Problém batohu hrubou silou a jednoduchou heuristikou

Dominik Plíšek

22. 10. 2015

Úvod k implementaci

Rozhodl jsem se pro implementaci v jazyce C, jelikož mám rád u podobných algoritmů pod svoji kontrolou paměťové nároky. Napsal jsem obecné struktury pro:

instanci problému id, kapacita, počet věcí a pole věcí

řešení problému váha, cena, bitová mapa značící přítomnost věcí z instance v batohu

věci v batohu váha, cena

Napsal jsem obecnou *main*() metodu pro řešení instancí ze *stdin* a výpis výsledků do *stdout*. Vytvořil jsem si hlavičku funkce pro obecné řešení instance problému. Implementaci této funkce v jednotlivých způsobech řešení zaměňuji.

Výpočet provádělo jedno vlákno čtyřjádrového procesoru 2,3 GHz Intel Core i7 s 6 MB L3 cache. Naměřené časy představují procesorový čas, tedy čas, kdy skutečně vlákno pracovalo.

1 Hrubou silou

Naprogramujte řešení problému batohu hrubou silou (tj. exaktně). Na zkušebních datech pozorujte závislost výpočetního času na n.

1.1 Diskuse

Řešení hrubou silou znamená vyzkoušet všechny stavy stavového prostoru, abychom se 100% jistotou nelezli nejlepší (nebo jedno z nejlepších) řešení.

Je možno jej implementovat buď iterativně, nebo rekurzivně. Iterativní metoda by v tom případě byla asi přehlednější, ale vzhledem k tomu, že úlohy, které budou následovat, bude patrně podstatně snažší řešit rekurzí, použil jsem rekurzi i pro hrubou sílu.

1.2 Implementace

Program rekurzivně volá funkci se zvětšujícím se indexem, která vždy zkusí věc s daným indexem do batohu přidat i nepřidat, načež zavolá dvakrát sama sebe, aby zkusila, co se bude dít v obou případech. Program věc do batohu nepřidá v případě, že se do něj již nevejde. Takto postupně vyzkouší všechny kombinace věcí v batohu. Nakonec vrátí výsledné tvrzení o nejvyšší možné ceně a jak jí dosáhnout.

Velmi hrubá kostra:

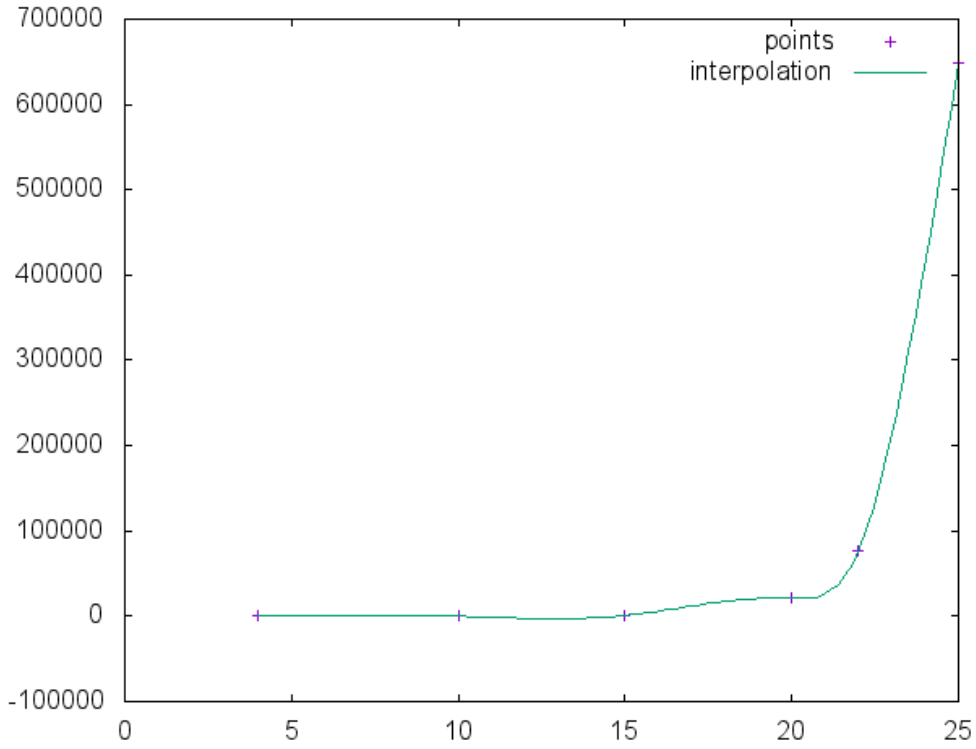
```
vyřešInstanci(instance) {
  vyhodnoť StavVýsledku(výsledek, 0)
  vrať výsledek
}

vyhodnoť StavVýsledku(výsledek, index) {
  nejsou-li již věci, vrať se
  vejde-li se věc s indexem do batohu (výsledku),
    pak si zapamatuj výsledek,
    dej ji tam
    a vyhodnoť StavVýsledku(výsledek, index)
  pak zkus vyhodnotitStavVýsledku(původníVýsledek, index)
    -- bez této věci
  byla-li první možnost výhodnější, vrať se k ní
}
```

1.3 Výsledky

Závislost času výpočtu na počtu věcí k přibalení do batohu je vidět z této tabulky:

Počet věcí	Čas výpočtu (millis)
4	0.520
10	14.634
15	709.653
20	21424.700
22	77237.619
25	648182.280

Pro lepší představu, jak rychle čas roste, přidávám proložený graf:



2 Jednoduchou heuristikou

Naprogramujte řešení problému batohu heuristikou podle poměru cena/váha. Pozorujte

- závislost výpočetního času na n. Grafy jsou vítány (i pro exaktní metodu).
- průměrnou a maximální relativní chybu (tj. zhoršení proti exaktní metodě)

2.1 Diskuse

Řešení se liší od hrubé síly především v tom, že se v každém kroku vyzkouší pouze jedna možnost, nikoli dvě. To znamená, že prostor řešení se prochází lineárně.

2.2 Implementace

Program opět rekurzivně volá funkci, která přidává věci do batohu. Tentokrát ale nepoužívá index, protože věc, kterou do batohu přidá, v každém kroku vybírá podle heuristiky. Konkrétně vždy spočítá poměr cena/váha a vybírá věc s nejvyšším.

Jakmile vybere věc, která se přidá, už ji neodebere. Považuje výběr za rozhodnutý a vstupuje do rekurze řešit další stav. Program věc do batohu nepřidá v případě, že se do něj již nevejde.

Nakonec vrátí výsledné tvrzení o nejvyšší možné ceně a jak jí dosáhnout.

Velmi hrubá kostra:

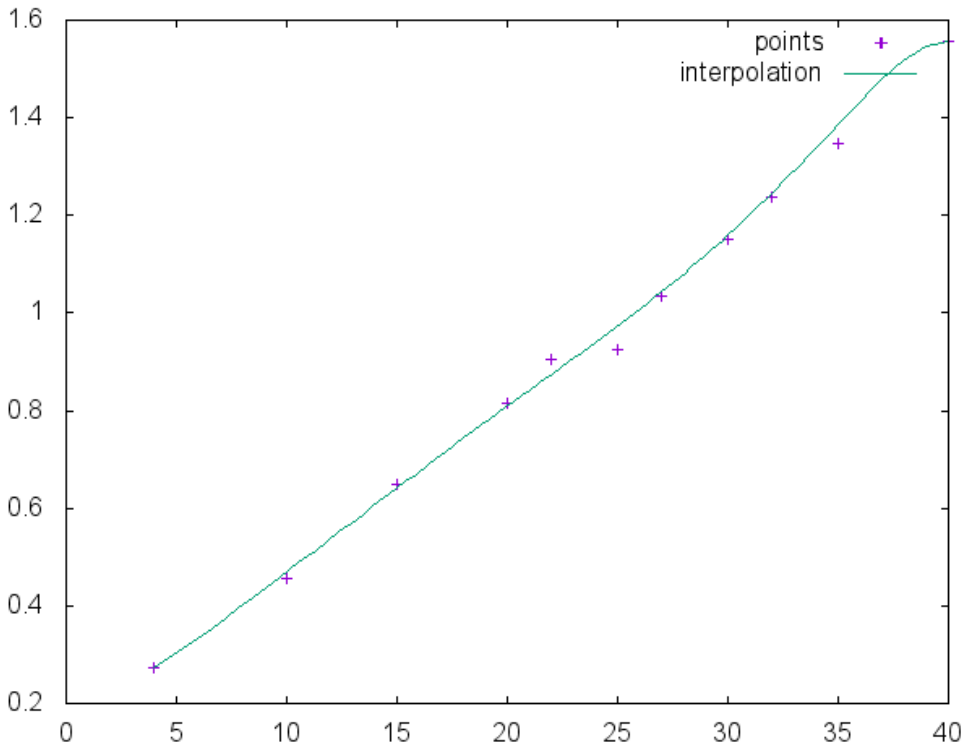
```
vyřešInstanci(instance) {
  vyhodnoť StavVýsledku(výsledek)
  vrať výsledek
}

vyhodnoť StavVýsledku(výsledek) {
  nejsou-li již věci, vrať se
  vyber věc s nejvýhodnějším poměrem
  vejde-li se věc do batohu (výsledku),
    dej ji tam
    a vyhodnoť StavVýsledku(výsledek)
}
```

2.3 Výsledky

Závislost času výpočtu na počtu věcí k přibalení do batohu je vidět z této tabulky:

Počet věcí	Čas výpočtu (millis)
4	0.272
10	0.455
15	0.649
20	0.815
22	0.906
25	0.924
27	1.036
30	1.150
32	1.238
35	1.346
37	1.554
40	1.556

Pro lepší představu, jak rychle čas roste, přidávám proložený graf:



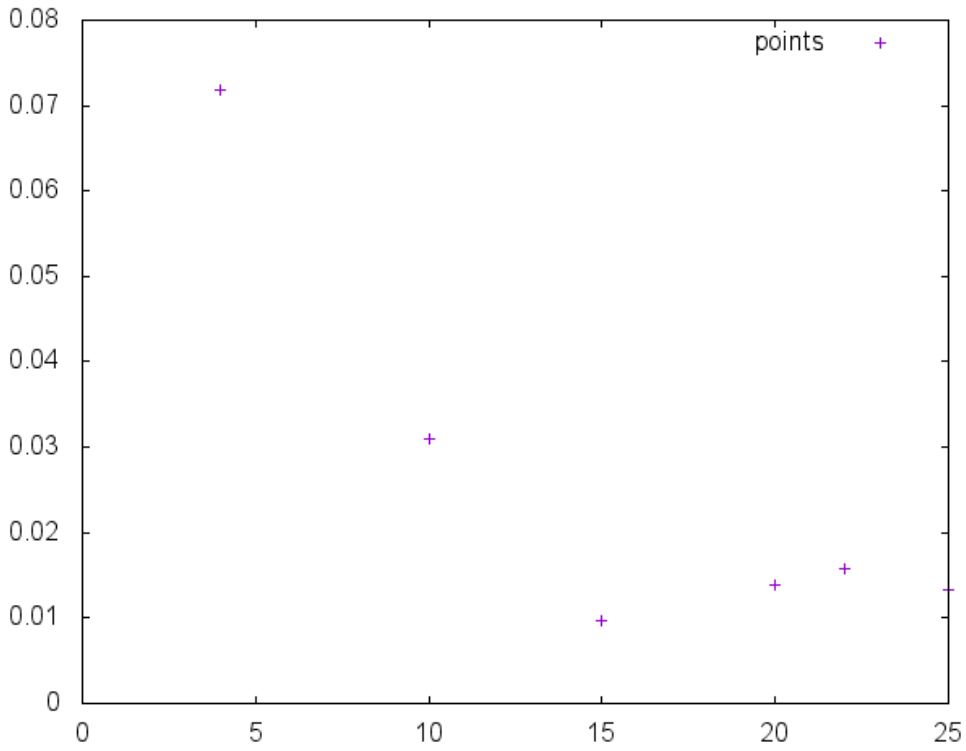
2.4 Relativní chyba

Heuristika počítá rychleji, ale ač není úplně hloupá, dopouští se chyby, protože nevyzkouší všechny možnosti. Tuto chybu jsem spočítal po každé jednotlivé

instanci pomocí vzorečku $(C(OPT) - C(APX))/C(OPT)$. Po každé sadě instancí jsem chybu zprůměroval. Výsledky jsou v následující tabulce:

Počet věcí	Relativní chyba
4	0.0719
10	0.0309
15	0.0097
20	0.0139
22	0.0157
25	0.0133
celkový průměr	0.0259

Po vynesení hodnot do grafu není na první pohled jasné, jestli existuje nějaký jednoduchý vztah mezi počtem věcí a relativní chybou:



3 Závěr

Je očividné, že heuristika způsobuje obrovské zrychlení. Oproti zdá se exponenciálnímu růstu potřebného času u hrubé síly je vztah času a počtu věcí u heuristiky více méně lineární.

Relativní chyba vypadá, že bude možná s přibývajícími věcmi klesat. To ale nemusí být nutně pravda. Potřebovali bychom větší počty věcí, jenže pro ty trvá hrubá síla extrémně dlouho.

4 Přílohy

sources.zip Obsahuje balík zdrojových kódú v jazyce C. Neobsahuje makefile.