Ovim putem želim se zahvaliti svojoj mentorici na pruženoj podršci i korisnim savjetima tijekom izrade ovog završnog rada. Posebnu zahvalnost dugujem svojoj obitelji i prijateljima, čija mi je neizmjerna podrška, razumijevanje i ohrabrenje bilo ključno tijekom cijelog studija. Hvala vam što ste uvijek bili uz mene!

Sadržaj

1. Uvod

Razvoj robotike tijekom posljednjih desetljeća ima snažan utjecaj na industriju, gospodarstvo i svakodnevni život. Roboti se danas koriste u raznim područjima – od automatizirane proizvodnje i logistike, preko medicine i istraživanja svemira, do kućne i uslužne robotike. U industrijskom kontekstu, a osobito u okviru koncepta Industrije 4.0, primjena robota postaje sve sofisticiranija, uz naglasak na autonomiju, fleksibilnost i suradnju.

Tradicionalni roboti, koji su obavljali unaprijed definirane, repetitivne zadatke u strogo kontroliranim uvjetima, danas se nadopunjuju naprednim sustavima koji mogu opažati okolinu, donositi odluke i međusobno komunicirati. Sve češće se nailazi na sustave s višestrukim robotima koji djeluju istovremeno, u kojima je ključno učinkovito upravljanje resursima, raspodjela zadataka i izbjegavanje konflikata među agentima.

Takvi sustavi traže algoritme koji mogu u realnom vremenu optimizirati ponašanje robota, uzimajući u obzir ograničenja poput vremena, prostora, fizičkog dosega i međusobnih interferencija. Jedan od tipičnih izazova u ovom području je raspoređivanje zadataka u proizvodnim sustavima u kojima objekti dolaze dinamički – primjerice, u obliku predmeta na pokretnoj traci.

Slika 1.1 Prikaz robotskih ruku kako preuzimaju pakete

Upravo takav problem promatra se u ovom završnom radu. Cilj rada je razviti algoritam za raspoređivanje zadataka robotima uz pokretnu traku s ciljem maksimalnog broja uspješno preuzetih predmeta, ravnomjerne raspodjele opterećenja među robotima i izbjegavanja međusobnih konflikata.

U ovom završnom radu razvijena su i uspoređena dva algoritma koja rješavaju opisani problem. Prvi je heuristički algoritam koji koristi lokalne informacije i pravila prioriteta za donošenje odluka. Drugi algoritam koristi formalni optimizacijski model temeljen na programiranju s ograničenjima *(*engl. *Constraint Programming*) pomoću Google OR-Tools biblioteke [1].

Analiza rezultata provedena je u simulaciji. Uspoređivana je efikasnost algoritma (broj preuzetih predmeta) i njegova vremenska složenost za različite ulazne vrijednosti.

Time je dobiven uvid u praktične razlike između jednostavnih i naprednijih algoritama raspodjele zadataka u višerobotskim sustavima.

Ovaj rad podijeljen je na sedam poglavlja. U prvom poglavlju dan je uvod u temu te objašnjena motivacija za rješavanje problema. Drugo poglavlje detaljno opisuje konkretan problem, s naglaskom na izazove koji proizlaze iz dinamičnosti sustava. U trećem poglavlju definiran je problem raspoređivanja zadataka kroz oblik fleksibilnog problema rasporeda poslova (engl. *Flexible Job Shop Scheduling Problem*). Četvrto poglavlje donosi matematičku formulaciju optimizacijskog modela koji je korišten u rješavanju problema. U petom poglavlju opisana je implementacija razvijenih algoritama u programskom jeziku Python. Šesto poglavlje sadrži simulacijsku analizu učinkovitosti algoritama na različitim skupovima ulaznih podataka. Na kraju, sedmo poglavlje donosi zaključke te ističe mogućnosti daljnjeg razvoja i primjene istraženih metoda.

2. Opis problema

Jedan od konkretnih i čestih scenarija u industrijskoj automatizaciji uključuje robotske ruke koje preuzimaju predmete s pokretne trake. Takvi sustavi široko se primjenjuju u sortirnim centrima, pakirnicama, automobilskoj industriji, farmaceutskoj proizvodnji, kao i u modernim skladištima.

Zajedničko svim takvim sustavima je potreba da roboti brzo i precizno manipuliraju objektima koji se gibaju, bez međusobnog konflikta i uz maksimalnu iskorištenost. Sustav treba u svakom trenutku odlučiti koji robot će preuzeti koji predmet, s obzirom na njegovu poziciju, brzinu gibanja i dostupnost robota.

U ovakvom sustavu izazov je višestruk:

- Objekti dolaze dinamički – njihov broj, pozicija i vrijeme pojave nisu unaprijed poznati.
- Roboti imaju ograničen radni prostor – mogu dohvatiti samo objekte s dijela trake.
- Vrijeme za odluku je ograničeno, jer objekt brzo napušta područje dohvatljivosti.

Problem se stoga može formulirati kao problem dodjele zadataka robotima uz poštivanje vremenskih i prostornih ograničenja, gdje je cilj:

- maksimizirati broj preuzetih objekata,
- ravnomjerno raspodijeliti opterećenje među robotima,
- izbjeći konfliktne situacije (npr. dva robota pokušavaju dohvatiti isti predmet).

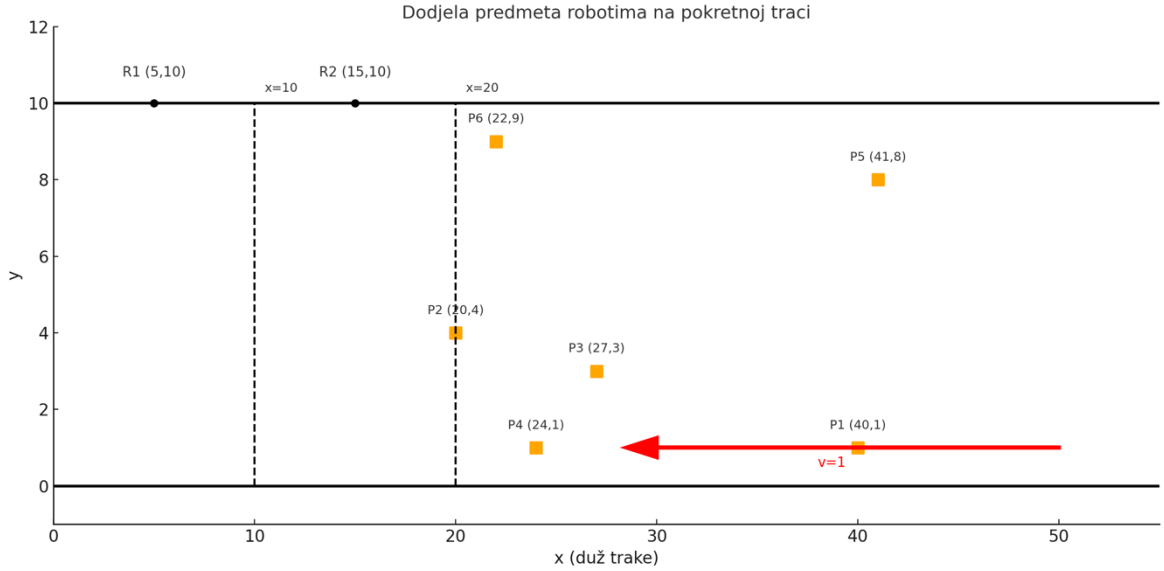
Na slici (Slika 2.2.) je prikazan sustav koji uključuje pokretnu traku, dva robota i niz predmeta čija se trenutna pozicija generira prilikom pokretanja simulacije. Traka se kreće unaprijed zadanom brzinom v = 1 u negativnom smjeru osi x.

U sustavu se nalaze:

- Robot 1 na poziciji (5, 10)
- Robot 2 na poziciji (15, 10)

Na slici su jasno prikazane i granice radnih prostora pojedinog robota (x = 10, x = 20), unutar kojih se nalazi prostor u kojem svaki robot može dohvatiti predmet.

Predmeti su označeni kvadratićima s pripadajućim koordinatama i predstavljaju operacije koje sustav treba obraditi. S obzirom na poznatu poziciju svakog predmeta i njegovo kretanje, potrebno je odlučiti koji robot će preuzeti koji predmet.



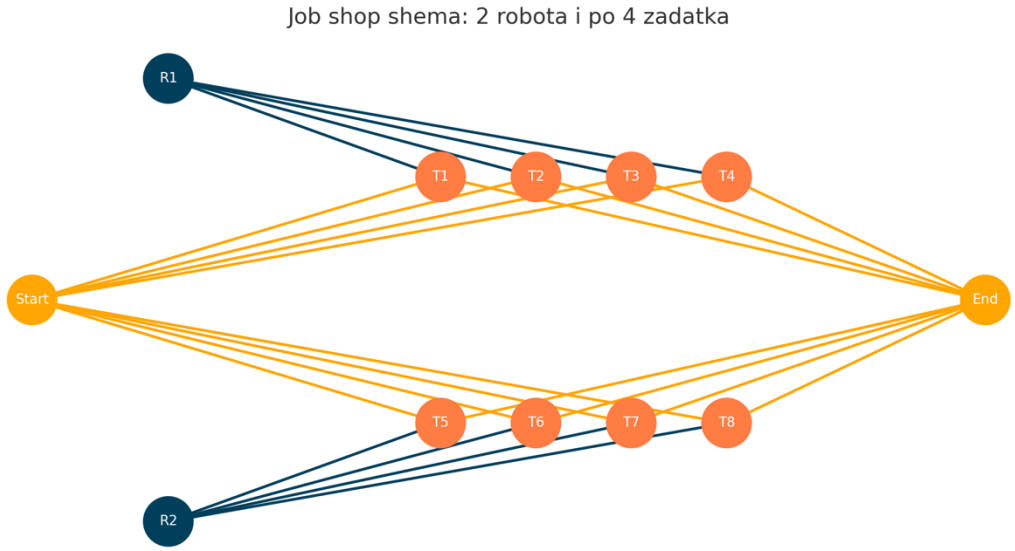
Slika 2.2 Prikaz sustava

Ovakav problem se može promatrati kao specifična varijanta problema fleksibilnog rasporeda poslova (engl. *Flexible Job Shop Scheduling Problem, FJSP*), gdje roboti preuzimaju ulogu strojeva, a objekti su operacije koje se izvršavaju.

3. Raspoređivanje zadataka

Problem raspoređivanja zadataka (engl. *Job Shop Scheduling Problem – JSSP*) jedan je od temeljnih problema u teoriji rasporeda i operacijskom istraživanju. Klasični oblik uključuje skup poslova, gdje se svaki sastoji od fiksnog slijeda operacija koje se moraju izvršiti na točno određenim strojevima. Svaki stroj može istovremeno obraditi najviše jednu operaciju, a cilj je pronaći raspored koji minimizira ukupno vrijeme trajanja (engl. *makespan*) ili neki drugi kriterij.

U fleksibilnoj varijanti ovog problema (engl. *Flexible Job Shop Scheduling Problem – FJSP*), uvodi se dodatna dimenzija slobode: svaka operacija može biti izvršena na jednom od više mogućih strojeva. Ovo proširenje znatno povećava složenost problema, ali i omogućuje realističnije modeliranje mnogih stvarnih sustava.



Slika 3.1 Shema job-shop sustava

Problem dodjele zadataka robotima uz pokretnu traku u ovom radu može se promatrati kao posebna varijanta FJSP-a, gdje:

- Predmeti predstavljaju operacije koje treba izvršiti,
- Roboti predstavljaju strojeve koji mogu izvršavati operacije,
- Svaki robot ima ograničen radni prostor i vrijeme, što uvodi dodatna ograničenja u usporedbi s klasičnim JSSP-om,

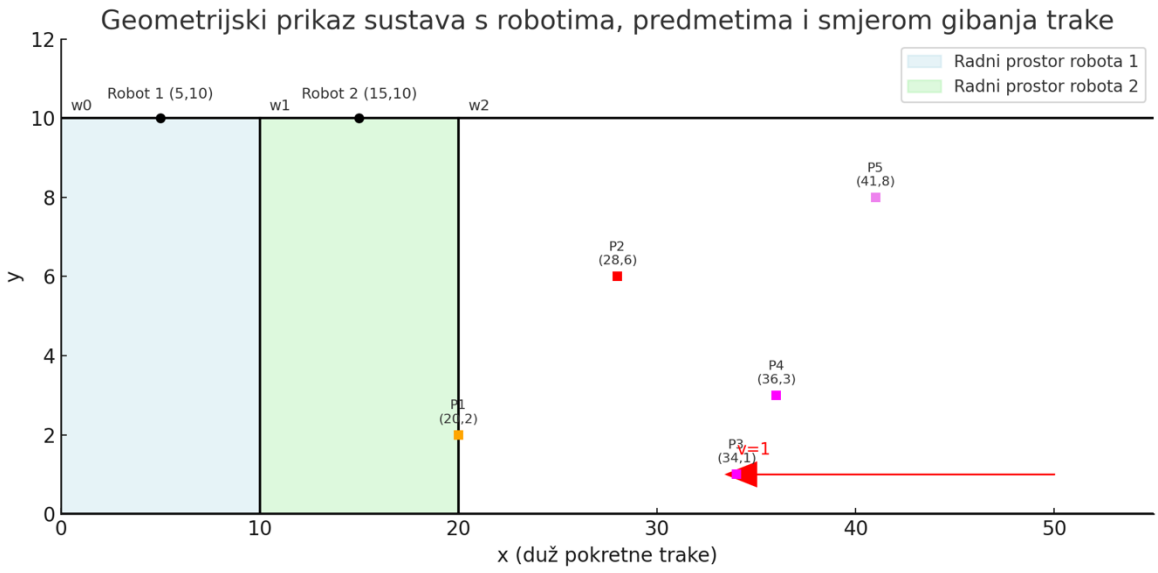
- Cilj nije minimizacija vremena već maksimizacija broja obavljenih zadataka (što je varijanta ciljne funkcije u FJSP-u).

Za rješavanje problema opisanog FJSP-om mogu se koristiti napredne metode rješavanja kao što su programiranje s ograničenjima, metaheuristike i kombinatorna optimizacija, što je upravo pristup korišten u ovom radu.

4. Raspoređivanje zadataka kao optimizacijski problem

4.1. Opis sustava

Zadan je skup zadataka J, pri čemu svaki zadatak $i \in J$ odgovara jednom predmetu na traci. Pozicija predmeta i u trenutku razmatranja je (x_i, y_i). Zadan je skup robota R. Za robota $k \in R$ definiraju se granice njegovog radnog prostora W_{k-1} i W_k . Također, definira se i za svakog robota k njegova pozicija (x^k, y^k). Brzine trake je v, a koordinatni sustav postavljen je prema slici 4.1.



Slika 4.1 Geometrijski prikaz sustava

4.2. Optimizacijski problem

Cilj je optimizacije odrediti vrijednosti sljedećih varijabli:

- $z_i \in \{0, 1\}$: označava je li paket i preuzet ili ne
- $\alpha_i^k \in \{0, 1\}$: označava je li zadatak i dodijeljen robotu k ili ne
- $t_i \in \mathbb{Z}_0$: vrijeme preuzimanja paketa
- $\beta_{ij} \in \{0, 1\}$: određuje redoslijed izvršavanja zadataka i i j na istom robotu

tako da se maksimizira broj preuzetih paketa, tj. sljedeća kriterijska funkcija:

$$max \sum_{i \in J} z_i \qquad (4\text{-}1)$$

Odabrano rješenje mora zadovoljavati ograničenja koja su opisana u sljedećem potpoglavlju.

4.3. Prostorno – vremenska ograničenja

Sustav se razmatra u trenutku $t = 0$. Definiraju se sljedeće vrijednosti:

tr_i^k - vrijeme kad posao $i \in J$ postaje dostupan robotu $k \in R$ (engl. *release time*). Ta vrijednost određena je radnim prostorom robota k i pozicijom predmeta i na traci.

$$tr_i^k = \frac{x_i - W_k}{v} \qquad (4\text{-}2)$$

td_i^k - vrijeme kad posao $i \in J$ postaje nedostupan robotu $k \in R$ (engl. *deadline time*), jer izlazi iz njegovog radnog prostota.

$$td_i^k = \frac{x_i - W_{k-1}}{v} \qquad (4\text{-}3)$$

t_i - trenutak u kojem robot uzima predmet i. Ako robot k uzima predmet, mora vrijediti:

$$tr_i^k \leq t_i \leq td_i^k \qquad (4\text{-}4)$$

T_i^k - vrijeme trajanja obrade posla i na stroju k, koje se sastoji od uzimanja predmeta te odlaganja predmeta. Ova vrijednost ovisi o položaju predmeta u trenutku kad se uzima s trake. Neka je t_i vrijeme u kojem robot k uzima predmet i s trake. Ako je u trenutku $t = 0$ predmet bio na poziciji (x_i, y_i), pozicija predmeta u trenutku t_i je $(x_i - v \cdot t_i, y_i)$, a T_i^k se može izračunati kao:

$$T_i^k = \frac{\sqrt{(x^k - x_i + v \cdot t_i)^2 + (y^k - y_i)^2}}{v} = f(t_i) \qquad (4\text{-}5)$$

Kako bi se osigurala linearna veza između T_i^k i t_i, koristi se aproksimirana vrijednost trajanja obrade:

$$T_i^k = \frac{x^k - x_i + v \cdot t_i}{v} + \frac{y^k - y_i}{v} \qquad (4\text{-}6)$$

Svaki zadatak koji se izvršava dodijeljen je najviše jednom robotu, tj. mora vrijediti:

$$\sum_{k \in R} \alpha_i^k = z_i \qquad (4\text{-}7)$$

Nadalje, ako se zadaci izvršavaju (ako je $z_i = 1$ i $z_j = 1$) i ako su dodijeljeni istom robotu (ako je $\alpha_i^k = 1$ i $\alpha_j^k = 1$), potrebno je osigurati da se ne izvršavaju istovremeno tj. da nema preklapanja zadataka na robotu. Ako je H oznaka za jako veliki broj, tada se ovo ograničenje može zapisati kao:

$$t_i \geq t_j + T_j^k - \left(4 - z_i - z_j - \alpha_i^k - \alpha_j^k + \beta_{ij}\right) \cdot H \qquad (4\text{-}8)$$

$$t_j \geq t_i + T_i^k - \left(5 - z_i - z_j - \alpha_i^k - \alpha_j^k - \beta_{ij}\right) \cdot H \qquad (4\text{-}9)$$

4.4. Primjer rješenja

Neka je zadano da u sustavu postoje dva robota $R = \{1,2\}$. Neka u sustavu postoji pet zadataka $J = \{1,2,3,4,5\}$. Ako se npr. predmet 5 preskače, a prva dva predmeta radi robot 1, a druga dva robot 2, rješenje će biti:

$$z_1 = 1, z_2 = 1, z_3 = 1, z_4 = 1, z_5 = 0$$

$$\alpha_1^1 = 1, \alpha_1^2 = 0$$

$$\alpha_2^1 = 1, \alpha_2^2 = 0$$

$$\alpha_3^1 = 0, \alpha_3^2 = 1$$

$$\alpha_4^1 = 0, \alpha_4^2 = 1$$

$$\alpha_5^1 = 0, \alpha_5^2 = 0$$

Ako se na robotu 1 prvo izvršava zadatak 1 pa zadatak 2, a na robotu prvo predmet 4, pa 3, vrijedit će:

$$\beta_{1,2} = 1, \beta_{2,1} = 0$$

$$\beta_{3,4} = 0, \beta_{4,3} = 1$$

Trebaju također biti određene vrijednosti t_1, t_2, t_3, t_4 koje su neki cijeli brojevi. Vrijednost $t_5 = 0$ jer se predmet ne uzima s trake. Rješenje koje nije moguće prekršit će barem jedan od napisanih uvjeta tj. ograničenja.

5. Implementacija rješenja

Implementacija rješenja izrađena je u programskom jeziku Python, korištenjem biblioteke Google OR-Tools, točnije CP-SAT rješavača koji omogućuje modeliranje i rješavanje problema programiranja s ograničenjima (engl. *Constraint Programming*).

U nastavku je opisana struktura programa, ključne varijable i funkcionalnosti, uz komentare i obrazloženja pojedinih implementacijskih odluka.

5.1. Korištene tehnologije

Google OR-Tools

Google OR-Tools je moćna, besplatna i open-source biblioteka za rješavanje problema iz područja operacijskog istraživanja (engl. *Operations Research*). Razvijena je od strane inženjera u Googleu kako bi se nosila s raznim izazovima rasporeda, rutiranja, optimizacije resursa i slično – u velikim, stvarnim sustavima poput onih koji se koriste u logistici, distribuciji i industrijskoj automatizaciji.

OR-Tools podržava nekoliko paradigmi rješavanja, uključujući:

- CP-SAT – moderni solver za programiranje s ograničenjima koji koristi kombinaciju propagacije, grananja i naprednih heuristika.
- Linearno i cjelobrojno programiranje (LP/MIP) putem popularnih rješavača kao što su GLOP, SCIP i Glpk.
- Probleme rutiranja (engl. *Routing Solver*) – posebno dizajniran za vozila, dostavne rute, planiranje putanja i sl.

CP-SAT rješavač

CP-SAT je glavni rješavač unutar Google OR-Tools biblioteke za rješavanje problema programiranja s ograničenjima (engl. *Constraint Programming, CP*). Naziv "SAT" dolazi od činjenice da se problemi transformiraju u varijantu problema zadovoljivosti Booleovih formula (SAT), čime se spaja snaga logičkog zaključivanja i kombinatoričke optimizacije.

Ključne značajke CP-SAT rješavača:

- **Podrška za Booleove i cjelobrojne varijable** – omogućuje modeliranje složenih uvjeta koji uključuju logičke odluke, vremenske ovisnosti i više-agentske interakcije (npr. više robota).
- **Učinkovita propagacija i domenska redukcija** – CP-SAT koristi napredne tehnike za eliminaciju nemogućih vrijednosti što značajno smanjuje veličinu pretražnog prostora.
- **Podrška za uvjetna ograničenja (engl. *reified constraints*)** – npr. OnlyEnforceIf(z[i]) omogućuje da se neko pravilo aktivira samo ako je zadatak aktivan, što je iznimno korisno u višerobotskim i dinamičkim sustavima.
- **Kombinacija CP + SAT + Integer Linear Programming** – CP-SAT je hibridni rješavač koji kombinira snagu različitih metoda: propagatora iz CP-a, zaključivanja iz SAT-a i presjeka domena iz ILP-a.
- **Skalabilnost** – iako je orijentiran prema "hard" kombinatoričkim problemima, CP-SAT je vrlo skalabilan i sposoban rješavati probleme s tisućama varijabli i ograničenja.

5.2. Inicijalizacija sustava

Na početku implementacije definiraju se ulazni parametri:

```
v = 1
robots = [(5, 10), (15, 10)] # pozicije robota
workspaces = [(10, 0), (20, 10)]  # dometi robota
T_MAX = 50
H = 1000   # velika konstanta
```

Zatim se korisnika traži unos broja predmeta, te se njihove početne pozicije generiraju nasumično unutar granica $x \in [20, 50]$ i $y \in [0, 10]$:

```
num_items = int(input("Unesi broj predmeta: "))
items = []
for i in range(num_items):
    x = random.randint(20, 50)
    y = random.randint(0, 10)
    items.append((x, y))
```

Na kraju se ispisuju generirane koordinate svakog predmeta:

```
for i, item in enumerate(items):
    print(f"Predmet{i+1} : ({item[0]}, {item[1]})")
```

5.3. Postavljanje modela i varijabli

Na početku implementacije definira se instanca `CpModel`, što je osnovni objekt za definiranje ograničenja i ciljne funkcije:

```
model = cp_model.CpModel()
```

Zatim se definiraju glavne varijable:

- `z[i]`: binarne varijable koje označavaju je li predmet i preuzet,
- `alpha[k][i]`: binarne varijable koje označavaju je li predmet i dodijeljen robotu k,
- `t[i]`: cjelobrojne varijable koje predstavljaju trenutak preuzimanja predmeta,
- `beta[i][j]`: binarne varijable za određivanje redoslijeda zadataka na istom robotu,
- `Tk[k][i]` – trajanje dohvaćanja predmeta i robotom k.

Varijable su kreirane korištenjem `NewBoolVar` i `NewIntVar` metoda, uz pažljivo zadane granice:

```
# z_i - indikator hoće li se predmet i uzeti
for i in range(num_items):
    z.append(model.NewBoolVar(f'z[{i}]'))

# t_i - cjelobrojno vrijeme početka dohvaćanja
predmeta i
for i in range(num_items):
    t.append(model.NewIntVar(0, T_MAX, f't[{i}]'))

# alpha_{k,i} - je li predmet i dodijeljen robotu k
for k in range(num_robots):
    row = []
    for i in range(num_items):
```

```
        row.append(model.NewBoolVar(f'alpha[{k}][{i}]'))
        alpha.append(row)

    # beta_{i,j} - je li i prije j na istom robotu
    for i in range(num_items):
        row = []
        for j in range(num_items):

    row.append(model.NewBoolVar(f'beta[{i}][{j}]'))
        beta.append(row)

    #Tk_{i,k} - vrijeme obrade posla i na stroju k
    for k in range(num_robots):
        row = []
        for i in range(num_items):
            var = model.NewIntVar(0, H,
    f'Tk[{k}][{i}]')
            row.append(var)
        Tk.append(row)
```

5.4. Implementacija ograničenja

Dodjela predmeta robotima

Prvo važno ograničenje je da svaki predmet može biti dodijeljen najviše jednom robotu:

```
for i in range(num_items):
   model.Add(sum(alpha[k][i] for k in
   range(num_robots)) == z[i])
```

Ako je `z[i] = 1`, tada je barem jedan `alpha[k][i] = 1`. Ako je `z[i] = 0`, tada nijedan robot ne smije dohvatiti taj predmet.

Računanje vremena trajanja obrade posla

Za svaki par (robot, predmet), izračunava se pozicija predmeta u trenutku dohvaćanja:

```
x_pos = xi - t[i] * v
```

Zatim se računa horizontalna udaljenost od robota:

```
abs_dx = |xk - x_pos|
```

Vertikalna udaljenost `|yk - yi|` se računa direktno jer je `yk` i `yi` konstanta. Ukupno trajanje obrade posla modelira se kao:

```
Tk[k][i] = abs_dx + dy
```

Vremenski prozor dostupnosti predmet

Predmet se može dohvatiti samo dok se nalazi unutar radnog prostora robota:

```
tr = max(0, int((xi - Wmin) / v))
td = max(0, int((xi - Wmax) / v))
model.Add(t[i] >= tr).OnlyEnforceIf(alpha[k][i])
model.Add(t[i] <= td).OnlyEnforceIf(alpha[k][i])
```

Ovo je implementirano putem `OnlyEnforceIf`, kako bi se uvjeti aktivirali samo ako je predmet stvarno dodijeljen robotu k.

Izbjegavanje preklapanja

Za svaki par predmeta (i, j) i svakog robota k, koristi se tzv. diskretna disjunkcija za izbjegavanje preklapanja:

```
model.Add(t[i] >= t[j] + Tk[k][j] - H * (4 - z[i] -
z[j] - alpha[k][i] - alpha[k][j] + beta[i][j]))

model.Add(t[j] >= t[i] + Tk[k][i] - H * (5 - z[i] -
z[j] - alpha[k][i] - alpha[k][j] - beta[i][j]))
```

Ove dvije linije zajedno osiguravaju da se zadaci `i` i `j` ne preklapaju ako ih isti robot dohvaća. Koristi se velika konstanta `H` da se izrazi uvjetna aktivacija, ovisno o stanju varijabli `z`, `alpha` i `beta`.

5.5. Ciljna funkcija

Cilj optimizacije je maksimizirati broj uspješno preuzetih predmeta:

```
model.Maximize(sum(z))
```

Time se model vodi prema rješenju gdje se obavi što više zadataka, ali unutar definiranih vremenskih i prostorno-vremenskih ograničenja.

5.6. Rješavanje i ispis rješenja

Rješenje optimizacijskog problema se dobiva primjenom CP-SAT algoritma:

```
solver = cp_model.CpSolver()
status = solver.Solve(model)
```

Ako rješenje postoji, ispisuju za svaki predmet sljedeće informacije, prikazane na Slici 5.1.

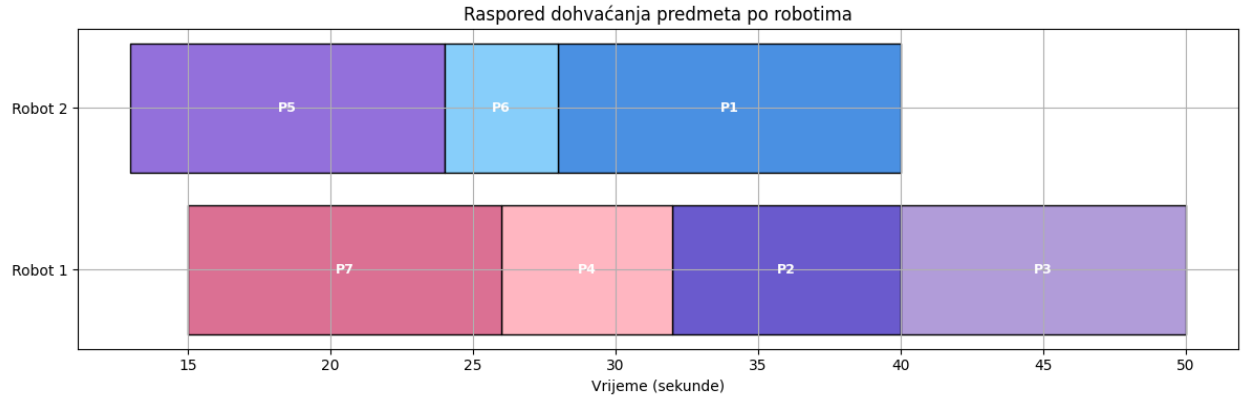
- je li predmet preuzet (`z[i]`)
- u kojem trenutku je preuzet (`t[i]`)
- koji robot ga je preuzeo (`alpha[k][i]`)
- koliko traje obrada posla (`Tk[k][i]`)

```
Unesi broj predmeta: 4
Predmet1 : (43, 10)
Predmet2 : (37, 6)
Predmet3 : (37, 1)
Predmet4 : (23, 8)

[RJEŠENJE] Broj uzetih predmeta: 4

Predmet1 uzet u t = 33
  uzima robot 2
  trajanje obrade posla: 5 sekundi

Predmet2 uzet u t = 28
  uzima robot 1
  trajanje obrade posla: 8 sekundi

Predmet3 uzet u t = 36
  uzima robot 1
  trajanje obrade posla: 13 sekundi

Predmet4 uzet u t = 12
  uzima robot 2
  trajanje obrade posla: 6 sekundi
```

Slika 5.1 Prikaz ispisa

5.7. Vizualizacija rasporeda

Za lakšu interpretaciju rješenja izrađuje se Ganttov dijagram pomoću funkcije `matplotlib`. Svaki predmet koji je preuzet prikazuje se kao obojana traka na vremenskoj osi za pripadajućeg robota. Predmet je označen oznakom `P1, P2`, itd.



Slika 5.2 Primjer Ganttovog dijagrama

Na ovaj način moguće je intuitivno vizualizirati redoslijed preuzimanja i stupanj iskorištenosti svakog robota.

5.8. Heuristički algoritam – pohlepna strategija

Osim optimizacijskog modela temeljenog na CP-SAT pristupu, implementirana je i pohlepan algoritam za dodjelu paketa robotima koji ih preuzimaju s pokretne trake. Cilj algoritma je svakom paketu dodijeliti najraniji mogući trenutak u kojem ga može preuzeti neki od raspoloživih robota, vodeći računa o njihovim radnim prostorima i međusobnom nepreklapanju zadataka.

Logika algoritma

Algoritam iterira kroz sve predmete sortirane po koordinati x (odnosno, po redoslijedu pristizanja s trake). Za svaki predmet se određuje je li u radnom području nekog robota i dodjeljuje se onom koji ga može prvi dohvatiti.

```
for i, (x, y) in enumerate(items):
    for k in range(num_robots):
        xk, yk = robots[k]
        Wmin, Wmax = workspaces[k]
        if Wmin <= x <= Wmax:
            tr = max(0, int((xi - Wmin) / v))
          td = max(0, int((xi - Wmax) / v))
          dy = abs(yk - yi)
          for t_candidate in range(tr, td + 1):
              x_current = xi - v * t_candidate
```

```
dist_x = abs(xk - x_current)
Tk_val = dist_x + dy
...
```

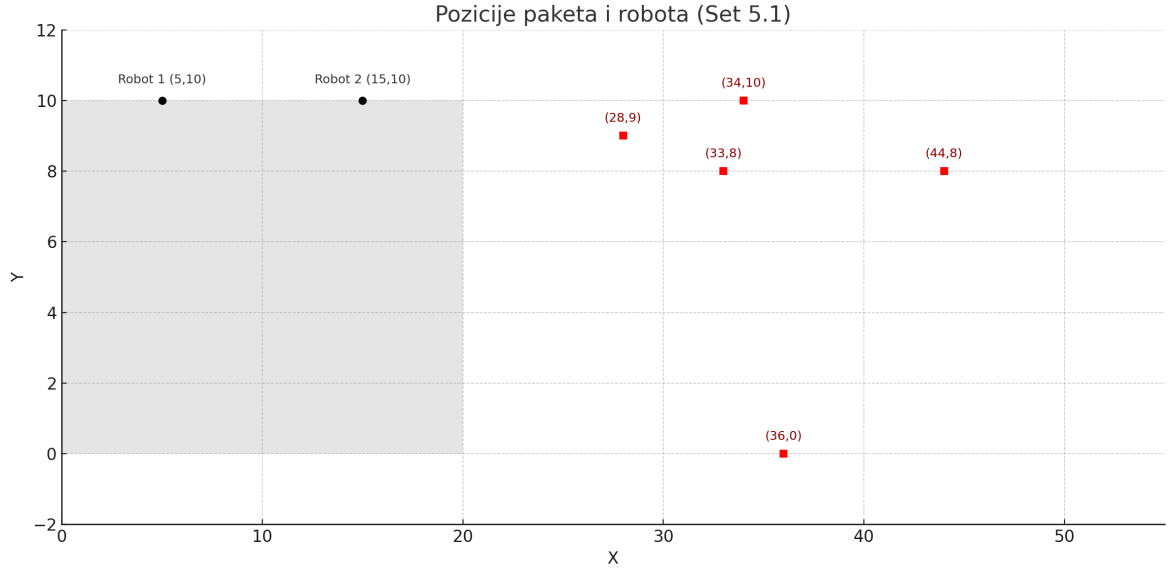
Ako se predmet nalazi unutar radnog prostora određenog robota, tada algoritam izračunava udaljenost između robota i predmeta (`dist_x + dy`) te provjerava je li robot slobodan za dohvat predmeta. U tom slučaju predmet se dodjeljuje tom robotu. U svakom trenutku kada je slobodan, robot dodjeljuje sebi najbliži dostupni predmet unutar svog radnog prostora. Budući da se predmeti obrađuju redom po x osi (što odgovara redoslijedu njihova dolaska), algoritam na prirodan način daje prednost onima koji prvi ulaze u sustav. Odluke se donose lokalno i trenutno – bez planiranja i bez uvida u cjelokupnu raspodjelu zadataka. Ovakav algoritam može dovesti do neravnomjerne raspodjele opterećenja među robotima i ne jamči maksimalnu iskorištenost sustava.

6. Analiza rezultata

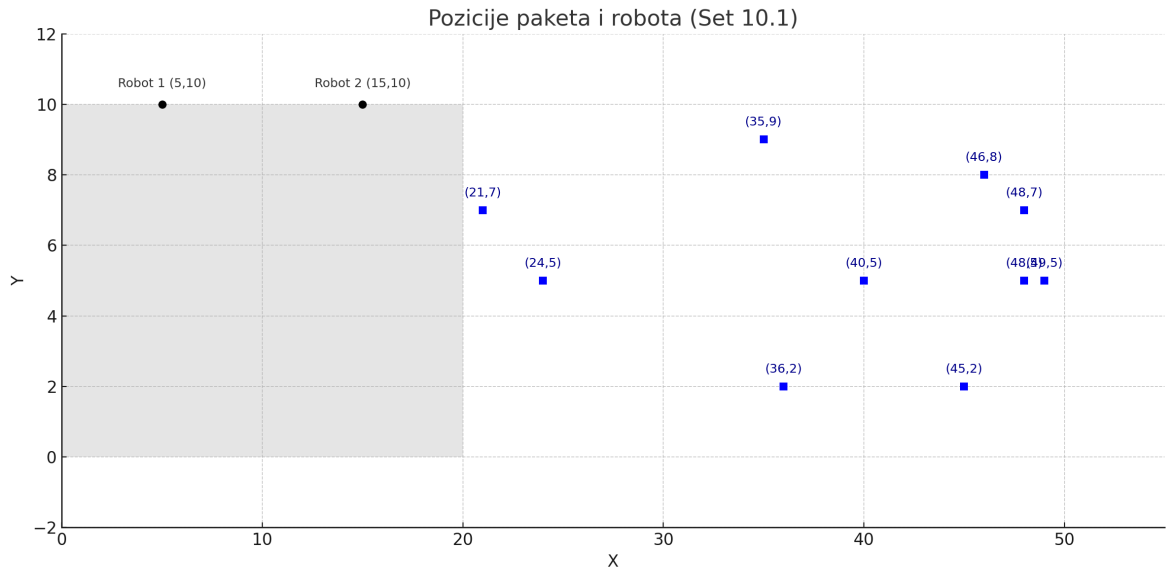
6.1. Usporedba rješenja pohlepnog i optimizacijskog algoritma

U sklopu ovog rada implementirana su dva pristupa za dodjelu zadataka robotima koji preuzimaju predmete s pokretne trake: pohlepni heuristički algoritam, koji u svakom trenutku uzima predmet najbliži robotu te algoritam koji koristi optimizacijski pristup s ciljem maksimizacije ukupnog broja preuzetih predmeta.

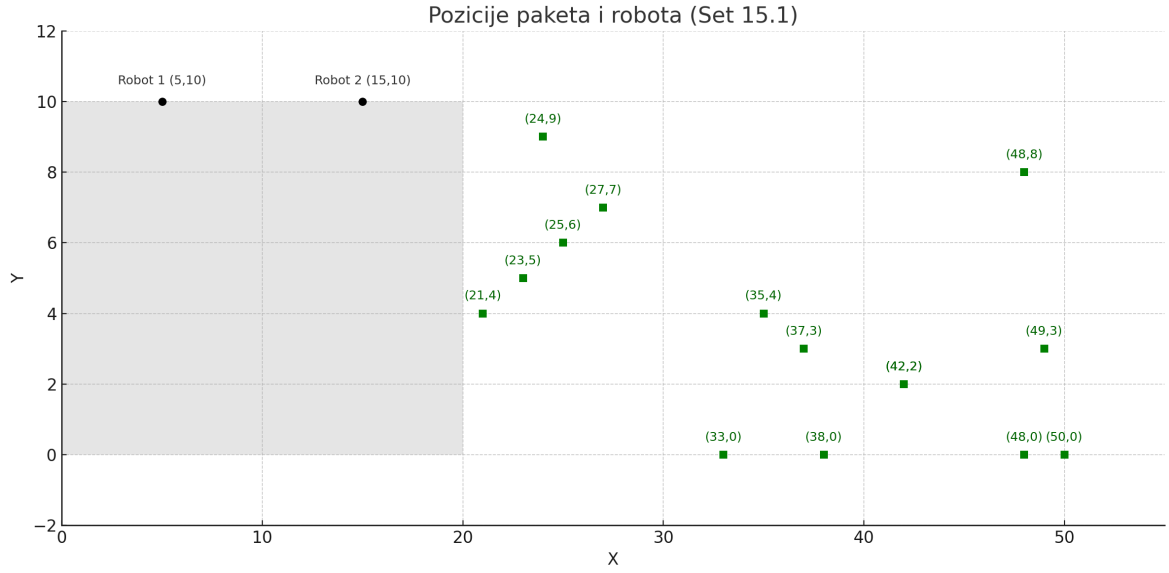
Učinkovitost algoritama evaluirana je na 30 unaprijed generiranih skupova podataka, pri čemu je analizirana izvedba na skupovima s 5, 10 i 15 predmeta (po 10 slučajeva za svaki broj predmeta). Skupovi se nalaze unutar `.json` datoteke priložene u radu. Primjeri nekih od odabranih instanci prikazani su na slikama 6.1 – 6.3, koje ilustriraju raspored predmeta i robota u prostoru.



Slika 6.1 Primjer testnog skupa sa 5 predmeta



Slika 6.2 Primjer testnog skupa sa 10 predmeta



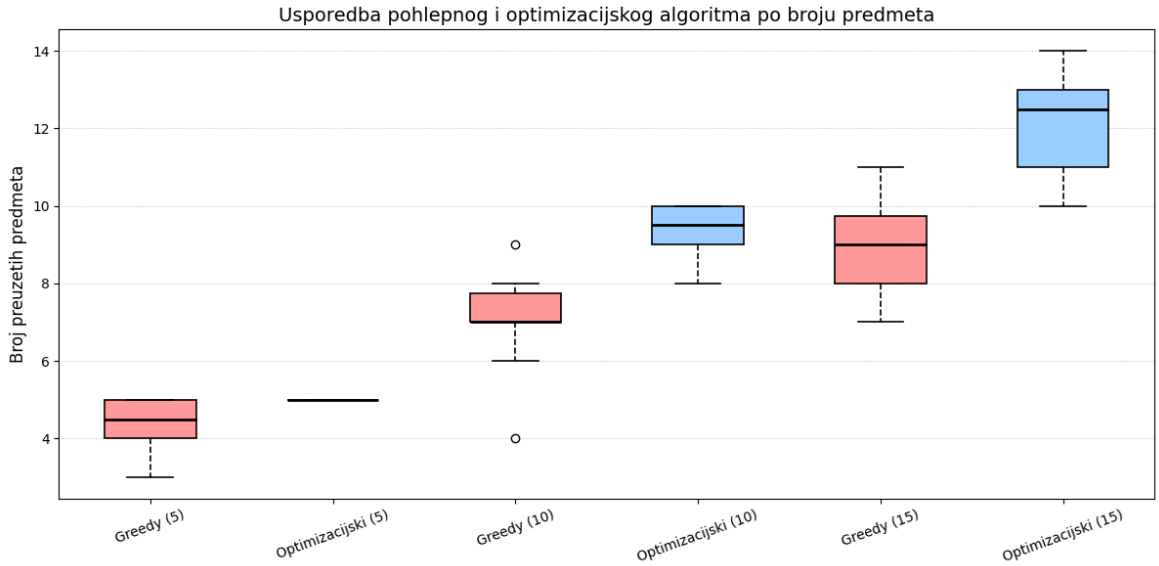
Slika 6.3 Primjer testnog skupa sa 15 predmeta

Za svaku instancu izvršeno je pokretanje oba algoritma, a rezultati su uspoređeni prema broju uspješno preuzetih predmeta. Rezultati su prikazani pomoću boxplot dijagrama (Slika 6.4), koji daje uvid u raspodjelu postignuća oba algoritma.

Iz dobivenih rezultata vidljivo je da optimizacijski algoritam nadmašuje pohlepan pristup u svim slučajevima. Za skupove od 5 predmeta razlika je minimalna jer su svi predmeti preuzeti. Međutim, kako broj predmeta raste, prednost optimiziranog algoritma postaje sve

izraženija. Pohlepan algoritam pokazuje veću varijabilnost rezultata i često ne uspijeva ostvariti maksimalan broj preuzetih predmeta.

Optimizacijski algoritam dosljedno ostvaruje rezultate blizu maksimuma, čime potvrđuje svoju učinkovitost u složenijim scenarijima. Međutim, zbog veće računalne složenosti njegova primjena može biti vremenski zahtjevnija, osobito kod većeg broja predmeta i robota. S druge strane, pohlepan algoritam, iako jednostavniji i brži, postiže slabiju ukupnu učinkovitost.



Slika 6.4 Dijagram usporedbe

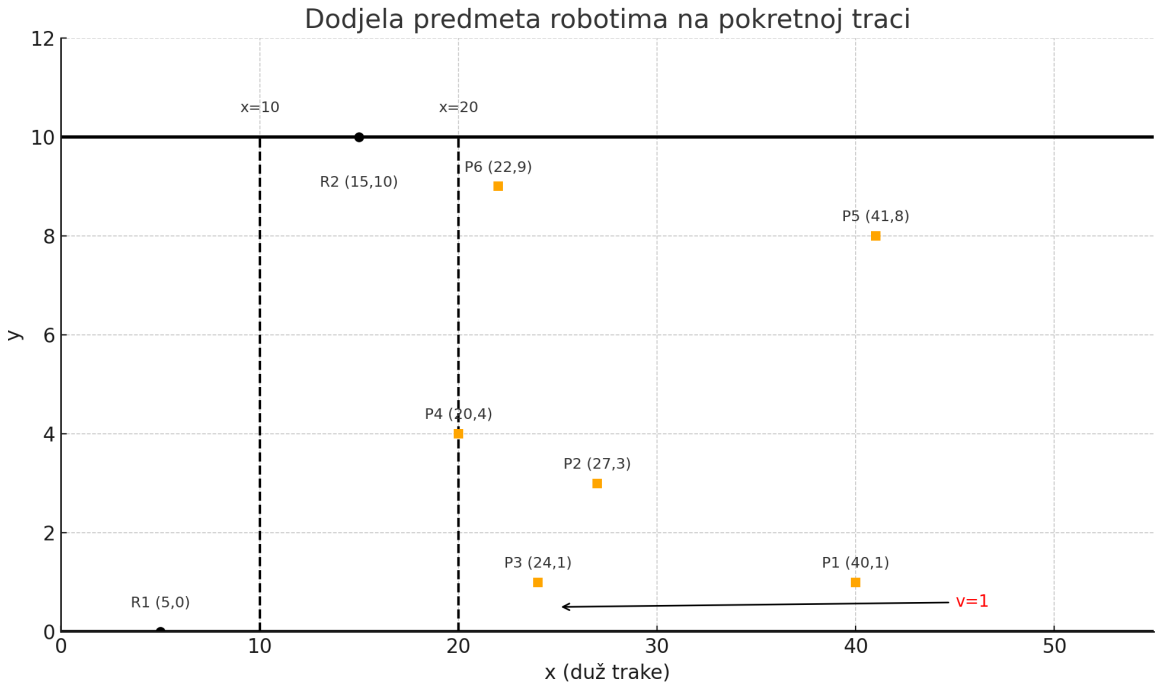
Na boxplot dijagramu (Slika 6.4) prikazane su raspodjele rezultata za pohlepni i optimizacijski algoritam pri različitom broju predmeta. Iz rezultata je uočljivo da optimizacijski algoritam daje vrlo konzistentne rezultate – svi izlazi su blizu maksimuma, a kutije (interkvartilni raspon) su uske ili potpuno stisnute, što ukazuje na nisku varijabilnost i visoku pouzdanost algoritma. Pohlepni algoritam, s druge strane, pokazuje veću disperziju rezultata, osobito kod 10 i 15 predmeta. Pojavljuju se slučajevi gdje broj preuzetih predmeta znatno odstupa od medijana, što znači da algoritam često ovisi o konkretnoj instanci zadatka.

Kod 5 predmeta razlike su manje izražene jer je problem dovoljno jednostavan da i pohlepan pristup često uspije pokupiti sve predmete. Kod 10 i 15 predmeta vidljiv je jasniji kontrast – optimizacijski algoritam uvijek bolje iskorištava resurse do maksimuma, dok pohlepni može propustiti do 4–5 predmeta. Brkovi (engl. *whiskers*) u boxplotu za pohlepni algoritam pokazuju prisutnost ekstremnih slučajeva gdje je izvedba izrazito loša, dok optimiziranog algoritam takvih odstupanja nema.

6.2. Usporedba rješenja za različite položaje robota

Za dodatnu evaluaciju optimizacijski algoritma provedeno je ispitivanje njegove osjetljivosti na početne položaje robota. Cilj je bio utvrditi koliko raspored robota u prostoru utječe na ukupnu učinkovitost preuzimanja predmeta.

U tu svrhu testirani su identični skupovi podataka kao u prethodnoj analizi (5, 10 i 15 predmeta), ali su pozicije robota bile postavljene u obrnutom redoslijedu u odnosu na originalnu konfiguraciju – svaki robot s jedne strane pokretne trake u odnosu na početan položaj u kojem su oba robota bila s iste strane trake. Prikaz takvog sustava vidljiv je na Slici 6.5.

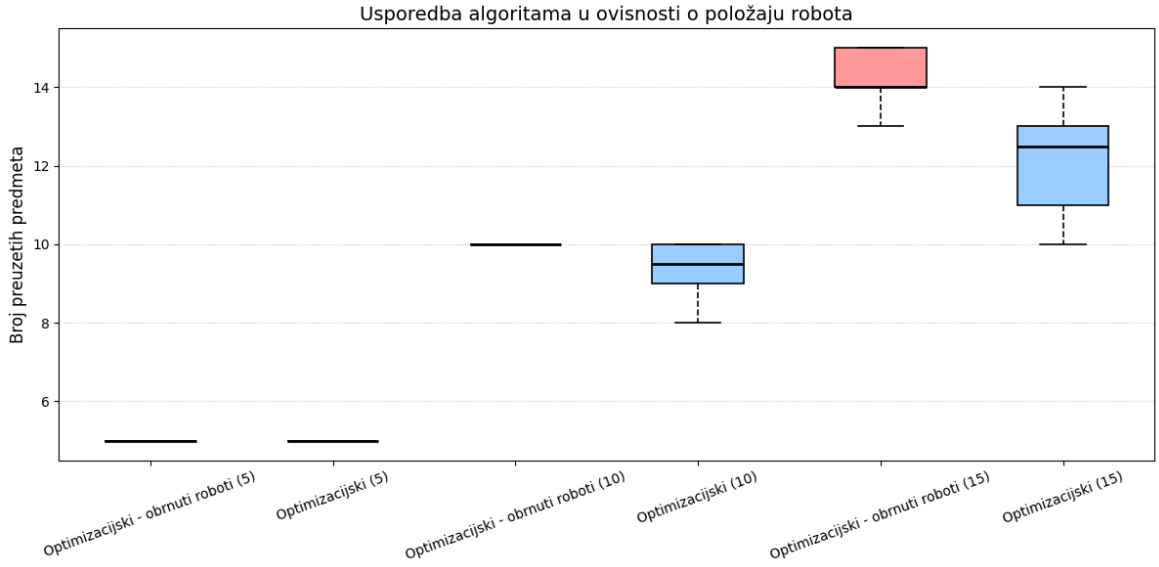


Slika 6.5 Prikaz sustava

Rezultati su prikazani u obliku boxplot dijagrama (Slika 6.6), gdje su uspoređene vrijednosti za originalnu i obrnuto postavljenu konfiguraciju robota za svaki testni slučaj.

Dijagram pokazuje da položaj robota ima značajan utjecaj na broj uspješno preuzetih predmeta, osobito kod većeg broja predmeta. Kod 5 predmeta razlike nisu značajne jer je broj mogućih kombinacija mali i jednostavan za optimizaciju. Međutim, kod 15 predmeta, prosjek u obrnutoj konfiguraciji nadmašuje standardnu, što ukazuje da u određenim situacijama i takvi rasporedi mogu rezultirati boljom koordinacijom.

Ovi rezultati ukazuju na važnost pažljivog dizajna početnih konfiguracija u stvarnim sustavima, ali i na robusnost algoritma u smislu prilagodbe na različite rasporede robota.



Slika 6.6 Dijagram usporedbe

Boxplot dijagram (Slika 6.6.) prikazuje utjecaj početnog rasporeda robota na učinkovitost preuzimanja predmeta.

Kod 5 predmeta razlike gotovo da nema – oba rasporeda rezultiraju identičnim rasponima i centralnim vrijednostima. Kod 10 predmeta, obrnut raspored robota postiže čak nešto bolje rezultate – svi rezultati su maksimalni, bez varijabilnosti. To sugerira da simetrična raspodjela robota uz traku može rezultirati boljom koordinacijom i raspodjelom posla. Kod 15 predmeta ta prednost postaje još izraženija: medijan i prosjek su viši, a varijabilnost manja. To ukazuje da raspored robota može imati značajan utjecaj kod većih opterećenja i složenijih konfiguracija. Boxplotovi za obrnuti raspored pokazuju uži interkvartilni raspon i kraće brkove, što znači da je sustav stabilniji u izvedbi.

Ovi rezultati potvrđuju važnost pažljivog dizajna rasporeda robota u stvarnim industrijskim primjenama, osobito kad je zadatkovno opterećenje visoko.

7. Zaključak

U ovom završnom radu istražen je problem dodjele zadataka više robota u sustavu s pokretnom trakom, što je tipičan scenarij u suvremenoj industrijskoj automatizaciji. Cilj je bio razviti i usporediti algoritme koji dodjeljuju predmete robotima na način koji maksimizira učinkovitost sustava, mjerenu kroz broj uspješno preuzetih predmeta.

Implementirana su dva pristupa: pohlepni algoritam temeljen na heuristici, koji donosi lokalno optimalne odluke, i optimizacijski algoritam zasnovan na programiranju s ograničenjima (CP-SAT), koji globalno maksimizira broj preuzetih predmeta uz poštivanje prostorno-vremenskih ograničenja.

Eksperimentalna analiza pokazala je da optimizacijski pristup daje značajno bolje rezultate, osobito u složenijim scenarijima s većim brojem predmeta. Iako je pohlepni algoritam brži i jednostavniji, njegova učinkovitost varira ovisno o rasporedu predmeta i robotima. Također je pokazano da raspored robota u prostoru može značajno utjecati na konačnu učinkovitost sustava, što potvrđuje važnost pažljive konfiguracije u praksi.

Dobiveni rezultati potvrđuju da primjena formalnih optimizacijskih metoda poput CP-SAT-a omogućuje pouzdano i učinkovito upravljanje višerobotskim sustavima, te otvaraju prostor za daljnji razvoj i proširenje modela, primjerice uvođenjem prioriteta, vremenskih prozora ili energetske optimizacije.

Literatura

[1] Google OR-Tools. Google Developers. Poveznica: https://developers.google.com/optimization

[2] Google Charts Documentation – Boxplot. Google Developers. Poveznica: https://developers.google.com/chart/interactive/docs/gallery/boxplot

[3] Jia, Y., Tang, H., Yang, B., Wang, Q. Multi-robot task assignment and scheduling for moving targets with time windows. *European Journal of Operational Research*, 311(2) (2023), str. 778–793. Poveznica: https://www.sciencedirect.com/science/article/pii/S037722172300382X

[4] Wikipedia. *Job-shop scheduling*. Poveznica: https://en.wikipedia.org/wiki/Job-shop_scheduling

[5] Plotly Technologies Inc. *Creating Gantt Charts in Python*. Poveznica: https://plotly.com/python/gantt/

[6] Tang, H., Zhang, Y., Zhang, W., & Tian, Y. A task allocation framework for multi-robot systems targeting moving objects. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Montreal, Canada (2019), str. 6153–6159. Poveznica: https://par.nsf.gov/servlets/purl/10149453

[7] Python Software Foundation. *Python Documentation*. Dostupno na: https://www.python.org/doc/

Sažetak

Naslov:

Algoritam za raspoređivanje zadataka za više robota uz pokretnu traku

Sadržaj:

Robotske ruke koje skupljaju predmete na pokretnoj traci koriste se u brojnim industrijskim primjenama. Ovisno o položaju objekata i robota, potrebno je robotima dodijeliti objekte tijekom rada sustava kako bi se postigli što bolji parametri učinkovitosti kao što su broj pokupljenih predmeta, iskorištenost robota itd. U ovom završnom radu implementirana su dva algoritma za raspoređivanje zadataka u sustavima s dvjema robotskim rukama koje prikupljaju predmete s pokretne trake. Opisan je optimizacijski problem dodjele zadataka robotima u uvjetima u kojima se položaji predmeta i robota mijenjaju tijekom rada sustava. Odabrana su dva pristupa rješavanju problema: algoritam dinamičkog programiranja i algoritam temeljen na pravilima prioriteta. Algoritmi su implementirani u programskom jeziku Python. Provedena je simulacijska analiza učinkovitosti rješenja za različite skupove ulaznih podataka. Uspoređeni su rezultati u pogledu broja uspješno pokupljenih predmeta te iskorištenosti robota. Na temelju dobivenih rezultata dani su zaključci o prikladnosti i učinkovitosti primijenjenih algoritama za opisani problem.

Ključne riječi:

roboti, Python, optimizacijski algoritam, predmeti, Or-Tools

Summary

Title:

Algorithm for task scheduling for multiple robots along a conveyor belt

Summary:

Robotic arms that pick up objects from a conveyor belt are used in numerous industrial applications. Depending on the positions of the objects and robots, it is necessary to assign objects to the robots during the system's operation in order to achieve optimal performance indicators such as the number of collected items, robot utilization, etc. In this thesis, two task allocation algorithms were implemented for systems with two robotic arms collecting items from a conveyor belt. The task allocation problem was described as an optimization problem under conditions where the positions of objects and robots change during system operation. Two solution approaches were selected: a dynamic programming algorithm and a priority rule-based algorithm. The algorithms were implemented in the Python programming language. A simulation-based performance analysis was conducted for different sets of input data. The results were compared in terms of the number of successfully picked items and the utilization of the robots. Based on the obtained results, conclusions were drawn regarding the suitability and efficiency of the applied algorithms for the described problem.

Keywords:

robots, Python, optimization algorithm, items, OR-Tools

Privitak

GitHub repozitorij sa svim kodovima, datotekama i dijagramima

https://github.com/ellagrkovic/ZavrsniRad