SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2852

Rješavanje problema pakiranja kutija korištenjem metaheuristika

Jan Čapek

Zagreb, kolovoz 2022.

SADRŽAJ

1. Uvod

Problem pakiranja kutija je klasičan optimizacijski problem gdje je ideja upakirati elemente različitih veličina u što manje kontejnera, tako da se minimizira neiskorišten prostor. Verzija ovog problema ima više: kutije mogu biti različitih veličina ili fiksne veličine, također one mogu biti multidimenzionalne, broj elemenata ne mora biti ograničen itd.

Problem je primjenjiv u raznim životnim okruženjima poput utovarivanja kamiona, brodskih kontejnera, minimizacije potrebnih računalnih resursa za aplikacije i slično. Danas digitalni oblak omogućuje bezbolno, gotovo neograničeno, skaliranje računalnih resursa, ali uz povećanje cijene stoga se i tu našao problem pakiranja što većeg broja aplikacija na što manji broj virtualnih računala kako bi se trošak minimizirao.

Postoji nekoliko heurističkih algoritama koji služe za rješavanje problema pakiranja, ali oni, naravno, nisu egzaktni jer je problem NP-težak što znači da će ga bilo koji deterministički vremenski polinomijalan algoritam teško riješiti. Ti algoritmi će također biti spomenuti u radu jer će se koristiti u simbiozi sa metaheurističkima.

Ovaj rad bavit će se pakiranjem jednodimenzionalnih elemenata u kutije istih veličina pomoću tri prirodom inspirirana metaheuristička algoritma. Svaki od tih algoritama koristit će isti heuristički algoritam koji će zapravo raditi pakiranje, a metaheuristički algoritam učit će redoslijed kojim treba elemente predavati klasičnom heurističkom algoritmu. Važno je naglasiti da spomenuti algoritmi pretpostavljaju da niti jedan element neće biti veći od same kutije.

2. Opis problema

Zajedničko svim varijantama problema pakiranja kutija je upakirati što veći broj elemenata s ograničenjem da kumulativna veličina elemenata u nekoj kutiji ne premašuje njenu veličinu. Također, pretpostavlja se da veličina jednog elementa neće premašiti veličinu prazne kutije, tj. da je svaki element moguće staviti barem u neku praznu kutiju koja je na raspolaganju.

Postoje verzije problema gdje je broj dozvoljenih kutija ograničen, ali njihove veličine ne moraju nužno biti iste i cilj je upakirati što više elemenata u njih. Također, kutije i elementi ne moraju biti jedne dimenzije, u stvarnom svijetu gotovo nikad i nisu. Npr. stvarna kutija ima visinu, širinu i dužinu pa stoga algoritmi koji rješavaju ovakvu vrstu problema moraju uzeti u obzir da element možda stane po širini, ali ne i visini i slično.

Kao što je u uvodu spomenuto, inačica problema koja se razmatra u radu je jednodimenzionalna s konstantnom veličinom kutija čiji broj nije ograničen. Elementi i kutije imat će samo jednu dimenziju, a sve kutije iste su veličine. Ekvivalent u stvarnom životu bio bi utovar stvari u kamione gdje je jedino ograničenje ne premašiti određenu težinu, a volumen stvari se zanemaruje. Također, svi dostupni kamioni su isti i neograničen ih je broj.

Svaki problem je stoga opisan veličinom kutije, te popisom veličina elemenata koji se moraju pakirati. Primjer je dan ispod.

$$veličina\ kutije = 10$$

$$veličine\ elemenata\ za\ pakiranje = [1, 5, 2, 7, 3, 5, 6, ...]$$

Izlaz iz algoritama je niz kutija koje sadrže popis elemenata koje su u njih stavljene.

$$[Bin(2, 3, 5), Bin(7, 2), Bin(5, 5), ...]$$

Za ocjenu dobrobiti rješenja, koristi se suma neiskorištenog prostora svih iskorištenih kutija. Što je neiskorištenog prostora više, to je rješenje lošije.

3. Klasični heuristički algoritmi

Postoji nekoliko klasičnih, vrlo jednostavnih heurističkih algoritama za problem pakiranja kutija. Rezultat većine njih jako ovisi o redoslijedu pakiranja elemenata. Što će dovesti do toga da će se ti algoritmi u ovom radu koristiti za pakiranje, a metaheurističkima će zadatak biti naučiti najbolji redoslijed pakiranja.

Prvi i najjednostavniji algoritam zove se *Next-Fit*. Ideja tog algoritma je da pakiranje počinje od prve kutije, algoritam uzima prvi element koji treba spakirati, on se stavi u trenutnu kutiju ako stane u nju. Ako element ne stane u trenutnu kutiju, onda se uzima nova u koju se taj element stavlja. Algoritam se nikada ne vraća na prošlu kutiju, jednom kad pređe na novu, novi elementi se stavljaju u tu kutiju ili opet novu, bez obzira imaju li stare kutije mjesta za te elemente. Vremenska i memorijska složenost ovog algoritma je $\mathbf{O(n)}$.

First-Fit algoritam, bolji je od *Next-Fit* algoritma po tome što ne zanemaruje stare kutije, radi na principu da element po element stavlja u najstariju kutiju u koju on stane. Tek ako element ne stane niti u jednu kutiju, otvara novu i tamo ga stavlja. Pošto za svaki element algoritam pretražuje najstariju kutiju u koju element stane, ako su kutije već sortirane po dostupnom prostoru i starosti, vremenska složenost mu je $\mathbf{O(n * \log m)}$, gdje je n broj elemenata, a m broj kutija. Memorijska složenost jednaka je prijašnjem algoritmu, što je $\mathbf{O(n)}$.

Best-Fit pokušava pakirati element u onu kutiju koja je najpunija, a da ga može primiti. Heuristika pokušava minimizirati neiskorišten prostor. Ako se algoritam implementira pomoću binarnog stabla koje drži kutije sortirane prema slobodnom prostoru, vremenska složenost je $\mathbf{O(n * \log m)}$ jer se za svaki element pretražuje kutija koju će on najbolje popuniti dok je prostorna ista kao kod prijašnja dva, $\mathbf{O(n)}$.

U ovom radu, koristit će se *Best-Fit* algoritam kao heuristika koja će pakirati kutije stoga će on određivati koliko je metaheuristički algoritam dobar odnosno loš.

4. Algoritam krijesnice

Algoritam krijesnice *(engl. Firefly Algorithm)* je prirodom inspiriran stohastička metaheuristička metoda optimizacije. Pripada kategoriji algoritama pod nazivom "Inteligencija roja" (*engl. Swarm Intelligence*) zato što se bazira na kolektivnom ponašanju samoorganizirajuće populacije (Beni i Wang, 1993).

Osmislio ga je Yang (2008). Algoritam je inspiriran bljeskajućim svjetlom krijesnica u prirodi te ne garantira pronalazak optimalnog rješenja problema. Također, kod svakog metaheurističkog pretraživanja, bitno je balansirati istraživanje i iskorištavanje (Fister et al., 2013). Istraživanje će pomoći da se nađu različita rješenja dok iskorištavanje koristi da se pretraživanje usmjeri prema, do sad pronađenim, boljim rješenjima.

Glavna karakteristika krijesnica je njihova svjetlucavost. Njena funkcija je privući potencijalne partnere i upozoriti predatore. Sam algoritam koristi samo prvu funkciju, privlačenje partnera. Također, metoda pretpostavlja određena pravila. Prvo, krijesnice su unisex što znači da svaka potencijalno može privući bilo koju drugu. Drugo, jedinka je privlačna proporcionalno koliko svijetli, a manje svjetlucave jedinke približavaju se jačima, ne obrnuto. Ako nema jedinke koja bi privukla krijesnicu, onda se ona kreće nasumično. I zadnje, svjetlost jedinke određena je funkcijom dobrote.

Valja razlikovati svjetlost od intenziteta. Intenzitet svjetlosti je objektivna mjera izračunata pomoću funkcije dobrote, dok je svjetlost mjera koja opisuje koliko jedna krijesnica svijetli drugoj s obzirom na udaljenost između njih jer se svjetlost smanjuje s udaljenošću, $\beta \propto 1/r^2$ (Munien et al., 2020). Tako da je svjetlost definirana formulom **(4.1)**. U slučaju pakiranja kutija, funkcija dobrote je negativna suma neiskorištenog prostora svih inicijaliziranih kutija.

$$\beta = \beta_0 * e^{-\gamma r^2} \tag{4.1}$$

gdje je β_0 zadani intenzitet primljen kao parametar, γ koeficijent apsorpcije svjetlosti te r udaljenost između jedinke i promatrača.

Samu udaljenost definiramo kao euklidsku. Stoga je ona izražena formulom **(4.2)**.

$$r = ||s_i - s_j|| = \sqrt{\sum_{k=1}^{n}(s_{ik} - s_{jk})^2} \tag{4.2}$$

gdje s_i i s_j predstavljaju krijesnice, a n njihovu dimenzionalnost.

Kad se krijesnica s_i pomakne prema krijesnici s_j, njenu novu poziciju dat će formula **(4.3)**.

$$s_i^{t+1} = s_i^t + \beta_0 e^{-\gamma r_{ij}^2}(s_j^t - s_i^t) + \alpha\epsilon_i^t \tag{4.3}$$

gdje α predstavlja koeficijent nasumičnog pomaka, a ϵ_i^t nasumično odabran broj iz Gaussove distribucije $N(0, 1)$. Iz formule se vidi da nova lokacija krijesnice i ovisi o trenutnoj, intenzitetu svjetlosti druge jedinke j, udaljenosti od nje do j, te nasumično odabranom koraku istraživanja.

Konkretno za problem pakiranja kutija treba definirati što jedna krijesnica predstavlja, a to je poredak elemenata kojim treba predavati elemente heurističkom algoritmu. Važno je primijetiti da algoritam krijesnice nije diskretan što znači da krijesnica predstavlja poredak, ali kontinuirani, te je potrebno dekodirati ju u diskretne brojeve koji govore "prvi element je onaj pod brojem 5, drugi pod brojem 1 itd.". To se može ostvariti rangiranjem vrijednosti *(engl. Rank Order Value - ROV)* vektora krijesnice tako da će na mjestu gdje je vrijednost vektora najmanja biti 0, druga najmanja 1 itd. Npr. neka je vrijednost krijesnice $[0.12, 0.5, 0.1]$, njena dekodirana vrijednost je $[1, 2, 0]$ (indeksirano od 0) što znači da će redoslijed elemenata za heuristički algoritam biti: 2. element, 3. element, 1. element.

Algoritam 1 Algoritam krijesnice

```
function FIREFLY(γ, α, β_0)
    Inicijaliziraj populaciju od N jedinki
    Izračunaj intenzitet krijesnica pomoću funkcije dobrote
    while (t < MaxGeneracija) do
        for i = 0 .. N do
            for j = 0 .. i do
                if I_j > I_i then
                    Pomakni i-tu krijesnicu prema j-oj prema formuli (4.3)
                    Izračunaj novi intenzitet jedinke
                end if
            end for
        end for
        Nađi najbolju krijesnicu
    end while
end function
```

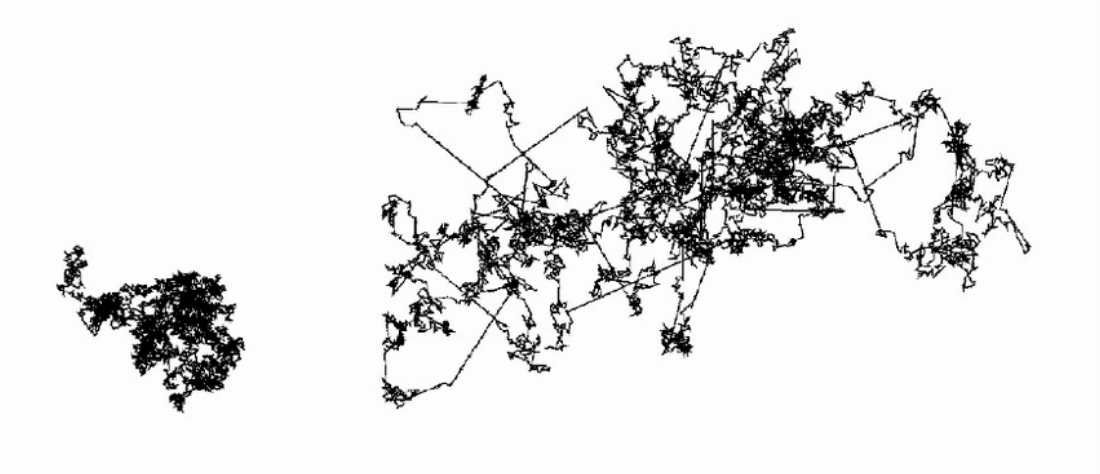
Munien et al. (2020) predlažu hibridni algoritam koji dodaje i mutaciju kako bi se smanjila vjerojatnost ostanka u lokalnom minimumu, pomoću novog parametra π koji stoji kao vjerojatnost mutacije. Ta vjerojatnost se smanjuje s brojem iteracija kako bi algoritam preferirao istraživati u ranijim fazama, a kasnije iskorištavao znanje o najboljim jedinkama. Sama mutacija se svodi na premetanje rasporeda elemenata koji se predaju heurističkom algoritmu tj. dva elementa zamijene svoja mjesta.

Algoritam 2 Hibridni algoritam krijesnice

function FIREFLY($\gamma, \alpha, \beta_0, \pi, degradacijaMutacije$)
Inicijaliziraj populaciju od N jedinki
Izračunaj intenzitet krijesnica pomoću funkcije dobrote
while (t < MaxGeneracija) **do**
for i = 0 .. N **do**
for j = 0 .. i **do**
if $I_j > I_i$ **then**
Pomakni i-tu krijesnicu prema j-oj prema formuli **(4.3)**
Mutiraj i-tu jedinku prema vjerojatnosti π
Izračunaj novi intenzitet jedinke
end if
end for
end for
$\pi = \pi * degradacijaMutacije$
Nađi najbolju krijesnicu
end while
end function

5. Kukavičji algoritam

U standardnom algoritmu, kukavica pretražuje pomoću Lévyjevog leta, Chechkin et al. (2008). On služi za postizanje nasumičnog pomaka, a njegovi koraci dobivaju se iz Lévyjeve distribucije s beskonačnom varijancom i srednjom vrijednošću. Lévyjeva distribucija spada u kategoriju distribucije teškog repa.



Slika 5.1: Usporedba putanja Gaussovog (lijevo) i Lévyjevog (desno) procesa. Lévyjeve putanje karakterizirane su otocima koje grade skupine kratkih koraka međusobno povezani dugim koracima (Chechkin et al., 2008).

Kukavičji algoritam *(engl. Cuckoo Search Algorithm)* nazvan je po pticama kukavicama koje liježu jaja u gnijezda drugih vrsta ptica, a evoluirale su tako da njihova jaja gotovo isto izgledaju kao od ptica domaćina. Algoritam se zasniva na tri pravila, koja su, naravno, umjetno savršena. Prvo, svaka kukavica liježe jedno jaje i stavlja ga u nasumično odabrano gnijezdo. Drugo, najbolja gnijezda, s najboljim jajima (rješenjima) prelaze u sljedeću generaciju. Posljednje, broj gnijezda je fiksan i ptica čije je gnijezdo može otkriti kukavičje jaje s vjerojatnošću $p_a \in [0, 1]$ što će rezultirati ili odbacivanjem jaja ili izgradnjom kompletno drugog gnijezda (Zakaria i Layeb, 2016). Ideja zadnjeg pravila je da se pri kraju jedne iteracije p_a najlošijih jaja (rješenja), koja su lakše prepoznatljiva kao kukavičja, izbaci i zamijeni novima.

Pošto svako gnijezdo u sebi ima samo jedno jaje, onda će ono predstavljati jedno rješenje. Iako, po definiciji kukavičjeg algoritma, jaje predstavlja rješenje. Kodiranje rješenja jednako je kao u prošlom algoritmu, jaje predstavlja vektor s onoliko dimenzija koliko ima elemenata za pakiranje, a onda se koristi ROV kako bi se točno odredio redoslijed.

Algoritam upotrebljava lokalni i globalni nasumični hod kako bi pretraživao prostor rješenja. Lokalni hod dan je formulom **(5.1)**.

$$x_i^{t+1} = x_i^t + \alpha s * H(p_a - \epsilon) * (x_j^t - x_k^t) \tag{5.1}$$

gdje su x_j^t i x_k^t dva različita, nasumično odabrana rješenja. α je skalirajući faktor koji je algoritmu predan kao parametar dok s predstavlja neku osnovnu veličinu koraka, također dobiven kod poziva algoritma. p_a, spomenut ranije, naziva se još i vjerojatnost prebacivanja *(engl. Switching Probability)*, a služi za balansiranje između lokalnog i globalnog pretraživanja. $H(u)$ je *Heaviside* funkcija za koju vrijedi $H(x) = 0$ kad $x \leq 0$ i $H(x) = 1$ kad $x > 0$, a $\epsilon \in [0, 1]$ predstavlja nasumično odabran broj, za svaku dimenziju vektora rješenja, iz uniformne distribucije.

Globalno pretraživanje izraženo je sljedećom formulom.

$$x_i^{t+1} = x_i^t + \alpha s L(\lambda) \tag{5.2}$$

gdje je $L(\lambda)$ generiran korak iz Lévyjeve distribucije.

Za izračun Lévyjevog koraka koristit će se Mantegnov algoritam, Yang i He (2020). Korak je izražen formulom **(5.3)**.

$$L(\lambda) = \frac{u}{|v|^{\frac{1}{\lambda}}} \tag{5.3}$$

gdje $u \sim N(0, \sigma^2)$, $v \sim N(0, 1)$.

Varijanca varijable u izražena je formulom **(5.4)** preko gamma funkcija.

$$\sigma^2 = \left[\frac{\Gamma(1+\lambda)}{\lambda\Gamma(1+\lambda)/2} * \frac{sin(\pi\lambda/2)}{2^{(\lambda-1)/2}} \right]^{1/\lambda} \tag{5.4}$$

U praksi, nije potrebno točno izračunati ovu varijancu već ju je dovoljno zadati prilikom pokretanja algoritma (Yang i He, 2020). Njena vrijednost utjecat će na veličine koraka i njihovu raspršenost te se mora postaviti na senzibilnu vrijednost za problem koji se rješava. Globalno pretraživanje i Lévyjev korak bit će tada izraženi sljedećim formulama.

$$x_i^{t+1} = x_i^t + \alpha s L(\lambda, \sigma^2) \tag{5.5}$$

$$L(\lambda, \sigma^2) = \frac{u}{|v|^{\frac{1}{\lambda}}} \tag{5.6}$$

Algoritam 3 Kukavičji algoritam

function CUCKOOSEARCH($\alpha, s, p_a, \lambda, \sigma^2$)
Inicijaliziraj N gnijezda
while (t < MaxGeneracija) **do**
Nasumično odaberi gnijezdo x_i i x_j koristeći k-turnir
Generiraj x_i' formulom **(5.2)**
if $dobrota(x_i') > dobrota(x_j)$ **then**
Izbaci x_j i zamijeni ga novim x_i'
end if
p_a najlošijih gnijezda zamijeni njihovim lokalnim nasumičnim hodom danim formulom **(5.1)**
Evaluiraj gnijezda i nađi najbolje
end while
end function

Vidljivo je da algoritam koristi funkciju dobrote. Ona je, kao i kod prošlog algoritma, izražena kao negativna suma neiskorištenog prostora svih inicijaliziranih kutija.

6. Genetski algoritam

Genetski algoritam je metoda optimizacije inspirirana prirodnim evolucijskim procesom, Yang (2014). Genetski algoritam koristi nekoliko operatora, a to su selekcijski postupak, križanje i mutacija. Također, genetski algoritam preuzima iz genetike i pojam kromosoma koji predstavlja jedno rješenje problema kojeg se optimira. To će u slučaju pakiranja kutija biti kodirani redoslijed elemenata koji se predaju heurističkom algoritmu. Taj redoslijed je kontinuiran, kao i kod prijašnja dva algoritma te će biti dekodiran već spomenutom *ROV* metodom.

Ovaj algoritam također je baziran na populaciji mogućih rješenja problema, stoga selekcijski postupak služi kako bi se odabrali roditelji iz kojih će nastati novo "dijete" rješenje pomoću operatora križanja. Konkretno, za ovaj problem, odabrana je k-turnirska selekcija gdje se nasumično odaberu tri rješenja, od kojih dva najbolja postaju roditelji novog, a ono lošije bude zamijenjeno s tim novim rješenjem.

Za križanje, odabrana je jednostavna metoda presjeka u jednoj točci. Presjek za problem pakiranja kutija bit će točno na polovici kromosoma roditelja. Pomoću ove metode, dijete će dobiti prvu polovicu kromosoma od prvog roditelja, a drugu od drugog. Naravno, moglo je biti i obrnuto, da prva polovica bude od drugog, a druga od prvog roditelja. Primjer križanja dan je u nastavku.

$$Roditelj\ 1:\ [0.1, 0.7, 0.3, 0.1]$$

$$Roditelj\ 2:\ [0.8, 0.3, 0.6, 0.2]$$

$$Dijete\ dobiveno\ križanjem\ roditelja\ 1\ i\ 2:\ [0.1, 0.7, 0.6, 0.2]$$

Također, križanje se moglo napraviti i tako da se gen djeteta izračuna kao srednja vrijednost odgovarajućeg gena svakog roditelja. U tom slučaju, kromosom djeteta izražen je formulom **(6.1)**

$$x_k = (x_i + x_j)/2 \tag{6.1}$$

gdje je x_k kromosom djeteta, a x_i i x_j kromosomi roditelja.

Operator mutacije zamjenjuje dva gena unutar kromosoma s vjerojatnošću π što rezultira mijenjanjem poretka elemenata koji se predaju heurističkom algoritmu. Primjer rezultata mutiranja gdje se 1. gen zamijenio s 3. može se vidjeti ispod.

$$[0.1, 0.7, 0.6, 0.2] \Rightarrow [0.6, 0.7, 0.1, 0.2]$$

Kao i kod prva dva algoritma, za funkciju dobrote koristi se negativna suma neiskorištenog prostora svih inicijaliziranih kutija.

Algoritam 4 Genetski algoritam

function GENETICALGORITHM(π)
    Inicijaliziraj N kromosoma
    while (t < MaxGeneracija) **do**
        Nasumično odaberi tri kromosoma koristeći k-turnir
        Napravi dijete od bolja dva koristeći operator križanja
        Operatorom mutacije mutiraj dobiveno dijete
        Zamijeni najlošiji kromosom dobiven iz k-turnira sa novo kreiranim djetetom
        Evaluiraj kromosome
    end while
end function

7. Rezultati

Svi rezultati dobiveni su na istom skupu podataka koji se sastoji od 1210 različitih problema (Scholl i Klein). Problemi su kategorizirani u dvije težinske razine, teški i ostali, ovisno o raspršenosti veličina elemenata te koliko elemenata u prosjeku stane u kutije. Kreatori skupa problema ne navode direktno koje su karakteristike problema koji nisu teški, ali će se ovdje to zaključiti iz njihove definicije teških problema. Teški problemi podrazumijevaju da će po kutiji biti mali broj elemenata i da su njihove veličine jako raspršene iz čega se može zaključiti da jednostavne probleme karakterizira slaba raspršenost veličina i to što puno elemenata stane u kutiju tako da male razlike u redoslijedu nemaju toliki utjecaj na rezultat. U ovom radu se spominju i problemi srednje težine koji su tako klasificirani jer pogledom u rezultate algoritama, vidi se da ih metaheuristički nisu uspjeli međusobno jednako dobro riješiti, ali razlike između njihovih performansi nisu velike kao kod teških problema. Konkretno, Scholl i Klein iznose da su veličine elemenata teških problema iz intervala $[20000, 35000]$, te u prosjeku u kutije stane 3 do 5 elemenata. Veličina kutija jednaka je 100000.

Svaki problem zadan je veličinom kutije i veličinama svakog od elemenata koje treba pakirati. Prilikom učitavanja problema, elementi su nasumično izmiješani kako bi se moglo lakše vidjeti kako redoslijed utječe na koji algoritam.

Za prikaz kvalitete rješenja, odabrana je suma neiskorištenog prostora inicijaliziranih kutija. Ova odluka bila je logična s obzirom na to da su svi navedeni algoritmi koristili funkciju dobrote koja je bila obrnuto proporcionalna neiskorištenom prostoru.

Svaki algoritam pokrenut je tri puta na istom problemu s istim poretkom elemenata kako bi mogao probati riješiti problem pomoću više od jedne inicijalne populacije. Od tri pokretanja, za svaki algoritam uzelo se njegovo najbolje rješenje koje će biti predstavljeno na grafovima.

Na grafovima, bit će prikazan i klasični *Best-Fit* algoritam kako bi postavio ljestvicu rješenja.

Svaki algoritam imao je iste postavke za sve probleme iz skupa podataka koji su navedeni u nastavku.

Parametar	Vrijednost
γ	0.3
α	0.01
β_0	0.01
π	0.05
$degradacijaMutacije$	0.8
$veličinaPopulacije$	20
$MaxGeneracija$	100

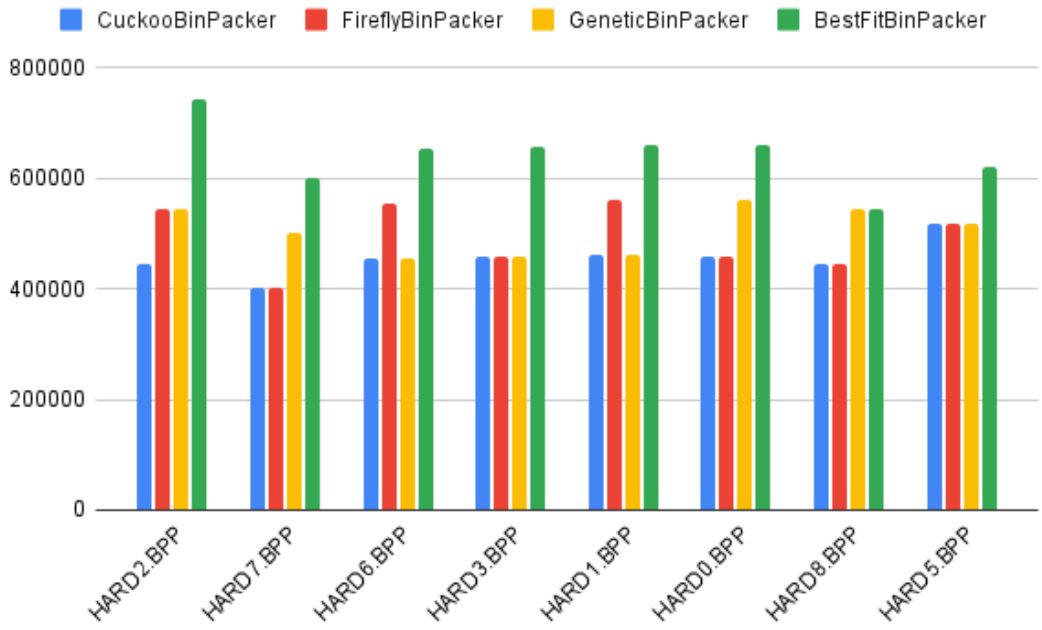
Tablica 7.1: Parametri algoritma krijesnice.

Parametar	Vrijednost
αs	2
p_a	0.1
λ	0.1
σ^2	2
$veličinaPopulacije$	20
$MaxGeneracija$	100

Tablica 7.2: Parametri kukavičjeg algoritma. α i s predani su algoritmu kao jedan parametar jer se u algoritmu uvijek koriste skupa u umnošku.

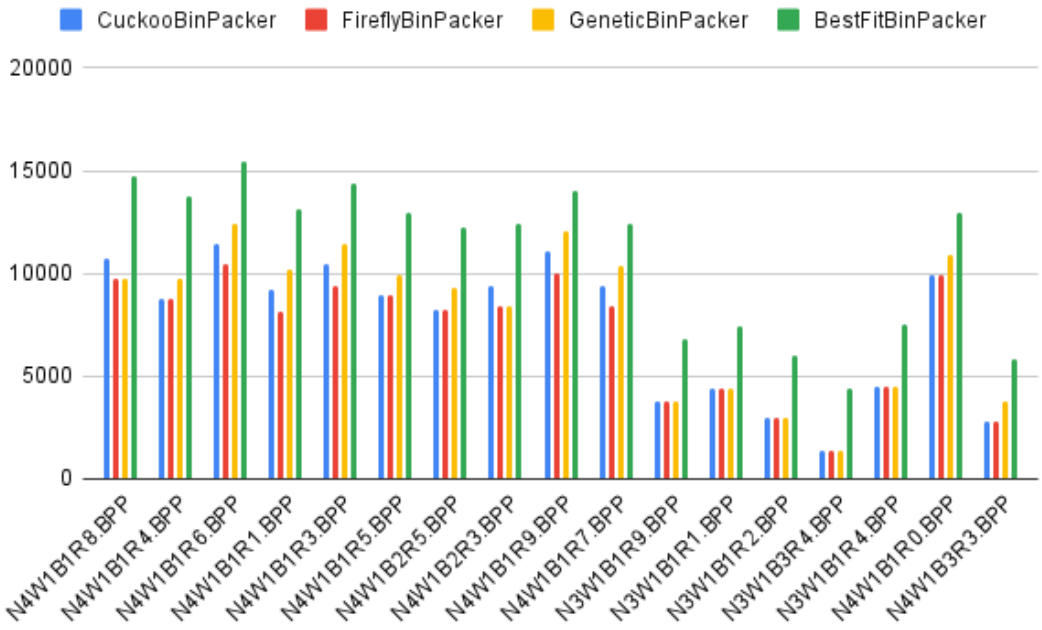
Parametar	Vrijednost
π	0.05
$veličinaPopulacije$	50
$MaxGeneracija$	200

Tablica 7.3: Parametri genetskog algoritma.



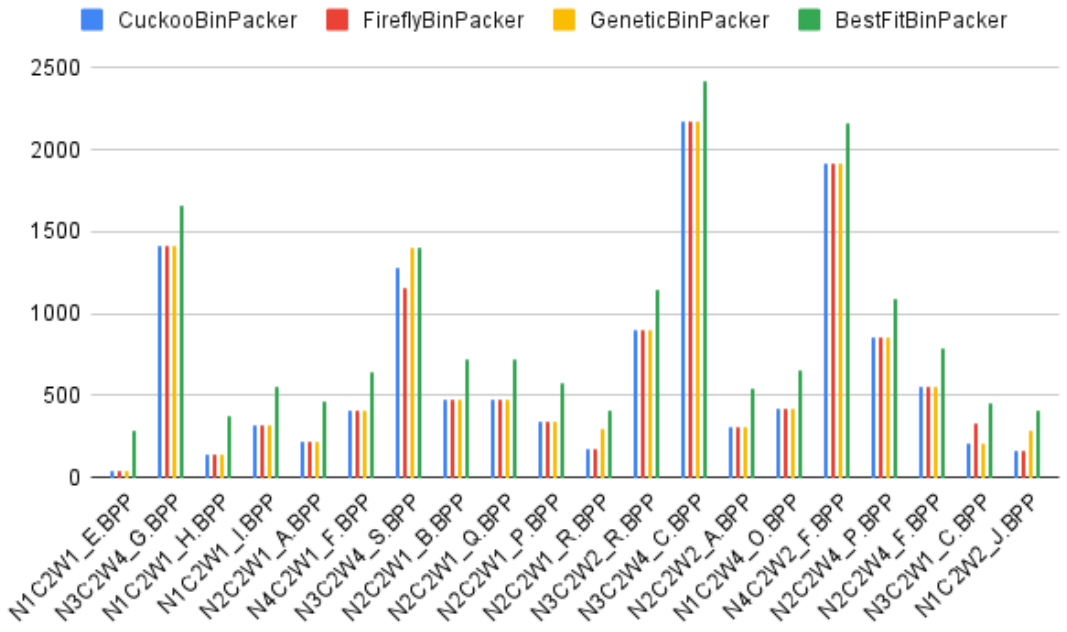
Slika 7.1: Neiskorišten prostor kutija na "teškom" skupu podataka po svakom od algoritama. Jedan *.BPP predstavlja jedan problem.

Slika 7.1 prikazuje neiskorišten prostor na teškom skupu. Problemi prikazani na grafu odabrani su zato što su najbolja rješenja različitih algoritama vrlo raspršena. U ovom slučaju, može se zaključiti da kukavičji algoritam biva najbolji dok algoritam krijesnice završava na drugom mjestu, genetski na trećem i klasična heuristika na očekivanom zadnjem mjestu jer znatno ovisi o poretku elemenata koji su na početku nasumično permutirani.



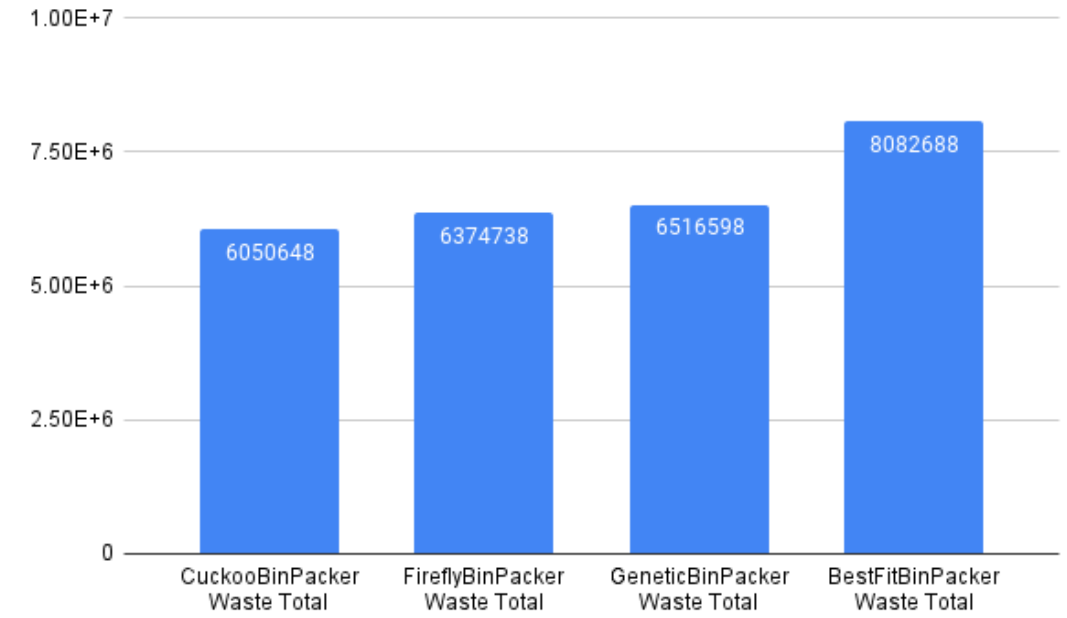
Slika 7.2: Neiskorišten prostor kutija na srednjim i lakšim problemima po algoritmima.

Algoritam krijesnice bio je pobjednik na problemima prikazanim na slici 7.2 dok je kukavičji završio na drugom mjestu, ispred genetskog i klasičnog heurističkog. Također, vidljivo je da na desnom skupu problema, svi algoritmi, osim heurističkog, postižu isti rezultat po čemu se može zaključiti da su ti problemi jednostavniji te da metaheuristički algoritmi ne ovise o početnom redoslijedu elemenata pakiranja.



Slika 7.3: Neiskorišten prostor kutija na lakšim problemima po algoritmima. Prikazuje da kod lakših problema, gotovo nema razlike u rješenjima metaheurističkih algoritama.

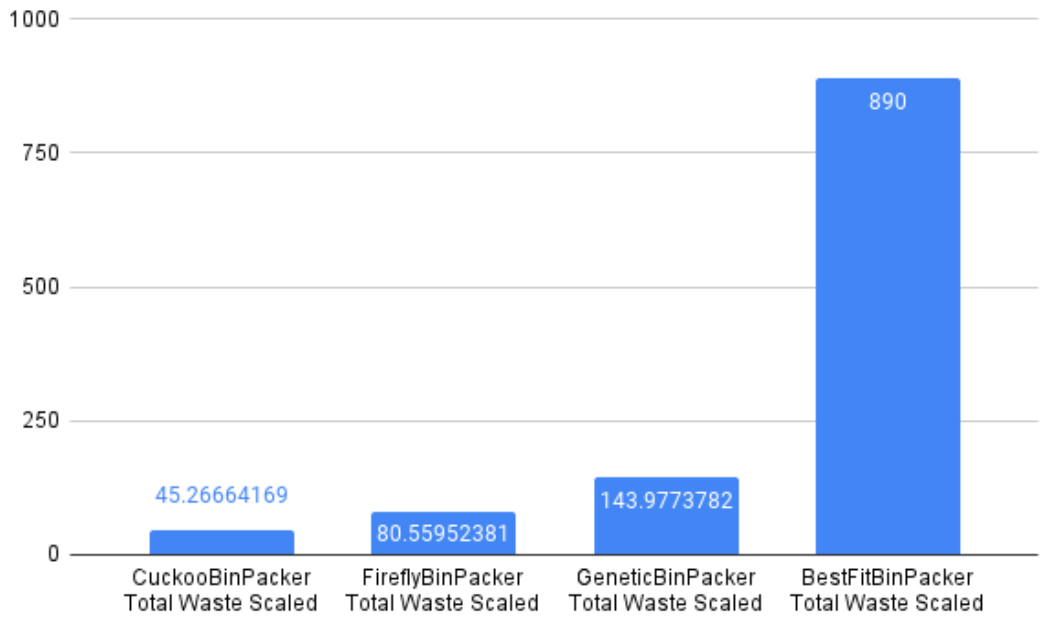
Konačno, graf 7.4 prikazuje ukupan neiskorišten prostor nad svih 1210 problema iz skupa podataka.



Slika 7.4: Neiskorišten prostor nad cijelim skupom problema.

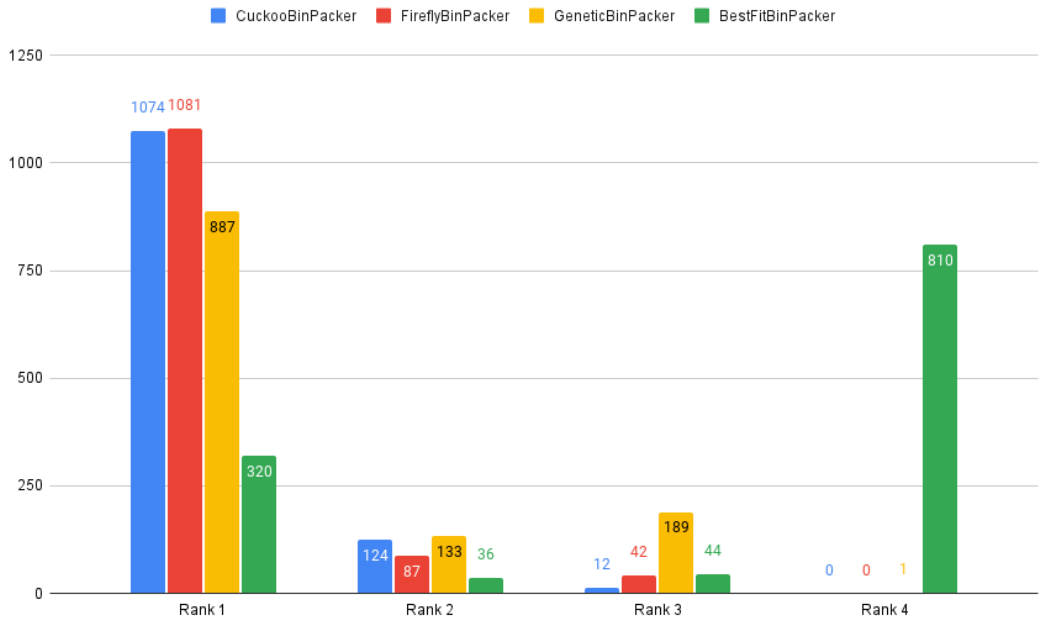
Iz priloženog, moglo bi se zaključiti kako je kukavičji algoritam najbolji za razne

težine problema. No, moguće da tako ispada samo zato što je bio najbolji nad teškim problemima gdje su veličine bile znatno većih magnituda. Slika 7.5 tu teoriju ipak opovrgava. Na slici se vidi i zapravo koliko je heuristički algoritam lošiji nad cijelim skupom problema, gotovo 20 puta od kukavičjeg algoritma. Graf je dobiven tako što se neiskorišten prostor od svakog algoritma prvo min-max skalirao za svaki problem, a zatim zbrojio za sve njih. Skaliranje je napravljeno da bi neiskorišten prostor najboljeg algoritma, za neki problem, ispao 0, najlošijeg 1, a ostalih negdje između ovisno o tome jesu li bliži najboljem ili najlošijem algoritmu. Tako se dobio odnos kumulativno neiskorištenog prostora između algoritama.



Slika 7.5: Ukupan neiskorišten prostor skaliran po problemu s obzirom na minimalni i maksimalni dobiven od svakog algoritma.

Konačno, slika 7.6 prikazuje koliko je puta neki algoritam bio na kojem mjestu po kvaliteti rješenja s obzirom na druge. Ako su svi bili jednako dobri, svakom algoritmu brojač prvog mjesta uvećan je za jedan. Isto je i ako bilo koja dva algoritma dijele mjesto, npr. ako dva algoritma dijele drugo mjesto za neki problem, obojici se brojač drugog mjesta uvećao, dok brojač trećeg mjesta ostaje nepromijenjen.



Slika 7.6: Rangirana rješenja algoritama.

Slika 7.6 pokazuje da je algoritam krijesnice najviše puta rezultirao najboljim rješenjem, ali jedva pobjeđuje kukavičji sa svega 7 pobjeda razlike dok je kukavičji 37 puta više rezultirao drugim najboljim rješenjem.

8. Zaključak

Problem pakiranja kutija vrlo je važan problem kojeg je mnogo znanstvenika pokušalo riješiti. No, on je NP-težak što znači da ga se ne može egzaktno riješiti u izglednom vremenu. Iz tog razloga, nastalo je mnogo algoritama koji ne garantiraju optimalno rješenje, ali imaju cilj naći što bolje rješenje u prihvatljivom vremenu. Nekoliko njih, metaheurističkih, opisano je u ovom radu.

Svaki od navedenih algoritama koristi populaciju nasumično generiranih rješenja koja predstavljaju poredak elemenata kojim se oni trebaju predavati heurističkom algoritmu kako bi ih on što bolje pakirao u kutije fiksne veličine. Bolje pakiranje smatra se što manji broj iskorištenih kutija tj. neiskorištenog prostora.

Iz rezultata vidljivo je da kukavičji algoritam daje najbolja rješenja, u postavljenoj konfiguraciji. Pretpostavka je da je razlog Lévyjev let koji znatno poboljšava pretraživanja i izbacuje rješenja iz lokalnog optimuma. Trebalo bi istražiti mogu li se bolje postaviti parametri navedenih algoritama tako da se postignu bolja rješenja. Možda iskoristiti neki od postojećih algoritama za optimizaciju parametara.

Kod pakiranja kutija postoji još jedan problem, ovaj rad pretpostavlja da će elementi i njihove veličine biti odmah poznate. No, to u stvarnom svijetu nije uvijek slučaj. Npr. u okruženju računalnog oblaka *(engl. Cloud)*, čest je slučaj da se svi elementi i njihove veličine (aplikacije i njeni resursi) ne znaju unaprijed. Aplikacije se gotovo uvijek dodaju jedna po jedna, onda kad je aplikacija napravljena i spremna. A često se i potrebni resursi već postojeće aplikacije moraju povećati zbog količine podataka koju ona obrađuje i sl. U tom slučaju, navedeni metaheuristički algoritmi ne mogu pomoći odmah kad aplikacije dolaze, ali mogu pomoći ako se koriste kao algoritmi za rebalansiranje. Npr. aplikacije dolaze i pakiraju se nekim heurističkim algoritmom, a nakon nekog vremena se pokrene metaheuristički koji će prepakirati jedan podskup aplikacija da resursi budu maksimalno iskorišteni.

LITERATURA

Gerardo Beni i Jing Wang. Swarm intelligence in cellular robotic systems. U Paolo Dario, Giulio Sandini, i Patrick Aebischer, urednici, *Robots and Biological Systems: Towards a New Bionics?*, stranice 703–712, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg. ISBN 978-3-642-58069-7.

Aleksei Chechkin, Ralf Metzler, Joseph Klafter, i Vsevolod Gonchar. *Introduction to the Theory of Lévy Flights*, stranice 129 – 162. 09 2008. ISBN 9783527622979. doi: 10.1002/9783527622979.ch5.

Iztok Fister, Iztok Fister, Xin-She Yang, i Janez Brest. A comprehensive review of firefly algorithms. *Swarm and Evolutionary Computation*, 13:34–46, 2013. ISSN 2210-6502. doi: https://doi.org/10.1016/j.swevo.2013.06.001. URL `https://www.sciencedirect.com/science/article/pii/S2210650213000461`.

Chanaleä Munien, Shiv Mahabeer, Esther Dzitiro, Sharad Singh, Siluleko Zungu, i Absalom Ezugwu. Metaheuristic approaches for one-dimensional bin packing problem: A comparative performance study. *IEEE Access*, PP, 12 2020. doi: 10.1109/ACCESS.2020.3046185.

Armin Scholl i Robert Klein. URL `https://www2.wiwi.uni-jena.de/Entscheidung/binpp/index.htm`.

Xin-She Yang. Nature-inspired metaheuristic algorithms. 2008.

Xin-She Yang. Chapter 5 - genetic algorithms. U Xin-She Yang, urednik, *Nature-Inspired Optimization Algorithms*, stranice 77–87. Elsevier, Oxford, 2014. ISBN 978-0-12-416743-8. doi: https://doi.org/10.1016/B978-0-12-416743-8.00005-1. URL `https://www.sciencedirect.com/science/article/pii/B9780124167438000051`.

Xin-She Yang i Xing-Shi He. Chapter 2 - bat algorithm and cuckoo search algorithm. U Xin-She Yang, urednik, *Nature-Inspired Computation and*

Swarm Intelligence, stranice 19–34. Academic Press, 2020. ISBN 978-0-12-819714-1. doi: https://doi.org/10.1016/B978-0-12-819714-1.00011-7. URL https://www.sciencedirect.com/science/article/pii/B9780128197141000117.

Zendaoui Zakaria i Abdesslem Layeb. *Adaptive Cuckoo Search Algorithm for the Bin Packing Problem*, stranice 107–120. 01 2016. ISBN 978-3-319-33409-7. doi: 10.1007/978-3-319-33410-3_8.

Rješavanje problema pakiranja kutija korištenjem metaheuristika

Sažetak

Problem pakiranja kutija vrlo je rasprostranjen problem, primjenjiv u mnogim situacijama stvarnog života. Rad razmatra jednodimenzionalnu inačicu tog problema s fiksiranom veličinom kutija te navodi algoritme za njegovo rješavanje. Svi spomenuti algoritmi koriste klasični heuristički algoritam u pozadini *(Best-Fit)*, a njihov zadatak bit će optimirati poredak elemenata koji se predaju *Best-Fit* algoritmu kako bi se maksimalno smanjio neiskorišten prostor svih inicijaliziranih kutija. Razmatraju se genetski, kukavičji *(engl. Cuckoo)* i algoritam krijesnice *(engl. Firefly Algorithm)* čiji se rezultati na kraju uspoređuju.

Ključne riječi: pakiranje kutija, metaheuristika, optimum, genetski algoritam, cuckoo, firefly, best fit.

Solving the Bin-Packing Problem Using Metaheuristics

Abstract

Bin packing problem is widely spread and applicable in multiple real-life situations. This paper studies the one-dimensional variant of the problem with fixed-size bins and offers a few algorithms for solving it. All offered algorithms use a classic heuristic algorithm in the background *(Best-Fit)* where their job is to optimize order of elements fed to *Best-Fit* in order to minimize wasted space of initialized bins. Studied algorithms are genetic, Firefly, and Cuckoo Search. Their results are compared at the end of the paper.

Keywords: bin packing, metaheuristic, optimum, genetic algorithm, cuckoo, firefly, best fit.