SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2853

Rješavanje problema realokacije kontejnera korištenjem metaheuristika

Domagoj Lokner

Zagreb, rujan 2022.

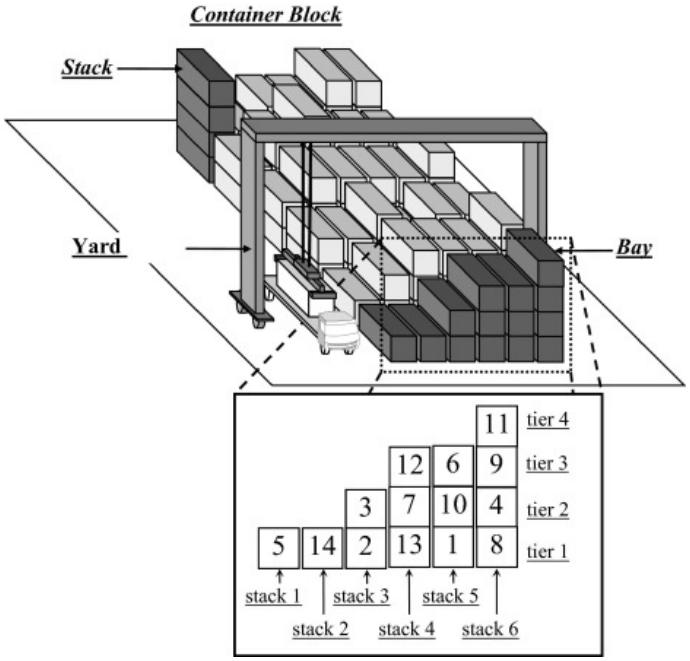
SADRŽAJ

1. Uvod

Gotovo 90% tereta internacionalnog transporta prolazi kroz terminale u lukama za pristanak brodova. Terminali su veliki logistički centri koji služe kao točka u kojoj je teret privremeno skladišten prije nego je transportiran na svoje sljedeće odredište. Optimiranjem procesa u terminalima možemo znatno smanjiti vrijeme i energiju potrebnu za transport tereta (Jovanovic et al., 2019). Na efikasnost transporta utječu mnogi čimbenici kao što su organizacija pristanka brodova u luci, skladištenje kontejnera u kontejnerskim terminalima i njihov utovar na teretne brodove, vlakove ili kamione. Za transport dobara najčešće su korišteni kontejneri, velike metalne kutije koje čuvaju teret od mogućih oštećenja te omogućuju visoku produktivnost. Ograničeni prostor s kojim raspolažemo u terminalima rezultira problemom skladištenja kontejnera. Iz toga razloga su kontejneri složeni jedni na druge te način na koji su raspoređeni određuje efikasnost njihovog prosljeđivanja u daljnje korake transporta. Optimiranje ovoga procesa je vrlo važno te je uzrokovalo porast broja radova koji proučavaju raspodijele i operacije dohvata kontejnera unutar terminala (Gulić et al., 2018).

2. Problem relokacije kontejnera

Problem relokacije kontejnera (engl. *container relocation problem*) također je poznat i kao problem relociranja blokova je NP-težak problem, što znači da najbolje rješenje s porastom veličine problema ne može biti određeno u realnom vremenu. Iz toga razloga razvijaju se metode optimiranja problema kako bi pronašli rješenja koja su zadovoljavajuća (Gulić et al., 2018). Kontejnerski terminali su organizirani u ukrcajne odjeljke ili redove kontejnera (engl. *bay*), a svaki odjeljak se sastoji od više kontejnera složenih u stogove (engl. *stack*). Broj odjeljaka najčešće je manji od 20, a u svakom odjeljku je od 3 do 7 stogova kontejnera (Zhu et al., 2012). Primjer jednog kontejnerskog terminala je prikazan slikom 2.1. Broj stogova u odjeljku, isto kao i maksimalan broj kontejnera u stogu, je ograničenje koje čini ovaj problem izazovnim i teškim.



Slika 2.1: Primjer kontejnerskog terminala (Ting i Wu, 2017)

Kontejneri se dohvaćaju i premještaju uz pomoć dizalice. Optimiziranjem rada dizalice smanjujemo vrijeme koje odvozno vozilo mora čekati pri utovaru dobara. Tako-

đer, optimiranjem akcija smanjujemo utrošak energije potreban za upravljanje dizalicom. Ovaj rad će se upravo fokusirati na optimiranje procesa rada dizalice, konkretno, optimiranje utroška energije, pri organizaciji i dohvatu kontejnera.

2.1. Vrste problema

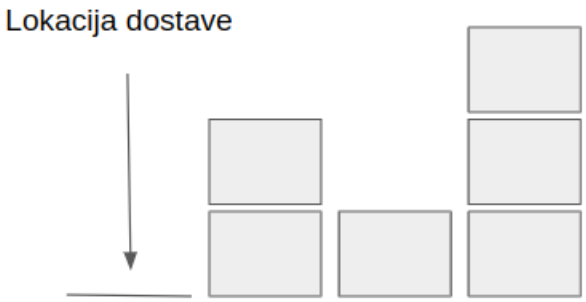
Problem realociranja kontejnera se može podijeliti na različite podtipove. Prva podjela uzima u obzir broj odjeljaka te tako razlikujemo probleme s jednim odjeljkom (engl. *single bay*) i probleme s više odjeljaka (engl. *multi bay*). Problemi s više odjeljaka moraju uzeti u obzir i vrijeme koje je potrebno dizalici da prijeđe na drugi odjeljak te naravno, imaju više mogućih izbora za realociranje kontejnera te ih to čini kompleksnijima i težima za optimirati.

Sljedeća podjela se odnosi na prioritet kontejnera. Kontejneri moraju biti dohvaćeni prema određenom redoslijedu kako je to zatraženo od strane koja preuzima dobra. Kontejneri mogu sačinjavati prioritetne grupe ili mogu imati točno pridijeljene prioritete. U slučaju prioritetnih grupa, nije važan redoslijed preuzimanja kontejnera unutar iste grupe, dok u drugome slučaju jasno je određen redoslijed kojim ćemo dohvaćati svaki pojedini kontejner.

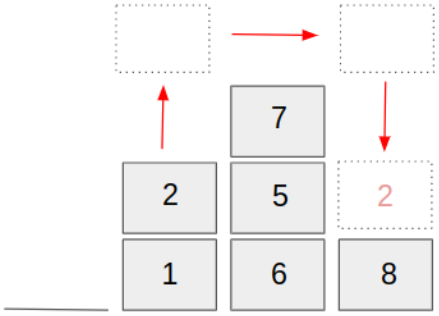
Još jedna moguća podjela je na probleme u kojima je prioritet dohvaćanja kontejnera predodređen te je moguće reorganizirati položaj kontejnera prije no što trebaju biti preuzeti kako bi preuzimanje bilo brže i efikasnije. U drugome slučaju, redoslijed dohvaćanja kontejnera poznat je tek u trenutku kada oni trebaju biti dohvaćeni te realokaciju izvodimo u isto vrijeme kada i preuzimanje kontejnera.

U ovome radu proučavat ćemo probleme s jednim odjeljkom koji imaju prioritete određene za svaki kontejner te su oni poznati tek u trenutku preuzimanja tereta.

Dodatnu podjelu moguće je napraviti i na razini algoritma koji će se koristiti pri dohvatu kontejnera. Ovaj rad podrazumijeva da se realocirati mogu samo kontejneri koji se nalaze neposredno iznad kontejnera najvećeg prioriteta. Također, u svrhu istraživanja pretpostavit ćemo da se preuzeti kontejneri moraju dostaviti na poziciju koja je neposredno lijevo u odnosu na odjeljak iz kojeg se kontejneri preuzimaju, točnije dostavnu poziciju možemo predočiti kao stog na poziciji 0. Pozicija dohvaćanja je prikazana slikom 2.2.



Slika 2.2: Primjer proučavanog problema



Slika 2.3: Primjer relokacije kontejnera

2.2. Optimiranje rada dizalice

Kako je zadaća ovoga rada opisati postupke kojima možemo optimirati rad dizalice u kontejnerskim terminalima opišimo ukratko kako ćemo računati cijenu akcije dizalice. Dizalica se može pomicati na tri načina prema gore, dolje ili horizontalno. Približimo ovo primjerom opisanim slikom 2.3. Kontejner označen brojem 1 je najvećeg prioriteta, kako bi ga dohvatili kontejner 2 mora biti realociran. Ako je naš algoritam odabrao treći stog kao odredišnu lokaciju za realociranje kontejnera 2 dizalica će napraviti sljedeće korake, kontejner će se podići za dva mjesta kako bi mogao proći iznad stoga tri, putovat će još dva koraka horizontalno kako bi došao iznad odredišnog stoga te će se po tome morati spustiti za dva mjesta na kontejner broj 8. Primijetimo kako će se svaka realokacija sastojati od samo ove tri operacije te kako će kontejner uvijek biti podignut do minimalne razine na kojoj može nesmetano obaviti horizontalni pomak.

Važno je napomenuti kako u ovome radu masu kontejnera nećemo uzimati u obzir

iako je očito očekivati da će ona doprinositi količini potrošene energije. Za izračunavanje potrošnje energije broj pomaka ćemo pomnožiti s koeficijentima koji će procijeniti stvarnu potrošnju dizalice za pridijeljene akcije. Koeficijenti su direktno preuzeti iz rada (**?**). Broj pomaka dizalice prema gore ćemo pomnožiti sa 0.90, horizontalne pomake sa 0.08, a broj spuštanja sa 0.02. U konačnici funkcija koja će predstavljati potrošnju dizalice će izgledati ovako:

$$cost = 0.9 * h + 0.08 * w + 0.02 * l$$

Varijabla	Opis
h	broj pomaka dizalice prema gore
w	broj horizontalnih pomaka dizalice
l	broj pomaka dizalice prema dolje

Tablica 2.1: Varijable funkcije potrošnje

3. Algoritmi

U ovome radu pristupit ćemo optimiranju potrošnje dizalice implementacijom tri različita algoritamska pristupa. U sljedećim poglavljima opisat ćemo sva tri pristupa te ćemo po tome usporediti rezultate na stvarnim instancama problema, to jest izračunati ćemo procjenu potrošene energije za realokacije predložene svakim pojedinim algoritmom.

3.1. Heuristički algoritmi

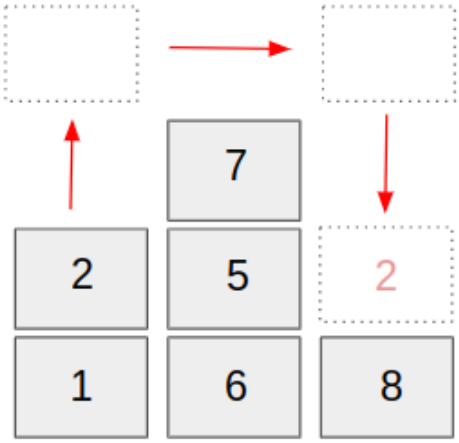
U ovome radu implementirano je nekoliko heurističkih metoda. Osnovna ideja svih metoda je da uz pomoć heurističke funkcije ocijenimo svaku moguću realokaciju kontejnera te odabiremo onu koja ima najbolju vrijednost. Već je jasno da neke heurističke funkcije mogu rezultirati s dvije ili više opcija koje su ocijenjene kao najbolje. Za obradu takvih slučajeva i donošenje konačne odluke moguće je koristiti neke od sljedećih metoda, nasumičan odabir, odabir najbližeg stoga ili je moguće izračunati cijenu svake realokacije te izabrati onu najjeftiniju. U ovome radu iz grupe rješenja odabranih heurističkim algoritmom konačno rješenje odabrano je slučajnim odabirom.

Najniža pozicija - ova heuristika će najbolje ocijeniti najnižu moguću poziciju na koju možemo realocirati kontejner, to jest bit će izabran stog koji trenutno ima najmanje kontejnera. Ovaj postupak je u grafovima referenciran kao *TLP* (engl. *to lowest position*). Jedan korak *TLP* algoritma prikazan je slikom 3.1. Kontejner 2 mora biti realociran kako bismo mogli dostaviti kontejner 1. Treći stog ima dva kontejnera manje nego drugi stog te će stoga *TLP* algoritam odabrati treći stog kao odredište realokacije.

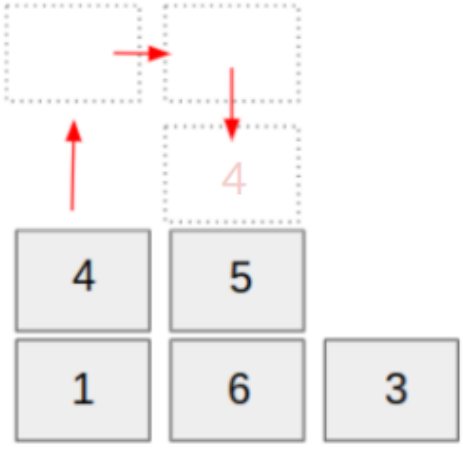
Reshuffle Index - heuristika koja će nastojati premjestiti kontejner na stog tako da pri tome blokira što manje prioritetnijih kontejnera. Drugim riječima, kontejner će biti realociran na stog koji ima najmanji broj kontejnera s većim prioritetom od realociranog kontejnera. Slikom 3.2 prikazan je jedan korak *Reshuffle Index* algoritma. U ovome slučaju kako bismo dohvatili kontejner jedan moramo realocirati kontejner

4. Drugi stog ima više kontejnera u usporedbi s trećim stogom, no na trećem stogu je skladišten jedan kontejner većeg prioriteta dok su na drugom stogu samo kontejneri nižeg prioriteta. Kako će *Reshuffle Index* algoritam odabrati stog sa što manjim brojem prioritetnih kontejnera u konačnici će kontejner 4 biti prebačen na drugi stog.

Pohlepna heuristika - heuristika koja će izračunati potrošnju energije za pomake dizalice u slučaju realociranja na svaki od stogova koji može prihvatiti kontejner te će izabrati realokaciju koja nosi najnižu potrošnju.



Slika 3.1: Primjer realociranja na najnižu poziciju



Slika 3.2: *Reshuffle Index* primjer

3.2. GRASP

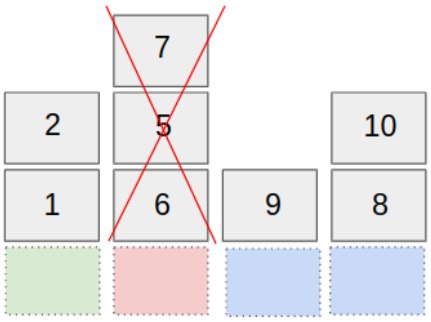
GRASP (engl. *greedy randomised adaptive search algorithm*) je metaheuristički algoritam koji je često upotrebljavan za rješavanje problema realociranja kontejnera (Diaz i Riff, 2016). Glavna ideja iza *GRASP* algoritma je da jednu od pohlepnih heuristika spomenutih u prošlome poglavlju proširimo uvođenjem nasumičnosti te dobiveno rješenje pokušavamo optimirati lokalnom pretragom (Jovanovic et al., 2019). Motivacija za ovakav pristup je da heuristička rješenja imaju dobre odluke pri realokaciji, no u nekim trenutcima odstupanje od takve odluke može rezultirati boljim konačnim rješenjem. *GRASP* algoritam možemo podijeliti u dvije faze, konstrukcijsku i fazu lokalne pretrage.

Algoritam 1 GRASP algoritam

$najbolje\_rjesenje = 0$
while maxIter > 0 **do**
    $max\_iter- = 1$
    $rjesenje = konstrukcijska\_faza()$
    $rjesenje = lokalna\_pretraga(rjesenje)$
    if $rjesenje > najbolje\_rjesenje$ **then**
        $najbolje\_rjesenje \leftarrow rjesenje$
    end if
end while

3.2.1. Konstrukcijska faza

U konstrukcijskoj fazi pokrećemo jedan od heurističkih algoritama. U ovoj fazi želimo uvesti nasumičnost u algoritam te ćemo stoga uvesti hiperparametar k . Modificirana heuristička metoda vratit će k najboljih rješenja te će slučajnim odabirom jedno od rješenja biti odabrano kao konačno. Proučimo pobliže ovu fazu na primjeru slike 3.3. Moramo realocirati kontejner 2 kako bismo mogli pristupiti kontejneru 1. Jedan korak konstruktivne faze u kojoj koristimo *TLP* heuristiku i hiperparametar k postavljen na vrijednost 2 izgledao bi ovako. Stogovi na koje je dopušteno premještanje su stogovi rednih brojeva 2, 3 i 4 te su im heuristikom redom pridijeljene vrijednosti 3, 1 i 2. Pošto je parametar $k = 2$, kao moguća rješenja ostaju samo stogovi 3 i 4. Idući korak je nasumično odabrati jedan od navedena dva stoga.



Slika 3.3: Primjer jednog koraka konstrukcijske faze za *TLP* heuristiku

3.2.2. Faza lokalne pretrage

Faza lokalne pretrage pokušava detektirati točke u rješenju dobivenom u konstruktivnoj fazi gdje je mogla biti donesena odluka koja će rezultirati boljim rješenjem. U ovome radu ćemo za detekciju ovih točaka iskoristiti metodu predloženu u radu Cifuentes i Riff (2020). U rješenju konstruktivne faze ćemo pronaći kontejner koji je realociran najveći broj puta. Te ćemo u fazi lokalne pretrage krenuti od zadnje realokacije toga kontejnera te procijeniti sve ostale moguće opcije u tome koraku te odabrati onu najjeftiniju. Jasno je da ostatak rješenja koji je slijedio nakon izmijenjenog koraka može postati nemoguć te zbog toga moramo imati algoritam oporavka, u tu svrhu možemo izabrati opet jedan od heurističkih algoritama. Kada smo stvorili novi prijedlog rješenja vršimo usporedbu s trenutnim i zadržavamo ono bolje. Nakon toga nastavljamo sa sljedećom realokacijom u nizu. Nakon što smo pretražili sve realokacije najčešće realociranog kontejnera odabiremo idući kontejner koji ima najduži lanac realokacija. Ovaj proces se ponavlja dok ne prođemo najveći određen broj iteracija ili pretražimo sve realokacije svih kontejnera.

3.3. GRH

GRH (engl. *global retrieval heuristic*) je heuristička metoda koju odlikuju parametri koje korisnik može definirati i tako utjecati na ponašanje algoritma. Svi parametri su brojevi između 0 i 1 te mogu biti ručno uneseni ili optimirani nekim optimizacijskim postupkom. Funkcija kazne izračunata je za svaku moguću realokaciju te je u konačnici izabrana realokacija s najmanjom pridruženom vrijednošću kazne. Definirajmo formalnije parametre korištene za izračun funkcije kazne (Hussein i Petering, 2012).

Objasnimo ukratko parametre algoritma. Već spomenuti parametri koji generalno opisuju instance problema i njihova ograničenja navedeni u tablici **Tablica 3.1**. Parametri *GRH* algoritma koji određuju način ponašanja algoritma i naglašuju preferirane karakteristike realokacije su opisani tablicom **Tablica 3.2**. Upravo te parametre ćemo optimirati kako bi algoritam donosio najbolje odluke. Tablicom **Tablica 3.3** su definirani parametri koje će algoritam određivati za sve moguće realokacije, a njihov utjecaj će biti naglašen ili oslabljen varijablama odluke iz tablice **Tablica 3.2** (Hussein i Petering, 2012).

Konačno funkcija kazne će biti izračunata formulom **Slika 3.4**.

$$penalty = \alpha(\frac{h_s}{maxHeight}) + \beta(\frac{l_s}{maxHeight}) + \ gamma(\frac{x_s}{S}) +$$
$$+\delta r_s + \varepsilon r_s(\frac{c - t_s}{C}) + \eta(1 - r_s)g_s + \Theta(\frac{k_s}{S}) + \mu(\frac{n_s}{maxHeight})$$

Slika 3.4: Funkcija kazne za *GRH* algoritam

Tablica 3.1: Inicijalne varijable problema

Varijabla	Opis
S	broj stogova
T	broj kontejnera na svakom stogu
C	ukupan broj kontejnera ($= S * T$)
maxHeight	najveći dopušten broj kontejnera na stogu

Tablica 3.2: Varijable odluke *GRH* algoritma

Varijabla	Opis
α	važnost minimiziranja podizanja
β	važnost minimiziranja spuštanja
γ	važnost minimiziranja horizontalnog pomaka
δ	važnost minimiziranja ponovnog realociranja
ε	važnost odgađanja ponovnog realociranja
η	*tijesnost* - koliko je kontejner blizu najmanjem kontejneru na stogu s
Θ	važnost realociranja što bliže mjestu istovara
μ	važnost održavanja stogova niskima

Tablica 3.3: Ulazne varijable *GRH* algoritma

Varijabla	Opis
h_s	broj podizanja potrebnih za realokaciju na stog s
l_s	broj spuštanja potrebnih za realokaciju na stog s
x_s	broj horizontalnih pomaka potrebnih za realokaciju na stog s
r_s	postoji li kontejner većeg prioriteta na stogu s (binarna varijabla)
t_s	kontejner s najnižim indeksom na stogu s
g_s	*tijesnost*$= (t_s - c - 1)/C$ (c - indeks kontejnera koji se realocira)
k_s	broj pomaka dalje od mjesta utovara ($= 0$ ako se približavamo)
n_s	broj kontejnera na stogu s

3.4. Genetski algoritam

Izazov *GRH* algoritma je što je teško odrediti optimalne parametre te oni mogu biti značajno različiti za različite veličine instanci problema. Ovome problemu ćemo pokušati doskočiti uporabom genetskog algoritma.

Genetski algoritam je adaptivni prirodom inspiriran metaheuristički algoritam. Ideja algoritma zasnovana je na prirodnom postupku evolucije, pri čemu najjače jedinke opstaju i dobivaju priliku prosljeđivati svoj gen u iduće generacije. U pravilu, genetski algoritam započinje sa slučajnim odabirom početne populacije. Svaka jedinka u populaciji predstavljena je kromosomom koji reprezentira karakteristike te jedinke. Jedinke su evaluirane te ovisno o toj vrijednosti probabilistički se određuje vjerojatnost svake jedinke da uđe u križanje s drugim jedinkama. Križanjem stvaramo nove jedinke koje će u konačnici formirati novu generaciju. Iz tog razloga novije generacije bi trebale imati u prosjeku bolje rezultate pri evaluaciji. Kako bi osigurali da algoritam ne zastane u lokalnom optimumu uvodimo neke stohastičke metode u procesima križanja i mutacije, a kako bismo osigurali da ne izgubimo najbolje jedinke iz trenutne generacije često ih nepromijenjene prosljeđujemo u novu generaciju, taj postupak je nazvan elitizmom (Kumar et al., 2010).

Parametre funkcije kazne *GRH* algoritma ćemo pokušati naučiti uz pomoć genetskog algoritma. Kromosom će formirati lista vrijednosti između 0 i 1, to jest parametri iz tablice **Tablica 3.2**. Populaciju će sačinjavati N kromosoma s nasumično pridijeljenim vrijednostima. Svaki kromosom u populaciji evaluira se na n slučajno generiranih instanci problema veličine $T \times S$. Upravo iz ovog razloga ćemo naučiti po jednu kolekciju parametara za svaku od mogućih veličina problema. Konačna evaluacija rješenja je izračunata kao suma svih zasebnih evaluacija na n instanci problema. Ispitivanjem na više različitih instanci nastojimo ostvariti objektivnost ocjene rješenja. Nakon evaluacije stvaramo novu generaciju, prvi dio nove generacije sačinjavat će N_e najboljih jedinki iz trenutne populacije, N_r jedinki ćemo generirati slučajno kao i jedinke iz prve generacije, a ostale jedinke ćemo dobiti križanjem. N_p najboljih jedinki će biti odabrani kao roditelji za križanje. Jedinke ćemo križati tako da izaberemo nasumičan broj koji se nalazi između brojeva koji odgovaraju roditeljskim genima. Ovaj postupak ponavljamo G puta te po tome vraćamo najbolju jedinku koja će definirati *GRH* algoritam koji ćemo u konačnici koristiti.

Algoritam 2 Genetski algoritam

Require:

N - velicina populacije

N_e - broj jedinki koje prenosimo elitizmom

N_r - broj nasumicno odabranih jedinki u novoj generaciji

max_iter - broj generacija

$populacija = nasumicno\_stvori\_jedinke(N)$

for $(i = 0; i < max\_iter; i++)$ **do**

$evaluiraj(populacija)$

$roditelji = odaberi\_roditelje(populacija)$

$potomci = krizanjzanje(roditelji, N - N_e - N_r)$

$nasumicni\_potomci = nasumicno\_stvori\_jedinke(N_r)$

$elitni\_potomci = elitizam(N_e, populacija)$

$populacija = stvori\_novu\_populaciju($
$potomci, nasumicni\_potomci, elitni\_potomci)$

end for

return $najbolje\_rjesenje(populacija)$

Varijabla	Opis
G	broj generacija
N	broj jedinki u generaciji
n	broj instanci problema korištenih za evaluaciju
N_r	broj nasumično stvorenih jedinki u generaciji
N_p	broj roditelja koji su odabrani u svakoj generaciji

Tablica 3.4: Parametri genetskog algoritma

4. Pregled rezultata

Za evaluaciju algoritama korištene su instance problema s definiranim prioritetima (tvorac instanci je Wenbin Zhua (Zhu et al., 2012)). Instance problema su podijeljene prema broju stogova, maksimalnoj inicijalnoj visini kontejnera na stogovima te ukupnom broju kontejnera. Broj stogova se proteže od 6 do 10, visina stogova od 3 do 7, a broj kontejnera od 15 pa sve do 69 kontejnera. Primjer jedne instance problema prikazan je slikom 4.1. Prvi redak tekstualne datoteke predstavlja broj stogova i broj kontejnera. Svi ostali redci započinju visinom stoga, a ostale vrijednosti predstavljaju prioritete kontejnera na tome stogu.

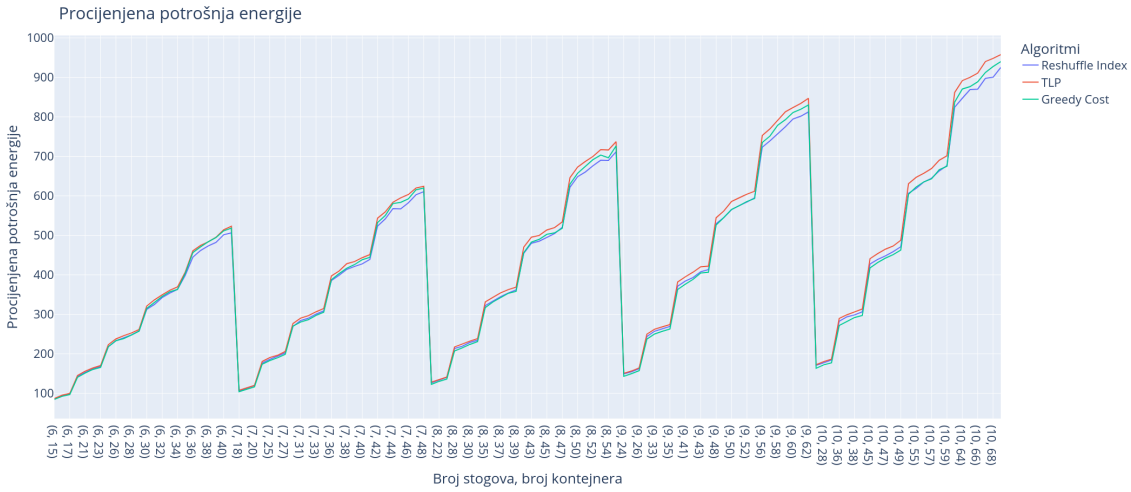
```
10 54
 6  9 35 49 24 26 31
 6 34 42 23  3 13 12
 6 28 52 51 50 53 48
 6 11 14 10  8 41 22
 1 45
 6 29 33 17 21 38 36
 6 20  7 39 37 16  1
 5 15 18 43 40 25
 6 44  6  4  2  5 47
 6 46 30 19 54 32 27
```

Slika 4.1: Primjer instance problema (10 stogova, visina 6, 54 kontejnera)

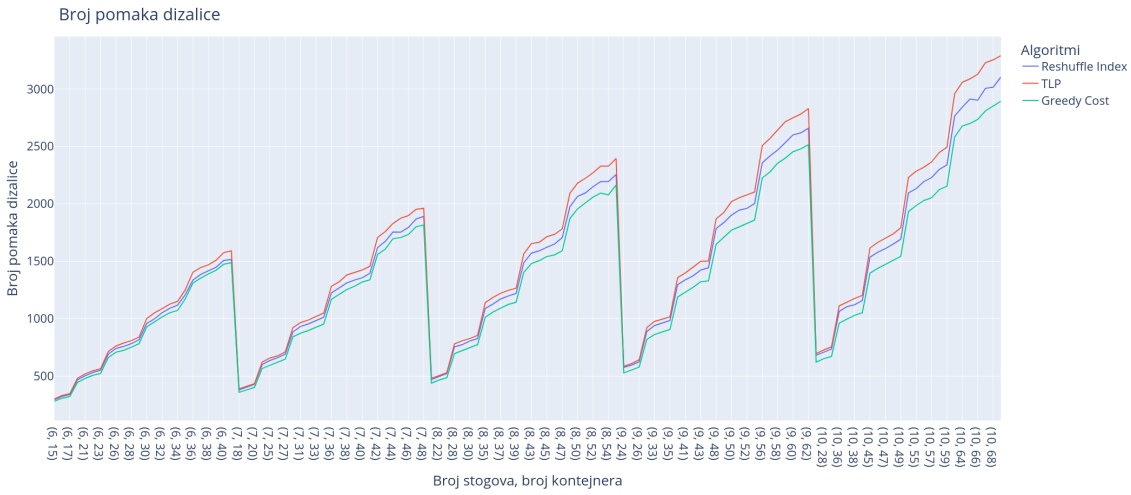
U nastavku će se evaluacije algoritama prezentirati linijskim grafovima na kojim će se vidjeti ukupna procijenjena potrošnja, zbroj jediničnih pomaka dizalice u svim smjerovima i broj realokacija u ovisnosti o broju kontejnera i broju stogova.

4.1. Heuristički algoritmi

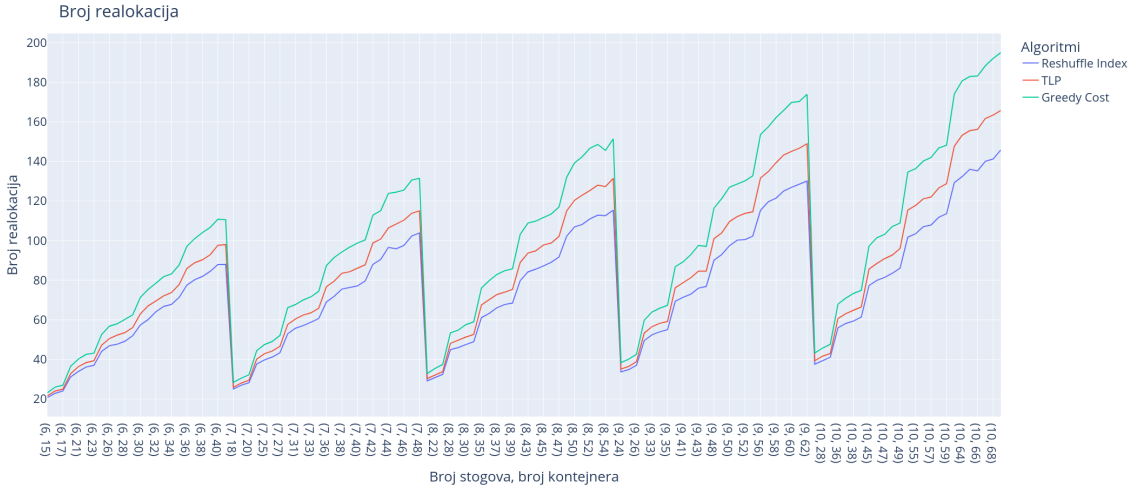
Usporedimo prvo heurističke algoritme. Slike 4.2, 4.3, 4.4 prikazuju procjenu potrošnje za heurističke metode. *Reshuffle Index* metoda ostvaruje znatno najbolje rezultate na cijelome skupu problema. Zbog toga ćemo tu metodu izabrati kao metodu za opravak u sklopu *GRASP* algoritma.



Slika 4.2: Izračun procjene potrošnje nad rješenjima kreiranim pomoću heurističkih metoda



Slika 4.3: Suma broja jediničnih pomaka dizalice nad rješenjima kreiranim pomoću heurističkih metoda



Slika 4.4: Broj realokacija u rješenjima kreiranim pomoću heurističkih metoda

4.2. Metaheurističke metode

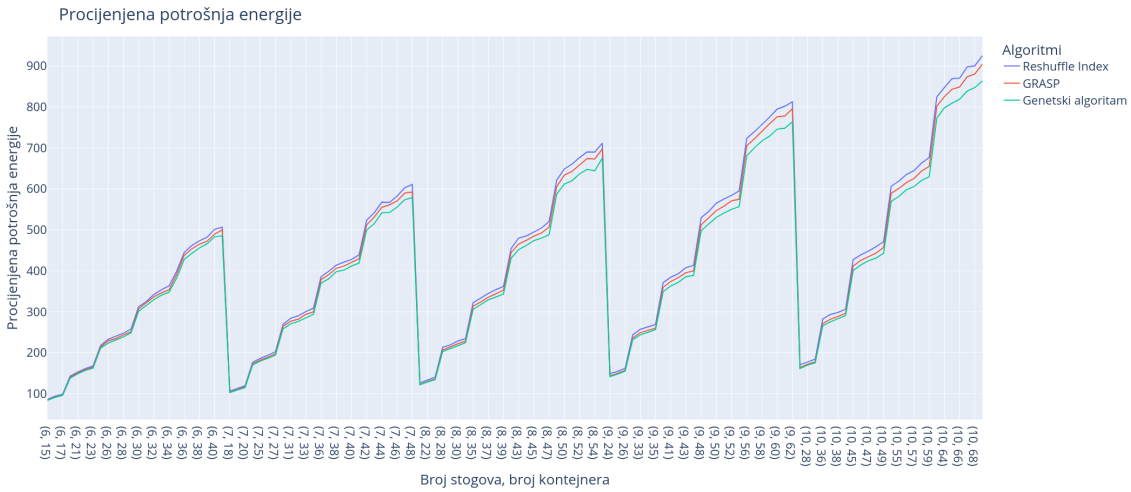
Prva metaheuristička metoda je *GRASP* algoritam. Algoritam je postavljen tako da je *TLP* heuristika iskorištena kao metoda u konstrukcijskoj fazi, parametar k je postavljen na 2, a strategija oporavka je *Reshuffle Index*.

Parametri *GRH* heuristike računati su 3 puta za različite veličine instanci problema, a pošto su rješenja približno jednaka na grafikonu će uvijek biti prikazana najbolja evaluacija. Parametri genetskog algoritma prikazani su tablicom **Tablica 4.1**.

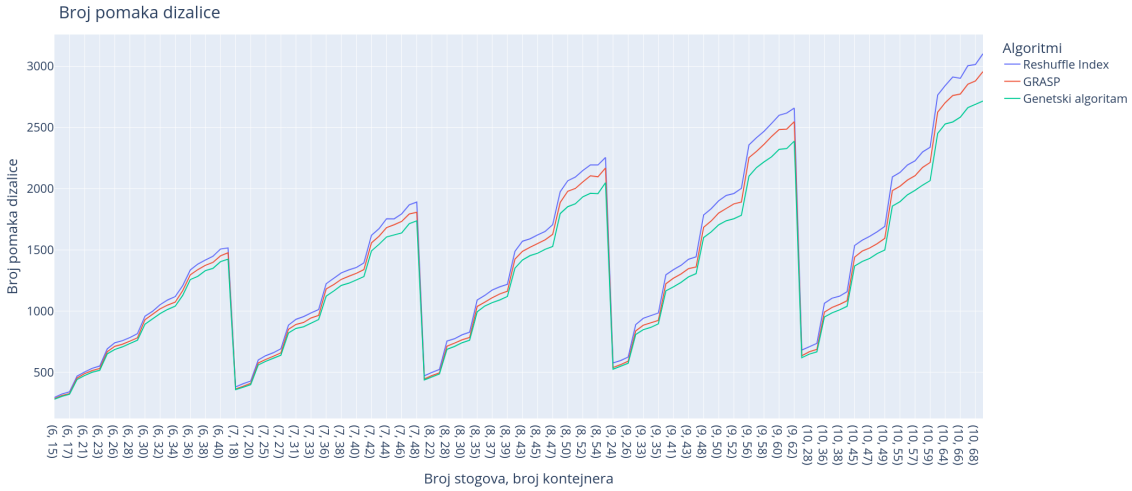
Tablica 4.1: Odabrani parametri genetskog algoritma

	veličina populacije - N	generacije - G	instance za evaluaciju - n
$GA1$	50	100	20
$GA2$	75	100	20
$GA3$	50	200	20

U nastavku su prikazani grafikoni koji prikazuju procjenu potrošnje dizalice, broj jediničnih pokreta koje je dizalica morala napraviti te broj realokacija koje su napravljene pri rješavanju problema metaheurističkim algoritmima te njihova usporedba sa *Reshuffle Index* metodom (slike redom 4.5, 4.6 i 4.7). Suma grafički prikazanih rezultata grupiranih prema broju stogova prikazana je tablicama **Tablica 4.2**, **Tablica 4.3** i **Tablica 4.4**.



Slika 4.5: Izračun procjene potrošnje nad rješenjima kreiranim pomoću metaheurističkih metoda



Slika 4.6: Suma broja jediničnih pomaka dizalice nad rješenjima kreiranim pomoću metaheurističkih metoda

Slika 4.7: Broj realokacija u rješenjima kreiranim pomoću metaheurističkih metoda

Zanimljivo je primijetiti kako rješenja dobivena optimiranjem genetskim algoritmom u konačnici rezultiraju s većim brojem realokacija, no nižom vrijednošću funkcije potrošnje.

Tablica 4.2: Suma procjene potrošnje svih instanci prema broju stogova

Procijenjena potrošnje					
Broj stogova	TLP	Reshuffle Index	Greedy Cost	GRASP	Genetski algoritam
6	733594.92	716782.32	723908.84	703568.64	691546.6
7	892238.28	867324.92	874283.64	848696.28	829407.8
8	1054803.84	1022535.84	1027660.16	997018.4	971338.52
9	1218060.28	1177819.56	1181484.48	1145762.04	1112131.44
10	1389686.16	1338135.84	1340307.08	1299123.16	1258929.88
suma	5288383.48	5122598.48	5147644.20	4994168.52	4863354.24

Tablica 4.3: Suma broja jediničnih pomaka dizalice za sve instance prema broju stogova

Broj pomaka dizalice					
Broj stogova	TLP	Reshuffle Index	Greedy Cost	GRASP	Genetski algoritam
6	2296392	2213050	2143016	2138262	2076368
7	2889850	2767046	2645714	2656662	2552318
8	3531528	3362220	3175512	3209616	3056226
9	4205592	3987868	3725338	3786820	3590894
10	4941296	4654136	4308202	4403550	4148210
suma	17864658	16984320	15997782	16194910	15424016

Tablica 4.4: Suma broja realokacija svih rješenja svih instanci prema broju stogova

Realokacija					
Broj stogova	TLP	Reshuffle Index	Greedy Cost	GRASP	Genetski algoritam
6	147045	134839	165539	136334	138827
7	174580	158535	198425	160442	163948
8	202046	181755	231365	184030	189067
9	229241	204812	264089	207383	213714
10	256117	227193	296250	230606	238937
suma	1009029	907134	1155668	918795	944493

5. Zaključak

Realociranje kontejnera u kontejnerskim terminalima je optimizacijski problem s važnom ulogom u stvarnom svijetu. Unaprjeđivanjem optimizacijskih algoritama adaptiranih za rješavanje ovog problem štedimo energiju i vrijeme potrebno za internacionalni transport dobara.

Ovim radom uspoređena su rješenja odabrana heurističkim metodama isto tako kao i rješenja formirana pomoću dva metaheuristička algoritma. Instance problema s malim brojem kontejnera najčešće su jednako dobro riješene heurističkim metodama kao i drugim naprednijim metodama, no na instancama s više kontejnera metaheuristički algoritmi pokazuju znatne prednosti i dostatno bolje rezultate.

Prostora za napredak je puno, trebalo bi istražiti djelotvornost predloženih algoritama na instancama s više odjeljaka. Također, algoritmi bi se mogli proširiti tako da dopuštaju i realokacije kontejnera koji nisu obavezno pozicionirani iznad kontejnera najvećeg prioriteta te bi tako možebitno ostvarili bolje rezultate.

LITERATURA

Camila Díaz Cifuentes i María Cristina Riff. G-crem: A grasp approach to solve the container relocation problem for multibays. *Applied Soft Computing*, 97:106721, 2020.

Camila Diaz i Maria Cristina Riff. New bounds for large container relocation instances using grasp. U *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, stranice 343–349. IEEE, 2016.

Marko Gulić, Livia Maglić, i Sanjin Valčić. Nature inspired metaheuristics for optimizing problems at a container terminal. *Pomorstvo*, 32(1):10–20, 2018.

Mazen Hussein i Matthew EH Petering. Genetic algorithm-based simulation optimization of stacking algorithms for yard cranes to reduce fuel consumption at seaport container transshipment terminals. U *2012 IEEE Congress on Evolutionary Computation*, stranice 1–8. IEEE, 2012.

Raka Jovanovic, Shunji Tanaka, Tatsushi Nishi, i Stefan Voß. A grasp approach for solving the blocks relocation problem with stowage plan. *Flexible Services and Manufacturing Journal*, 31(3):702–729, 2019.

Manoj Kumar, Dr Husain, Naveen Upreti, Deepti Gupta, et al. Genetic algorithm: Review and application. *Available at SSRN 3529843*, 2010.

Ching-Jung Ting i Kun-Chih Wu. Optimizing container relocation operations at container yards with beam search. *Transportation Research Part E: Logistics and Transportation Review*, 103:17–31, 2017.

Wenbin Zhu, Hu Qin, Andrew Lim, i Huidong Zhang. Iterative deepening a* algorithms for the container relocation problem. *IEEE Transactions on Automation Science and Engineering*, 9(4):710–722, 2012.

Rješavanje problema realokacije kontejnera korištenjem metaheuristika

Sažetak

Ovaj rad bavi se proučavanjem problema realokacije kontejnera. U radu je formalno objašnjen problem realociranja kontejnera u kontejnerskim terminalima, njegove varijante i ograničenja. Napravljen je pregled osnovnih heurističkih metoda koje rješavaju problem. Implementirane su dvije metaheurističke metode koje imaju za cilj optimirati rješenja heurističkih metoda. Rezultati svih metoda su izračunati za instance različitih težina te su grafički prikazani i međusobno uspoređeni.

Ključne riječi: problem realokacije kontejnera, CRP, BRP, metaheuristika, heuristika, GRASP, GRH, genetski algoritam, evolucijsko računarstvo, optimizacija

Solving the container relocation problem using metaheuristics

Abstract

This paper studies the container relocation problem. Container relocation is formally described, different variants are defined as well as problem limitations. Basic heuristic methods for solving a problem are explained. Two different metaheuristic algorithms are implemented in order to improve heuristic methods. Results calculated for different problem instances using different algorithms are plotted and compared.

Keywords: container relocation problem, CRP, BRP, metaheuristic, heuristic, GRASP, GRH, genetic algorithm, evolution, optimization