Rješavanje problema premještanja kontejnera pomoću Reshuffle Index with Look-Ahead prioritetnog pravila

Butumović, Iva

Sveučilište Josipa Jurja Strossmayera u Osijeku

Fakultet primijenjene matematike i informatike

Sveučilišni prijediplomski studij Matematika i računarstvo

Rješavanje problema premještanja kontejnera pomoću *Reshuffle Index with look-ahead* prioritetnog pravila

Završni rad

Mentor:

doc. dr. sc. Mateja Đumić

Student:

Iva Butumović

Osijek, 2024

Sadržaj

1 | Uvod

Kontejnerski transport ključni je element suvremene globalne trgovine koji omogućuje brz i efikasan prijevoz robe diljem svijeta. Kontejneri su postali standard u međunarodnom pomorskom transportu, a njihov broj u upotrebi raste iz godine u godinu.

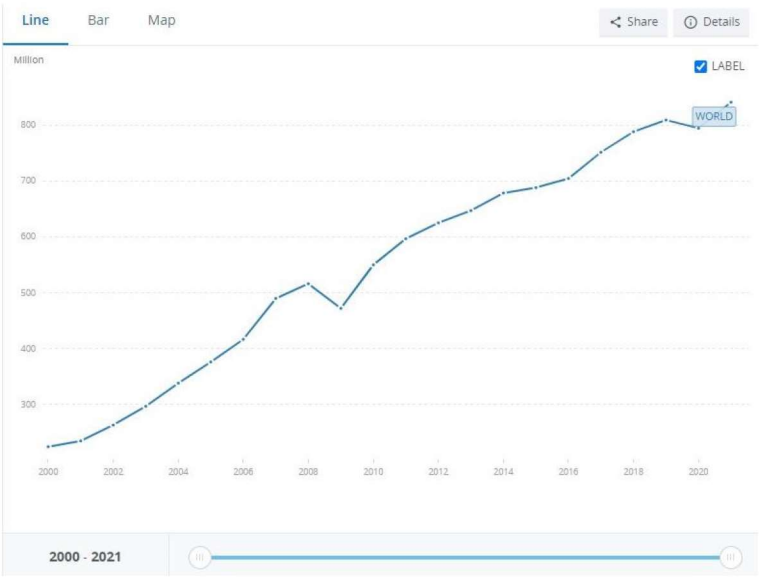
Jedan od glavnih izazova u upravljanju kontejnerskim terminalima je problem premještanja kontejnera (engl. *Container Relocation Problem*, CRP). Zbog složenosti operacija utovara i istovara, bitno je optimizirati premještanje kontejnera kako bi se minimiziralo vrijeme zadržavanja brodova i transportnih vozila, a time i poboljšala ukupna učinkovitost terminala.

Prvo poglavlje ovoga rada daje detaljan opis problema premještanja kontejnera, pružajući kontekst i objašnjenje važnosti ovog problema u radu kontejnerskih terminala. Drugo poglavlje predstavlja prioritetna pravila koja se često koriste za rješavanje CRP-a. Treće poglavlje sadrži implementaciju grafičkog sučelja koje simulira dohvaćanje kontejnera koristeći pravilo *Reshuffle index with lookahead*.

2 | Problem premještanja kontejnera

2.1 Motivacija

Veliki dio ukupnog svjetskog trgovinskog prijevoza odnosi se na prijevoz robe u kontejnerima. Od kada su uvedeni 1960-ih, kontejneri su postali standardna mjerna jedinica za međunarodni teretni transport. Korištenje kontejnera u međunarodnom pomorskom transportu značajno je poraslo posljednjih godina. Dok je 1985. godine u svijetu bilo u prometu oko 50 milijuna TEU-a[1], taj je broj do 2021. godine porastao na više od 840 milijuna.[3] Danas se približno 85% tereta prevozi u kontejnerima koji su smješteni na kontejnerske brodove.[2] Na slici 2.1 možemo vidjeti kretanje kontejnerskog prometa za razdoblje od 2000. do 2021. godine.



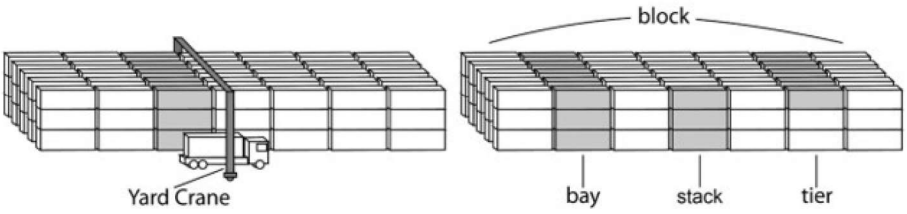
Slika 2.1: Kretanje broja kontejnera u međunarodnoj trgovini. Izvor: [12]

[1] akr. od engl. Twenty Foot Equivalent Unit, normirana jedinica izmjere transportnih kontejnera (tzv. ISO-kontejneri) duljine 20 stopa (6,1 m), širine 8 stopa (2,44 m) i visine 8,5 stopa (2,6 m), mase do 20 t.

2.2 Kontejnerski terminali

Kontejnerski terminali mjesta su na koja se doprema veliki broj kontejnera. Terminali omogućuju efikasan protok kontejnera između brodova i kopnenih prijevoznih sredstava. Kontejneri koji se prevoze od kamiona ili vlaka do broda nazivaju se izvozni kontejneri, a kontejneri koji se prevoze obrnutom rutom nazivaju se uvozni kontejneri. Izvozni kontejneri prevoze se u skladište terminala te se ondje skladište neko vrijeme prije nego što budu prevezeni na utovar. [4]

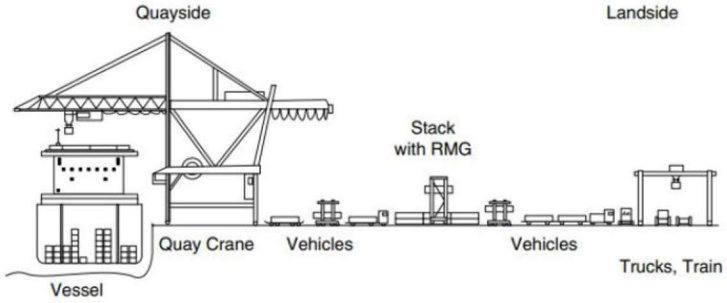
Skladište je podijeljeno na nekoliko blokova, svaki blok ima nekoliko odjeljaka (engl. *bays*). Svi odjeljci unutar istog bloka imaju isti broj stupaca (engl. *stacks*), a svaki stupac ima više redova (engl. *tiers*). Na svaki stupac može biti pohranjen jednak broj kontejnera. [11] Ilustracija kontejnerskog skladišta prikazana je na slici 2.2.



Slika 2.2: Ilustracija kontejnerskog skladišta. Izvor: [11]

Kontejneri se posebnim dizalicama premještaju iz prostora skladištenja do transportnog vozila. Ovakve dizalice odjednom mogu transportirati samo jedan kontejner te pristupiti samo kontejnerima koji se nalaze na vrhu stupca. Ako se kontejner koji je potrebno dohvatiti ne nalazi na vrhu stupca, sve kontejnere koji se nalaze iznad tog kontejnera potrebno je premjestiti na druge stupce. Premještanje kontejnera sa stupca na stupac naziva se premještanje (engl. *relocation*).

Prioriteti dohvaćanja kontejnera iz skladišta određeni su planom smještaj na brod. Plan smještanja na brod određuje, do određenog stupnja, redoslijed utovara kontejnera. Kako bi se očuvala stabilnost, teži kontejneri smještaju se na niže pozicije, a lakši na više. Stoga, teži kontejneri su uglavnom ukrcani prije lakših. Odredišne luke, također utječu na redoslijed utovara. Kontejneri za bliže luke postavljaju se iznad kontejnera za udaljenije luke. Efikasnost ukrcaja kontejnera na brod poboljšava se smanjenjem broja premještanja između stupaca tijekom transfera iz skladišta na brod. Slika 2.3 prikazuje glavne elemente procesa rada na kontejnerskom terminalu.



Slika 2.3: Prikaz rada u kontejnerskom terminalu.
Izvor: [6]

Pri smještanju kontejnera u skladište nastoji se osigurati da kontejneri koji će se kasnije utovarivati na brod ne blokiraju one koji trebaju biti ukrcani ranije. Međutim, tijekom izrade plana smještaja na brod mogu se pojaviti nepredviđene okolnosti, poput promjena odredišnih luka, zbog čega je ponekad teško osigurati da svi prioritetni kontejneri budu postavljeni na pozicije gdje ih drugi neće blokirati. [4]

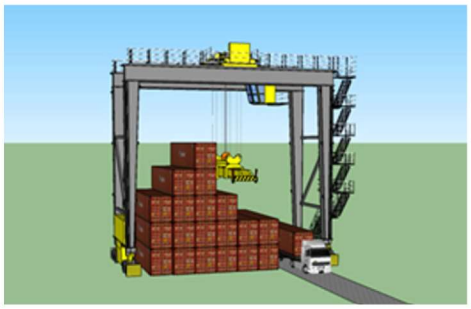
Premještanje kontejnera prilikom gore opisanih operacija utovara na brod i spremanja kontejnera u skladište predstavlja optimizacijski problem koji uključuje pronalaženje optimalnog slijeda operacija za dohvat kontejnera iz skladišta ili broda, minimizirajući dodatna premještanja kontejnera koji blokiraju one koje trebamo dohvatiti. [8]

2.3 Definicija i opis problema

Problem realokacije kontejnera (engl. *Container Relocation Problem*, *CRP*) je kombinatorni optimizacijski problem koji se može opisati na sljedeći način: Imamo niz stupaca S, pri čemu svaki stupac može primiti do H predmeta. U početku, n predmeta ($n \leq S \cdot H$) raspoređeno je nasumično među stupcima. Svaki predmet ima dodijeljen prioritet određen rednim brojem – što je manji redni broj, to je prioritet predmeta veći. Predmete možemo premještati s vrha jednog stupca na vrh drugog (premještanje) ili ih možemo dohvatiti s vrha stupca. Dohvaćanje je moguće samo ako je predmet na vrhu stupca onaj s najvećim prioritetom. Cilj je odrediti redoslijed operacija koji će s minimalnim brojem poteza dohvatiti sve predmete iz stupca. [4]

Postoje dvije verzije CRPa: ograničena (engl. *restricted*) i neograničena (engl. *unrestricted*). U ograničenoj verziji, predmeti se mogu premještati s vrha stupca samo ako u tom stupcu postoji predmet koji želimo dohvatiti. S druge strane, neograničena verzija dopušta premještanje predmeta s bilo kojeg stupcu, bez obzira na to gdje se nalazi predmet za dohvaćanje. [1]

Problem premještanja kontejnera (engl. *Container Relocation Problem*, CRP) pojavljuje se prilikom dohvaćanja kontejnera iz skladišta za utovar na brod, kao i prilikom iskrcaja kontejnera s broda. U ovom ćemo slučaju razmatrati scenarij gdje je potrebno dohvatiti kontejnere iz skladišta. Dohvaćanje kontejnera iz skladišta prikazano je na slici 2.4.

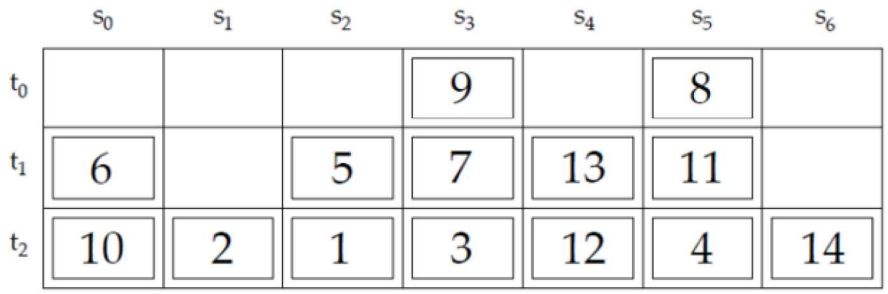


Slika 2.4: Ilustracija dohvaćanja kontejnera.
Izvor: [1]

Kontejneri su unutar odjeljka označeni brojevima, pri čemu manji broj označava veći prioritet dohvaćanja, dok veći broj znači manji prioritet. Kontejner kojeg želimo dohvatiti naziva se ciljni kontejner (engl. *target container*) i on trenutno ima najveći prioritet za dohvaćanje. Ako je ciljni kontejner na vrhu stupca, dizalica ga može izravno prenijeti na transportno vozilo. Međutim, ako se iznad ciljnog kontejnera nalaze drugi kontejneri, potrebno ih je premjestiti kako bi se do njega moglo doći.

3 | Metode za rješavanje CRPa

Fokusirat ćemo se na rješavanje statičkog i ograničenog problema realokacije kontejnera unutar jednog odjeljka (engl. *single bay*). Odjeljak sadrži određeni broj stupaca i redova. Statičnost problema podrazumijeva da su svi kontejneri već smješteni u skladište te da se novi kontejneri ne dodaju tijekom operacija dohvaćanja.



Slika 3.1: Dvodimenzionalni prikaz jednog odjeljka.
Izvor: [1]

Na slici 3.1 prikazan je dvodimenzionalni model jednog odjeljka. Ovaj odjeljak sastoji se od 7 stupaca, označenih od S_0 do S_6, te 3 reda, označenih od t_0 do t_2. Redni brojevi unutar pravokutnika predstavljaju prioritete dohvaćanja kontejnera. U nastavku ćemo prikazati kako se CRP može riješiti primjenom prioritetnih pravila.

Prioritetna pravila određuju kriterije za odabir stupca na koji treba premjestiti kontejnere koji blokiraju pristup ciljnome kontejneru. Ova pravila pomažu u izboru stupca za premještaj s ciljem minimizacije broja potrebnih dodatnih premještanja, čime se poboljšava cjelokupna učinkovitost operacija u skladištu.

3.1 Pravilo najniže pozicije

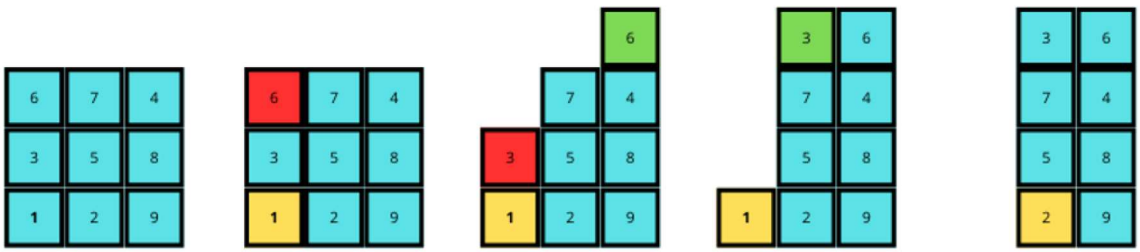
Pravilo najniže pozicije (engl. *The Lowest Point*, TLP) broji sve stupce i bilježi broj kontejnera u svakom stupcu. Zatim, odabire stupac s najmanjim brojem kontejnera kao odredište za premještanje kontejnera koji blokiraju pristup ciljnom kontejneru. Ako postoji više stupaca s istim najmanjim brojem kontejnera, odabir se vrši nasumično među tim stupcima. [9]

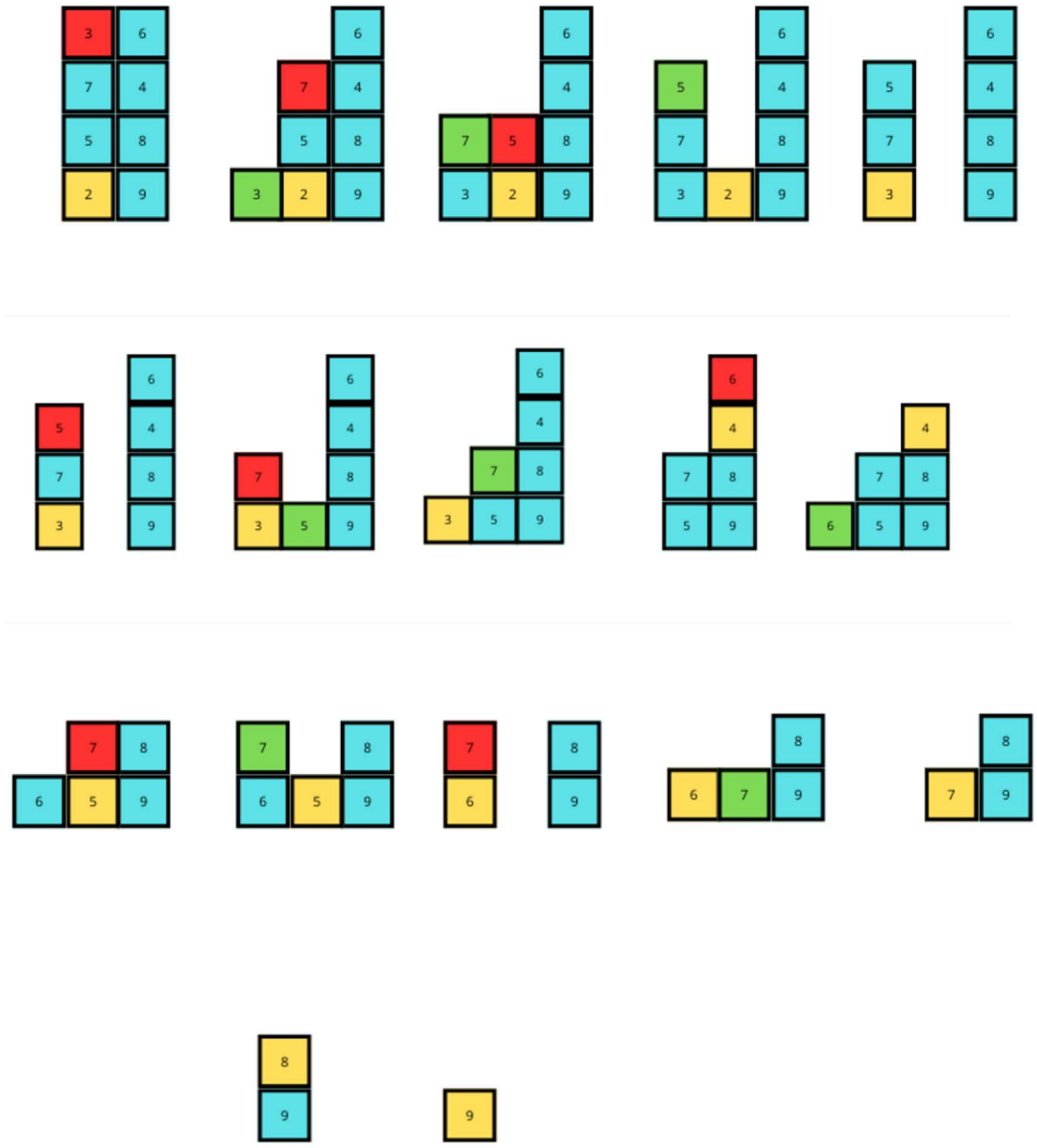
Dohvat kontejnera iz jednog odjeljka primjenom TLP pravila zahtjeva sljedeće korake:

1. Odredi ciljni kontejner.
2. Ako je ciljni kontejner na vrhu stupca, može se odmah dohvatiti. U suprotnom, odredi blokirajući kontejner.
3. Odredi stupac s najmanjim brojem kontejnera. Ako postoji više stupaca s jednakim minimalnim brojem kontejnera, nasumično odaberi jedan od tih stupaca.
4. Premjesti blokirajući kontejner na vrh odabranog stupaca.
5. Ponavljaj korake 2-4 dok ciljni kontejner ne bude dostupan za dohvat.
6. Ponavljaj cijeli postupak dokle god ima kontejnera.

Primjer 1. *Primjena TLP pravila na odjeljak koji sadrži 3 reda i 3 stupca, pri čemu maksimalna visina stupca iznosi 5 redova.*

Žuto obojeni kvadrat označava ciljni kontejner, crveni kvadrat predstavlja najviši blokirajući kontejner, dok je zeleni kvadrat korišten za prikaz lokacije na koju je blokirajući kontejner premješten.





3.2 Pravilo *Reshuffle Index*

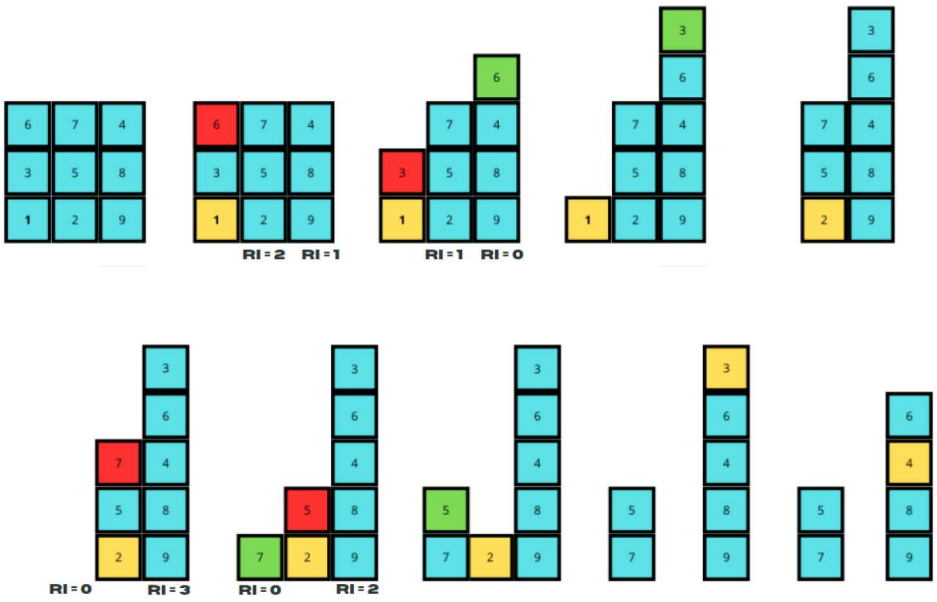
Pravilo *Reshuffle index* (RI) izračunava *reshuffle index* za stupce na koje može biti premješten blokirajući kontejner. Odredište za blokirajući kontejner je stupac s najmanjim RI, gdje RI označava broj kontejnera u stupcu koji imaju viši prioritet od blokirajućeg kontejnera. Ako više stupaca ima isti RI, nasumično se bira na koji će stupac biti premješten blokirajući kontejner. [9]

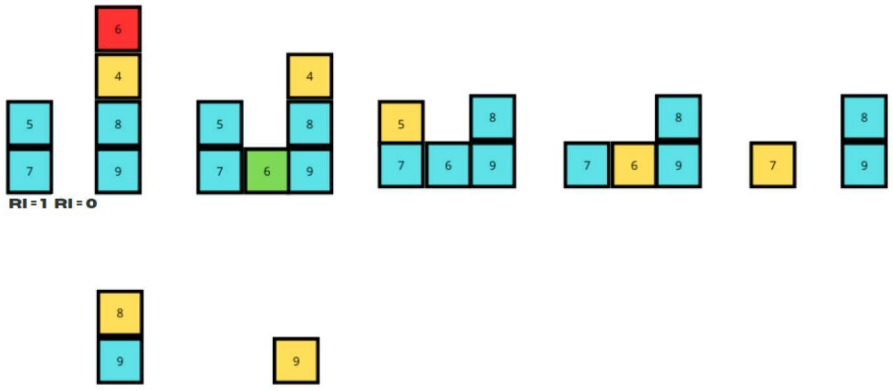
Dohvat kontejnera iz jednog odjeljka primjenom RI pravila zahtjeva sljedeće korake:

1. Odredi ciljni kontejner.
2. Ako je ciljni kontejner na vrhu stupca, može se odmah dohvatiti. U suprotnom, odredi blokirajući kontejner.
3. Odredi RI za svaki stupac.
4. Odaberi stupac s najmanjim RI. Ako postoji više stupaca s jednakim RI, nasumično odaberi jedan od tih stupaca.
5. Premjesti blokirajući kontejner na vrh odabranog stupca.
6. Ponavljaj korake 2-5 dok ciljni kontejner ne bude dostupan za dohvat.
7. Ponavljaj cijeli postupak dokle god u odjeljku ima kontejnera.

Primjer 2. *Primjena RI pravila na odjeljak koji sadrži 3 reda i 3 stupca, pri čemu maksimalna visina stupca iznosi 5 redova.*

Boje u ovom primjeru imaju iste oznake kao u Primjeru 1. Ispod stupaca na koje je moguće premjestiti blokirajući kontejner izračunat je reshuffle index.





3.3 Pravilo *Reshuffle Index with look-ahead*

Pravilo *Reshuffle index with look-ahead* (RIL) proširenje je RI prioritetnog pravila. Ovo pravilo, osim što izračunava RI, pokušava predvidjeti buduća premještanja kontejnera i izbjeći poteze koji bi mogli uzrokovati dodatna premještanja. Pored izračunavanja RI za svaki stupac, pravilo bilježi najmanji element u svakom stupcu. Odredište za premještanje blokirajućeg kontejnera je stupac s najmanjim RI. Ako više stupaca ima isti RI, kao odredište se odabire stupac koji ima najveći najmanji element. [9]

Dohvat kontejnera iz jednog odjeljka primjenom RIL pravila zahtjeva sljedeće korake:

1. Odredi ciljni kontejner.
2. Ako je ciljni kontejner na vrhu stupca, može se odmah dohvatiti. U suprotnom, odredi blokirajući kontejner.
3. Odredi RI za svaki stupac.
4. Odredi najmanji element u svakom stupcu.
5. Odaberi stupac s najmanjim RI. Ako postoji više stupaca s jednakim RI, odaberi stupac koji ima najveći najmanji element.
6. Premjesti blokirajući kontejner na vrh odabranog stupca.
7. Ponavljaj korake 2-6 dok ciljni kontejner ne bude dostupan za dohvat.
8. Ponavljaj cijeli postupak dokle god u odjeljku ima kontejnera.

4 | Implementacija RIL prioritetnog pravila

Ovo poglavlje prikazat će implementaciju RIL prioritetnog pravila te izradu grafičkog sučelja koje omogućuje simulaciju rješavanja instance problema. Prikazat će funkcije koje su ključne za izradu samog grafičkog sučelja, a detaljna implementacija dostupna je na GitHubu[1]. Na kraju poglavlja bit će prikazan primjer s nasumično raspoređenim kontejnerima, koji je riješen unutar implementiranog sučelja.

Za prikaz grafičkog sučelja koje će omogućiti vizualno prikazivanje premještanja kontejnera koristeći pravilo RIL koristit ćemo ćemo programski jezik *Python*, zajedno s paketom *tkinter*, koji omogućuje kreiranje prozora aplikacije, dodavanje kontrola kao što su gumbi i izbornici, te upravljanje obavijestima putem *messagebox-a*. Za simulaciju rasporeda kontejnera u skladištu koristit ćemo funkcije *randint* i *shuffle* iz modula *random* za generiranje nasumičnih brojeva i permutacija. Podaci o luci i kontejnerima moći će se učitavati iz JSON datoteka uz pomoć *filedialog-a*, a za slučaj grešaka pri parsiranju JSON datoteka, koristit ćemo JSONDecodeError.

Programski kod 4.1 prikazuje uvoz navedenih biblioteka i modula potrebnih za implementaciju grafičkog sučelja, generiranje nasumičnih podataka te učitavanje i obradu informacija iz JSON datoteka.

```
1  import tkinter as tk
2  from tkinter import ttk, filedialog, messagebox
3  from json import JSONDecodeError
4  from random import randint, shuffle
```

Programski kod 4.1: Uvoz potrebnih biblioteka

Kontejner unutar aplikacije definiran je unutar zasebne klase, a njegov konstruktor sadrži 3 atributa: *id*, *position* i *reshuffle index*. Detalji implementacije ove klase navedeni su u Programskom kodu 4.2.

```
1  class Container:
2    def __init__(self, container_id, position, reshuffle_index):
```

[1]Implementacija je dostupna na sljedećem linku: https://github.com/andreamatasovic/ZPP

```
3       self.id = container_id
4       self.position = position
5       self.reshuffle_index = reshuffle_index
```

Programski kod 4.2: Klasa *Container*

Kako bi korisniku omogućili učitavanje podataka iz JSON datoteke definirana je funkcija *parse input* dana s Programskim kodom 4.3 koja podatke o kontejnerima iz JSON datoteke pretvara u objekte klase Container.

```
1 def parse_input(file_path):
2   with open(file_path, 'r') as file:
3     data = json.load(file)
4   stacks = []
5   for stack_data in data['stacks']:
6     stack = []
7     for container_data in stack_data:
8       container = Container(container_data['id'], container_data['
     position'],container_data['reshuffle_index'])
9       stack.append(container)
10    stacks.append(stack)
11  return stacks
```

Programski kod 4.3: Funkcija *parse input*

U slučaju da korisnik odluči ne koristiti podatke iz JSON datoteke, funkcija *initial stacks* dana Programskim kodom 4.4 dodjeljuje kontejnerima nasumične ID-jeve. Prvo, funkcija dodjeljuje redom brojeve od 1 do ukupnog broja kontejnera, koji se dobija množenjem broja stupaca i broja redova. Nakon što su svi kontejneri inicijalizirani, redoslijed kontejnera u listi se nasumično izmiješa koristeći funkciju *shuffle*. Na kraju, funkcija kreira liste za svaki stupac i raspoređuje kontejnere među njima tako da svaki stupac dobije određeni broj kontejnera.

```
1   def initial_stacks(self, tiers, stacks):
2     all_containers = []
3     container_id = 1
4     for _ in range(stacks * tiers):
5       container = Container(container_id, container_id)
6       all_containers.append(container)
7       container_id += 1
8
9     shuffle(all_containers)
10
11    stacks_list = [[] for _ in range(stacks)]
12    for i, container in enumerate(all_containers):
13      stack_index = i % stacks
14      stacks_list[stack_index].append(container)
15
16    return stacks_list
```

Programski kod 4.4: Funkcija *initial stacks*

Kontejneri se iscrtavaju pomoću funkcije *draw canvas* dane Programskim kodom 4.5. Ova funkcija svakom kontejneru dodjeljuje odgovarajuću boju i ispisuje tekst s pozicijom kontejnera unutar pravokutnika. Funkcija također pohranjuje referencu na svaki nacrtani pravokutnik za buduću upotrebu.

```
1    def draw_canvas(self):
2     self.tlp_canvas.delete("all")
3     self.container_rects = {}
4     container_height = 50
5     container_width = 50
6     bottom_margin = 20
7
8     canvas_height = self.tlp_canvas.winfo_height()
9     base_y = canvas_height - bottom_margin - container_height
10
11     for i, stack in enumerate(self.stacks):
12      for j, container in enumerate(stack):
13        x1 = 50 * i + 10
14        y1 = base_y - (j + 1) * container_height
15        x2 = x1 + container_width
16        y2 = y1 + container_height
17        color = "#9400D3" if container.id == self.
    get_current_container_id() else "#4B0082"
18        rect = self.tlp_canvas.create_rectangle(x1, y1, x2, y2,
    fill=color, tags=f"container_{container.id}", outline="black")
19        self.tlp_canvas.create_text((x1 + x2) // 2, (y1 + y2) // 2,
     text=f"{container.position}", fill="white")
20        self.container_rects[container.id] = rect
```

Programski kod 4.5: Funkcija *draw canvas*

Slika 4.1 prikazuje implementirano grafičko sučelje koje je kreirano koristeći prethodno navedene programske kodove. U ovom primjeru, korisnik je odabrao 4 reda i 4 stupca, nakon čega je klikom na gumb *Start* generirano 16 kontejnera s nasumično dodjeljenim prioritetima.

Slika 4.1: Implementirano grafičko sučelje.

Za implementaciju RIL pravila potrebna nam je *calculate reshuffle index* funkcija dana s Programskim kodom 4.6 koja računa *reshuffle index* za dani stupac i blokirajući kontejner.

```
def calculate_reshuffle_index(stack, blocking_container_id):
  reshuffle_index = 0
  for container in stack:
    if container.id < blocking_container_id:
      reshuffle_index += 1
  return reshuffle_index
```

Programski kod 4.6: Funkcija *calculate reshuffle index*

Funkcijom RIL, a koja je dana s Programskim kodom 4.7, implementirano je pravilo *Reshuffle Index with Look-Ahead*. U prvoj petlji funkcija identificira blokirajući kontejner, dok se u drugoj petlji računaju reshuffle indeksi. Na temelju izračunatih indeksa, odabire se stupac koji ima manji reshuffle indeks, odnosno onaj s najvećim najmanjim elementom.

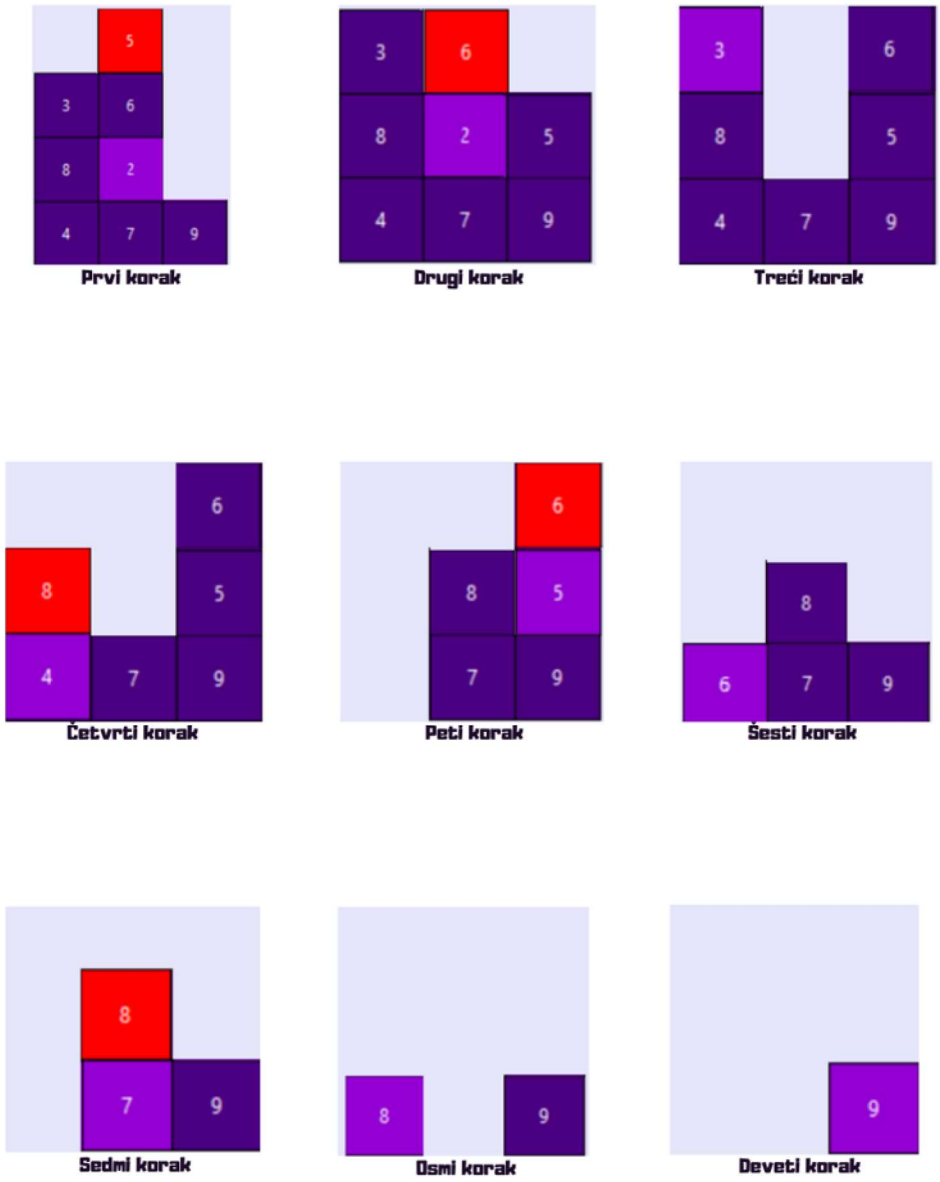
```
1   def RIL(stacks):
2     min_reshuffle_index = float('inf')
3     target_stack = None
4     current_stack = None
5     blocking_container_id = None
6     min_container_id = float('inf')
7     min_max_id= float('-inf')
8
9     for stack in stacks:
10      for container in stack:
11        if container.id < min_container_id:
12          min_container_id = container.id
13          current_stack= stack
14          blocking_container_id = current_stack[-1]
15    for stack in stacks:
16      current_reshuffle_index = calculate_reshuffle_index(stack,
    blocking_container_id)
17      min_in_stack=min(stack)
18      if current_reshuffle_index < min_reshuffle_index:
19        min_reshuffle_index = current_reshuffle_index
20          target_stack = stack
21          elif current_reshuffle_index == min_reshuffle_index:
22            if min_in_stack > min_max_id:
23            min_max_id = min_in_stack
24            min_reshuffle_index = current_reshuffle_index
25            target_stack = stack
26
27    return target_stack
```

Programski kod 4.7: Funkcija *Reshuffle Index with Look-Ahead*

Primjer 3. *Koristeći grafičko sučelje, prikazat ćemo primjenu pravila RIL na primjeru s 3 stupca i 3 reda, pri čemu su kontejneri nasumično raspoređeni.*

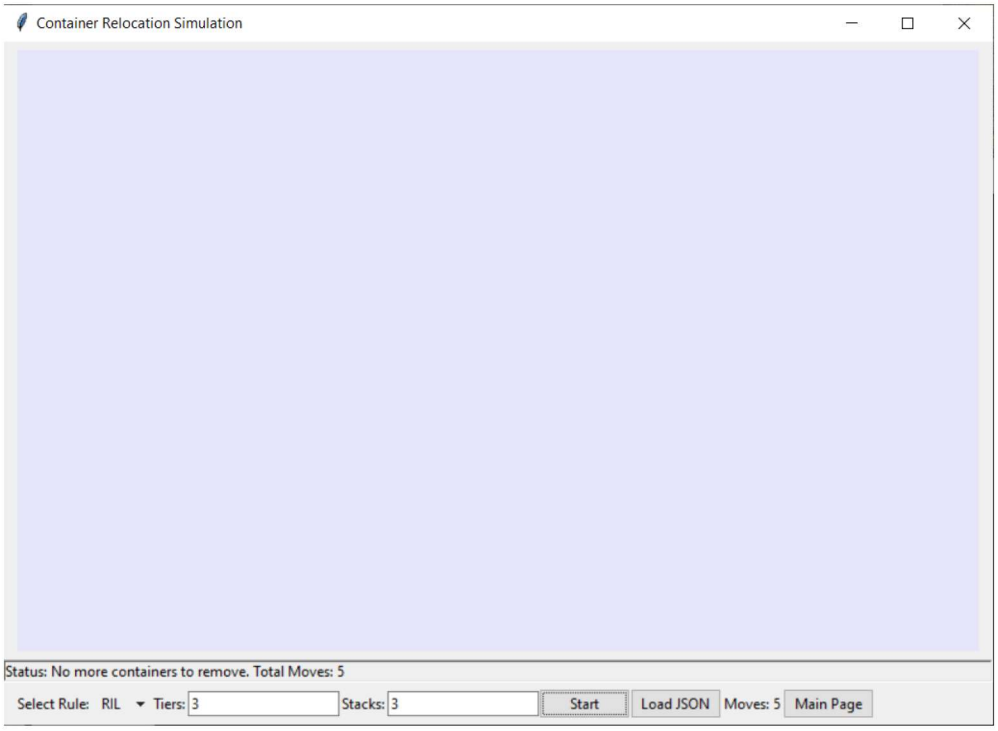
Slika 4.2: Početni izgled sučelja prilikom pokretanja programa.

Slika 4.2 prikazuje početak simulacije. Nakon unosa broja redova i stupaca, pritiskom na gumb "Start" generiran je prikaz devet kontejnera s nasumično raspoređenim identifikacijskim oznakama.



Slika 4.3: Koraci simulacije.

Slika 4.3 prikazuje korake kojim sučelje, koristeći RIL pravilo, postupno uklanja kontejnere, jedan po jedan. Svaki korak ilustrira način na koji se kontejneri reorganiziraju i uklanjaju iz stupaca.



Slika 4.4: Završetak simulacije.

Slika 4.4 prikazuje sučelje nakon što je simulacija dovršila dohvaćanje svih kontejnera. Ukupan broj poteza (engl. moves) predstavlja broj premještanja koja su nužna kako bi se omogućilo dohvaćanje svih kontejnera u danom redoslijedu.

Literatura

[1] B. Jin, W. Zhu, A. Lim, *Solving the container relocation problem by an improved greedy look-ahead heuristic*, Innovative Applications of O.R., 2024.

[2] D. Krstulović, *Automatizirani pomorski kontejnerski terminali*, Undergraduate thesis, University of Split, Faculty of Maritime Studies, 2018. Dostupno na: https://urn.nsk.hr/urn:nbn:hr:164:403035.

[3] D. Perčić, *Obilježja suvremenih kontejnerskih i ro-ro terminala*, Undergraduate thesis, University of Rijeka, Faculty of Maritime Studies, Rijeka, 2023. Dostupno na: https://urn.nsk.hr/urn:nbn:hr:187:379761.

[4] F. Forster, A. Bortfeldt, *A tree search procedure for the container relocation problem*, Computers & Operations Research, University of Hagen, Hagen, Germany, 2024.

[5] M. Gulic, L. Maglic, T. Krljan, L. Maglic, *Solving the Container Relocation Problem by Using a Metaheuristic Genetic Algorithm*, Sciences, Faculty of Maritime Studies, University of Rijeka, Rijeka, Croatia, 2024.

[6] K. Hwan, Hans-Otto G., *Container Terminals and Cargo Systems*, Helmut Petri, Berlin Heidelberg, 2007, str. 13.

[7] M.E.H. Petering, M.I. Hussein, *A new mixed integer program and extended look-ahead heuristic algorithm for the block relocation problem*, University of Wisconsin—Milwaukee, Department of Industrial & Manufacturing Engineering, Milwaukee, USA, 2024.

[8] Tiru S. Arthanari, *Container Relocation Problem with Time Windows for Container Departure*, European Journal of Operational Research, 2016.

[9] C.-C. Wu, C.-J. Ting, *A Beam Search Algorithm for Minimizing Reshuffle Operations at Container Yards*, Journal of Industrial Engineering and Management, Department of Industrial Engineering and Management, Yuan Ze University, Chung-Li, Taiwan, 2024.

[10] C. Zhang, H. Guan, Y. Yuan, W. Chen, T. Wu, *Machine learning-driven algorithms for the container relocation problem*, Journal of Operations Research, Research Center for Modern Logistics, Shenzhen International Graduate School, Tsinghua University, Shenzhen, China; Department of Industrial Engineering, Tsinghua University, Beijing, China; Department of Supply Chain

Management, Rutgers Business School-Newark and New Brunswick, Rutgers University, New Jersey, United States; School of Economics & Management, Tongji University, Shanghai, China, 2024.

[11] W. Zhu, H. Qin, A. Lim, H. Zhang, *Iterative Deepening A* Algorithms for the Container Relocation Problem*, IEEE Transactions on Automation Science and Engineering, vol. 9, no. 4, October 2012.

[12] World Bank, *Trade in goods and services*, Dostupno na: https://data.worldbank.org/indicator/IS.SHP.GOOD.TU?end=2021&start=2000&view=chart, pristupljeno 2024.

Sažetak

Problem realokacije kontejnera (engl. *Container Relocation Problem*) jedan je od ključnih izazova u upravljanju kontejnerskim terminalima, posebno tijekom operacija utovara i istovara brodova. U ovom problemu, svaki kontejner ima određeni prioritet. Cilj je dohvatiti sve kontejnere u definiranom redoslijedu, a koji je određen danim prioritetom. Kada je kontejner koji treba dohvatiti blokiran drugim kontejnerima, nužno je premjestiti te blokirajuće kontejnere kako bi se omogućio pristup kontejneru s većim prioritetom.

U ovom radu prikazana su tri prioritetna pravila The Lowest Point (TLP), Reshuffle Index (RI) i Reshuffle Index with Lookahead (RIL) koja se koriste u rješavanju CRPa. Za pravila TLP i RI dani su primjeri njihove primjene, dok je pravilo RIL implementirano zajedno s grafičkim sučeljem koje prikazuje korake rješavanja problema.

Ključne riječi

kontejner, terminal, problem realokacije kontejnera, optimizacija, prioritetna pravila, TLP, RI, RIL

Solving the Container Relocation Problem using the Reshuffle Index with Look-Ahead Priority Rule

Summary

The Container Relocation Problem (CRP) is one of the key challenges in managing container terminals, particularly during loading and unloading operations. In this problem, each container has a specified priority, and the goal is to retrieve all containers in the defined priority order. When other containers block a container that needs to be retrieved, it is necessary to move those blocking containers to allow access to the container according to its priority.

This paper presents three priority rules for solving the Container Relocation Problem (CRP): The Lowest Point (TLP), Reshuffle Index (RI), and Reshuffle Index with Lookahead (RIL). Examples of applying the TLP and RI rules are provided, while the RIL rule is implemented along with a graphical interface that displays the steps for solving the problem.

Keywords

container, terminal, container relocation problem, optimization, priority rules, TLP, RI, RIL

Životopis

Rođena sam 23. ožujka 2002. godine u Slavonskome Brodu. Pohađala sam Osnovnu školu "Bogoslav Šulek" u Slavonskom Brodu. Po završetku osnovne škole upisala sam Gimnaziju "Matija Mesić", Slavonski Brod, smjer prirodoslovno-matematička gimnazija. Nakon završene gimnazije upisala sam prijediplomski studij Matematike i računarstva na Fakultetu primijenjene matematike i informatike u Osijeku.