Návrh hybridného algoritmus umiestňovania dvojrozmerných útvarov na prepravnú plochu

Proposition of hybrid algorithm for solving two dimensional strip packing problem

Ján Pittner

Abstract:

Proposed article deals with proposition of hybrid algorithm designed to solve strip packing problem in two dimensional space. After mentioning the common algorithms, that are used to solve strip packing problem, we propose parallel hybrid algorithm, that reduces time to solve selected problems.Currently there are several existing heuristic algorithms, namely genetic algorithms, that are achieving very good results, however based on previous analysis, their weak point is usually in generating initial solutions, or any operation like mutation or crossover that works with deterministic algorithms. This problem together with genetic algorithm that we are currently working on was the main factor of creating this article. Although deterministic algorithms like those mentioned in this paper seem to be throughoutly mapped, in our opinion there is still place for additional research. Proposed algorithm is not perfect, it has it's flaws, however for purpose of our future genetic algorithm, it is suitable and useful.

Keywords:

object placing, hybrid algorithm, parallelization, optimalisation, two dimensional space

ACM classification: D1.3, G.0, G.1.0, G.4

Úvod

V súčasnej dobe otvorenej ekonomiky, keď sa na trhu vyskytuje veľký počet subjektov rastie potreba zefektívňovania jednotlivých procesov, pretože len efektívne nastavené procesy môžu zaručiť naplnenie cieľa, to jest maximalizácie zisku. Nesprávne nastavené procesy môžu naopak spôsobovať spomedzi mnohého aj zbytočné časové straty a tým pádom klesá aj objem zisku.

Na trhu tovarov sa spravidla nachádzajú tri kategórie subjetkov. Prvou kategóriu sú firmy(maloobchody) ponúkajúce výrobky zákazníkom, teda druhej kategórii. Firmy zväčša sprostredkúvajú predaj tovarov zákazníkom, pričom zisk dosahujú stanovením si určitej marže. Treťou kategóriou sú napokon výrobcovia tovarov, pričom spravidla figurujú ako dodávavatelia. Práca sa zaoberá dodávateľským reťazcom a jeho optimalizáciou, priblížme si teda tento proces.

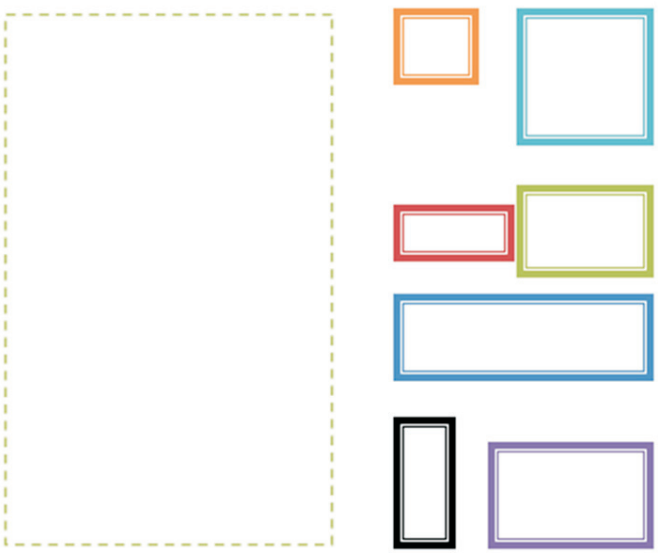
Dodávateľ je fyzická alebo právnická osoba, ktorá ma odberateľovi jednorázovo dodať, alebo mu dodáva za odplatu v dohodnutom čase alebo lehotách určitý výrobok. Zvyčajne sa nestáva, že by odberateľ zakúpil len jednu jednotku výrobku, respektíve výrobok, ktorý by sa skladal len z jednej časti.

Majme fast-food, ktorý od svojho dodávateľa odoberá potraviny a nápoje, ktoré predáva zákazníkom. Dodávka sa uskutočnuje raz mesačne, pričom potraviny sú uskladňované v chladiacich boxoch, aby vydržali dlhšie čerstvé. Fast-food má vysokú návštevnosť a preto aj dodávky bývajú zväčša objemnejšie. Požiadavkou odberateľa je teda včasná dodávka tovarov, aby nenastala situácia, v ktorej fast-food nie je schopný ponúknuť zákazníkom svoje služby a teda situácia ktorej dôsledkom je pokles zisku.

Dodávateľ na druhej strane síce z objednávky vie aké tovary bude musieť dodať, avšak pri maximalizácii zisku musí pozerať aj na prepravné náklady[1)]. Preto sa snaží pri dodávkach tovaru (či už priamo do predajnej prevádzky, alebo do skladu) pokryť čo najviac miesta v prepravnom priestore. Čím viac tovarov pojme jedna jednotka prepravného zariadenia, tým menej prepravných jednotiek musí expedovať, čím šetrí náklady na dopravu. Predpokladajme, že v tomto prípade dodávateľ využíva špedičnú kamiónovú spoločnosť a prenajíma si kontajnery, do ktorých jeho zamestnanci ukladajú tovary určené na expedíciu. Tovary sú balené z hygienických dôvodov v plastových foliách, ktoré sú uložené v kartónových boxoch.

Dodávateľ teda rieši problém optimalizácie prepravného miesta v kontajneroch (prepravných plochách). Problém vzniká vtedy, keď nie sú všetky boxy rovnakej veľkosti. V dôsledku krehkosti tovarov, predpokladajme, že boxy nemožno umiestňovať na seba, ale len vedla seba.

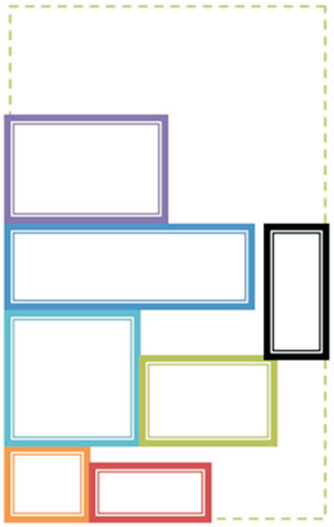
Máme teda prepravnú plochu a boxy, ktoré na ňu treba vhodne umiestniť:



Obrázok č. 1: *Prepravná plocha a boxy – zadanie*

Na obrázku 2 vidíme jedno z možných riešení. Samozrejme, aby sme vedeli ohodnotiť toto riešenie, naskytá sa otázka, akým algoritmom bolo vypočítané? Existuje mnoho variantov na výpočet tohto typu problému. Postupne prejdeme najznámejšie a spolu s grafickou ukážkou si osvojíme princípy ich fungovania.

1) Dodávky materiálu tvoria rozsiahlu časť logistiky, avšak v tejto práci sa zameriavame len na dodávku z miesta A do miesta B z hľadiska optimalizácie prepravného miesta.



Obrázok č. 2: *Prepravná plocha a boxy – možné riešenie*

Ako teda stanoviť vhodnosť jednotlivých algoritmov? V nasledujúcej kapitole si exaktne vysvetlíme fungovanie jednotlivých algoritmov, vyberieme tie, ktoré sú z hľadiska efektívnosti a času vhodné a pokúsime sa ich včleniť do hybridného algoritmu.

1. Definícia problému

Tento typ problému sa vo všeobecnosti nazýva problém balenia koša, alebo problém balenia pásu a je NP-úplným problémom. Formálna definícia hovorí, majme objekty rôznych objemov, ktoré musia byť zabalené do konečného počtu košov, ktoré majú určitú kapacitu tak, aby sme minimalizovali počet potrebných košov. Existujú mnohé variácie tohto problému ako napríklad 2D balenie, lineárne balenie, balenie podľa váhy, balenie podľa ceny atď. Praktické využitie je potom napríklad plnenie kontajnerov, nakladanie kamiónov obmedzených váhovou kapacitou, rozmiestňovanie technológií na mikroprocesor atď.

Z predchádzajúceho výkladu teda môžeme očakávať, že máme zadanú ohraničenú oblasť O a množinu útvarov S_i kde $i=1,2,3....n$ pričom n je dostatočne veľké. Úlohou je potom rozmiestniť útvary S_i tak, aby bola minimalizovaná plocha, potrebná na ich umiestnenie.

2. Algoritmy riešiace problém balenia pásu

Problém hľadania optimálneho riešenia pre problém balenia pásu je NP-úplný, tým pádom sa výskum zameriava hlavne na vývoj aproximačných algoritmov. Aproximačné algoritmy nájdu takmer optimálne riešenia, ale nezaručujú nájdenie optima pre každú skupinu dát.

Aproximačné, alebo aj heuristické algoritmy však implicitne využívajú deterministické algoritmy pre potreby generovania inicializačných riešení, resp. pri operáciách ako je mutácia alebo prekríženie. Pri analýze existujúcich aproximačných algoritmov sme zistili, že hoci v súčasnosti existujú takmer dokonalé algoritmy, mnohé z nich sa sústredia na nastavovanie evolučných parametrov, vhodné metódy selekcie a opomíňajú základné algoritmy, ktoré predstavujú podstatnú časť záťaže procesora.[2)] Motiváciou tohto článku bola práve táto oblasť, pretože optimalizácia týchto fundamentálnych princípov spôsobí podľa nášho názoru nárast výkonnosti aproximačného algoritmu.

2) KRAUSPE, Kamil. 2011. An evolutionary algorithm for mixed postman problem. In *Mezinárodní Baťova konference pro doktorandy a mladé vědecké pracovníky : [recenzovaný zborník] : 7. ročník : 12. dubna 2011* [elektronický zdroj]. – Zlín : Univerzita Tomáše Bati ve Zlíně, 2011. ISBN 978-80-7454-013-4, s. [1-11].

Väčšina algoritmov funguje tak, že umiestňuje útvary na pás využívajúc pri tom jeden z piatich prístupov.

- algoritmus "Dolu vlavo"

  Tento typ algoritmu sa pri hľadaní pozície pre útvary snaží nastaviť pozíciu čo najbližšie spodku kontajnera a potom čo najviac naľavo, bez toho aby súčasný útvar prekryl už uložený útvar. Neplatí tu podmienka zoradenia útvarov pred ukladaním.

- úrovňovo orientované algoritmy

  Algoritmus začína zoradením útvarov podľa výšky (od najvyššieho po najnižší). Následne sa uskutočnuje balenie na rôznych úrovniach, ktoré určujú pozíciu spodku útvaru. Prvú úroveň predstavuje spodok kontajnera a nasledovné úrovňe sú definované výškou najvyššieho útvaru, ktorý bol umiestnený na predchádzajúcej úrovni.

- algoritmus delenia

  Pre vysvetlenie uvedieme porovnanie s úrovňovo orientovanými algoritmami. Úrovňovo orientované algoritmy sú v podstate podmnožinou algoritmov delenia, keďže delia kontajner horizontálne na bloky určitej šírky. Pod pojmom algoritmus delenia máme na mysli potom algoritmus, ktorý rozdelí vertikálne kontajner na menšie časti, v závislosti od šírky obdĺžnikov. Algoritmus vždy začína zoradením útvarov podľa šírky pred ukladaním.

- algoritmus regálov

  Tieto algoritmy sú modifikáciou úrovňovo orientovaných algoritmov, ktoré sa vyhýbajú problému zoradenia zoznamu útvarov pred ich ukladaním. Pri stanovovaní nasledovnej úrovňe však narozdiel od úrovňovo orientovaných algoritmov, kde je spodok nasledovnej úrovne určený výškou najvyššieho útvaru na súčasnej úrovni, sa pri algoritme regálov stanovuje ich výška parametrom *r,* pre ktorý platí :

  $$0 < r < 1$$

- hybridné algoritmy

  Tieto špeciálne typy algoritmov, využívajú dva alebo viac typov algoritmov popísaných vyššie v kombinácii s heuristickými algoritmami. Môžu, ale nemusia zahŕňať zoradovanie útvarov pred samotným ukladaním.

2.1 Podriadené algoritmy (The Slave Algorithm)

Podriadený algoritmus je prístup využívaný na rozhodnutie, do ktorého regálu sa má útvar umiestniť v prípade, že boli všetky regály stanovené.

Pre úrovňovo orientované algoritmy, ktoré som si v projekte zvolil ďalej skúmať, má každá úroveň (teda regál) v koši rovnakú šírku *C.* Tým pádom sa po vypočítaní úrovne stáva rozhodnutie typom jedno-rozmerného plnenia nádoby. To znamená, že počet používaných úrovní zodpovedá počtu jednorozmernej nádoby kapacity *C.* Ďalej je potom tento jednorozmerný problém riešený podriadenými algoritmami. Existuje množstvo jednorozmerných algoritmov, ktoré môžu byť využité ako podriadené algoritmy. Ako príklad zvolme algoritmy Next-fit a First-fit. Všetky algoritmy majú svoje výhody a nevýhody.

Porovnajme teda tieto dva algoritmy tým, že vysvetlíme na akom princípe pracujú.

Next-fit algoritmus umiestňuje každý útvar na najvyššiu úroveň na ktorú sa zmestí. Naproti tomu first-fit algoritmus umiestni každý útvar na najnižšiu úroveň na ktorú sa zmestí. Z hľadiska úspornosti je najčastejšou voľbou práve First-fit algoritmus, pretože umožnuje umiestniť útvary na všetkých úrovniach s cieľom čo najnižšieho umiestnenia, zatiaľ čo next-fit iba umožnuje využiť súčasnú a teda najvyššiu úroveň, pričom spätné umiestňovanie (napríklad pri zistení miesta

pre súčasný útvar o 3 úrovne nižšie) nieje možné. Čo sa týka prípadov použitia je first-fit algoritmus vhodným algoritmom pre umiestňovanie pokiaľ je možné umiestniť všetky útvary do kontajnera pred vykonaním určitej akcie.

Next-fit algoritmus by bol potrebný v prípade, že akcia (napr. práca) začala a súčasne musíme umiestnovať do nádoby ďalšie útvary. Nech je výška nádoby čas a útvary trvania určitých prác, plynutím času sa pomyselný kontajner „zmenšuje“ čiže už nie sme schopní vrátiť sa v čase a naplánovať práce inak, pretože určitý čas už uplynul.

2.2 On-line a off-line algoritmy

Pri zvolení si správneho algoritmu je rozhodujúcim faktorom taktiež to, či musia byť jednotlivé útvary pred umiestnením zoradené. Majme situáciu, v ktorej útvary prichádzajú jeden po druhom. Pri príchode každého útvaru, musíme tomuto útvaru priradiť miesto v nádobe. Až po tejto akcii zistíme parametre nasledujúceho útvaru.

Tomuto typu prostredia hovoríme on-line prostredie. Tým pádom všetky on-line algoritmy musia predpokladať, že nebudú mať žiadnu znalosť o nasledujúcich útvaroch, takže zoradenie pred umiestnovaním nepripadá do úvahy.

Naproti tomu v off-line prostredí je možné zobraziť celý zoznam útvarov dopredu a ľubovoľne ich zoradiť do akéhokoľvek poradia predtým, než sú umiestnené do nádoby. Off-line algoritmy teda predpokladajú znalosť celého problému pred vykonávaním umiestňovania.

On-line algoritmus býva využívaný v situáciách kde sa práce musia vykonávať v určitom poradí. Napríklad jednotlivé predmety môžu byť predmetom rôznych priorít alebo termínov.

Off-line algoritmus býva potom využívaný v situáciách kde na poradí nezáleží. Ako príklad uveďme výrobu kusu oblečenia. Prípad, že z kusu materiálu vystrihneme ako prvý rukáv a až potom golier nemá význam, pretože na poradí nezáleží a až po vystrihnutí všetkých častí môžeme zostrojiť tričko.

Jedným z ďalších faktorov, ktoré môžu zavážiť pri výbere algoritmu je potrebné množstvo pamäte. Off-line algoritmus v tomto prípade zaberá viac pamäťových prostriedkov, keďže pred samotným umiestňovaním triedi.

2.3 Úrovňové a regálové algoritmy

Priblížme si teraz tieto dve skupiny algoritmov, ich funkcionalitu a názornú ukážku. Ako prvé si predstavíme niektoré regálové algoritmy. Regálové algoritmy sú typom úrovňových algoritmov, ktoré nezoraďujú útvary. Preto narozdiel od úrovňových algoritmov sú regálové algoritmy radené do triedy on-line algoritmov. Úrovne sú teda určované nie najvyšším útvarom ale fixnými „regálmi“, ktorých výška je určená parametrom r. Každý nový útvar je potom klasifikovaný podľa jeho výšky. Samotné balenie prebieha ako konštrukcia skupiny „regálov“, pričom útvary podobných výšok sú ukladané na rovnaký regál. Veľkosť regálov má potom tvar r^k, každý útvar má určitú výšku h a je umiestnený do regálu, ktorý má výšku r^k. Tým pádom platí rovnica

$$r^{k+1} \leq h = r^k$$

$$\text{kde } k \in Z$$

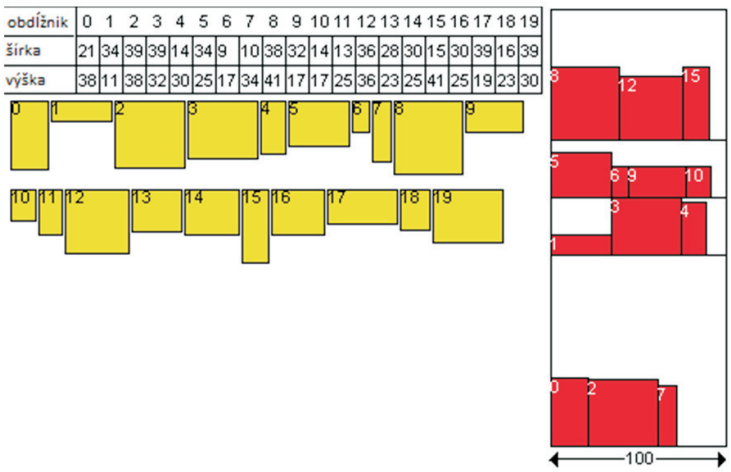
Parameter r je známy vopred a jeho cieľom je obmedziť počet nevyužitého miesta povoleného na každom regáli. Vysvetlime si, ako nastavovanie parametra r ovplyvňuje balenie.

V prípade, že parameter r nadobúda malú hodnotu ($r \approx 0$) je rozsah výšok povolených na každom regáli veľky. Často sa stretávame so situáciou keď máme len malé množstvo útvarov na umiestnenie a nechceme, aby sa na jednotlivých regáloch nachádzal len jeden útvar, s tým, že

regále by zostali neobsadené. Všeobecne pre malé *r* platí, že máme na výber menej možností výberu výšok regálov, pričom každý regál obsahuje útvary s väčším rozsahom výšok.

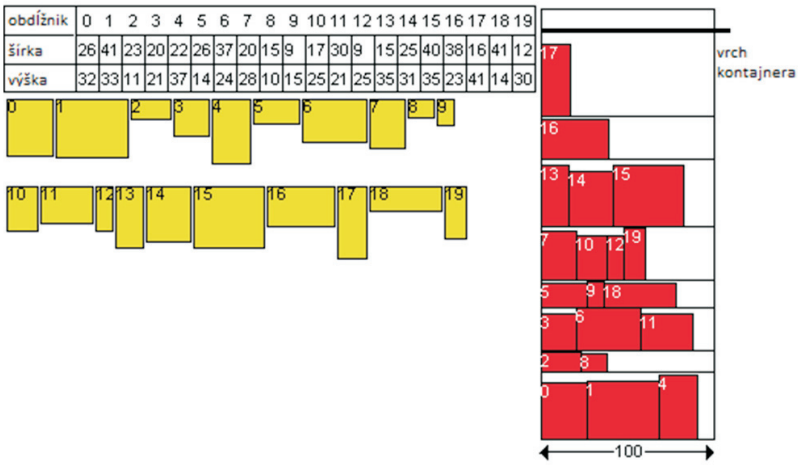
Naopak v prípade, že parameter *r* nadobúda veľkú hodnotu (*r≈1)* bude každý z regálov obsahovat len útvary veľmi podobné svojou výškou. Využitie tohto prípadu je užitočné, v prípade, že útvary v zozname majú veľmi podobné výšky, čize algoritmus musí rozlišovať aj menšie rozdiely vo výškach. Všeobecne platí, že vysoká hodnota parametra *r* spôsobí širší výber výšok regálov s útvarmi veľmi podobných výšok.

O výsledkoch riešenia v závislosti od parametra pojednávajú nasledujúce obrázky



Obrázok č. 3: *Výsledok regálového Next-fit algoritmu s parametrom r=0,3*[3)]

Ako môžeme vidieť z obrázku 3, máme 20 útvarov so stanovenou šírkou a výškou. Regálový next-fit algoritmus bude vysvetlený v nasledovnej kapitole, avšak za povšimnutie tu stoja hranice regálov. Tie boli stanovené vďaka parametru *r* tak, že zobrazená časť nádoby nebola schopná generovať dostatočný počet regálov. Pri ešte nižšej hodnote parametra *r* očakávame ešte benevolentejšie hranice regálov. Je teda zrejmé, že parameter *r* je nutné nastavovať aj podľa toho, ako sme limitovaný výškou kontajneru.



Obrázok č. 4: *Výsledok regálového Next-fit algoritmu s parametrom r=0,8*[4)]

3) Zdroj obrázku: výstup php algoritmu naprogramovaného podľa http://users.cs.cf.ac.uk

4) Zdroj obrázku: výstup php algoritmu naprogramovaného podľa http://users.cs.cf.ac.uk

Zmena parametra *r* z 0,3 na 0,8 je zrejmá už len z pohľadu na obrázky 3 a 4. Hranice sú oveľa striktnejšie a taktiež sa na jednotlivých regáloch nachádza väčší počet útvarov.

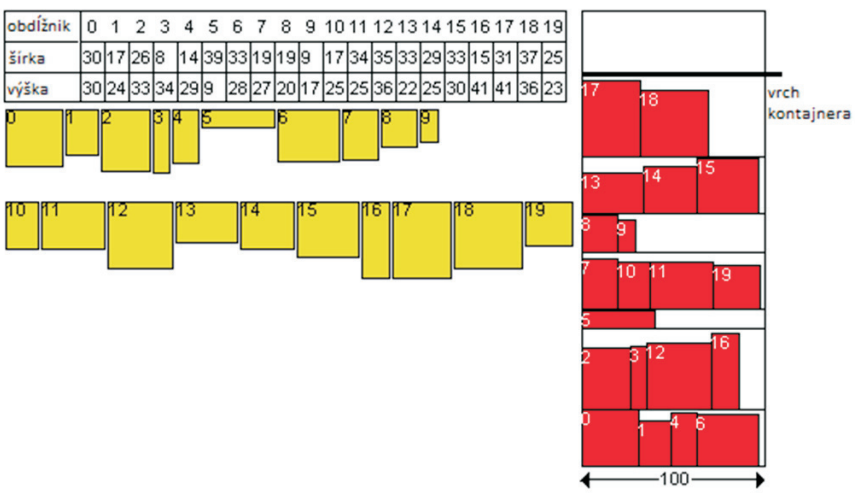
Poďme si teraz priblížiť niektoré algoritmy nachádzajúce sa v skupine regálových algoritmov.

2.3.1 Regálový Next-fit algoritmus

Tento typ algoritmu sa pri balení každého útvaru snaží zvoliť jeho pozíciu najviac vľavo, ako je to možné na najvyššom regáli. V prípade, že na danom regáli nieje miesto, alebo regál s takouto výškou neexistuje je vytvorený nový regál, ktorý sa stáva aktívnym pre danú výšku. Regále rovnakej výšky nachádzajúce sa pod aktívnym regálom nie sú zohľadňované. Ako vidíme napríklad z obrázku 4, pre útvar číslo 3 bol vytvorený nový regál, pretože jeho umiestnenie do druhého regálu nepripadalo kvôli výškovému obmedzeniu v úvahu[5)].

2.3.2 Regálový First-fit algoritmus

Tento algoritmus funguje tak, že je vybraný najspodnejší regál korektnej výšky, na ktorý je možné daný útvar umiestniť. Ak nemáme regál požadovanej výšky, alebo žiadny z regálov nespĺňa požiadavku na miesto, vytvorí sa nový regál s výškou podobnou danému obdĺžniku.



Obrázok č. 5: *Regálový First-fit algoritmus s parametrom r=0,7*[6)]

Ako vidíme z obrázka 5 už pre druhý útvar nezodpovedala výška prvého regálu (max. 30, útvar 2 má 33) a preto bol vytvorený nový regál.

Úrovňové algoritmy sú narozdiel od regálových off-line algoritmy. Z predchádzajúcich definícií teda vieme, že off-line algoritmy zahŕňajú znalosť rozmerov všetkých útvarov a ich následné zoraďovanie podla výšky zostupne. Prvý útvar, ktorý bude balený má teda najvyššiu výšku a posledný najnižšiu. Balenie sa potom uskutočnuje na postupnosti úrovní. Každý útvar je postupne balený do kontajnera umiestnením spodnej hrany tak, že kopíruje hranicu jednej z úrovní. Prvá úroveň je teda spodok kontajnera a každá nová úroveň je potom definovaná horizontálnou priamkou rovnobežnou a prechádzajúcou vrcholom najvyššieho útvaru predchádzajúcej úrovne.

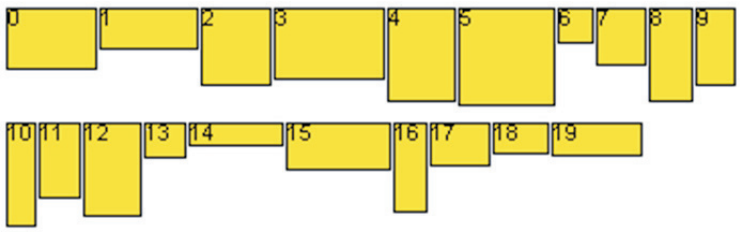
5) V tomto prípade zohral najväčšiu úlohu parameter r, ktorý bol nastavený na vysokú hodnotu. Pri pohľade na obrázok 4 je zrejmé, že situácia sa radikálne mení pri zmene parametra *r*.

6) Zdroj obrázku: výstup php algoritmu naprogramovaného podľa http://users.cs.cf.ac.uk

2.3.3 Úrovňový algoritmus next-fit so znižujúcou sa výškou

Tento algoritmus je úrovňový algoritmus, ktorý využíva prístup next-fit na balenie zoradeného zoznamu útvarov. Útvary sú postupne balené tak, že sú zarovnané zľava na jednej úrovni, až pokial sa nasledujúci útvar na danú úroveň nezmestí. Tento útvar je potom využitý na stanovenie novej úrovne a balenie pokračuje na tejto úrovni. Predchádzajúce úrovne dalej nie sú zohľadnované.

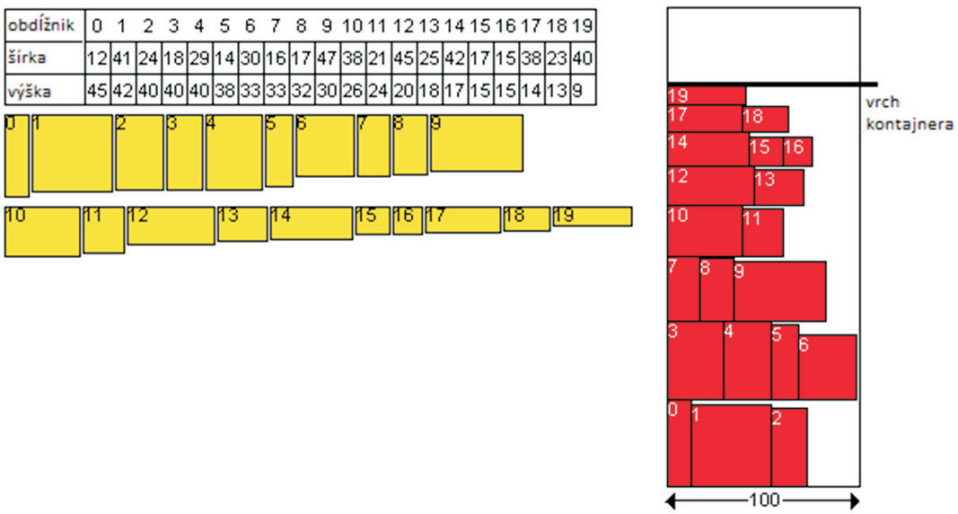
obdĺžnik	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
šírka	38	42	30	47	29	41	15	21	18	16	12	17	24	17	40	45	14	25	23	38
výška	26	17	33	30	40	42	15	24	40	33	45	32	40	15	9	20	38	18	13	14



Obrázok č. 6: *Vygenerované nezoradené útvary*[7)]

Po vygenerovaní útvarov nasleduje ich zoradenie podľa výšky a následné ukladanie pomocou algoritmu.

obdĺžnik	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
šírka	12	41	24	18	29	14	30	16	17	47	38	21	45	25	42	17	15	38	23	40
výška	45	42	40	40	40	38	33	33	32	30	26	24	20	18	17	15	15	14	13	9



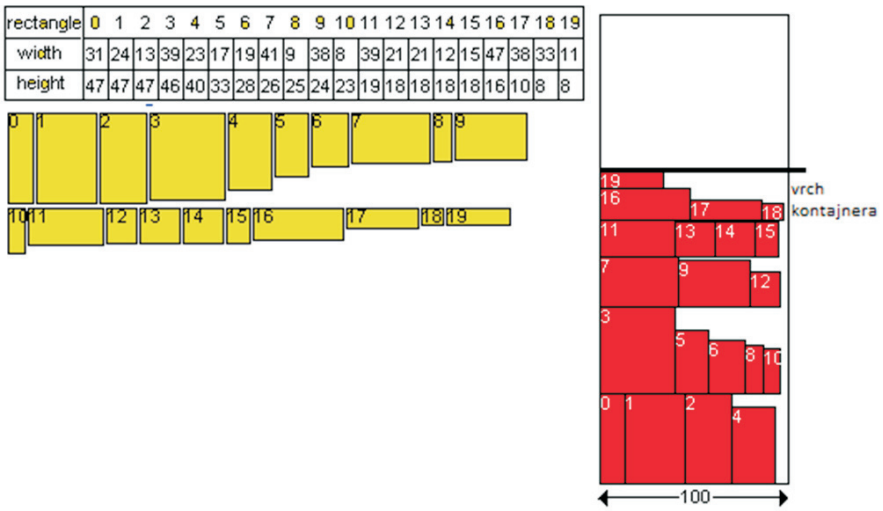
Obrázok č. 7: *Zoradenie a uloženie pomocou algoritmu next-fit*[8)]

2.3.4 Úrovňový algoritmus First-fit so znižujúcou sa výškou

Úrovňový algoritmus First-fit so znižujúcou sa výškou pracuje tak, že každý útvar je umiestnený na prvú úroveň kam sa zmestí. V prípade, že sa nezmestí do žiadnej z existujúcich úrovní, vytvorí sa nová úroveň. Rozdiel oproti Next-fit algoritmu je teda práve v tom, že útvar je umiestnený na prvé miesto kde sa zmestí, to znamená že sa testuje či sa na predchádzajúcich úroveniach nachádza plocha volná na jeho umiestnenie. Toto môže znamenať veľké úspory miesta v kontajneri.

7) Zdroj obrázku: výstup php algoritmu naprogramovaného podľa http://users.cs.cf.ac.uk

8) Zdroj obrázku: výstup php algoritmu naprogramovaného podľa http://users.cs.cf.ac.uk



Obrázok č. 8: *Umiestnenie zoradených útvarov do kontajnera úrovňovým algoritmom First-fit*[9)]

2.3.5 Ostatné typy algoritmov geometrickej optimalizácie 2D priestoru

Hore uvedené typy algoritmov zďaleka nie sú všetky, ktoré má potencionálny riešiteľ k dispozícii, avšak detailné rozpísanie všetkých z nich by presahovalo rozsah práce. Za spomenutie určite stojí inicializačná fáza kombinatorického algoritmu Benderových rezov[10)], algoritmus Akeba H.[11)], ktorý uplatňuje prístup hľadania a spätných krokov či heuristika SPEA2[12)] v kombinácii so štandardnými algoritmami v evolučnom algoritme umiestňovania útvarov na 2D pás od Colheo, Wannera, Souza a Carrana.

Práve z kombinácie vhodných vyššie spomenutých algoritmov sa skladá náš hybridný paralelný algoritmus.

3. Hybridný paralelný algoritmus balenia pásu

V súčasnosti, keď viacjadrové počítače predstavujú štandard prichádza potreba vytvárať algoritmy využívajúce túto hardvérovú kapacitu. Napriek obtiažnostiam pri paralelizácii môžeme pozorovať podstatný nárast výkonu, čo môže spôsobiť, že dané problémy sú riešené omnoho rýchlejšie. Druhou výhodou je to, že vznikajú rôzne hybridné algoritmy, ktoré sú niekedy efektívnejšie čo sa týka počtu krokov smerujúcich k vyriešeniu úlohy ako pôvodné algoritmy.

Pri návrhu paralelného algoritmu bola najdôležitejšia otázka, ako vhodne využiť výpočtovú kapacitu viacerých jadier. Museli sme teda odpovedať na otázku, ako je problém možné efektne deterministicky rozdeliť. Ak bližšie preskúmame obrázky jednotlivých algoritmov, možeme konštatovať nasledovné: jediným prvkom, o ktorom neviem dopredu povedať akú bude mať

9) Umiestnovaniu predchádzalo zoraďovanie, čo sme však z hladiska šetrenia miesta nezahrnuli do práce ako obrázok. Zdroj obrázku: výstup php algoritmu naprogramovaného podľa http://users.cs.cf.ac.uk

10) Cote, J., Dell'Amico, M., Iori, M.: Combinatorial Benders' Cuts for the Strip Packing Problem [online], 20.4.2013 https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2013-27.pdf

11) Akeb, H., Hifi, M., Lazure, D.: An improved algorithm for the strip packing problem. In *Computer Science and Information Systems (FeDCSIS)*, str. 357 – 364, ISBN:9781467307086

12) Coelho, D.G., Wanner E.F., Souza S.R., Carrano, E.G: A multiobjective evolutionary algorithm for the 2D Guillotine Strip Packing Problem, In *Evolutionary Computation (CEC)*, str. 1-8, ISBN:9781467315104

veľkosť je ušetrená plocha. Tento prvok je teda nedeterministický. Pri našom nasledovnom postupe s ním teda tak budeme uvažovať.

Filozofiou nášho algoritmu je teda algoritmus navrhnúť tak, aby každé jedno jadro spracovávalo deterministický algoritmus. Tým pádom sme však obmedzený na riešenie pomerne špecifických prípadov.

1. Algoritmus predpokladá využitie 100 % možnej kapacity pásu. Táto podmienka nám teda dopredu hovorí, že efektívnosť algoritmu bude v rýchlosti spracovania a nie v optimalizácii ušetreného miesta
2. V súvislosti s bodom jedna potom uvažujeme o rozmiestňovaní útvarov, ktorých celková plocha je vyššia ako celková plocha pásu. Teda vzniká situácia, že aj v prípade optimálneho riešenia nebude možné uložiť na pás všetky útvary.
3. Predpokladáme, že ukladané útvary majú tvar obdĺžnika resp. štvorca.

Náš hybridný algoritmus kombinuje dva klasické prístupy; regálový a úrovňový algoritmus. Algoritmy boli vybrané z hľadiska úspory času a ich efektívnosti. Parameter *r* je asi najdôležitejšou časťou algoritmu, pretože jeho spôsoby výpočtu môžu zefektívniť, ale taktiež aj zhoršiť kvalitu algoritmu.

3.1 Konštrukcia algoritmu

Činnosť algoritmu možno zhrnúť nasledovne:

1. Zisti počet jadier procesora.
2. Náhodne rozdeľ balené útvary do skupín, ktorých počet je zhodný s počtom jadier procesora.
3. Rozdel pás na rovnako veľké úseky, ktorých počet sa rovná počtu jadier procesora.
4. Pomocou metódy *sort()* zoraď útvary v jednotlivých skupinách od najväčšieho po najmenší.
5. V prípade, že ľubovolný najväčší útvar je vyšší ako je výška ľubovolného úseku, zlúč dva úseky do jedného a k nim príslušné skupiny do jednej. Tieto skupiny znovu zoraď pomocou metódy *sort()*.
6. V paralelnom cykle *Paralel.For* vykonaj pre každé jadro[13] deterministický *first-fit* algoritmus.
7. Po ukončení behu algoritmu na všetkých jadrách ulož zvyšné útvary do poľa a zoraď toto pole vzostupne.
8. Pre každú časť pása vykonaj deterministický *first-fit* a ulož útvary, ktoré je možné uložiť.

Výhodou takéhoto algoritmu z pohľadu teórie je dĺžka exekúcie. Ak predpokladáme optimálnu situáciu a teda, že sa využijú všetky jadrá, očakávame zrýchlenie v priemere o 15 % až 50 % na jedno jadro procesora.

Na testovanie algoritmu sme si vytvorili generátor útvarov s náhodnými veľkosťami, ktoré však neprekročia 20 % z plochy pásu[14]. Veľkosť pásu je generátoru zasielaná ako parameter.

3.2 Dosiahnuté výsledky

Dosiahnuté výsledky sme zhrnuli do tabuľkovej formy doplnenej o graf. Každý z algoritmov bol kvôli odstráneniu odchýliek spustený 500 krát. Napriek tomu treba uvažovať pri interpretácii

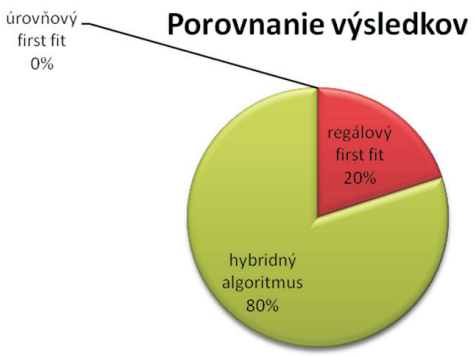
13) Ktorých počet zistíme ako počet častí pásu

14) V prípade, že by veľkosť jedného útvaru prekročila 20 % dosahoval by v prípade, že počítač na ktorom algoritmus beží má viac než 8 jadier suboptimálne výsledky.

aj s miernou odchýlkou. Tabuľka č. 1 nám ukazuje priemerný čas behu algoritmu s rozličným počtom ukladaných útvarov pre rôzne veľkosti pásu.

Tabuľka č. 1 *Dosiahnuté výsledky*[15)]

veľkosť pásu	počet útvarov	priemerná veľkosť útvaru	exekučný čas úrovňového first fit	exekučný čas regálového first fit	exekučný čas nášho algoritmu
550	250	5,8146	20,12	25,9	14,7
600	280	6,128	24,128	30,844	20,12
700	350	8,55	27,6	32,8	22,11
800	400	7,65	31,5	37,77	26,24
850	420	6,782	33,97	39,43	29,56
900	450	5,97	36,71	42,11	31,5
1000	570	7,138	39,97	45,49	34,27
1050	720	6,71	42,45	48,16	37,71
2000	1200	8,12	75,12	94,58	76
4000	2500	7,54	162,44	164,844	170,15
10000	5000	4,54	380,15	394,74	450,12



Graf č. 1 *Porovnanie výsledkov*[16)]

4. Diskusia a návrhy na zlepšenie

Nami navrhnutý hybdriný algoritmus dosiahol v 80 % testovaných prípadov lepšie výsledky ako dva konkurenčné algoritmy. Algoritmus teda potvrdil hypotézu, že paralelizácia prinesie isté zlepšenie v otázke exekučného času. Algoritmus však pracoval pod pomerne špecifickými podmienkami, ktoré významne obmedzujú jeho využitie v praxi.

Algoritmus bol naprogramovaný v jazyku ?#, ktorý má značnú podporu paralelného programovania. Naproti tomu konkurenčné algoritmy boli spracované v jazyku PHP. Rozdiel v rýchlosti týchto dvoch jazykov pravdepodobne penalizoval riešenie v jazyku PHP. Odborná literatúra uvádza, že jazyk ?# je zhruba o 15 %-20 % rýchlejší ako jazyk PHP.

Ako sa počas testovania ukázalo, náš algoritmus má tendenciu zhoršovať svoje exekučné časy vzhľadom na zvyšujúcu sa veľkosť pásu a množstva ukladaných útvarov. Toto je možné vysvetliť veľmi jednoducho. Totiž algoritmus po tom, ako sa všetky parciálne pásy naplnili uloží do pola neumiestnené predmety, zoradí ich a snaží sa vložiť do jednotlivých parciálnych pásov

15) Výsledky sú uvádzané v sekundách. Vlastný zdroj.

16) Vlastný zdroj

možné útvary. S rastúcim počtom útvarov potom rastie aj časové kvantum potrebné na zoradenie a pokusné vkladanie do častí pásu. Z toho možno usudzovať, že čím viac bude ukladaných útvarov, tým horšie časy bude algoritmus vykazovať.

Záver

Paralelizácia je jednoznačným prínosom pre počítanie úloh operačného výskumu. Otázka, ktorú sa snaží mnoho ľudí riešiť je jej vhodná implementácia. Paralelizácia nevhodných častí kódu môže potom spôsobiť, že čas behu sa namiesto očakávaného zníženia zvýši a preto je vhodné algoritmus ako celok analyzovať.

Medzi základné typy paralelizácie patrí paralelizácia jednoduchých činností, hlavne teda cyklov. Náš algoritmus v podstate rieši namiesto jednej úlohy *n*-rovnakých úloh, kde *n* predstavuje počet jadier procesora. Toto nám v tomto konkrétnom prípade zabezpečí maximálne možné vyťaženie procesora, pretože sú v ideálnom prípade využívané všetky jadrá. Efektívnejšia paralelizácia by bola taká, kde by sa na jednotlivé vlákna delegovali špecifické úlohy jedného algoritmu. Ak je však algoritmus rekurzívny, teda treba napríklad zoradiť útvary pred ich samotným uložením, je paralelizácia prakticky nemožná.

Preto by do budúcna bolo vhodné zamyslieť sa aj nad samotnými matematickými krokmi daného algoritmu a pokúsiť sa ich upraviť tak, aby boli paralelizovateľné. V našom hybridnom algoritme sa to čiastočne podarilo, avšak za cenu niektorých obmedzení.

Cieľom navrhnutého algoritmus nebolo konkurovať heuristikým algoritmom, ktoré riešia rozličné variácie problému balenia pásu. Navrhnutý algoritmus by mal znamenať prínos pre niektoré heuristické algoritmy (v závislosti od ich spôsobu generovania a modifikácie riešení) a v budúcnosti bude zohrávať práve túto úlohu v genetickom algoritme, ktorý v súčasnosti pripravujeme.

Použitá literatúra:

- [1] MAURICIO G. C. R., PINHO DE SOUSA, J.: *Metaheuristics: Computer Decision Making Applied Optimization.* Springer, 2003. ISBN 9781402076534.
- [2] ANDONOV, R. et al.: *Optimal semi-oblique tiling.* New York : ACM, 2001. ISBN1581134096.
- [3] MURPHEZ, R., PARDALOS, P.: *Cooperative Control and Optimization.* [s.l.] : Springer, 2002. ISBN1402005490.
- [4] RIBEIRO, C., MARTINS, S.: *Experimental and Efficient Algorithms.* Berlin : Springer, 2004. ISBN3540220674.
- [5] ROBINSON, S. et al.: *C# Programujeme profesionálně.* Brno : Computer Press, 2003. ISBN8025100855.
- [6] YANG, X., TEO, K.,CACCETTA, L.: *Optimization methods and applications.* Dordrecht : Springer, 2001. ISBN9780792368663.
- [7] DUPAĽ A., BREZINA I.: Logistika v manažmente podniku Bratislava : Sprint vfra, 2006. – 326 s. : grafy, obr., sch., tab. – ISBN 80-89085-38-5
- [8] BREZINA I., ČIČKOVÁ Z., REIFF M.: Kvantitatívne metódy na podporu logistických procesov – Bratislava : Vydavateľstvo EKONÓM, 2009. – 191 s. – ISBN 978-80-225-2648-7
- [9] GRELL, M., HYRÁNEK E.: Maticové modely na meranie výkonnosti produkčných systémov. In *E + M. Ekonomie a management : vědecký ekonomický časopis*. – Liberec : Ekonomická fakulta Technické univerzity v Liberci, 2012. ISSN 1212-3609, 2012, roč. 15, č. 1, s. 73-87. VEGA 1/165/08.
- [10] CHOCHOLATÁ M.: Lineárne programovanie pre manažérov, Bratislava : Vydavateľstvo EKONÓM, 2013. – 234 s. [14,021 AH]. – ISBN 978-80-225-3562-5

- [11] KRAUSPE, K.: An evolutionary algorithm for mixed postman problem. In *Mezinárodní Baťova konference pro doktorandy a mladé vědecké pracovníky : [recenzovaný zborník] : 7. ročník : 12. dubna 2011* [elektronický zdroj]. – Zlín : Univerzita Tomáše Bati ve Zlíně, 2011. ISBN 978-80-7454-013-4, s. [1-11].
- [12] BREZINA I.JR., ČIČKOVÁ Z.: *Solving the Travelling Salesman Problem Using the Ant Colony Optimization,* Management Information Systems, vol. 6, pp. 10-14, 2011.
- [13] COTE, J., DELL'AMICO, M., IORI, M.: *Combinatorial Benders' Cuts for the Strip Packing Problem* [online], 20.4.2013 https://www.cirrelt.ca/DocumentsTravail/CIRRELT-2013-27.pdf
- [14] AKEB, H., HIFI, M., LAZURE, D.: An improved algorithm for the strip packing problem. In *Computer Science and Information Systems(FeDCSIS),* str. 357 – 364, ISBN:9781467307086
- [15] COELHO, D.G., WANNER E.F., SOUZA S.R., CARRANO, E.G.: A multiobjective evolutionary algorithm for the 2D Guillotine Strip Packing Problem, In *Evolutionary Computation (CEC),* str. 1-8, ISBN:9781467315104
- [16] *Distributed & Scientific Computing, Cardiff University* [online], 1.4.2012 http://www.cs.cf.ac.uk/research/dsc.

Ing. Ján Pittner, PhD.
Katedra aplikovanej informatiky
Fakulta hospodárskej informatiky, Ekonomická univerzita v Bratislave,
Dolnozemská cesta 1, 852 35 Bratislava,
Telefón: 0903 630 151
e-mail: janci.pittner@gmail.com