# MI-PAA — Domácí úkol č.1

Řešení problému batohu metodou hrubé síly a jednoduchou heuristikou

Jakub Trhlík

25. října 2018

# 1 Zadání

## 1.1 Základ problému

Je dáno:
celé číslo n (počet věcí)
celé číslo M (kapacita batohu)
konečná množina V = v1, v2, ... ,vn (hmotnosti věcí)
konečná množina C = c1, c2, ... ,cn (ceny věcí)

## 1.2 0/1 problém batohu

0/1 problém batohu je nejznámější varianta, obvykle je tím co se myslí "problémem batohu".
Zkonstruujte množinu X=x1, x2, ... ,xn , kde každé xi je 0 nebo 1, tak, aby platilo

$$v1x1 + v2x2 + ... + vnxn < = M$$

(aby batoh nebyl přetížen). Výraz

$$c1x1 + c2x2 + ... + cnxn$$

nabýval maximální hodnoty pro všechny takové množiny (cena věcí v batohu byla maximální).

# 2 Rozbor možných variant řešení

Možností řešení je velice mnoho. Tento úkol se specializuje na řešení heuristikou a na řešení Hrubou silou.

## 2.1 Možná řešení heuristikou

Jako heuristiku lze zvolit různé vlastnosti, například cenu předmětu vkládaného doo batohu, jeho hmotnost, nebo poměr mezi cenou a hmotností.

## 2.2 Možná řešení Hrubou silou

Problém batohu lze řešit hrubou silou pomocí rekurze, nebo pomocí cyklu.

V případě řešení rekurze vzniká strom volání, kde kde každý uzel takového stromu představuje jinou verzi batohu.

V případě cyklu, je nejlpve třeba vygenerovat možné verze batohu a poté je porovnat mezi sebou, respektive najít nejvýhodnější konfiguraci batohu.

# 3 Řešení Hrubou silou

## 3.1 Popis Řešení

Pro řešení byl použit jazyk C++

## 3.2 Třída BackpackProblem

Jde o třídu která pracuje s ukazately na pole cen položek a hmotnosti položek.

Třída má bitové pole, které značí která položka byla do batohu vložena a která nebyla.

```
int id,n,M,Price,initM;
int * Weights, * Prices, * itemNum;
```

## 3.3 Algoritmus

Jde o přímý rekursivní algoritmus.

Vytvoří se počáteční BackpackProblem, bez vložených položek.

Funkce bruteForceCalculate se spustí s danou počáteční instancí BackpackProblem a i=0, tedy položku o které se bude rozhodovat, zda do batohu patří. Maximální kapacitou batohu.

V případě, že je batoh plný, funkce vrátí tuto instanci, v případě že již v batohu nejsou další položky, funkce také vrátí danou instanci.

Funkce spustí samu sebe pro batoh s položkou přidanou do batohu a s položkou nepřidanou do batohu.

Z verzí batohu které funkce získala z volání, vybere tu lepší a vrátí ji.

## 3.4 Měření

Měření bylo vykonáno na Xiaomi Air a procesorem core i5 7 generace pro Ultrabooky.

Menší soubory byly spuštěny v cyklu až 10000x a výsledný celkový čas průměrován.


Obrázek 1: Doba běhu souborů instanci knap 4.inst.dat až knap 27.inst.dat


Obrázek 2: Doba běhu souborů instanci knap 4.inst.dat až knap 27.inst.dat v Logaritmiském měřítku času

### 3.4.1 Způsob Měření

Pro účely měření je využita funkce multipleruns a funkce knap, které automatizují vícenásobné spouštění a porovnávání souborů typu knap x.inst.dat

## 3.5 Výsledky měření

Na grafech je jednoznačně vidět exponenciální povaha algoritmu.

# 4 Řešení Hrubou silou

## 4.1 Popis Řešení

Pro řešení byl použit také jazyk C++ a také třída BackpackProblem viz řešení hrubou silou

## 4.2 Algoritmus

Jako Heuristiku jsem použil poměr ceny a hmotnosti.

$$H = W/P$$

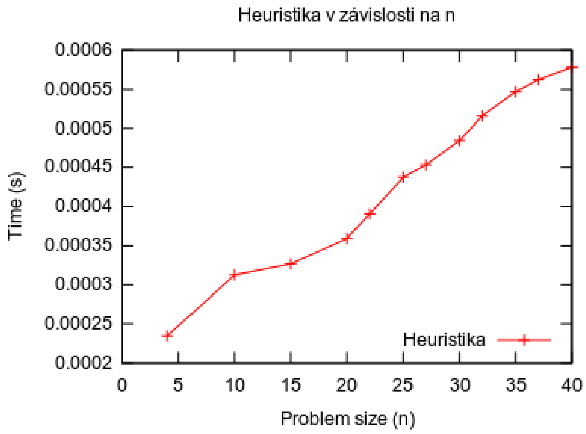
Pro každou možnou položku batohu byl vypočítán koeficient heuristiky H.
Položky byly seřazeny podle H a postupně přidávány do batohu.
Položky, které vložit do batohu nebylo možné byly přeskočeny.

## 4.3 Měření

Měření bylo vykonáno na Xiaomi Air a procesorem core i5 7 generace pro Ultrabooky.
Menší soubory byly spuštěny v cyklu až 10000x a výsledný celkový čas průměrován.

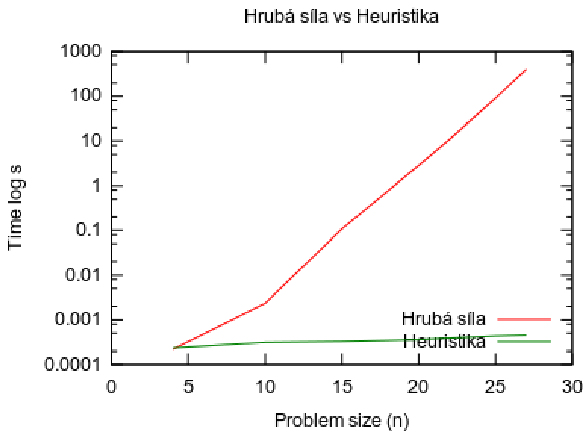
## 4.4 Výsledky měření

Na grafech je jednoznačně vidět lineární zpomalení algoritmu.




Obrázek 3: Doba běhu souborů instanci knap 4.inst.dat až knap 27.inst.dat

### 4.4.1 Hrubá síla vs Heuristika




Obrázek 4: Doba běhu souborů instanci knap 4.inst.dat až knap 27.inst.dat

### 4.4.2 Průměrná relativní chyba

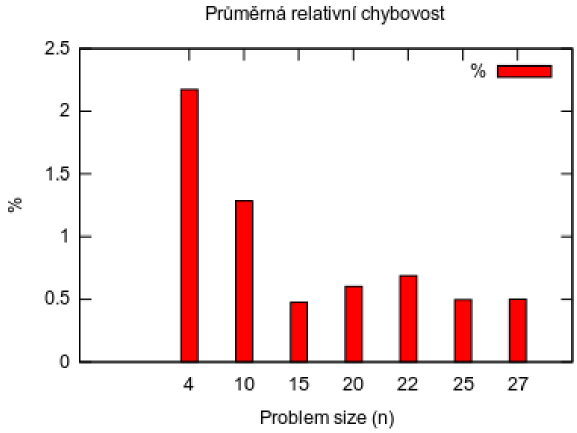
Každá instance byla vypočítána heuristikou i hrubou silou, z nich byla vypočítána relativní chyba. Relativní chyba byla poté průměrována pro celý soubor.

Výpočet relativní chyby

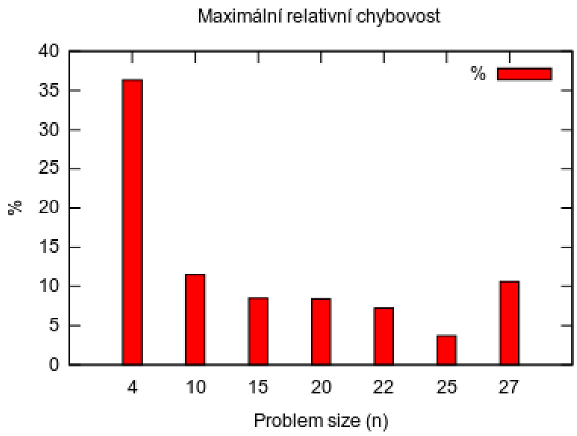
$$E = (C(OPT) - C(APX))/C(OPT)$$

### 4.4.3 Maximální relativní chyba

Maximální relativní chyba v každém souboru testovacích instancí.




Obrázek 5: Průměrná relativní chybovost knap 4.inst.dat až knap 27.inst.dat




Obrázek 6: Maximální relativní chybovost pro knap 4.inst.dat až knap 27.inst.dat