně je důležitá volba hodnoty (či intervalu), kterou proměnné přiřadíme. Zde se naopak hojně využívá strategie „succeed-first“, neboli se vybírá hodnota, která ponechá co nejvíce otevřených možností, tj. nezpůsobí příliš drastickou filtraci domén [3].

Rovněž volba vhodných propagátorů silně závisí na povaze problému, v některých případech je dokázáno, že záleží dokonce na jejich pořadí. Ukážeme to na příkladu problému URP (Unary Resource Problem).

URP je dán množinou úloh T (tasks), přičemž pro každou úlohu *t* jsou specifikovány

- $est(t)$ - čas, kdy může úloha nejdříve začít (earliest possible start time),
- $lpt(t)$ - čas, kdy může úloha nejpozději končit (latest possible finish time)
- $dur(t)$ - trvání úlohy

Řešením URP je rozvrh, který každé úloze přiřadí její počáteční čas $s(t)$ tak, že se žádné dvě úlohy nepřekrývají [3][11]. Například pro instanci T s úlohami

$$t_1,\ est(t_1) = 0,\ lpt(t_1) = 6,\ dur(t_1) = 2,$$
$$t_2,\ est(t_2) = 1,\ lpt(t_2) = 5,\ dur(t_2) = 2,$$

$$t_3,\ est(t_3) = 4,\ lpt(t_3) = 6,\ dur(t_3) = 2,$$

máme řešení

$$s(t_1)=0,\ s(t_2)=2,\ s(t_3)=4.$$

Pro URP existuje několik druhů propagátorů, které z domén například odvodí, že daná úloha nemůže začínat jako první, nebo že nutně musí začínat po jiné úloze a tato fakta následně využijí při filtraci domén [9][10][11][12]. Mezi takové propagátory pro URP patří kupříkladu edge-finding, not-first/not-last či detectable precedences. Ukazuje se, že na konečné omezení domén pořadí propagátorů nemá vliv, pro časovou náročnost výpočtu je však výhodné, aby byly tyto propagátory spuštěny právě v pořadí, v jakém jsou zde uvedeny [3][12].

Ukazuje se, že při řešení výpočetně náročných CP úloh je přínosné mít možnost použít vlastní způsob výběru proměnné (branching) i vlastní propagátor tak, aby co nejvíce odrážely povahu problému [9][10].

EXISTUJÍCÍ ŘEŠENÍ

Existují komerční i freewarové nástroje, které nabízejí knihovny různých nelineárních řešičů pro C++. Tyto nástroje implementují větší množství algoritmů vhodných pro konkrétní typy problémů s tím, že je možné je určitým způsobem parametrizovat. Dílčí řešiče jsou ovšem většinou atomické, tj. nelze ovlivnit jejich vnitřní logiku z vnějšku.

Mezi komerční nástroje nabízející řešení nelineárních problémů patří například

- Produkty firmy ILOG (v této oblasti zřejmě nejznámější výrobce), ILOG Solver, respektive vývojové prostředí ILOG OPL Development Studio využívající speciálního jazyka OPL (Optimization Programming Language). ILOG ovšem dodává také knihovny pro C, C++, C#, Java, Visual Basic a FORTRAN, například CPLEX Callable Library s rozhraním pro jazyk C. ILOG ve svých produktech nabízí velmi širokou škálu nástrojů. např. spoustu předdefinovaných propagátorů, algoritmů užitečných pro řešení konkrétních typů problémů atd. Nevýhodou je vysoká cena.
- Koalog Constraint Solver, který nabízí knihovnu v jazyce Java,
- balíček Optimization Toolbox pro MATLAB,
- balíček MathOptimizer pro program Mathematica,
- CHIP V5 od společnosti COSYTEC, který obsahuje knihovny pro C, C++ a PROLOG. Knihovna pro C++ implementuje některé vysokoúrovňové funkce, které mohou usnadnit modelování a zvýšit míru znuvupoužitelnosti. Navíc umožňuje vykonat uživatelsky definované callback funkce (tj. vykonat uživatelskou akci) při změně domény proměnné a takto ovlivnit průběh výpočtu.
- různé varianty PROLOGU (SICStus, IF/PROLOG)
- a mnohé další

Otevřených freewarových nástrojů existuje rovněž celá řada, mezi jinými

- Mozart – zajímavé vývojové prostředí založené na jazyce Oz, který je jakýmsi hybridem C++ a prologu a podporuje jak deklarativní, tak procedurální a objektově orientované programování. Oz je přímo určen pro programování pomocí omezujících podmínek, je ovšem poměrně náročný na zvládnutí. Některé části řešení mohou být definovány v externích dynamických knihovnách (napsaných například právě v C++), je tedy možné říci, že se dá ve větší míře ovlivnit průběh výpočtu. Je ovšem stále potřeba mít minimálně část kódu v Ozu a API není příliš příjemné [5].
- Lp_solve – lineární solver postavený na simplexové metodě, který ovšem umožňuje i řešení specifických typů nelineárních problémů. Je implementovaný v ANSI C a dá se

volat z mnoha různých jazyků. Umožňuje nastavovat parametry solveru, navíc nabízí možnost definovat uživatelské callback funkce, které mohou určovat směr prohledávání prostoru řešení. Nenabízí ovšem možnost definovat uživatelské funkce u propagací omezujících podmínek [4]

- CLP – další lineární simplexový řešič, otevřený a navíc postavený objektově v C++. Je jedním z mnoha podprojektů projektu COIN-OR (COmputational INfrastructure for Operations Research), který obsahuje mnoho řešení pro různé typy optimalizačních úloh. [6]

Nejobecnějším řešením pro řešení složitých úloh pomocí, které je autorovi známo, je systém Gecode, kterému je věnována následující kapitola.

GECODE

Gecode (Generics Constraint Development Environment) je vývojové prostředí pro tvorbu aplikací založených na CP [1]. Prostředí je napsané v C++ a je jak objektové, tak silně parametrické [1]. Je v něm implementována celá řada standardních strategií, umožňuje ovšem také definovat uživatelské funkce pro propagaci podmínek, strategie větvení a prohledávání prostoru řešení pomocí odvozování z bázových tříd. Například funkčnost propagace resp. strategie větvení (které jsme zmiňovali v části o CP) jsou implementovány ve třídách Gecode::Propagator, resp. Gecode::Branching.

Z našeho pohledu má Gecode několik důležitých vlastností:

- Otevřenost
  - Jde o otevřený systém, do kterého lze přidávat nové propagátory, strategie větvení i vyhledávací enginy. Mohou být rovněž přidány nové typy proměnných (Gecode sám definuje celočíselný typ IntVar).
- Volná šiřitelnost
  - Veškeré zdrojové kódy, dokumentace, příklady atd. jsou volně k dispozici pod licencí MIT.
- Přenositelnost
  - Implementace se drží standardu jazyka C++ a umožňuje kompilaci na široké škále systémů. Z našeho pohledu jsou důležité systémy Windows a Linux a potažmo kompilátory Microsoft Visual C++, resp. gcc.
- Výkon
  - Podle srovnávacích testů přímo na stránkách Gecode dosahuje systém co do časové a paměťové náročnosti na vybraných úlohách výborných výsledků v porovnání s jinými solvery (v testech jsou zahrnuty ILOG Solver, SICStus Prolog a Mozart Oz). Jak sami autoři podotýkají, tento fakt neimplikuje, že bude Gecode lepší pro řešení nějakého jiného specifického problému.

Podstatná je pro nás zejména prvně jmenovaná vlastnost, tj. snadná rozšiřitelnost knihovny. Gecode nám sám o sobě poskytne podporu technické stránky věci, tj. možnost definování proměnných a jejich domén, definování prostoru všech řešení, efektivní implementaci prohledávání tohoto prostoru (depth first search, branch and bound, limited discrepancy search) atd [1]. To je velmi výhodné, neboť odladění takovéhoto systému je vysoce netriviální záležitost. Budeme se tak moci plně věnovat implementaci propagátoru a strategie pro branching.

V neposlední řadě je Gecode velmi zajímavým otevřeným projektem, který má smysl dále rozvíjet. Pokud jde o nedostatky, Gecode poměrně dlouhou dobu trpěl absencí skutečné dokumentace (pokud nepočítáme komentáře ve zdrojovém textu zpracované Doxygenem) a

příkladů použití či tutoriálů. Vytvoření byť i jednoduché aplikace tak nebylo zcela triviální záležitostí; nedávno se ovšem na webu objevil velmi užitečný dokument [2].

DALŠÍ PRÁCE

Naším dalším cílem bude implementace propagátoru pro Gecode, který bude řešit určitý typ úloh; půjde nám o takové problémy, pro které existuje „poměrně rychle“ řešitelná relaxace, jejíž řešení (tj. částečné řešení původního problému) budeme využívat jako heuristiku při. procházení prostoru řešení původního problému.

LITERATURA

[1] Online dokumentace projektu Gecode (Generic Constraint Development Environment) http://www.gecode.org/

[2] Schulte, Ch., Tack, G., Lagerkvist, M.: Modeling With Gecode, http://www.gecode.org/doc-latest/modeling.pdf

[3] Barták, R.: Online Guide to Contraint Programming, http://kti.mff.cuni.cz/~bartak/constraints/

[4] Online dokumentace projektu lp_solve, http://lpsolve.sourceforge.net/5.5/

[5] Online dokumentace projektu Mozart, http://www.mozart-oz.org/

[6] Web projektu COIN-ORG, http://www.coin-or.org/projects/

[7] Stroustrup, Bjarne: The C++ Programming Language (Third Edition and Special Edition), Addison-Wesley, ISBN 0-201-88954-4 and 0-201-70073-5.

[8] Korte, B., Vygen, J.: Combinatorial Optimization - Theory and Algorithms, Fourth Edition, Springer-Verlag 2008, ISBN 978-3-540-71843-7

[9] Mercier, L., Hentenryck, P. V.: Edge Finding for Cumulative Scheduling, INFORMS JOURNAL ON COMPUTING, Vol. 20, No. 1, Winter 2008, pp. 143-153

[10] Vilím, P.: O(n log n) Filtering Algorithms for Unary Resource Constraint. In Proceedings of the First International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR’04), pages 319–334, Nice, 2004.

[11] Vilím, P.: Globální podmínky, Diplomová práce, Univerzita Karlova v Praze, Matematicko-fyzikální fakulta, Katedra teoreticcké informatiky a matematické logiky, 2001

[12] Vilím, P.: Global Constraints in Scheduling, Ph.D. Thesis, Charles University in Prague, Faculty of Mathematics and Physics, Department of Theoretical Computer Science and Mathematical Logic, 2007

[13] Nocedal, J., Wright, S.: Numerical Optimization, Springer-Verlag, ISBN 0-387-98793-2