Kombinatorická optimalizace

Semestrální práce

Tomáš Trnka

13. května 2014

Zadání

V semestrální práci bych chtěl řešit úlohu plánování s podmínkami a hierarchií jednotlivých událostí. V základu úlohy jde o plánování rozvrhu. Úloha ale má některá rozšíření a specifika. Zásadní rozdíl spočívá v tom, že událostí, které potřebuji naplánovat je několik druhů. Za prvé mám události, které se musí uskutečnit v danou hodinu a trvat určitou dobu - říkejme jim `striktní události`. Další události jsou takové, že mají nějaký deadline a které chci pokud možno rozvrhovat, tak aby byly v jednom kuse. Tyto události je potřeba splnit pouze před deadlinem, žádná jiný omezení na ně kladena nejsou a je mi jedno, kdy se uskuteční. Říkejme jim `střednědobé události`. Další druh události se musí odehrát každý den. Na ně klademe omezení, že musí mít nějakou minimální dobu trvání (čím více času na ně můžeme vyhradit tím lépe). Jejich délka se může měnit v určitém rozsahu a proto jim říkejme `natahovací události`. Posledním typem událostí jsou takové, které se odehrát mohou ve zbývajícím volném čase v rozvrhu, ale když se do rozvrhu nevejdou, tak se prostě zruší. Označme je `volitelné události`. S většinou událostí můžeme něco dělat, protože mají mezi sebou hierarchické vztahy je potřeba je nějak zakódovat. Přišlo by mi ideální celou úlohu řešit jako optimalizaci nějaké kriteriální funkce, která se bude odvíjet od množství porušení ideálních stavů pro každou událost. Výše zmíněnou hierarchii bych se pokusil zahrnout do penalizační funkce pomocí rozdílných hodnot penalizace pro porušení ideálních podmínek jednotlivých událostí.

Se `striktními událostmi` nemůžeme dělat téměř nic, tvoří jakousi základní strukturu rozvrhu. V extrémním případě a za velkou penalizaci dají zrušit úplně - taková situace by ale neměla téměř nastat. Pro události `střednědobého` typu je jedno, kam se v rozvrhu umístí, ale nutně to musí být před termínem jejich dokončení. Pokud se událost nevejde v jednom kuse (např 4 hodinový blok) je možné jí rozdělit s nějakou penalizací. Dělení lze provádět do rozumné hodnoty, počítejme, že minimální délka rozděleného bloku události bude minimálně 45 minut. `Natahovací událost` má každý den nějaký základní objem času, který je možno opět s penalizací zmenšovat. Zmenšovat nejde neomezeně, ale opět jen do nějaké rozumné hranice a to do 60% původního požadavku. Navíc by tato snížení neměla

klesnout dlouhodobě (v rámci týdne/měsíce) součtu pod hranici 80% součtu původních hodnot. Do zbývajících mezer v rozvrhu se umístí `volitelné události`, tak aby se minimalizoval prostor mezi jednotlivými událostmi.

Standardní rozvrhovací a plánovací úlohy se dají řešit pomocí ILP. Soudím ale, že moje úloha zahrnuje několik dílčích věcí, které je potřeba vyřešit než je možné jí takto řešit. Mám za to, že nejprve bude potřeba prohledat prohledat prostor stavů, na které se mi úloha může rozpadnout (dělením `střednědobých událostí`). Pro splnitelnost bych mohl použít standardní solvery, které prohledávají za pomoci heuristik stavový prostor. V momentě, kdy bych byl schopen říci, že nějaká posloupnost akcí mi umožní vygenerovat splnitelný rozvrh, bych se mohl pustit do optimalizace daného rozvrhu právě pomocí ILP.

Související práce

Pro řešení plánování bych mohl použít klasického plánovače, který prohledává celý stavový prostor, který prořezává za pomoci heuristik a snaží se optimalizovat nějakou kriteriální funkci, jako klasického zástupce bych mohl použít například plánovač `lama2008`, o kterém pojednává tento článek [Richter et al.(2011)Richter, Westphal,, Helmert]. Prohledávání probíhá pomocí váženého A* a je urychlováno heuristikami. Obávám se, že tento přístup by byl schopen řešit problémy v rámci jednotlivých, maximálně týdne.

Práce, které by se zabývali přesně mým tématem jsem nenašel. Nicméně úloha je nakonec podobná spíše než klasickému vytváření rozvrhů pro mnoho uživatelů a učeben plánování procesů pro jeden procesor. Úlohy plánování pro jeden procesor jsou podobné tomu, že potřebujeme vytvořit takovou posloupnost procesů, abychom je mohli všechny splnit v požadovaném čase.

Řešení plánování pro jeden procesor se dá rozdělit na dva případy, jeden je takzvané online plánování – jde o situaci, kdy máme procesor, který potřebuje vykonat nějakou předem danou sekvenci procesů, ale může být zároveň kdykoli vyrušen přerušením vyvolaným nějakým vstupem, obsloužit tuto událost a zároveň stihnout předchozí požadované úlohy. Tato úloha je obecně NP-hard, jak dokládá starý článek [Ullman(1975)]. Problémem plánování pro tyto systémy se zabývají například tyto práce [Braun et al.(2013)Braun, Chung,, Graham] a [Baruah et al.(1990)Baruah, Mok,, Rosier]. Problém jde redukovat na offline plánování, což mnohem více odpovídá našemu zadání. Události do kalendáře nastavím na počátku, a pak je možné je již jednou naplánovat - i při přidání nebo ubrání nejsem omezen obsluhou těchto událostí. Úlohy tohoto typu lze řešit v polynomiálním čase, jako například zde [Baptiste et al.(2007)Baptiste, Chrobak,, Dürr]. Algoritmus publikovaný v tomto článku má výpočetní složitost $O(n^5)$. Ve své práci bych mohl použít i poznatky z tohoto článku [Browne – Yechiali(1990)Browne, Yechiali], protože by dávalo smysl, aby se `střednědobá událost` upřednostňovala tím více, čím se blíží její limit pro splnění.

Návrh řešení

Způsobů řešení vidím několik. První z nich by mohl být formalizovat úlohu pro klasické plánování například PDDL. Na takto formalizovanou úlohu bych mohl použít standardní plánovače jako třeba Probe nebo Lama2008. Pro takovou formalizaci bych musel definovat kriteriální funkci, kterou bych optimalizoval. Pro jednotlivé události bych definoval akce, které by s nimi mohly provádět akce typu dělení, přesun do jiných dní a každá akce by typicky přidávala nějakou penalizaci – tj. za rozdělení události, za zrušení a jiné.

Další možností je použití plánovacích algoritmů v publikovaných článcích. Myslím, že řešením by bylo použití plánovacích algoritmů z článků – zejména již zmíněných polynomiálních například [Baptiste et al.(2007)Baptiste, Chrobak,, Dürr]. V tomto článku je předmět redukován na plánování s co nejmenším počtem „děr“, algoritmus (který navazuje na starší práce autorů) je založen na dynamickém programování. Pro mojí práci bych musel použít zmíněnou složitější variantu, která umožňuje aby jednotlivé události měli i jinou než jednotkovou délku. Pro implementaci budu muset vymyslet jak přepsat `události` do `procesů`. To by nemělo být příliš složité, protože každá událost má někdy začít a má nějaký deadline, podobně jako procesy. Navíc s tím zjednodušení, že v našem problému některé deadline nemají nebo se dá relaxovat (dá se s nám pohybovat). Při implementaci bych se rád řídit článkem, kde je podrobně popsáno, jak vyplňovat tabulku typickou pro dynamické programování a jak pak získat nejlepší hodnoty a na jejich základě vytvořit optimální plán událostí. Mám pocit, že v této reprezentaci bude možné zakódovat i trochu speciální dělící se události, protože při plánování je optimální plán s $n-1$ mezerami a pokud událost rozdělím, přidám další a zvýším počet mezer na n, a v takovém případě bude lepší plán s menším počtem událostí.

Kriteriální funkce, kterou budu potřebovat v každém případě, ať už se rozhodnu pro kteroukoli implementaci, bude vypadat zhruba následovně (předpokládám, že její parametry budu ladit v průběhu implementace, v závislosti na tom, jak ručně vyhodnotím výsledky optimalizace na reálných datech):

$$\begin{aligned} Q &= \min \sum_{p=1}^{P} \sum_{n=1}^{N} c_{str} \cdot str_n \cdot n + std_n \cdot n + nat_n \cdot n + vol_n \cdot n \\ & c_{str} = 100, c_{std} = 10, c_{nat} = 15, c_{vol} = 5 \end{aligned}$$

Kde P je počet dní pro které optimalizuji, N je počet událostí v jednotlivém dni. Parametry std, str, nat, vol jsou přiřazení událostí do příslušné kategorie (viz. výše), pro které platí následující podmínky. Přiřazení do skupin str_n a vol_n nabývají pouze hodnot 0 a 1, tedy, že se událost uskuteční a nebo neuskuteční. Pro std_n jde o celočíselnou hodnotu, která odpovídá počtu dělení dané událost, toto číslo by se mělo pohybovat v intervalu $\langle 0, 5 \rangle$. Pro události typu nat_n se bude jednat o celočíselnou hodnotu, kterou získám jako $nat_n = 8 - |nat_n|$, tedy od ideální délky události odečtu její skutečnou délku a tím získám příspěvek k penalizaci. V tomto nastavení parametrů je přítomna hierarchická struktur – která má i slovní interpretaci – například 10 rozdělení střednědobé události má stejnou váhu, jak zkrácení

natahovací události na čtvrtinu původní doby nebo vypuštění jedné striktní události.

Řešení

Pro řešení jsem se rozhodl zkombinovat klasické enumerativní řešení používané pro NP-těžké úlohy a dynamickým programováním. Existují však i jiné cesty, například použití randomizovaného algoritmu jako v tomto článku[Goemans et al.(2002)Goemans, Queyranne, Schulz, Skutella,, Wang]. Já jsem první část řešení implementoval podle článku[Cheng(1990)] z roku 1990. Jde o algoritmus dynamického programování, který je schopen úlohy seřadit v optimálním pořadí s ohledem na minimalizaci překročení deadlineů jednotlivých úloh. Protože je úloha koncipována v původním zadání jako $1|r_j|\sum Cj$ musel jsem trochu upravit výpočet hodnoty v jednotlivém kroku - v mém případě je ještě potřeba počítat s deadlinem. Když pak budu procházet tuto posloupnost událostí a splním ji, vím že žádnou jinou ji nemohu vylepšit. Tato vlastnost pomáhá prořezávat strom stavového prostoru.

Po prohledání optimální sekvence událostí se zkusí vytvořit relaxovaná varianta rozvrhu, tj. ponechají se jen události typu `striktní`. Pokud jen s nimi je rozvrh nesplnitelný, úloha se dále neřeší a považuje se za nesplnitelnou. Pokud takový to rozvrh mám, použiji jeho částečný plán ve struktuře `TimeTable` pro další výpočty, abych je urychlil – když se pokouším přiřadit danou úlohu pro daný čas, tak kontroluji jestli v dané části rozvrhu ještě není nic naplánováno. Stejně tak použiji i hodnotu kritéria, které jsem si spočítal, pokud byl relaxovaný rozvrh splnitelný. Pak se snažím vytvořit rozvrh pro všechny původní události. Pokud je takový rozvrh splnitelný, je dosaženo hodnoty kritéria 0 a není potřeba nic dalšího prohlédat, lepší hodnotu nelze s danou funkcí obdržet.

Pokud však je daný rozvrh nesplnitelný, je potřeba provést úpravy některých událostí. Abych se vyhnul komplikacím s další NP-těžkou úlohou, tj. vybrání těch událostí, které je potřeba zkrátit nebo zmenšit – snažím se je procházet opět postupně a pokud odhalím, že daná událost je příčinou toho, že rozvrh není splnitelný, tak nad ní vykonám akci, která je s daným typem události kompatibilní (rozdělím, zruším). Úlohy se zase procházejí podle jejich seřazení. Úlohy typu `volitelné` se primárně ruší, úlohy typu `střednědobé` se půlí. Zkracování událostí jsem nakonec vypustil z možných úprav rozvrhu - jelikož nemá cenu zkracovat události bez jejich kontextu, kontext mi ale v rámci lokálního prohledávání chybí, protože prohledávám bez návaznosti na změny jiných událostí. Úlohu jsem si zjednodušil tedy tím způsobem, že spolu většinou kolidují (nebo jinak interferují dvě události jiné než `striktní`) a pokud jednu z nich zruším mohu splnitelnosti pomoci. Tedy jde o lokální prohledání se snahou nalézt suboptimálního řešení. V rámci praktických pokusů jsem předefinoval původně navrhovanou ohodnocovaní funkci takovýmto způsobem:

$$\begin{aligned} Q &= \min \sum_{p=1}^{P} \sum_{n=1}^{N} c_{str} \cdot str_n(n) + std_n(n) \cdot div_{std} + c_{nat} + vol_n(n) \cdot c_{vol} \cdot n \\ & c_{str} = 5000, div_{std} = 50, c_{nat} = 50, c_{vol} = 150 \end{aligned}$$

kde jsem v rámci požadovaných výsledků upravil konstanty pro jednotlivé penalizace. A kde jsou logičtěji definované příslušnosti do jednotlivých kategorií pro jednotlivé úlohy. Všechny přiřazení vrací hodnotu $\{0, 1\}$, jen $std_n(n)$ vrací hodnotu dělení dané úlohy, tedy z intervalu $\{0, 1\}$.

Experimenty

Algoritmus jsem testoval na zkušebním rozvrhu pro 14 dní. Kde se po týdnech opakují `striktní` události a mezi se snaží algoritmus vyplnit události jiných druhů. D takového časového úseku se vejde zhruba mezi 40 - 50 úlohami, které je potřeba naplánovat.

Při experimentování jsem ověřil, že je velmi důležité, jak jsou jednotlivé úlohy seřazeny, je to dáno tím, že při lepším řazení se mi stavový prostor bude lépe prořezávat a já mohu ušetřit výpočetní čas, když budou úlohy seřazeny náhodně, tak jsem dostával vůbec pro vyřešení relaxované úlohy zhruba desetinásobné časy oproti seřazeným úlohám. Tedy v konkrétních údajích, plánování pro seřazené úlohy se řeší v rámci jednotek sekund, pro neseřazené to jsou až desítky. Doba řešení úlohy také závisí na počtu `střednědobých` úloh. Protože nemají žádný release date a proto mohou být vloženy do každého volného místa rozvrhu, čím zvyšují počet rekurzivních volání a zpomalují celý algoritmus.

Přestože algoritmus z článku [Cheng(1990)] používá dynamické programování, tak škáluje v závislosti na kombinatorických číslech podle počtu úloh. Bohužel tedy není zase tak moc efektivní a funguje rozumně zhruba pro 20 úloh.

Pro daný počet úloh (40-50) skončí výpočet poměrně rychle, časově a paměťově nejvíce náročné právě hledání optimální posloupnosti úloh. V takovém případě je potřeba vytvořit a držet v paměti poměrně velké datové struktury. Jejich hodnotu můžeme vyjádřit jako kombinační číslo, kdy vybíráme úlohy do možného pořadí. Pro n úloh bude největší číslo takovéto:

$$x = \sum_{i}^{n} \binom{n}{n} \text{pro můj konkrétní případ tedy dosadím}$$

$$x = \sum_{i}^{20} \binom{20}{i} = 1048575$$

Z výpočtu vyplývá, že pro tento počet úloh ale musí projít 1048575 všech kombinací pro různé velikosti sekvencí. A nejde jen o procházení kombinací, v každé iteraci se navíc ještě testuje pořadí poslední přidané úlohy. Pro každou další úlohu bude kombinací zhruba dvojnásobný, tedy časová složitost bude v třídě $O(2^n)$ kde n je počet úloh. Protože jsem pro jednoduchost implementace použil HashMapy, tak program při plánování spotřebuje zhruba až 1GB RAM. Při drobném zvýšení nad empiricky ověřenou hranici počtu úloh jednak plánování trvá velmi dlouho - řádově minuty a navíc program narazí na omezení standardní velikosti heapu Javy - tedy ani výpočet nedokončí.

Závěr

Při implementaci a podrobném zkoumání, to, čeho bych rád docílil jsem zjistil, že původní plán byl až příliš ambiciózní. V úloze bych totiž neřešil jednu striktně NP-těžkou úlohu, ale hned dvě. Uvědomil jsem si, že rozhodnout jaké události rozdělit, jaké vypustit, jaké zkrátit je další NP-těžký problém. A to i při uvažovaném zjednodušení, že bych události mohl zkracovat o několik dopředu daných různých délek, případně je dělit několika možnými způsoby a natož, kdybych uvažoval o případném nesymetrické dělení. Z toho hlediska se tedy naplnění záměru nepodařilo, protože jsem na něj v průběhu rezignoval.

Co se však podařilo je implementace funkčního plánovacího algoritmu, který dokáže vyřešit lehké konflikty mezi jednotlivými úlohami za pomocí vypuštění některých méně důležitých událostí případně rozdělením takových, u kterých lze tuto operaci provést. Toho se mi podařilo dosáhnout na základě několika kroků při výpočtu, které ho zjednodušují a urychlují. Jednak pomocí dynamického programování vyhledám optimální řazení akcí, ve kterém je pak plánuji do rozvrhu, který může mít mezery mezi jednotlivými událostmi (ty se při hledání sekvencí neuvažují). Pak pro úlohu pokouším vyřešit jako relaxovaný problém - tj. pro úlohy, které v rozvrhu musí nutně být, ostatní neuvažuji. Pokud jsem schopen nalézt takové řešení, použiji tuto vyřešenou „kostru“ pro řešení se všemi úlohami. Pokud tuto úlohu nejsem schopen splnit, tak následuje lokální prohledávání pro řešení konfliktů mezi úlohami. Z tohoto prohledávání se většino vytvoří několik rozvrhů, které si pak všechny můžu uživatelsky přívětivě prohlédnout.

Výstupem programu je totiž grafické okno, kde se zobrazí několik rozvrhů - jednak zobrazuje relaxovaný rozvrh, který se spočítal jako iniciální řešení. Dále umožňuje zobrazit všechny rozvrhy u kterých se podařilo vyřešit pomocí změn konflikt a nějak je naplánovat. Mezi těmito rozvrhy je možné v okně programu přepínat. U každého naplánovaného rozvrhu je uvedena jeho penalta, kterou nasbíral ze porušení ideálního naplánování všech úloh. Rozvrhy zobrazuji všechny, abych nemusel pokaždé měnit vnitřní parametry pro počítání postihu a mohl si vybrat z několika rozvrhů podle aktuálních potřeb.

Reference

[Baptiste et al.(2007)Baptiste, Chrobak,, Dürr] BAPTISTE, P. – CHROBAK, M. – DüRR, C. Polynomial Time Algorithms for Minimum Energy Scheduling. In ARGE, L. – HOFFMANN, M. – WELZL, E. (Ed.) *Algorithms – ESA 2007*, 4698 / *Lecture Notes in Computer Science.* : Springer Berlin Heidelberg, 2007. s. 136–150. doi: 10.1007/978-3-540-75520-3_14. Available from: http://dx.doi.org/10.1007/978-3-540-75520-3_14. ISBN 978-3-540-75519-7.

[Baruah et al.(1990)Baruah, Mok,, Rosier] BARUAH, S. – MOK, A. – ROSIER, L. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *Real-Time Systems Symposium,*

1990. Proceedings., 11th, s. 182–190, Dec 1990. doi: 10.1109/REAL.1990.128746.

[Braun et al.(2013)Braun, Chung,, Graham] BRAUN, O. – CHUNG, F. – GRAHAM, R. Single-processor scheduling with time restrictions. *Journal of Scheduling*. 2013, s. –. Available from: http://link.springer.com/10.1007/s10951-013-0342-0.

[Browne – Yechiali(1990)Browne, Yechiali] BROWNE, A. S. – YECHIALI, U. Scheduling Deteriorating Jobs on a Single Processor. *Operations Research*. 1990, s. 495–498.

[Cheng(1990)] CHENG, T. Dynamic programming approach to the single-machine sequencing problem with different due-dates. *Computers*. 1990, vol. 19, issue 2, s. 1–7. Available from: http://linkinghub.elsevier.com/retrieve/pii/089812219090001Z.

[Goemans et al.(2002)Goemans, Queyranne, Schulz, Skutella,, Wang] GOEMANS, M. X. et al. Single machine scheduling with release dates. *SIAM Journal on Discrete Mathematics*. 2002, 15, 2, s. 165–192.

[Richter et al.(2011)Richter, Westphal,, Helmert] RICHTER, S. – WESTPHAL, M. – HELMERT, M. LAMA 2008 and 2011. *The 2011 International Planning Competition*. 2011, s. 50.

[Ullman(1975)] ULLMAN, J. D. NP-complete Scheduling Problems. *J. Comput. Syst. Sci.* June 1975, 10, 3, s. 384–393. ISSN 0022-0000. doi: 10.1016/S0022-0000(75)80008-0. Available from: http://dx.doi.org/10.1016/S0022-0000(75)80008-0.