3D bin packing

Marko Lazarević, Uroš Ivetić

Avgust 2025

Sadržaj

1 Uvod

1.1 Opis problema

Problem trodimenzionalnog pakovanja u kontejnere (3D Bin Packing Problem – 3D-BPP) predstavlja klasičan kombinatorni optimizacioni problem koji se javlja u brojnim praktičnim domenima, poput logistike, skladištenja i transporta. Suština problema ogleda se u potrebi da se skup pravougaonih objekata (*stavki*) rasporedi unutar jednog ili više pravougaonih kontejnera (*binova*), uz striktno poštovanje uslova da se stavke međusobno ne preklapaju i da u potpunosti budu smeštene unutar granica kontejnera. Cilj pakovanja, u okviru našeg rada, jeste **minimizacija broja korišćenih kontejnera**, čime se postiže efikasnija upotreba raspoloživog prostora.[1]

Kako bi problem bio jasno definisan i prilagođen implementaciji, usvojena su sledeća pravila pakovanja:

Pravila za kontejnere

- Kontejneri se moraju puniti **od vrha nadole**, čime se obezbeđuje konzistentan postupak pakovanja.
- Nije dozvoljeno da bilo koja stavka prelazi granice kontejnera, odnosno nijedan deo objekta ne sme da viri van zapremine kontejnera.

Pravila za pozicioniranje kutija

- Sve stavke se pozicioniraju u unapred zadatoj orijentaciji, što znači da rotacije duž koordinatnih osa nisu dozvoljene.

Varijanta problema

U našoj implementaciji razmatrana je **oflajn** verzija problema trodimenzionalnog pakovanja. To podrazumeva da je čitav skup stavki poznat unapred, pre samog procesa pakovanja, čime se omogućava primena složenijih optimizacionih metoda i detaljnija analiza kvaliteta dobijenih rešenja.

2 Opis rešenja

2.1 Reprezentacija podataka i proces pakovanja

Za rešavanje problema trodimenzionalnog pakovanja kutija u kontejnere implementirane su ključne klase `Box` i `Bin`, kao i funkcija `pack_boxes_into_bins`, koje zajedno omogućavaju modelovanje i izvršenje procesa pakovanja.

2.1.1 Klasa Box

Klasa `Box` predstavlja osnovnu jedinicu koja se pakovanjem raspoređuje unutar kontejnera. Svaka kutija opisuje se trodimenzionalno, atributima `width` (širina), `depth` (dubina) i `height` (visina). Pored dimenzija, kutija sadrži i atribut `position`, koji definiše njenu koordinate (x, y, z) unutar `Bin`-a i služi za praćenje rasporeda kutija.

2.1.2 Klasa Bin

Klasa `Bin` modeluje kontejner u koji se kutije smeštaju. Svaki `Bin` definisan je dimenzijama `width`, `depth` i `height`, i sadrži sledeće ključne strukture:

- listu `boxes`, koja sadrži sve kutije već smeštene u kontejner,
- matricu `top_surface`, koja za svaku tačku osnove (x, y) beleži trenutnu visinu zauzetog prostora,
- skup `candidate_positions`, predstavljajući potencijalne pozicije za postavljanje novih kutija.

Centralna metoda klase `Bin` je `add_box`, koja pokušava da postavi kutiju u kontejner prema jednoj od sledećih heuristika [1]:

- **DBL (Deepest Bottom Leftmost)** – prioritetno postavljanje na najnižu i najlevlju poziciju,
- **MC (Maximum Contact)** – izbor pozicije koja maksimizuje kontakt sa već postavljenim kutijama,
- **SE (Smallest Extrusion)** – minimizacija visinskog izbočenja novododate kutije,
- **NS (Neighbour Score)** – maksimizacija broja susednih kutija.

U okviru ovog rada, odabrana je **DBL** strategija, pri čemu su svi rezultati dobijeni korišćenjem ove heuristike. Dodatne pomoćne funkcije, poput `can_fit`, `contact_score`, `z_extrusion_score` i `neighbour_score`, koriste se za proveru validnosti pozicije i ocenjivanje kvaliteta potencijalnog rasporeda.

Metoda `add_box` Ova metoda predstavlja srž pakovanja unutar pojedinačnog `Bin`-a. Njen zadatak je da pronađe validnu poziciju za datu kutiju i, ukoliko je pronalazak moguć, ažurira stanje kontejnera. Proces rada obuhvata:

1. filtriranje kandidata iz skupa `candidate_positions` na osnovu dimenzija kutije i kontejnera,
2. proveru visinskih ograničenja $(z + h)$ i minimalne visine potrebne za stabilno postavljanje,
3. provere sudara sa postojećim kutijama pomoću metode `can_fit`,
4. izbor najbolje pozicije prema definisanoj heuristici,
5. ažuriranje liste `boxes`, matrice `top_surface` i poziva metode `_update_candidates` radi osvežavanja potencijalnih pozicija.

Ako nijedna pozicija nije validna, metoda vraća `False`, u suprotnom `True`.

Metoda `can_fit` Proverava mogućnost postavljanja kutije na poziciju (x, y, z) kroz tri koraka:

1. **Granice kontejnera:** osigurava da kutija ostaje unutar širine, dubine i visine `Bin`-a,
2. **Kolizija sa postojećim kutijama:** detektuje preklapanje u sve tri dimenzije,
3. **Podrška osnove:** osigurava da sve tačke osnove kutije leže na ili iznad `top_surface`.

Metoda `_update_candidates` Ažurira skup potencijalnih pozicija za dalja pakovanja:

1. uklanja koordinate unutar osnove novopostavljene kutije,
2. dodaje nove pozicije neposredno uz ivice kutije, unutar granica kontejnera.

2.1.3 Funkcija `pack_boxes_into_bins`

Funkcija `pack_boxes_into_bins` implementira glavni proces pakovanja:

- uzima listu kutija, dimenzije kontejnera i strategiju pakovanja,
- iterativno pokušava postavljanje svake kutije u postojeće kontejnere,
- u slučaju neuspeha otvara novi `Bin` i dodaje kutiju u njega,
- vraća kompletan skup kontejnera sa raspoređenim kutijama.

2.2 Algoritam grube sile

2.2.1 Generalni opis algoritma

Brute-force metod rešavanja problema pakovanja zasniva se na iscrpnom pretraživanju svih mogućih rasporeda kutija. Za svaku permutaciju ulaznog skupa:

1. generiše se nova permutacija kutija,
2. proverava se raspored kutija u kontejnerima,
3. ako je rešenje bolje (manji broj kontejnera), ono se pamti kao optimalno.

Prednost ovog pristupa je garancija pronalaska globalnog optimuma, dok je glavna slabost ekstremno visoka složenost $O(n!)$.

2.2.2 Naša primena

U našoj implementaciji, funkcija `brute_force` koristi prethodno definisane funkcije za pakovanje (`pack_boxes_into_bins` i `pack_boxes_into_bins_interactive`) i vraća najbolje rešenje u smislu minimalnog broja korišćenih kontejnera. Ovaj pristup omogućava verifikaciju heurističkih metoda za male instance problema.

Primer upotrebe U nastavku je dat primer korišćenja `brute_force` funkcije za pet kutija i kontejnere dimenzija $3 \times 3 \times 3$:

```
boxes = [
    Box(1, 1, 1),
    Box(1, 1, 1),
    Box(1, 1, 1),
    Box(3, 3, 1),
    Box(3, 3, 1),
]

bin_size = (3, 3, 3)
bins = brute_force(boxes, bin_size, interactive=False)

print(f"Number of bins used: {len(bins)}")
for i, bin in enumerate(bins, 1):
    print(f"Bin {i}: {bin}")
```

Na sledećoj slici 1 prikazana je vizuelizacija rešenja grubom silom primera navedenog iznad.

Slika 1: Primer optimalnog pakovanja dobijenog algoritmom grube sile.

2.3 Algoritam simuliranog kaljenja

2.3.1 Generalni opis algoritma

Simulirano kaljenje (*Simulated Annealing*) predstavlja metaheuristički pristup inspirisan fizičkim procesom kaljenja metala. Osnovna ideja algoritma je da tokom pretrage prostora rešenja bude dopušteno prihvatanje i onih rešenja koja privremeno pogoršavaju vrednost ciljne funkcije. Na taj način se smanjuje rizik od zarobljavanja u lokalnim optimumima. Tokom izvršavanja algoritma, verovatnoća prihvatanja lošijih rešenja postepeno opada, analogno postepenom smanjenju temperature u procesu kaljenja.

Intenzifikacija se ostvaruje fokusiranjem pretrage na oblasti prostora rešenja u kojima se nalaze najbolje permutacije kutija. Algoritam sistematski prihvata bolja rešenja i sve ređe lošija rešenja, čime se povećava kvalitet konačnog rešenja. **Diverzifikacija** je obezbeđena stohastičkim prihvatanjem lošijih rešenja u početnim iteracijama, što omogućava istraživanje šireg prostora rešenja i smanjuje rizik od prerane konvergencije u lokalni minimum.

2.3.2 Naša primena

U okviru ovog rada, simulirano kaljenje primenjeno je na problem trodimenzionalnog pakovanja kutija u kontejnere. Početno rešenje generiše se slučajnom permutacijom kutija,

nakon čega se u svakoj iteraciji primenjuju operacije nad susedstvom (*neighbor operations*) koje menjaju redosled kutija.

- **Intenzifikacija:** svaka nova permutacija koja daje bolje rešenje uvek se prihvata. Time algoritam sistematski poboljšava postojeće rešenje i usmerava pretragu ka oblastima sa perspektivnim rasporedima kutija.
- **Diverzifikacija:** realizuje se kroz verovatnoću prihvatanja lošijeg rešenja $q = \frac{1}{it}$, gde it označava trenutnu iteraciju. U ranim fazama pretrage (male vrednosti it) veća je verovatnoća prihvatanja lošijih rešenja, dok se tokom napredovanja algoritma verovatnoća smanjuje, prelazeći iz faze istraživanja u fazu fokusirane eksploatacije.

Konačni rezultat algoritma predstavlja najbolje pronađeno rešenje, odnosno raspored kutija u kontejnere koji minimizuje broj korišćenih kontejnera i/ili maksimizuje iskorišćenost dostupnog prostora.

3 Eksperimentalni rezultati

3.1 Generisanje setova podataka

Za potrebe evaluacije i testiranja algoritama za trodimenzionalno pakovanje kutija u kontejnere, implementiran je postupak *slučajnog generisanja* kutija unutar definisanog kontejnera. Ideja je da se od jedne početne kutije datih dimenzija postepeno vrše nasumična deljenja na manje kutije, pri čemu se kontroliše ukupan broj generisanih objekata i očuvanje zapremine.

3.1.1 Funkcija `try_split`

Funkcija `try_split` pokušava da podeli datu kutiju duž odabrane ose (x, y ili z), stvarajući niz novih kutija koje zajedno popunjavaju zapreminu originalne. Osnovni principi njenog rada su:

- Postoji minimalna dozvoljena dimenzija (≥ 1) ispod koje kutija ne može biti podeljena.
- Nasumično se bira broj preseka (između 1 i $\min(5, \texttt{max\_splits})$), koji određuje koliko novih segmenata će nastati.
- Tačke preseka unutar dimenzije odabrane ose biraju se slučajno. Ako neki segment bude manji od minimalne dimenzije, pokušaj deljenja se odbacuje.
- U slučaju uspešnog deljenja, generišu se nove kutije koje zajedno popunjavaju zapreminu originalne kutije.

3.1.2 Funkcija `split_bin_randomly`

Funkcija `split_bin_randomly` koristi `try_split` kako bi generisala kompletan skup kutija unutar jednog kontejnera. Postupak funkcije je definisan sledećim koracima:

1. Inicijalno se ceo kontejner tretira kao jedna velika kutija.

2. U ograničenom broju pokušaja (do 20) nasumično se bira kutija i pokušava njeno deljenje.
3. Osa deljenja bira se proporcionalno veličini dimenzije kutije, pri čemu veće dimenzije imaju veću verovatnoću da budu izabrane.
4. Ako deljenje uspe, nove kutije se dodaju u skup, uz proveru da ukupan broj kutija ne premašuje vrednost `max_boxes`.
5. Ako deljenje nije moguće, kutija se vraća u prvobitnom obliku.

Rezultat ovog postupka predstavlja listu kutija različitih dimenzija, koja služi kao ulaz za testiranje algoritama, uz mogućnost procene minimalnog broja kontejnera potrebnog za njihovo pakovanje.

3.1.3 Funkcija `write_boxes_to_file`

Funkcija `write_boxes_to_file` omogućava trajno čuvanje generisanih kutija u tekstualni fajl, radi reproduktivnosti eksperimenata i jednostavne evaluacije. Format zapisa uključuje:

- U prvoj liniji upisuju se dimenzije kontejnera (`Bin`).
- Za svaku kutiju beleže se njene dimenzije: `width`, `depth` i `height`.
- Na kraju fajla dodaje se separator i očekivani minimalni broj kontejnera potreban da se sve kutije spakuju.

3.2 Rezultati brute force algoritma

Testove smo podelili u pet kategorija, `noob`, `easy`, `moderate`, `medium` i `hard`.
Važe sledeći parametri:

Noob

```
num_examples = 30
num_bins_range = (1,2)
size_range = (2,4)
max_boxes_range = (2,3)
```

Easy

```
num_examples = 30
num_bins_range = (1,2)
size_range = (5,10)
max_boxes_range = (2,3)
```

Moderate

```
num_examples = 30
num_bins_range = (3,4)
size_range = (5,10)
max_boxes_range = (2,4)
```

Medium

```
num_examples = 30
num_bins_range = (5,20)
size_range = (5,30)
max_boxes_range = (2,30)
```

Hard

```
num_examples = 30
num_bins_range = (20,200)
size_range = (30,300)
max_boxes_range = (2,1000)
```

Parametar `num_examples` je parametar koji određuje broj generisanih setova, `num_bins_range` određuje minimalan i maksimalan ukupan broj kontejnera u koje treba spakovati kutije, `size_range` daje minimalne i maksimalne dimenzije kontejnera po svakoj osi, dok `max_boxes_range` određuje minimalan i maksimalan broj kutija koje treba dobiti sečenjem.

Testiranje je sprovedeno nad **30** nasumično generisanih test primera svake težine i mereno je vreme kao i tačnost.

Algoritam grube sile mogli smo da primenimo samo na `noob` i `easy`, već na `moderate` primerima vreme izvršavanja bilo je nepraktično dugačko, odnosno 180 minuta, dok za `medium` primere kôd koji smo pokrenuli nije uspeo da se izvrši u iole razumnom vremenu, a naše očekivanje jeste da bi samo izvršavanje trajalo **više od sedam dana.**

Vreme izvršavanja za `noob` i `easy` primere bilo je 0.1 sekunda i 1.9 sekundi, redom.

3.3 Rezultati simuliranog kaljenja

Pristup simuliranim kaljenjem pokazao je značajno bolje performanse u odnosu na druge metode. Naime, test primeri označeni kao `noob` i `easy` izvršeni su za 1.1 sekundu, odnosno 5 sekundi. Za `moderate` test primere vreme izvršenja iznosilo je 28 sekundi, dok su `medium` primeri obrađeni za 60 sekundi.

Simulirano kaljenje je proizvelo očekivani broj kontejnera za `noob`, `easy` i `moderate` test primere. Za `medium` test primere, prosečna odstupanja u odnosu na očekivani broj kontejnera iznosila su dva dodatna kontejnera. Ovi rezultati mogu se smatrati zadovoljavajućim u praktičnim primenama, s obzirom na složenost problema i znatno kraće vreme izvršenja u poređenju sa egzaktnim algoritmom.

Broj iteracija nad kojima je simulirano kaljenje radilo je **100.**

4 Zaključak

4.1 O rezultatima

Na osnovu dobijenih rezultata može se zaključiti da je za veoma male test primere povremeno efikasnije koristiti algoritam grube sile, dok se heurističke metode, kao što je simulirano kaljenje, nameću kao neophodne čim složenost problema počne da raste. Simulirano kaljenje omogućava pronalaženje rešenja bliskih optimalnom uz znatno kraće vreme izvršenja, što ga čini praktično primenjivim za realne instance problema.

4.2 Moguća unapređenja i optimizacije

Analizom implementiranog algoritma za generisanje setova podataka i postupka pakovanja, moguće je identifikovati nekoliko oblasti u kojima se performanse i efikasnost mogu dodatno poboljšati:

4.2.1 Efikasnija evaluacija funkcije cilja

Trenutna implementacija zahteva potpuno ponavljanje pakovanja svih kutija za svaku permutaciju, što ima složenost reda $O(n^2)$. Ova operacija može se optimizovati:

- Korišćenjem inkrementalnog računanja fitness funkcije. Pošto funkcija `neighbor_operations` generiše permutacije koje su u bliskom susedstvu trenutne permutacije, nije potrebno ponovo pakovati sve kutije od početka. Dovoljno je evaluirati samo one kutije na koje promena direktno utiče.

4.2.2 Pametnije biranje kontejnera za pakovanje

Trenutna strategija iterativnog pokušaja pakovanja može dovesti do situacije u kojoj se kutija smešta u kontejner sa ograničenim preostalim prostorom, što može onemogućiti smeštaj drugih kutija.

- Alternativni pristup podrazumeva prvo biranje kontejnera sa najvećom preostalom zapreminom ili sa najvećim dostupnim prostorom duž određene ose. Ovaj pristup može poboljšati iskorišćenost prostora i smanjiti broj potrebnih kontejnera.

Literatura

[1] Sam D. Allen. „Algorithms and Data Structures for Three-Dimensional Packing”. Deposited: 05 September 2012; Last modified: 28 February 2025. PhD thesis. University of Nottingham, 2011. URL: https://eprints.nottingham.ac.uk/12779/.