Přiblížení řešeného problému

Zadáním bylo seznámit se a vyzkoušet si jednu z přednášených pokročilých iteračních technik pro řešení konstruktivního problému batohu. Konstruktivní problém batohu je optimalizační úloha, kde v každé instanci problému máme zadanou kapacitu batohu a **n** věcí. Každá věc má definovanou svou váhu a cenu. Cílem je sestavit množinu předmětů, jejichž celková váha nepřekračuje kapacitu batohu a zároveň je jejich celková cena nejvyšší možná.
Měli jsme na výběr ze 3 algoritmů: simulované ochlazování, genetický algoritmus a tabu search. Rozhodl jsem se implementovat a zkoumat simulované ochlazování. Úkolem je algoritmus implementovat a za použití datových sad z druhé úlohy prozkoumat a pokusit se nalézt co nejlepší hodnoty jeho parametrů pro řešení úlohy batohu.

Kompletní zadání úlohy je na adrese: moodle-vyuka.cvut.cz/mod/assign/view.php?id=89702

Popis algoritmu Simulované ochlazování (Simulated Annealing)

Simulované ochlazování je algoritmus založený na jednoduché heuristice, která je obohacena o techniku, která ze simulovaného ochlazování dělá poměrně výkonný algoritmus. Hlavním cílem tohoto algoritmu je snaha vylepšit heuristické hledání řešení tím, že se pokusíme předejít uváznutí v lokálním optimu během hledání nejoptimálnějšího řešení. Tato snaha je založena na jednoduchém principu.
Algoritmus prohledává stavový prostor optimalizačního problému. Začíná v nějakém počátečním stavu a v každém kroku vybere náhodného souseda a porovná kvalitu jeho řešení s kvalitou řešení pro aktuální pozici. Pokud soused přináší zlepšení, přejde do něj, pokud nepřináší, tak za určité pravděpodobnosti se do jeho stavu algoritmus také rozhodne přejít. Právě tato pravděpodobnost a možnost přijmutí přechodu do horšího stavu umožňuje algoritmu vyřešit problém uváznutí v lokálním optimu. Vzorec pro tuto pravděpodobnost je založen na aktuální hodnotě parametru teploty, více o něm dále.
Algoritmus jsem implementoval následovně. Na začátku si vytvořím náhodnou instanci potencionálního řešení. Znám také již pevně nastavené hodnoty parametrů pro počáteční teplotu, koncovou teplotu a parametr pro řízení chlazení. Více o těchto parametrech a jak jsem k nim dospěl popisuji v další sekci.
Hlavní tělo algoritmu tvoří dvojitý cyklus. Během popisu budu hovořit o dvou proměnných, které si uchovávám. Tzv. globální nejlepší řešení a tzv. lokální nejlepší řešení. Na počátku algoritmu jsou obě dvě tyto proměnné rovny vytvořené náhodné instanci řešení.

Vnitřní cyklus běží vždy pro aktuální hodnotu teploty a opakuje prohledávání sousedů. Vždy vygeneruje náhodný index a pro lokální nejlepší řešení vezme předmět na daném indexu a na základě toho, zda v batohu daný předmět je, nebo není, ho z batohu odejme, respektive ho do batohu přidá. Následně porovná hodnotu takto vygenerovaného souseda s lokálním nejlepším řešením. Pokud je hodnota souseda větší a instance je platná (váha předmětů nepřesahuje kapacitu batohu), tak ho přijme a aktualizuje na něj hodnotu lokálního nejlepšího řešení. Pokud má tato nalezená instance vyšší hodnotu i než globální nejlepší řešení, aktualizuje i jej. Pokud instance není platná, nebo má hodnotu předmětů nižší, tak se vygeneruje náhodný float z intervalu [0, 1) a porovná se v následující rovnici:

```
random.nextFloat() < Math.exp(delta/t)/2
```

, kde delta je rovna rozdílu mezi hodnotou řešení nalezeného souseda a hodnotou lokálního nejlepšího řešení. Hlavním trikem celého algoritmu je, aby hodnota na pravé stráně nerovnice výše byla schopna na základě teploty přijímat horší řešení s rozumnou pravděpodobností, která se snižující se teplotou bude klesat. S tímto cílem bylo nutné pro hodnoty, které mi v pilotních experimentech vycházely, nutné, abych do vzorce přidal na pravou stranu rovnice dělení 2, které v běžné implementaci nebývá. Tento vnitřní cyklus

běží buďto 2***n**krát, kde **n** je počet dostupných předmětů pro daný problém a nebo běží do té doby, než proběhne celkem **n** posunů po stavovém prostoru.

Vnější cyklus má na starosti spouštění cyklu vnitřního, vhodnou inicializaci parametrů počítajících iterace a má také na starost aktualizaci hodnoty teploty. Teplota má při prvním běhu cyklu hodnotu rovnou parametru počáteční teploty. Vždy po skončení vnitřního cyklu se aktualizuje hodnota aktuální teploty pomocí vzorce t = t*α, kde t je aktuální teplota a α je hodnota parametru, který řídí ochlazování. Vnitřní cyklus běží do té dobu, než teplota dosáhne, nebo se sníží pod úroveň hodnoty parametru konečné teploty, nebo do té doby, než se nejlepší globální řešení aktualizuje **n***100krát. Druhá podmínka má za úkol zastavit algoritmus v případě, že je nejlepší hodnota aktualizována neúměrně často, tedy nejpravděpodobněji nastala ve stavovém prostoru situace, kdy přebíháme mezi dvěma srovnatelnými sousedy.

Postup při hledání vhodných hodnot parametrů

Algoritmus je ve svém principu obecný a lze jej nasadit na různé problémy. Pro každý problém je ovšem vhodná jiná kombinace parametrů. Pro optimalizační problém batohu jsou některé parametry jasné. Například omezení počtu cyklů v závislosti na počtu předmětů je zcela přirozené. Ovšem je potřeba přijít na nejvhodnější hodnotu tří parametrů. Tyto parametry jsou: počáteční resp. koncová teplota (označme t_0 resp. t_{final}) a parametr řídící rychlost chlazení (značíme α). Rozhodl jsem se tedy provést experiment nad jednou instancí dat a na ní pozorovat chování algoritmu a jeho reakce na parametry. Cílem bylo nalézt nejvhodnější hodnoty těchto parametrů a ty si následně ověřit na pár dalších instancích. Teprve po provedení těchto experimentů jsem pustil algoritmus na všechny datové sady. Pro pozorování skutečné citlivosti algoritmu na změnu parametrů jsem danou instanci nechal algoritmem vyhodnotit 200 krát a pozoroval průměr naměřených hodnot relativních chyb výpočtů.
Parametr α byl jeden z nejlehčích na experimentování. Jeho doporučená hodnota je z intervalu (0.8, 1). Algoritmus určuje rychlost ochlazování a při hodnotách blížících se 0.8 teplota až moc rychle klesala a algoritmus kvůli tomu velmi často skončil v lokálním optimu. Pro hodnoty blížící se 1 sice algoritmus provádí více iterací, ale také mnohem více horších instancí dostalo díky vyšším teplotám šanci a díky tomu byla šance na uváznutí mnohem menší. Hodnoty blížící se 1 také ukazovali nejlepší výsledky. Výsledná hodnota α, kterou jsem použil je 0.9998.

Parametry t_0 a t_{final} mi zabraly nejdelší čas vyladit. Tyto parametry mají na kvalitu výsledků nejvyšší vliv. Hlavním cílem bylo zvolit hodnotu počáteční teploty takovou, aby v kombinaci s hodnotami instancí batohu byla procentuální šance na přijetí horšího řešení v rozumné hodnotě. Mým cílem, který se mi povedlo uskutečnit, bylo, aby počáteční teplota vytvářela v průměru 50 % hranici na přijetí horšího řešení a tato hranice s ubývající teplotou rovnoměrně klesala. Vzhledem k tomu, že nikde není definováno, jakých hodnot má počáteční teplota nabývat jsem experimentoval s hodnotami z intervalu [30, 90 000]. Vypozoroval jsem, že nízké teploty nejsou vhodné, protože algoritmus provede pouze pár iterací a také malá hodnota teploty má nepříznivý vliv na procentuální hranici přijetí horšího výsledku. Konkrétně způsobuje, že skoro každé horší řešení je přijato. Vysoká hodnota počáteční teploty sice

zařídila velký počet iterací, ale výpočty algoritmu byly zbytečně dlouhé kvůli pomalému ochlazování. Nakonec jsem jako nejvhodnější interval pro volbu počáteční teploty vytyčil interval [4000, 6000]. Z čehož při mých experimentech nejlépe vycházela hodnota průměrné relativní chyby pro 200 spuštění pro hodnotu $t_0 = 5300$.

U parametru t_{final} , určující konečnou teplotu, velké hodnoty způsobovaly, že mi algoritmus doběhl dříve než stačil nalézt alespoň trochu dobré řešení. Pro extrémně malé hodnoty zase způsoboval zbytečné navýšení počtu iterací, které zdržovalo algoritmus opět bez přínostu lepších výsledků. Přitom při nízkých teplotách již šance na přijetí horšího řešení byla tak malá, že skoro žádné horší řešení nebylo nikdy přijato. Nakonec jsem ukotvil hodnotu tohoto parametru na hodnotě t_{final}=1000.

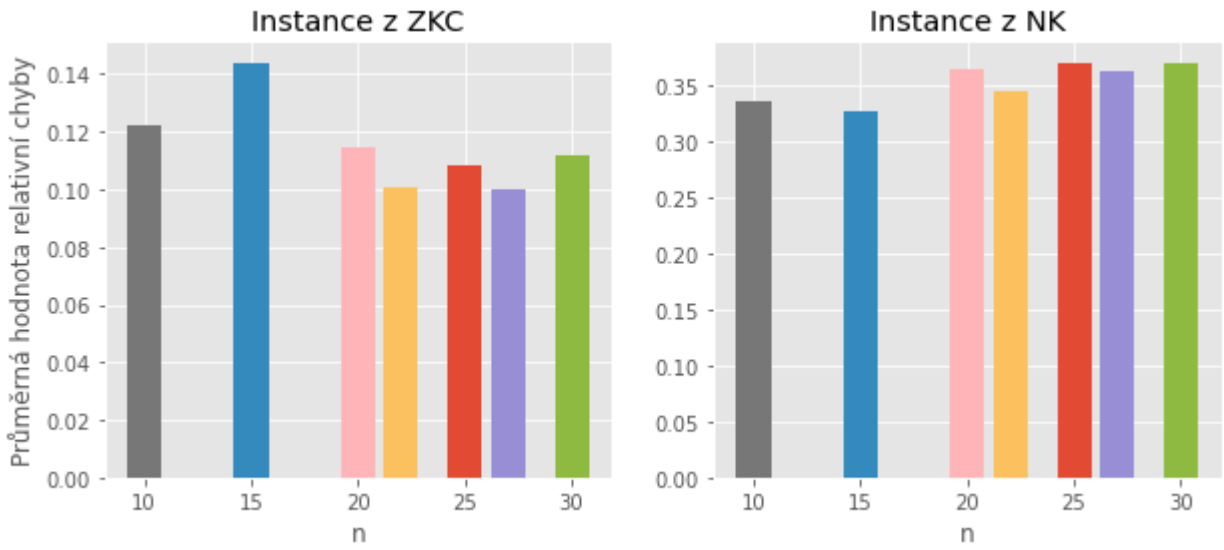
Výše testované hodnoty jsem odladil na jedné instanci problému batohu s **n**=22 předměty. Jelikož se stále jedná o aproximační algoritmus, který prohledá jen zlomek stavového prostoru, je dost běžné, že nemusí najít optimální řešení, ovšem relativní chyba by měla být malá. Finální hodnoty parametrů jsem tedy zvolil tak, aby tuto relativní chybu měli co nejmenší. Instanci jsem nechal algoritmus vyhodnotit 200x a zprůměroval, abych získal pravý vliv parametrů. Pro mnou vybrané parametry jsem z testovací sady ZKC, která je dostupná zde, dostal hodnotu relativní chyby 0.7 %.

```
Average best price mismatch in 200 executions if pilot test: 0,007
DONE
```

Následně jsem zkusil vyhodnotit jiné instance za použití stejných parametrů. Jednalo se o instanci s **n**=20 ze stejné datové sady a instanci s **n**=20 z druhé datové sady jménem NK. Pro každou zmíněnou instanci jsem spustil opět 200 měření. Výsledky mě potěšili, jelikož nabývaly do 0.012, tedy hodnot relativní chyby do 1.2 %, což považuji za dobrý výsledek. Rozhodl jsem se tedy celé obě zmíněné datové sady vyhodnotit a do grafu zachytit naměřené hodnoty relativních chyb pro jednotlivá **n** v datových sadách.

Naměřené hodnoty na datových sadách ZKC a NK

Skutečnou úspěšnost zvolených parametrů jsem se rozhodl posoudit na výsledcích naměřených na 2 datových sadách a porovnat výsledky mezi s sebou. Grafy níže zachycují průměrnou hodnotu relativní chyby pro jednotlivá **n**.



Je možné pozorovat, že pro instance, jejichž **n** se lišilo od **n**, které jsem zkoumal v pilotním experimentu, je hodnota průměrné chyby větší. Také je možné sledovat rozdíl mezi hodnotami naměřenými na datové sadě ZKC a datové sadě NK. Je vidět, že pro data z NK jsou výsledky výrazně horší, což naznačuje, že algoritmus je velmi citlivý na podobu vygenerovaných dat.

Závěr

Myslím, že se mi povedlo na základě pilotních experimentů vybrat hodnoty klíčových parametrů poměrně vhodně. Je možné vidět, že hodnota průměrné relativní chyby je mírně horší, než kterou jsem měl u pilotních experimentů, ale to se dalo očekávat, protože instance jsou od sebe velmi rozdílné. I přesto si myslím, že jsou výsledky povedené. Také si myslím, že velké zlepšení výsledků by se objevilo, kdybych algoritmus pro každou instanci pustil vícekrát. Díky tomu by pokaždé začínal prohledávat stavový prostor z jiného místa a šance nalezení správného řešení by se výrazně zvýšila. Toto opakované spouštění je pro tento algoritmus běžné i v praxi. Pro velký počet zkoumaných instancí v hlavním experimentu jsem se rozhodl toto opakované spouštění vynechat. Také jsem chtěl pozorovat výsledky algoritmu po jednom běhu, protože si myslím, že to lépe vypovídá o jeho výpočetní síle.
Povedlo se mi tedy pochopit a implementovat algoritmus simulovaného ochlazování, odladit hodnoty parametrů během prvotních experimentů a následně porovnat naměřené hodnoty a zhodnotit správnost parametrů na velkém množství jiných instancí.

Ondřej Schejbal
gitlab.fit.cvut.cz/schejond/ni-kop/tree/master/KOP-4