1 Zadání úlohy

Celé zadání se skládá ze tří částí:

1. Naprogramujte řešení problému batohu metodou větví a hranic (BB) tak, aby omezujícím faktorem byla hodnota optimalizačního kritéria. Tj. použijte ořezávání shora (překročení kapacity batohu) i zdola (stávající řešení nemůže být lepší než nejlepší dosud nalezené),
2. Naprogramujte řešení problému batohu metodou dynamického programování (dekompozice podle kapacity nebo podle cen),
3. Naprogramujte řešení problému batohu FPTAS algoritmem, tj. s použitím modifikovaného dynamického programování s dekompozicí podle ceny.

2 Rozbor možných variant řešení

Problém batohu lze řešit:

- hrubou silou,
- metodou větví a hranic,
- metodou dynamického programování (dekompozice podle ceny),
- metodou dynamického programování (dekompozice podle váhy),
- algoritmem FPTAS.

V rámci této práce jsem se zaměřil na pět způsobů řešení problému batohu.

2.1 Hrubou silou

Řešení problému batohy hrubou silou probíhá tak, že se nechají vygenerovat všechny možné kombinace výskytu jednotlivých položek v batohu. Následně se spočítá, která z vygenerovaných kombinací vyhovuje našim kritériím (největší cena, suma hmotnosti položek v batohu nepřekročí jeho nosnost). Složitost tohoto algoritmu je exponenciální.

2.2 Metodou větví a hranic

U řešení metodou větví a hranic vytváříme binární strom, kde jeden syn symbolizuje, že věc byla do batohu přidána a druhý syn, že ne. U této metody předvídáme zdáli generovaná kombinace má potenciál dosáhnou cenově výšky alespoň takové jakou měla dosud nejlepší nalezena kombinace. To znamená, že při generování kombinací si pamatujeme nejlepší dosaženou cenu a některé kombinace nemusíme ani generovat, neboť víme, že jejich horní hranice ceny batohů pokud budeme dále v rekurzi sestupovat, nemůže dosahovat nejlepší dosud nalezené ceny. Ačkoli se zdá, že výsledná složitost bude lepší nežli exponenciální, není tomu tak. Jelikož se může stát, že nebudeme moci ukončit generování žádné kombinace.

2.3 Metodou dynamického programování

2.3.1 Dekompozice podle ceny

Další metodou je dynamické programování s dekompozicí podle ceny zdola nahoru. U této metody se připraví tabulka $(n+1)*(c+1)$, kde n je počet vkládaných věcí a c je suma jejich cen. Řádky tabulky symbolizují cenu a sloupce vkládanou věc. Tabulka se poté vyplňuje zdola nahoru takovým způsobem, že:

- $T[0][0] = 0$,
- $T[c][0] = \infty; \forall c > 0$,
- $T[c][i+1] = min(T[c][i], T[c - c_{i+1}][i] + w_{i+1}); \forall i > 0$, kde w_{i+1} je váha $(i+1)$. věci a c_{i+1} je cena $(i+1)$. věci.

Tudíž vybíráme zdali je lepší věc do batohu přidat nebo nepřidat. Na konci získáme nejlepší řešení tak, že budeme postupně procházet poslední sloupec tabulky shora dokud nedojdeme na řádek, jehož buňka bude mít hmotnost menší nebo rovnu hodnotě kapacitě batohu. Číslo tohoto řádku je poté nejlepší cena. Výsledná složitost odpovídá velikosti tabulky. Je tedy závislá na velikosti vstupních dat, ale i na konstantě. Takový algoritmus se nazývá pseudopolynomiální.

2.3.2 Dekompozice podle váhy

Při dekompozici podle váhy vytváříme tabulku o velikosti $(n+1)*(M+1)$, kde M je maximální kapacita batohu a n je počet přidávaných věcí. Řádky reprezentují velikosti kapacity. Zde tabulku vyplňujeme také zdola nahoru, ale poněkud jiným způsobem:

- $T[c][i+1] = max(T[w][i], T[w - w_{i+1}][i] + c_{i+1}); \forall i > 0$, kde w_{i+1} je váha $(i+1)$. věci a c_{i+1} je cena $(i+1)$. věci.

Vybíráme zdali získáme lepší cenu při této hmotnosti, pokud věc do batohu vložíme či nikoliv. Na závěr získáme nejlepší kombinaci v buňce v n-tém sloupci v M-tém řádků.

2.4 FPTAS

Tento algoritmus je založený na dynamickém programování dekompozicí podle ceny. Obsahuje konstantu ϵ $(0 < \epsilon \leq 1)$, která symbolizuje hodnotu přípustné chyby. Na základě výpočtu $b = \lfloor \log_2 \frac{\epsilon C_M}{N} \rfloor$ vypočítáme počet bitů, které můžeme v ceně věci zanedbat. $Cm = max(c_1, \ldots, c_N)$. Po zanedbání těchto bitů v cenách všech věcí aplikujeme metodu dynamického programování dekompozicí podle ceny pro získání řešení. Toto řešení poté posuneme o b bitů zpět a máme výsledek s jistou mírou nepřesnosti. Tím, že jsme zmenšili cenu věcí jsme také zmenšili tabulku potřebnou k výpočtu řešení a to na $(n+1)*(\frac{P}{2^b})$. Díky tomu je výpočet řešení rychlejší, ale výsledek je nepřesný. Zanedbáváním bitů v cenách věcí získáváme algoritmus polynomiální.

3 Popis postupu řešení

V první řadě jsem se zaměřil na řešení problémů batohu metodou větví a hranic, neboť hrubou sílu jsem měl již naimplementovanou z minulého úkolů. Metodu větví a hranic jsem vytvořil částečným předěláním hrubé síly, kde jsem zaměnil generování kombinací ve for cyklu za rekurzi, přidal jsem proměnné a zapamatování dosavadně nejlepší vygenerované ceny a do rekurze přidal podmíněné volání podproblémů. Další metodu, kterou jsem implementoval byla metoda dynamického programování. Nejdříve jsem začal dekompozicí podle ceny a poté jsem přidal i dekompozici podle váhy. FPTAS algoritmus jsem vytvořil jako rozšíření dynamického programování dekompozicí podle ceny. Na závěr jsem provedl měření doby běhu implementovaných metod a zabýval jsem se srovnáním maximální naměřené chyby s předpokládanou chybou u FPTAS algoritmu.

4 Kostra algotritmu

Modul, který řeší problém batohu přijímá 5 parametrů:

1. datový soubor s počátečními hodnotami,
2. jméno algoritmu, kterým bude problém řešit,
3. zdali chceme, aby počítal čas, který algoritmus stráví na CPU,
4. kolik iterací daného algoritmu budeme chtít pro výpočet průměrného času,
5. volitelný parametr, který určuje předpokládanou chybu u FPTAS.

Po přečtení vstupních parametrů se načtou data ze souboru do pole. Podle naší volby proběhne jeden z 5 algoritmů. Pro výběr algoritmu jsem implementoval factory návrhový vzor. Výstupy tohoto algoritmu jsou dále zapsány do souboru.

4.1 Problém batohu hrubou silou

Pro vygenerování všech možných kombinací výskytu položky v batohu jsem použil for cyklus, který má 2^n iterací. V každé iteraci se do pole primitivního typu int zapíše jednička na místa, která jsou reprezentována binární podobou čísla probíhající iterace. Takto vzniklou kombinaci položek otestuji, zdali nepřesáhla maximální nosnost batohu. Pokud ano, pokračuji další kombinací. Pokud ne, porovnám aktuální hodnotu položek s největší předchozí hodnotou. Pokud je větší, zapamatuji si tuto novou maximální hodnotu a kombinaci položek, které ji vytváří. Po skončení for cyklu získám nejlepší kombinaci a její hodnotu.

4.2 Problém batohu metodou větví a hranic

Možné kombinace generuji pomocí rekurze, kdy první volání metody je na všechny věci, které chceme do batohu vložit. Současně spočítám maximální

cenu, kterou mohou věci generovat. V každém volání rekurze se poté rozhodnu zdali se aktuálně zvolená věc ještě vejde do batohu (ořezávání shora). Pokud ano, přičtu cenu věci k aktuální ceně věcí v batohu, odečtu její hmotnost od zbývající volné kapacity batohu a zavolám stejnou metodu nyní však bez přidávané věci a s horní hranicí ceny, kterou mohou zbývající věci ještě generovat. Závěrem zavolám stejnou metodu bez přidané aktuální věci. Na samotném začátku rekurze zkontroluji jestli horní hranice ceny, kterou mohou věci vygenerovat společně s aktuální hodnotou věcí v batohu je větší nebo rovna největší dosavadně získané hodnotě (ořezávání zdola). Pokud není ukončuji tuto větev. Po skončení rekurze je největší cena uložena v největší dosavadně získané hodnotě.

4.3 Problém batohu metodou dynamického programování

4.3.1 Dekompozice podle ceny

Základem bylo vytvoření požadované tabulky. Buňka tabulky je typu `KnapsackCell`, která obsahuje cenu a kombinaci prvků, které tuto cenu tvoří. Tabulku vyplním podle pravidel uvedených v podkapitole 2.3.1. Metoda pro výpis výsledků mi poté projde poslední sloupec tabulky shora a najde buňku s nejlepším řešením. Nejlepší cena odpovídá řádku buňky a kombinace je uložena v buňce.

4.3.2 Dekompozice podle váhy

Aplikováním podmínek pro vytváření tabulky dekompozicí podle váhy získám vyplněnou tabulku. Řešení se poté nachází v posledním sloupcí nahoře, kde buňka obsahuje nejlepší cenu i kombinaci.

4.4 Problém batohu pomocí algoritmu FPTAS

Pro implementaci tohoto algoritmu jsem zdědil třídu `KnapsackProblemSolverFPTAS` od třídy s metodou dynamického programovaní dekompozicí podle ceny. Před spuštěním algoritmu nad instancí batohu projdu věci, které mám vložit do batohu a všem bitově posunu jejich cenu o b míst doprava. Metoda dynamického programování mi poté vrátí řešení, jehož cenu posunu o b míst doleva.

4.5 Výpočet maximální chyby

Pro měření maximální chyby jsem vytvořil separátní modul, který má na vstupu dva parametry:

1. soubor s naměřenými výsledky,
2. soubor se správnými výsledky.

Algoritmus přečte oba soubory a vytvoří pole relativních chyb ($\frac{opt(n)-apx(n)}{opt(n)}$) veškerých instancí obsažených v souboru. Následně vypočítá průměrnou relativní chybu, maximální relativní chybu a tyto hodnoty zapíše do souboru. Pro spouštění modulu se všemi soubory jsem vytvořil skript.

4.6 Měření času

Abych co možná nejlépe změřil čas, který algoritmus stráví na CPU, použil jsem java třídu ThreadMXBean. Pokud chci změřit čas, dám modulu na vstupu parametr kolikrát chci daný algoritmus zopakovat. Po každé iteraci změřený čas zapíšu do pole a po skončení všech iterací hodnoty v poli zprůměruji. Výsledek zapíši do souboru.

5 Naměřené výsledky

Pro implementaci jsem využil programovací jazyk Java. Testy běžely na operačním systému macOS.

5.1 Srovnání výpočetního času algoritmů

Z naměřených výsledků zobrazených v tabulce 1 je vidět, že hrubá síla je výrazněji pomalejší nežli všechny ostatní metody. Výrazného zlepšení jsme dosáhli pomocí B&B metody. Nicméně metoda dynamického programování se ukázala jako efektivnější. Pokud nám nevadí ztráta přesnosti můžeme použít FTPAS algoritmus. Ten pro $\epsilon = 0.5$ beží o více než polovinu kratší čas, nežli tomu bylo u čistého dynamického programovaní.

Počet položek	Hrubá síla	B&B	Dynamické podle ceny	FPTAS $\epsilon = 0.5$
4	1,84 ms	0,99 ms	12,28 ms	3,96 ms
10	2,231 ms	2,01 ms	32,90 ms	10,99 ms
15	103,65 ms	2,64 ms	59,88 ms	21,33 ms
20	6,22 s	5,72 ms	109,97 ms	31,36 ms
22	20,87 s	13,91 ms	121,95 ms	44,44 ms
25	4,91 min	27,32 ms	163,81 ms	49,06 ms
27	22,74 min	48,35 ms	165,96 ms	68,33 ms
30	1,02 h	112,01 ms	211,19 ms	81,97 ms

Table 1: Srovnání doby běhu algoritmů

5.2 FPTAS

5.2.1 Závislost výpočetního času na zvolené přesnosti zobrazení

Rychlost výpočet pomocí algoritmu FPTAS závisí na volbě hodnoty epsilon. Jak můžeme vidět z grafu 1, s rostoucím ϵ klesá doba výpočtu.

5.2.2 Závislost chyby algoritmu na zvolené přesnosti zobrazení

Velikost chyby algoritmu roste s hodnotou zvoleného epsilon. Jak je vidět z tabulky 2, maximální naměřená chyba nikdy nepřesahuje teoreticky předpokládanou chybu. V mnoha případech je dokonce několikanásobně menší.

5.3 Srovnání různých dekompozic v dynamickém programování

Na základě výsledků zobrazených na grafu 2 lze lehce dojít k myšlence, že dekompozice podle váhy je lepší. Nicméně je třeba brát v úvahu, že vše záleží na vstupních datech. V případě, že by maximální kapacita batohu byla větší nežli suma všech cen věcí, vyplatí se zvolit dekompozici podle ceny.

6 Závěr

Exponenciální řešení problému batohu je pro větší instance špatně použitelný, díky dlouhé době běhu výpočtu. Sice se B&B metoda ukázala jako významným vylepšením, nicméně v porovnáním s dynamickým programování stále zaostává a to hlavně díky tomu, že je B&B stále exponenciální. Rychlejší metodou bylo dynamické programování. Co se týče výběru mezi dynamickým programováním dekompozicí podle ceny nebo váhy, nejdříve bych zjistil, která dekompozice bude pro mou instanci rychlejší. V případě, že by mi nešlo o naprosto správný výsledek, volil bych algoritmus FPTAS s takovou předpokládanou chybou, která by vyhovovala mým potřebám.

7 Přílohy

	Maximální chyba u algoritmu FPTAS			
Počet položek	$\epsilon = 0.1$	$\epsilon = 0.25$	$\epsilon = 0.5$	$\epsilon = 0.75$
4	2.3622 %	5.7325 %	13.3758 %	28.6624 %
10	0.6319 %	2.0173 %	4.3228 %	9.1892 %
15	0.00 %	1.7619 %	4.5045 %	4.5045 %
20	0.00 %	0.5594 %	1.5020 %	3.5048 %
22	0.00 %	0.5764 %	1.5801 %	3.4594 %
25	0.00 %	0.5780 %	1.4249 %	1.4249 %
27	0.00 %	0.5922 %	1.4799 %	1.4799 %
30	0.00 %	0.4735 %	1.4795 %	1.7544 %
32	0.00 %	0.2227 %	0.5803 %	1.4829 %
35	0.00 %	0.2004 %	0.6012 %	1.3878 %
37	0.00 %	0.1821 %	0.5405 %	1.6852 %
40	0.00 %	0.1669 %	0.5008 %	1.4370 %

Table 2: Procentuální srovnání maximálních naměřených chyb v závislosti na zvolené přesnosti zobrazení

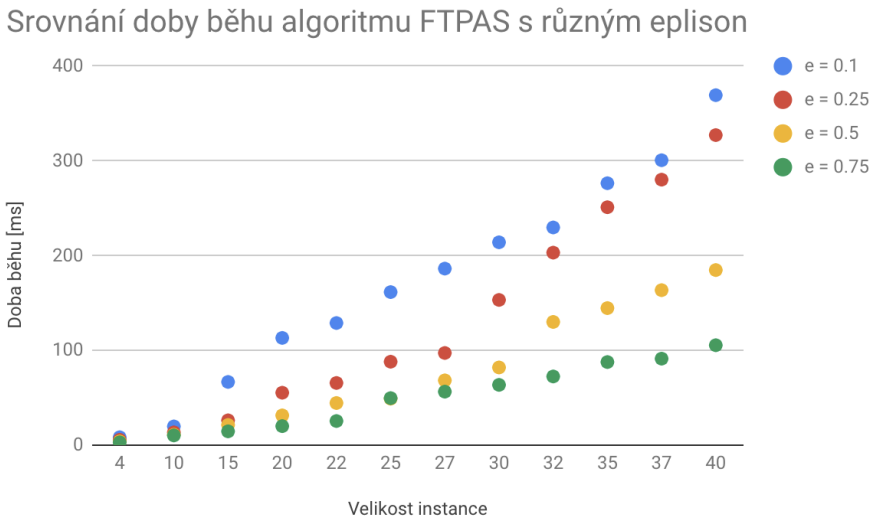

Figure 1: Srovnání doby běhu algoritmu FTPAS s různým eplison

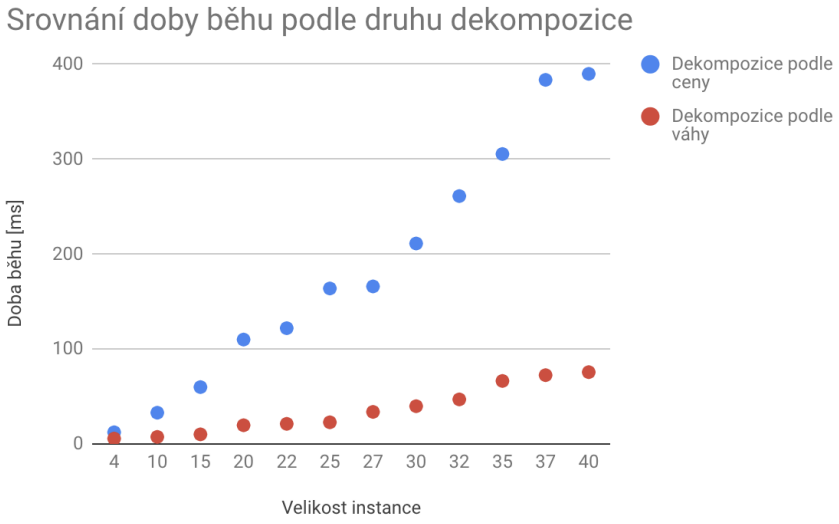

Figure 2: Srovnání doby běhu podle druhu dekompozice u dynamického programování