Semestrální práce z předmětu KIV/PRO
Bin Packing problem

Dudchuk Olesya

18. listopadu 2018

Obsah

Kapitola 1

Úvod

1.1 Zadání

Bin Packing Problem (dále jen BPP) slouží obecně k minimalizaci nevyužitého místa. Jeho řešení má mnohá použití. Nejčastěji slouží k optimalizaci uložení zboží v kontejneru a v následném uložení těchto kontejnerů v nákladním prostoru dopravních prostředků.

BPP patří ke standardním optimalizačním problémům. Je to kombinatorický NP-těžký problém, kde je množina předmětů o různých velikostech uložena do co nejmenšího počtu kontejnerů (binů).

1.2 Vstupy

Jako vstupy pro řešení BPP jsou definovány množina předmětů a množina homogenních kontejnerů.

Množina obdélníkových předmětů R_i o určité šířce w_i a výšce h_i.

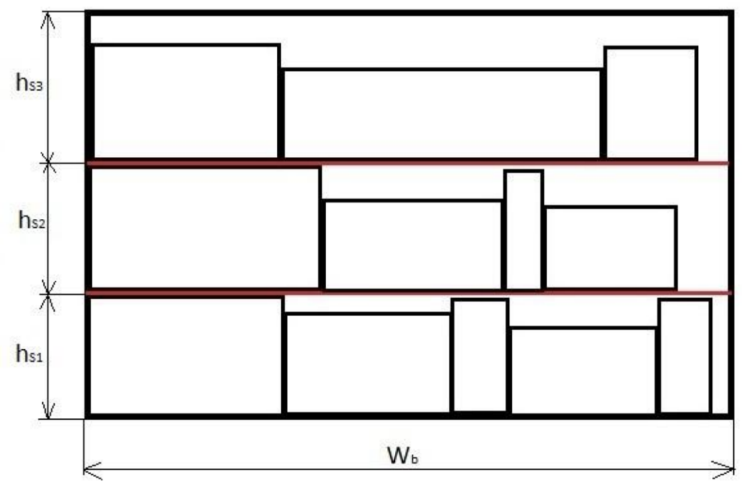
Množina homogenních obdélníkových kontejnerů. Všechny kontejnery mají stejné rozměry: šířku W_b, výšku H_b.

Kapitola 2

Existující metody

2.1 Kratky seznam existujících metod

2.1.1 Policove algoritmy



Obrázek 2.1: Uložení předmětů pomocí policového algoritmu.

Hodně velká čast literatury ohledně BPP používá přibližné algoritmy. Pro důkladnou análizu použijemé průzkum Coffmana, Gareyho a Johnsona (1984)[2], který má bibliografie s vic než odkazy na literaturu, která tady bude použitá.

Next-Fit (NF)

Nejjednodušší přistup k řešení daného problemu je Next-Fit (NF) algoritm. První předmět je vložen do kontejneru čislo 1. Předměty 2,... ,n potom se ukladají nasledujicím zpusobem: algoritmus se pokusí předmět uložit na výšku (pokud se nejedná o úplně první předmět, ten je uložen vždy na šířku), pokud nelze předmět takto uložit, otočí ho a pokusí se ho uložit na šířku. Pokud nelze ani to, zavírá daný kontejner a otevírá nový. Pokud není dostatek místa pro novou polici (s předmětem uloženým na šířku) algoritmus končí. Časova náročnost algoritmu je O(n). Ale výsledky tohoto algoritmu jsou hodně vzdáleny od výsledků optimálního uložení.

First-Fit (FF)

U metody FF je předmět vždy ukládán do první (nejspodnější) polici, do které předmět může být uložen. U této metody předmět, který lze uložit do některé z uzavřených polic, šetří místo na polici otevřené.

Best Width Fit (BWF)

Tato metoda vychází z metody FF. Algoritmus nejprve projde všechny police, a zjistí, do kterých by mohl být předmět uložen. U tohoto algoritmu bude z těchto polic následně vybrána ta police, ve které bude po uložení předmětu šířka zbývajícího volného místa nejmenší.

Best Height Fit (BHF)

Protože hrany oddělující police jsou přímé čáry, při uložení předmětu o menší výšce, než je výška police, vzniká pruh nevyužitého místa mezi horní hranou předmětu a stropem police. Algoritmus nejprve projde všechny police, a zjistí, do kterých by mohl být předmět uložen. Pro minimalizaci zmíněného nevyužitého místa uloží algoritmus předmět do takové police, kde bude rozdíl výšek $h_s - h$ minimální.

Best Area Fit (BAF)

U BHF algoritmu může nastat situace, kdy bude shodný rozdíl výšek h_s-h při uložení předmětu na výšku a na šířku (v různých policích). Pokud tato situace nastane, vynásobí se tento rozdíl aktuální šířkou předmětu. Tento součin je roven ploše mezi horní hranou předmětu a stropem police. Předmět bude uložen tak, aby byla tato plocha minimální.

Worst Width Fit (WWF)

Zatímco BWF se snaží zaplnit šířku každé police, co nejvíce je to možné. WWF se snaží o úplný opak. Snaží se na každé polici udržet co největší možnou šířku nevyužitého místa. Algoritmy BWF a WWF jsou si úplnými opaky, přesto není možné o jednom tvrdit, že by jeho uložení předmětů bylo lepší než u toho druhého. U algoritmu WWF je navíc přijato pravidlo, že pokud je ukládán předmět o šířce w a je nalezena police, ve které zbývá právě šířka w volného místa, je okamžitě vybrána tato police a předmět je do ní uložen. Podle stejného vzoru lze definovat i algoritmy Worst Height Fit a Worst Area Fit. Ale protože policové algoritmy nevyužívají místo mezi horní stranou každého předmětu a stropem police, zvyšování tohoto rozdílu by vedlo k maximalizaci nevyužitého místa, a tím pádem jsou tyto metody suboptimální.

Kapitola 3

Zvolené řešení

Martello-Toth algoritm

Martello a Toth (1989)[1] navrhlí algoritm, MTP, který se bazoval na "first-fit decreasing"strategii. Původně předměty jsou poskladané sestupně. Algoritm indexuje předměty v tom pořadí, v jakym oni jsou inicializované. V každém rozhodovacím uzlu, první volný (tj. největší) předmět je vložen do proveditelně inicializovanýhou kontejneru a zaroven i do novýho kontejneru. V každém dalším kroku:
(a) volají se procedury L_2 a potom L_3 které zkoumají daný uzel, a redukují problem;
(b) Kdýž nedochazí k vylepšení, aplikujou se algoritmy FF, BHF a WWF (viz. sekce 2.1. Existujicí metody) pro vylepšení dosud nejlepšiho možnýho řešení. WWF algoritm seřadí předměty sestupně a vloží každy předmět do největšiho zbylého kontejneru (pokud takový existuje). Backtracking potom zaručuje odstranění aktualního prvku $i*$ z aktualního kontejneru $j*$ a vložení tohoto prvku do dalšího proveditelnýho kontejneru (backtracking nastavá pouze v připadě, když předmět $i*$ je inicializovan kontejnerém $j*$, protože inicializace $i*+1$ předmětu $j*$ kontejnerém ve vysledku vytvoří identickou situaci). Pokud z je hodnota dosud nejlepšího možnýho řešení, kdykoliv se má použit backtracking, tak se použiva na poslední předmět vložený do kontejneru s indexem ne větším než $z - 2$.

Nasledně, mezi každým rozhodovacím uzlem použiva se tzv. dominantní kritérium. Když aktualní předmět $j*$ je vložen do kontejneru $i*$, u kterého zbytková kapacita $\overline{c_{x*}}$ je menší než $w_{j^*} + w_n$, toto vložení má přednost před všemí ostatnimi vloženimi do kontejneru $i*$, předmětů $j > j*$, ktere nedovolí žadné další vkladaní. Proto takovýho typu vkladání zavírá kontejner $i*$, ve smyslu, že po backtrackingu na prvek $j*$, žadny prvek $j \in [k > j* : W_k + W_n > \overline{c_{i*}}]$ už nejde vložit. Proto že v každém rozhodovacím uzlu zbytková kapacita každého kontejneru je jiná, funkci vypočtů nižších hranic L_2 a L_3 musí s tím počitat.

3.1 Pseudokod algoritmu MTP

Pseudokod algoritmu MTP.[1]

procedure MT1:
input: $n, c, (p_j), (w_j)$;
output: $z, (x_j)$;
begin
1. [initialize]
 $z := 0$;
 $\hat{z} := 0$;
 $\hat{c} := c$;
 $p_{n+1} := 0$;
 $w_{n+1} := +\infty$;
 for $k := 1$ **to** n **do** $\hat{x}_k := 0$;
 compute the upper bound $U = U_2$ on the optimal solution value;
 $\overline{w}_1 := 0$;
 $\overline{p}_1 := 0$;
 $\overline{r}_1 := 1$;
 $\tilde{r} := n$;
 for $k := n$ **to** 1 **step** -1 **do** compute $m_k = \min\{w_i : i > k\}$;
 $j := 1$;
2. [build a new current solution]
 while $w_j > \hat{c}$ **do**
  if $z \geq \hat{z} + \lfloor \hat{c} p_{j+1}/w_{j+1} \rfloor$ **then go to** 5 **else** $j := j + 1$;
 find $r = \min\{i : \overline{w}_j + \sum_{k=\overline{r}_j}^{i} w_k > \hat{c}\}$;
 $p' := \overline{p}_j + \sum_{k=\overline{r}_j}^{r-1} p_k$;
 $w' := \overline{w}_j + \sum_{k=\overline{r}_j}^{r-1} w_k$;
 if $r \leq n$ **then** $u := \max(\lfloor (\hat{c} - w')p_{r+1}/w_{r+1} \rfloor,$
  $\lfloor p_r - (w_r - (\hat{c} - w'))p_{r-1}/w_{r-1} \rfloor)$
 else $u := 0$;
 if $z \geq \hat{z} + p' + u$ **then go to** 5;
 if $u = 0$ **then go to** 4;
3. [save the current solution]
 $\hat{c} := \hat{c} - w'$;
 $\hat{z} := \hat{z} + p'$;
 for $k := j$ **to** $r - 1$ **do** $\hat{x}_k := 1$;
 $\overline{w}_j := w'$;
 $\overline{p}_j := p'$;
 $\overline{r}_j := r$;
 for $k := j + 1$ **to** $r - 1$ **do**
  begin
   $\overline{w}_k := \overline{w}_{k-1} - w_{k-1}$;
   $\overline{p}_k := \overline{p}_{k-1} - p_{k-1}$;
   $\overline{r}_k := r$

end;
for $k := r$ **to** $\tilde{r}$ **do**
begin
$\overline{w}_k := 0$;
$\overline{p}_k := 0$;
$\overline{r}_k := k$
end;
$\tilde{r} := r - 1$;
$j := r + 1$;
if $\hat{c} \geq m_{j-1}$ **then go to** 2;
if $z \geq \hat{z}$ **then go to** 5;
$p' := 0$;

4. [update the best solution so far]
$z := \hat{z} + p'$;
for $k := 1$ **to** $j - 1$ **do** $x_k := \hat{x}_k$;
for $k := j$ **to** $r - 1$ **do** $x_k := 1$;
for $k := r$ **to** n **do** $x_k := 0$;
if $z = U$ **then return** ;

5. [backtrack]
find $i = \max\ \{k < j : \hat{x}_k = 1\}$;
if no such i **then return**;
$\hat{c} := \hat{c} + w_i$;
$\hat{z} := \hat{z} - p_i$;
$\hat{x}_i := 0$;
$j := i + 1$;
if $\hat{c} - w_i \geq m_i$ **then go to** 2;
$j := i$;
$h := i$;

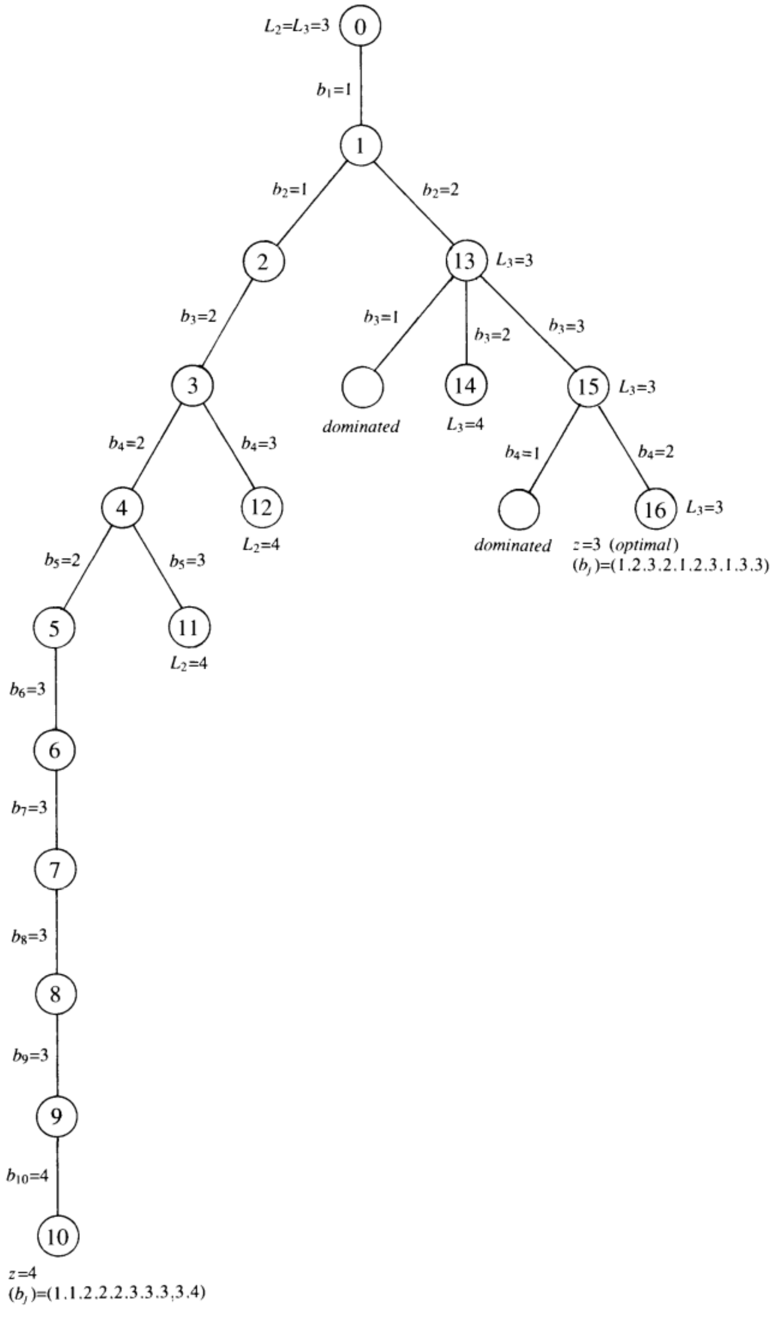
6. [try to replace item i with item h]
$h := h + 1$;
if $z \geq \hat{z} + \lfloor \hat{c} p_h / w_h \rfloor$ **then go to** 5;
if $w_h = w_i$ **then go to** 6;
if $w_h > w_i$ **then**
begin
if $w_h > \hat{c}$ or $z \geq \hat{z} + p_h$ **then go to** 6;
$z := \hat{z} + p_h$;
for $k := 1$ **to** n **do** $x_k := \hat{x}_k$;
$x_h := 1$;
if $z = U$ **then return**;
$i := h$;
go to 6
end
else
begin
if $\hat{c} - w_h < m_h$ **then go to** 6;
$\hat{c} := \hat{c} - w_h$;
$\hat{z} := \hat{z} + p_h$;
$\hat{x}_h := 1$;
$j := h + 1$;

```
        w̄_h := w_h;
        p̄_h := p_h;
        r̄_h := h + 1;
        for k := h + 1 to r̃ do
            begin
                w̄_k := 0;
                p̄_k := 0;
                r̄_k := k
            end;
        r̃ := h;
        go to 2
    end
end.
```

3.2 Přiklad

Přiklad

Vezmemé instanci MTP definovanou nasledujícími vlastnostmi:
n = 10;
$(w_j) = (49, 41, 34, 33, 29, 26, 26, 22, 20, 19)$;
c = 100;
definujemé proveditlné řešení přes vektor (b_j), kde
$b_j =$ kontejner do kterého je vložený předmět $j (j = 1...n)$
z =4;
$(w_j) = (1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4)$;
Obrazek 2.2 ilustruje rozhodovací strom, produkovaný algoritmem MTP. Zpočatku, veškeré vypočty nižších hranic davají hodnotu 3, zatim co FF algoritm dává první proveditelné řešení, odpovidající rozhodovacím uzlum 1 až 10.



Obrázek 3.1: Rozhodovací strom

Kapitola 4

Experimenty a výsledky

V teto sekci zkoumamé pruměrnou efektivitu ve vypočtech. Program byl naprogramován v C.
Třida 1: w_j rovnoměrně náhodný v rorzmezí [1,100]
Třida 2: w_j rovnoměrně náhodný v rorzmezí [20,100]
Třida 3: w_j rovnoměrně náhodný v rorzmezí [50,100]

		c = 100		c = 120		c = 150	
Class	n	Time	Nodes	Time	Nodes	Time	Nodes
1	50	0.006	0	0.005	0	0.096	11
	100	0.012	1	15.022(17)	3561	0.156	29
	200	5.391	1114	0.062	6	0.140	10
	500	10.236	2805	10.340	887	2.124	28
	1000	20.206(16)	2686	6.596	244	8.958	44
2	50	0.005	0	0.008	1	0.183	61
	100	0.012	1	0.030	9	26.599(15)	4275
	200	0.047	11	0.073	18	69.438(7)	8685
	500	0.127	28	10.062	1663	—	—
	1000	15.524(17)	3896	30.148(14)	4774	—	—
3	50	0.005	0	0.005	0	0.005	0
	100	0.010	0	0.010	0	0.010	0
	200	0.019	0	0.020	0	0.018	0
	500	0.049	0	0.050	0	0.051	0
	1000	0.102	0	0.104	0	0.105	0

Obrázek 4.1: Tabulka s vysledky

Kapitola 5

Závěr

Všichni instance třidy 3 byly vyřešené hodně rychle, protože metodaL_3 vždycky vracela optimalní řešení. Pro třidu 1 výsledky jsou velice uspokojivé, až na nekteré vyjimky. Ve třidě 2, chování algoritmu bylo lepší než ve třidě 1 prro c = 100, skoro stejné pro c = 120, a jednoznačně horší pro c = 150. Ale pouze v jedinečných připadech optimalní řešení bylo nalezeno pomocí policových algoritmu.

Literatura

[1] Silvano Martello and Paolo Toth: KNAPSACK PROBLEMS,Algorithms and Computer Implementations
http://www.or.deis.unibo.it/kp/KnapsackProblems.pdf

[2] E. C. Coffman Jr, M. R. Garey and D. S. Johnson : Approximation algo thms for bin-packing-an updated survey, In Algorithm Design for Computer System Design (
http://www.labri.fr/perso/eyraud/pmwiki/uploads/Main/BinPackingSurvey.pdf