SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

ANITA GUDELJ

OPTIMALIZACIJA SUSTAVA S DISKRETNIM DOGAĐAJIMA PRIMJENOM PETRIJEVIH MREŽA I GENETSKIH ALGORITAMA

Doktorska disertacija

VARAŽDIN, 2010.

PODACI O DOKTORSKOJ DISERTACIJI

I. Autor

Ime i prezime	Anita Gudelj
Datum i mjesto rođenja	4. veljače 1970. Split
Naziv fakulteta i datum diplomiranja	Prirodoslovni matematički fakultet, Split 5.11. 1993.
Sadašnje zaposlenje	Viši predavač, Sveučilište u Splitu, Pomorski fakultet Split

II. Doktorska disertacija

Naslov	Optimalizacija sustava s diskretnim događajima primjenom Petrijevih mreža i genetskih algoritama
Broj stranica, slika, tabela, priloga, bibliografskih podataka	
Znanstveno područje, smjer i disciplina iz koje je postignut akademski stupanj	
Mentor ili voditelj rada	Voditelj: Prof. dr. sc. Stjepan Vidačić Suvoditelj: Prof.dr.sc. Danko Kezić
Fakultet na kojem je rad obranjen	Fakultet organizacije i informatike u Varaždinu
Oznaka i redni broj rada	

III. Ocjena i obrana

Datum prihvaćanja teme od Znanstveno_nastavnog vijeća	
Datum predaje rada	
Datum sjednice ZNV-a na kojoj je prihvaćena pozitivna ocjena rada	
Sastav Povjerenstva koje je rad ocjenilo	
Datum obrane rada	
Sastav Povjerenstva pred kojim je rad obranjen	
Datum promocije	

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

ANITA GUDELJ

OPTIMALIZACIJA SUSTAVA S DISKRETNIM DOGAĐAJIMA PRIMJENOM PETRIJEVIH MREŽA I GENETSKIH ALGORITAMA

Doktorska disertacija

VARAŽDIN, 2010.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

ANITA GUDELJ

Optimalizacija sustava s diskretnim događajima primjenom Petrijevih mreža i genetskih algoritama

Doktorska disertacija

Voditelji rada:

Prof.dr. sc. Stjepan Vidačić

Suvoditelj:
Prof. dr. sc. Danko Kezić

VARAŽDIN, 2010.

Predgovor

Određivanje rasporeda poslova je važno pitanje u planiranju i upravljanju proizvodnim, prometnim, te drugim tehničkim i netehničkim sustavima i ima veliko značenje za postizanje visoke ekonomske učinkovitosti. Treba odrediti slijed izvođenja poslova na skupu resursa tako da ukupno vrijeme izvođenja bude što kraće. Posljednjih 15-tak godina, genetski algoritmi se primjenjuju na mnoge kombinatoričke probleme, uključujući i raspoređivanje. Razlog tome je što se zna da genetski algoritmi učinkovito traže rješenje u velikom prostoru pretraživanje, bez eksplicitnih zahtijeva za dodatnim informacijama o funkciji cilja koja se optimalizira. Problem je što raspoređivanje spada u skupinu NP teških problema. To znači da što je sustav složeniji, to uključuje više resursa i poslova, više ograničenja povećava prostor pretraživanja, te se vrijeme izvođenja algoritma eksponencijalno povećava s brojem poslova. Dakle, potrebno je podesiti parametre genetskog algoritma tako da se teži problemi riješe u što je moguće kraćem vremenu.

Pri određivanju optimalnog rasporeda ne smije se zanemariti činjenica da su resursi ograničenog kapaciteta, da resursi mogu mogu posluživati jedan ili više poslova, te da se mora zadovoljiti ograničenje na slijed izvođenja poslova. Problemi koji proizlaze iz postojanja višeradnih resursa su nedozvoljena stanja u kojima se sustav može naći, primjerice konflikti, zastoji i sudari, te stoga treba primijeniti određenu strategiju izbjegavanja tih stanja. Radi toga je potrebno projektirati nadzornik koji ima zadatak da nadzire stanje u sustavu i blokira određene događaje radi izbjegavanja nedozvoljenih stanja. Time problem raspoređivanja postaje još složeniji, jer je važno da sustav pouzdano i sigurno izvršava zadani slijed poslova.

Rezultat genetskog algoritma u određivanju rasporeda poslova je numeričko rješenje iz kojeg krajnji korisnik ne vidi eventualno postojanje nelogičnog slijeda poslova. Zato je uz optimalizaciju, potrebno sustav i vizualno simulirati kako bi se uočile i ispravile eventualne nepravilnosti u izvođenju slijeda poslova. Proizvodni i prometni sustavi mogu se opisati kao sustavi s diskretnim događajima (DES), trebalo je pronaći takav model DES koji bi, s jedne strane, osigurao stabilan, siguran i optimalan rad u izvođenju poslova, a s druge strane, koji bi bio dovoljno jednostavan, lagan za implementaciju i općenito učinkovit u praktičnoj primjeni.

U ovom se radu sustavno obrađuje metodologija višekriterijskog raspoređivanja poslova u sustavu s diskretnim događajima (DES) pomoću genetskog algoritma i simulacijskog modela promatranog sustava pomoću MRF_1 Petrijevih mreža. MRF_1 je podklasa Petrijevih mreža za sustave proizvodnih linija ili engl. *Flowline System Petri Net* (FPM). FPM je namijenjena analizi sustava koji pripadaju klasi nazvanoj višeprolazne proizvodne linije ili engl. *multiple reentrant flowlines* (MRF). Riječ je o klasi sustava koja opisuje sustave sa slijedom zadataka, a jedna od značajki te klase jest postojanje višeradnih resursa. Odabran je MRF_1 model, jer se takav sustav može opisati matričnim modelom koji se jednostavno integrira u GA. MRF1 model omogućava jednostavan način proračuna nadzornika koji upravlja sustavom na način da ne dođe do konflikta i zastoja. Kada se jednom riješi pitanje stabilnosti i zastoja, podacima iz tih istih matrica mogu se pronaći bolji i ekonomičniji rasporedi izvršavanja zadataka. Kada će koji skup resursa biti korišten za izvođenje određenih poslova određeno je upravljanjem.

U ovom radu je razvijen algoritam integracije koji je testiran računalnom simulacijom na dva primjera pomorskih prometnih sustava. Prvi sustav je sustav kanala i bazena u kojem

do potpunog zastoja može doći neodgovarajućim zauzimanjem kanala i/ili bazena od strane brodova koji prolaze suprotnim smjerovima. Pokazano je da integracija GA i PM omogućava proračun upravljanja uz optimalan raspored poslova kako bi brodovi što brže prošli sustavom kanala. Algoritam uspješno upravlja prometnom signalizacijom sustava kanala i bazena na način da sprječava moguću pojavu potpunog zastoja, a da u isto vrijeme što manje ometa promet brodova.

Drugi sustav je kontejnerski terminal. Razmatra se problem rasporeda poslova za automatski upravljana vozila (AGV) koja prevoze kontejnere unutar kontejnerskog prostora. AGV sustav je važan dio svakog automatiziranog kontejnerskog terminala i rad ovog sustava se odražava na učinkovitost cijelog sustava, odnosno na vrijeme koje pojedini brod provede u luci. Stoga je za održavanje konkurentne prednosti i povećanja učinkovitosti kontejnerskog terminala potrebno odrediti odgovarajući broj AGV-a za implementaciju, te formulirati dobru strategiju raspodjele poslova za AGV-ove. Veliki problem AGV-ova su zastoji. Potpuni zastoj može nastati, primjerice, u slučaju otkaza jedne zajedničke skladišne dizalice. Mnogo je radova koji su se bavili problemom raspoređivanja AGV-ova u okruženju kontejnerskog terminala. Međutim, u tim istraživanjima se ne razmatra problem konflikta i potpunog zastoja uslijed korištenja zajedničkih resursa. Stoga je cilj ovog rada razviti algoritam upravljanja bez konflikta i zastoja uz minimalno vrijeme čekanja AGV-ova na dizalice i vrijeme koje brod provede u luci, na način da se integrira Petrijeva mreža i genetski algoritam. Primjenom integracije MRF_1 Petrijeve mreže i genetskog algoritma koji se predlaže u ovom radu, kao i matrične metode proračuna nadzornika za sprječavanje potpunog zastoja, omogućava se uspješan protok kontejnera na terminalu, dodjeljivanje zajedničkih resursa bez konflikta i potpunog zastoja. Algoritam se može koristiti i za određivanje potrebnog broja AGV-ova u konkretnom sustavu.

Razvijeni algoritmi su vrednovani s različitim veličinama populacije kako bi se ispitao utjecaj tog parametra na konvergenciju rezultata ka konačnom rješenju. Kod statičke veličine populacije, uočeno je da broj jedinki u populaciji može biti jako velik (200 i više), a da pri tome nema značajnijeg poboljšanja rezultata. Rješenje problema je u dinamičkoj prilagodbi veličine populacije trenutnom rješenju. Kako bi se poboljšao rezultat, a da se pri tome značajno ne poveća vrijeme izvođenja i računalni napor, razvijen je GA kod kojeg se reprodukcija odvija prema Fibonaccijevom nizu i predložen je mehanizam dinamičke promjene veličine populacije. Rezultati testiranja pokazali su ispravnost ovog pristupa.

Iskreno se zahvaljujem voditelju Prof. dr.sc. Stjepanu Vidačiću i suvoditelju Prof. dr.sc. Danku Keziću na poticanju, razumijevanju, korisnim sugestijama i vođenju u izradi ovog rada.

Sadržaj

Popis oznaka

A	Skup usmjerenih lukova Petrijeve mreže
$\mathbf{A} = \left[a_{i,j} \right]_{m \times n}$	Matrica događanja Petrijeve mreže
$\mathbf{A}^{+} = \left[a_{i,j}{}^{+} \right]_{m \times n}$	Ulazna matrica događanja
$\mathbf{A}^{-} = \left[a_{i,j}{}^{-} \right]_{m \times n}$	Izlazna matrica događanja
$\mathbf{A'} = \left[a_{i,j}{}' \right]_{m \times n}$	Modificirana matrica događanja
$\mathbf{A}^{+\prime} = \left[a_{i,j}{}^{+\prime} \right]_{m \times n}$	Modificirana ulazna matrica događanja
$\mathbf{A}^{-\prime} = \left[a_{i,j}{}^{-\prime} \right]_{m \times n}$	Modificirana izlazna matrica događanja
CO	Kontrolno mjesto u PM
c_{max}	Ukupno vrijeme izvođenja svih poslova (makespam)
ERT_i	Najraniji početak (engl. *Earliest Release Time*) za projekt *i*
DD_i	Krajnji rok (engl. *Due Date*) za izvođenje *i*-tog projekta.
$\boldsymbol{DelayGen_j}$	Vrijeme kašnjenja posla *j*
d_j	Vrijeme trajanja izvođenja posla *j*
F_V	Matrica potrebnih poslova
F_R	Matrica potrebnih oznaka
F_U	Ulazna matrica
F	Matrica dobivena kombiniranjem matrica F_V i F_R
F_j	Vrijeme završetka posla $j \in \mathcal{J}$.
$f(m, t_j)$	Funkcija prijelaza stanja Petrijeve mreže
$f(v)$	Funkcija cilja
$I = \{1, 2, ..., n\}$	Skup od n redova matrice događanja $\mathbf{A}$
$\mathbf{I}$	Jedinična matrica ograničenja dimenzije $n_c \times n_c$
$\mathcal{J}$	Skup svih poslova u sustavu
$J(CW)$	Skup poslova u kružnom čekanju
$J(r)$	Skup poslova resursa *r*
$L(m_0)$	Skup svih mogućih sekvenci okidanja Petrijeve mreže koji su mogući iz stanja $\mathbf{m_0}$
m	Ukupni broj mjesta Petrijeve mreže
$m(p)$	Broj oznaka u mjestu p Petrijeve mreže
$m_0(p)$	Početni broj oznaka u mjestu p Petrijeve mreže
$\mathbf{m}$	Vektor stanja Petrijeve mreže
$\mathbf{m_0}$	Vektor početnog stanja Petrijeve mreže
m_f	Zavrsni dio mjesta
m_i	Ulazni dio mjesta
$\mathbf{m}\left[t_j > \mathbf{m'} \right.$	Prijelaz iz stanja $\mathbf{m}$ u stanje $\mathbf{m'}$ okidanjem prijelaza t_j Petrijeve mreže

n	Ukupni broj prijelaza Petrijeve mreže
N	Broj jedinki u populaciji u svakoj generaciji
$Nbit$	Broj bitova od kojih se sastoji binarni niz jedinke
$NE = (V, E)$	Graf Petrijeve mreže
P	Skup mjesta Petrijeve mreže
$\mathcal{P}$	Skup projekata
P_j	Skup svih poslova koji prethode poslu j
$PV(g)$	Veličina populacije za bilo koju generaciju g
(PM, m_0)	Opća Petrijeva mreža u početnom stanju
$PT = (P, T, A, w, m_0)$	PT Petrijeva mreža
p_{mut}	Vjerojatnost mutacije gena
p_{sel}^i	Vjerojatnost odabira i-te jedinke
$\bullet Q$	Skup prijelaza iz kojih idu lukovi prema zamki,
$Q \bullet$	Skup prijelaza u koje idu lukovi iz zamke.
$R(m_0)$	Skup svih mogućih dostupnih stanja PM koji se mogu doseći iz početnog stanja $\mathbf{m_0}$
$R = \{r_i\}$	Skup svih resursa u sustavu
$r_{j,k}$	Broj jedinica resursa $r_k \in R$ koji je potreban za izvođenje posla j
S	Sifon Petrijeve mreže
$\bullet S$	Skup prijelaza iz kojih idu lukovi prema sifonu,
$S \bullet$	Skup prijelaza u koje idu lukovi iz sifona.
S_V	Matrica pokretanja poslova
S_R	Matrica opuštanja resursa
S_Y	Izlazna matrica
$S^k{}_{uv}$	Vrijeme setiranja resursa r_k između poslova u i v
s_j	Vrijeme početka posla $j \in \mathcal{J}$
T	Skup prijelaza Petrijeve mreže
$\mathbf{u}$	Vektor okidanja Petrijeve mreže
$u(t)$	Vektor ulaznih varijabli u sustav
V	Skup čvorova Petrijeve mreže
$w(p_i, t_j)$	Faktor težine luka između p_i i t_j
X	Skup svih stanja u sustavu
$\mathbf{x}$	P-postojanost
$\|\mathbf{x}\|$	Podrška P-postojanosti
$\mathbf{X}$	Prostor stanja u sustavu s diskretnim događajima
$y(t)$	Vektor izlaznih varijabli iz sustava
$\mathbf{y}$	T-postojanost
$\|\mathbf{y}\|$	Podrška T-postojanosti
2^E	Partitivni skup skupa E (skup svih podskupova skupa E)

$\sigma = t_1 t_2 ... t_n$	Niz okidanja prijelaza Petrijeve mreže
Σ	Skup simbola koji predstavljaju događaje u sustavima s diskretnim događajima (abeceda)
μ	Srednja vrijednost
σ^2	Varijanca
$I(x)$ ili $\bullet x$ -	Skup ulaznih čvorova u čvor x Petrijeve mreže
$O(x)$ ili $x \bullet$	Skup izlaznih čvorova iz čvora x Petrijeve mreže
$PN(g)$	Parcijalni računalni napor po generaciji
d(x)	Funkcija dobrote,
e	Događaj u sustavu s diskretnim događajima
E	Skup događaja u sustavu s diskretnim događajima
g	generacija
J	Skup svih poslova u sustavu
PM	Opća Petrijeva mreža
PNtimes	Vektor vremena pridruženih mjestima PM
Q	Zamka
R	Skup resursa
RD(*i*)	Kapacitet *i*-tog resursa
RN	Računalni napor
s	Selekcijska razlika
s_I	Selekcijski intenzitet
v	Jedno rješenje optimalizacijskog problema
$\overline{d_p}$	Prosječne vrijednosti dobrote preživjelih jedinki
$\overline{d}$	Prosječne vrijednosti dobrote svih jedinki u svakoj iteraciji
$\Theta \subset R^N$	*N*-dimenzionalan prostor pretraživanja (odlučivanja)
$\Omega \subset R^n$	*n*-dimenzionalan prostor cilja (kriterija)
$G(x)$	Funkcijski vektori označava ograničenja u obliku nejednakosti
$H(x)$	Funkcijski vektor označava ograničenja u obliku jednakosti

Popis simbola

$\in$ - pripada skupu
$\subset$ - podskup
$\mathbf{A}$ - matrica
$\mathbf{A}^\mathbf{T}$ - transponirana matrica
$\bigcup$ - unija skupova
$\bigcap$ - presjek skupova
$\setminus$ - razlika skupova
$\times$ - operacija sinkronog umnoška automata
$\bullet$ - funkcija povezivanja procesa i nadzornika
$\varnothing$ - prazni skup
$\wedge$ - logički operator I
$\vee$ - logički operator ILI

Pokrate

AGV	*automatski upravljano vozilo*
AKT	*automatiziranih kontejnerskih terminala*
ASCs	*automatske skladišne dizalice* (engl. *Automated Stacking Cranes*)
CB	*kružno blokiranje*
CPM	*obojena Petrijeva mreža*
CW	*kružno čekanje*
DEDS	*dinamički sustav sa diskretnim događajima*
DES	*sustav sa diskretnim događajima*
DTPNs	*deterministične vremenske Petrijeve mreže*
EA	*evolucijski algoritam*
EDD	*Najkraći rok (engl. Earliest Due Date)*
EP	*evolucijsko programiranje*
ERT	*najraniji početak* (engl. *Earliest Release Time*)
FCFS	*koji prvi stigne, prvi je i poslužen* (engl. F*irst-come, First served*)
FIFO	*metoda "Prvi unutar, prvi vani" (engl. First In First Out)*
FMS	*fleksibilni proizvodni sustav*
FPM	*Petrijeva mreža za sustave proizvodnih linija*
GA	*genetski algoritam*
JSP	*problem rasporeda poslova*
MFR	*višeprolazne proizvodne linije (engl. Multiple Reentrant Flowlines)*
MOGA	*više-ciljni genetski algoritam*
PM	*Petrijeva mreža*
PTN	*Place-Transition Nets*
PTPN	*Place Timed Petri Nets*
QC	*obalna (kontejnerska) dizalica* (engl. *quay* (*gantry*) *cranes*)
SGA	*jednostavni genetski algoritam*
SC	*skladišna dizalica* (engl. *stack crane*)
SPMK	*sustav prometa morskim kanalima*
TEU	*Twenty-foot equivalent unit*
TPM	*vremenska Petrijeva mreža*
TTPN	*Transition Timed Petri Nets*
VEGA	*Vector Evaluated Genetic Algorithm*
VP	*varijacija populacije*
WIP	*poslovi koji su u obradi*

Popis slika

Popis tablica

1. UVOD

1.1. Uvodna razmatranja

Područje istraživanja sustava s diskretnim događajima (pokrata: DES) je relativno novo i kombinira različite formalizme, metodologije i alate iz teorije upravljanja, informacijskih znanosti i operacijskih istraživanja. Istraživanja DES su usmjerena na različite aplikacijske domene kao što su računalni i informacijski sustavi, proizvodni sustavi, sustavi automatizacije, sustavi za dijagnostiku kvarova, transportni sustavi i dr.

Područje istraživanja DES-a može se podijeliti u dvije domene:

- *Logički modeli* koji opisuju strukturna svojstva DES-a i definiraju slijed događaja u sustavu. Svrha im je upravljati slijedom događaja koje generira DES i ispitati strukturnu ispravnost toga slijeda. Kako bi se ostvario ovaj cilj, jedino je važno zadržati slijed u izvođenju događaja, a zanemaruje se vrijeme pojavljivanja događaja. Uobičajeni problemi koji se razmatraju unutar ove domene su izbjegavanje nedozvoljenih događaja ili nedozvoljenih stanja, izbjegavanje konflikata i zastoja, itd.
- *Vremenski modeli* koji se koriste za opisivanje kvantitativnih svojstava s ciljem upravljanja vremenima nastupanja događaja. Ovi modeli trebaju odgovoriti na pitanje kada će nastupiti događaj i nužno je zadržati vremenski slijed pojavljivanja svih događaja u sustavu. Uobičajeni problemi koji se razmatraju unutar ove domene su zadovoljiti vremenska ograničenja, optimalizirati vrijeme izvođenja, određivanje rasporeda, itd.

Kao metoda modeliranja i analize sustava s diskretnim događajima, Petrijeve mreže (pokrata: PM) su se pokazale uspješnim alatom za prikazivanje složenih sustava čije su osnovne značajke asinkronost, te postojanje paralelnih ili konfliktnih operacija [MUR89, GIR03]. PM su pogodne jer mogu jednostavno grafički prikazati problem i jednostavno modelirati zastoje, konflikte i ograničenja u sustavu [CAS99].

Model sustava s diskretnim događajima modeliran Petrijevom mrežom sadrži puno više strukturnih informacija o sustavu kojeg opisuje, nego što sadrži diskretni model dobiven primjerice teorijom automata. Jednostavniji je i pogodniji za modeliranje složenijih sustava te nije toliko izražen problem velikog broja različitih stanja (Valmari, 1996 citirano u [KEZ04]). Tako se pojednostavnjuje spoznavanje i razumijevanje proučavanog sustava. PM se intenzivno primjenjuje za analizu i proračun upravljanja proizvodnih sustava. Vrlo dobar pregled korištenja PM dan je u [MOO96, ZUR94].

Od nastanka Petrijevih mreža do današnjih dana, Petrijeve mreže se neprestano usavršavaju i nalaze sve veću primjenu posebno u područjima kao što su računalni sustavi i mreže, komunikacijski protokoli, fleksibilni proizvodni sustavi i slično. Do danas su razvijene razne algebarske metode za statičku i dinamičku analizu Petrijevih mreža. Tako je iz modela PM izveden matrični model [BOG06] koji primjenom 6 osnovnih matrica sustava može jednoznačno opisati promatrani sustav. Takav model je primijenjen i u ovome radu.

PM su našle primjenu i u automatskom ili polu-automatskom okruženju proizvodnih sustava. Pokazale su se jako učinkovitim alatom u modeliranju pojedinih dijelova složenih automatskih i transportnih sustava [CAM93, MOO96, ZUR94].

Također, mnogi istraživači su svoj rad usmjerili na razvijanje metoda za brzo rješavanje stvarnih problema rasporeda, posebno problema rasporeda poslova u fleksibilnom proizvodnom sustavu (pokrata: FMS) [BOG06, LEE07, TAC97]. FMS su poznati kao "živi" sustavi [BOG06] sastavljeni od skupa resursa i poslova koji trebaju te resurse pri svom izvođenju. Ovi sustavi mogu biti različitih struktura i zahtijevaju različite strategije upravljanja ovisno o zadanim nalozima (npr. robot se uvodi u sustav) ili ovisno o stanju u kojem se dio FMS može naći (npr. robot je izvan funkcije). Pod takvim uvjetima, slijed izvođenja poslova, paralelizam naloga, propust planiranih poslova i sraz istodobnih naloga samo su neki od problema s kojima se suočavaju FMS menadžeri.

Jedan od najvažnijih problema koji se javlja u sustavima s diskretnim događajima jest problem potpunog zastoja sustava (engl. *deadlock*). Potpuni zastoj sustava je pojava kod koje daljnji tijek operacija u sustavu nije moguć i koji u sustavima automatizacije može izazvati velike materijalne štete, a mogu ga uzrokovati resursi koji se dijele između više procesnih operacija (Abdallah, ElMaraghy, 1998, navedeno u [KEZ04]). Nenadani otkaz pojedinog dijeljenog resursa može izazvati potpuni zastoj cijelog sustava, nepotrebne troškove, nedovoljnu uporabu skupih i važnih resursa ili dulje vrijeme izvođenja procesa. Jedan od glavnih zadataka projektanata sustava automatskog upravljanja je predviđanje takvih situacija i projektiranje nadzornika kojemu će biti glavni zadatak promatranje događaja u sustavu i poduzimanje odgovarajućih akcija samo u onom slučaju kada prijeti neposredna mogućnost pojave stanja potpunog zastoja. Mnogi autori su pokušali riješiti problem zastoja pomoću Petrijevih mreža. Barkaoui (1995) je razvio metodu izbjegavanja zastoja pomoću kontrolnih mjesta (citirano iz [KEZ09]). Također, Ezpelta je razvio algoritam za rješavanje problema zastoja u S^3PR klasi PM-a. Ovaj rad se smatra prvim koji, za FMS, primjenjuje strukturnu analizu za kreiranje PM nadzornika koji nadzire 'živost' mreže (citirano iz [KEZ09]). Lautenbach (1996) je istraživao algoritam za pronalaženje minimalnog sifona unutar mreže kao i algoritam za sprječavanje zastoja uvođenjem kontrolnih mjesta u standardnu PM koja ne sadrži ulazna mjesta. Nadalje, Lewis je razvio učinkovit algoritam za sprječavanje zastoja u posebnoj klasi PM koje opisuju FMS [TAC97].

Određivanje rasporeda u DES pomoću PM je predmet istraživanja brojnih znanstvenika. Kako u sustavima s diskretnim događajima, distribucija resursa može dovesti do konfliktnih situacija i zastoja, mnogi istraživači su razvili rasporede bez zastoja.

Dobro je poznata činjenica da problem određivanja rasporeda, matematički, spada u skupinu NP-teških problema, što je razlog da ne postoji učinkovit algoritam koji bi taj problem optimalno riješio. Stoga se na njegovo rješenje primjenjuje heuristika, osobito genetski algoritmi.

Genetski algoritam (GA) je računalni model koji primjenjuje Darwin-ove principe biološke evolucije, kao što su preživljavanje najboljih, mutacije jedinki i dr., a služi za rješavanje optimalizacijskih problema. Preciznije rečeno, genetski algoritam je recept koji kazuje što treba raditi s genetskim materijalom kako bi se s određenom vjerojatnošću nakon određenog vremena postiglo zadovoljavajuće rješenje zadanog optimalizacijskog problema. Genetski materijal je skup svojstava koji opisuju neku jedinku. Genetski algoritam ne određuje na koji način je genetski materijal pohranjen u radni spremnik, niti kako treba manipulirati genetskim materijalom, već samo kaže da se genetski materijal treba razmjenjivati, slučajno mijenjati, a bolje jedinke trebaju s većom vjerojatnošću preživljavati selekciju. Kako će se bolje jedinke selektirati za reprodukciju, te kako će se i s kolikim vjerojatnostima obaviti križanje i mutacija, određuje se na temelju iskustva i eksperimentalno,

tj. heuristički. Stoga postoji velika sloboda u izradi genetskih algoritama. Cijena te slobode su loši ili nikakvi rezultati koji se najčešće dobivaju s genetskim algoritmom koji nema podešene parametre ili nema genetske operatore prilagođene problemu.

Rješavanjem problema rasporeda, najčešće za industrijsko okruženje, uporabom GA bavili su se brojni istraživači [DAV85, BIE90, BIE99, CRO95,GON05]. Temeljem tih istraživanja i dobrih rezultata koje su postigli razvijeni algoritmi, u ovoj disertaciji će se rješavati složeni problemi rasporeda poslova za DES pomoću GA.

DES su najčešće složeni sustavi za koje nije jednostavno razviti algoritam koji generira optimalan raspored poslova. Kako modelirati složeni DES predstavlja veoma izazovan zadatak. Dobar model bi trebao ne samo pomoći korisniku da na jednostavan način koristi algoritam generiranja rasporeda, nego mu pomoći da učinkovito prati stanje resursa i da u pravom trenutku promjeni strategiju upravljanja i raspoređivanja.

U području modeliranja, kako je već navedeno, PM imaju značajnu ulogu. To je jedan od razloga što su sadašnja istraživanja usmjerena na kombiniranje heurističkih metoda kao što su genetski algoritmi koji s PM rješavaju problema optimalnog rasporeda primjerice proizvodnih sustava [GAN04, CHE01.

Temeljem dobrih rezultata prethodnih istraživanja, u ovom radu se navedena integracija pokušala primijeniti u području transportnih sustava. Pomoću PM će se modelirati primjer pomorskog transportnog sustava, koji se po svojim značajkama može opisati kao sustav s diskretnim događajima.

Kako bi se u model ugradile sve željene značajke promatranog sustava, u ovom radu je predstavljen nov pristup integracije PM i GA (pokrata: PMGA) za optimalnog nadzornika koji određuje optimalni raspored poslova i pri tome sprječava stanje potpunog zastoja u sustavu. Predlaže se primjena *P-vremenske* MRF_1 Petrijeve mreže (vrijeme se pridružuje mjestima) koja odgovara rasporedu poslova u sustavu. U prethodnim pristupima [CHE01, LLO95, ONA91] modeliranju sustava integracijom PM i GA, mjesta su predstavljala resurse i uvjete, dok su prijelazi predstavljali poslove ili operacije. Za razliku od takvog pristupa, u ovom radu će se i poslovi prikazati mjestima. Mjestima će se pridružiti vrijeme koje određuje trajanje izvođenja pojedinog posla. Također, na sličan način će se modelirati i 'set-up' vrijeme – vrijeme pripreme resursa za izvođenje posla.

Kako bi se postigli što bolji rezultati, predlaže se integracija GA s matričnim opisom MRF sustava tako da se generira parametrizirani aktivni raspored (detaljno objašnjenje je u poglavlju 6). Predloženi PMGA model pogodan je za pronalaženje optimalnog rješenja više-kriterijskog i više-ciljnog problema rasporeda poslova u sustavu s više različitih resursa, od kojih su neki djeljivi i ograničenog kapaciteta. Postupak počinje na način da se prvo načini model procesa pomoću Petrijeve mreže, a nakon toga se primjenom matričnog modela odredi mogućnost nastanka nedozvoljenih stanja, primjerice potpunog zastoja. Ukoliko u sustavu postoje stanja potpunog zastoja, postojećoj mreži se dodaje nadzornik koji se sastoji od skupa kontrolnih mjesta. Zadatak nadzornika je detekcija događaja koji nastaju u sustavu i djelovanje na sustav na način da se spriječe potencijalna stanja potpunog zastoja. U proračun nadzornika je uključen GA koji traži optimalan raspored poslova u sustavu sa stajališta iskoristivosti resursa i ukupnog izvođenja svih poslova u sustavu. Predloženi algoritam je primjenjiv za svaki sustav automatskog upravljanja koji se može opisati kao sustav s diskretnim događajima.

Također, pokušat će se razviti takav genetski algoritam, koji će pronaći optimalno rješenje, će skratiti vrijeme optimalizacije. Kraće vrijeme optimalizacije povećava uporabljivost programskog rješenja i dodatno pomaže da se eksperimentiranjem pronađe najbolji raspored. U tom kontekstu eksperimentirat će se s GA operatorom selekcije i sa statičkom, odnosno dinamičkom veličinom populacije.

1.2. Ciljevi istraživanja i hipoteze

Temeljni cilj ovog rada je dati doprinos metodologiji modeliranja svih sustava s diskretnim događajima na način pronalaženja optimalnog načina upravljanja ovih sustava, s posebnim naglaskom na proizvodne i prometne sustave.

U ovom radu predlaže se modeliranje sustava s diskretnim događajima pomoću PM simulatora koji, integriran s genetskim algoritmom, određuje optimalan rasporeda poslova u sustavu. Pri upravljanju proizvodnim ili transportnim sustavom i rasporedom poslova, PM simulator bi trebao omogućiti modeliranje sustava i njegovog okruženja. PM se može koristiti za modeliranje složenih procesnih tokova i detaljnih značajki sustava, kao što su postavljanje strojeva, otkazi strojeva, uvođenje novih poslova, paralelni procesi, itd. Tako bi se pojednostavnilo spoznavanje i razumijevanje proučavanog sustava.

Genetski algoritam će se koristiti za rješavanje višekriterijskog problema dinamičkog rasporeda poslova u PM modelu. Kromosomi u GA će se oblikovati izravno iz PM modela. Prilikom razmatranja problema dinamičkog rasporeda, minimalizacija vremena koje je potrebno za izvršavanje svih poslova (*total production time, makespan*) nije i najbolji kriterij optimalizacije, jer je nužno povremeno prerasporediti poslove te *makespan* ostavlja velike praznine u rasporedu. Zato će se u ovom radu pokušati riješiti kombinirano više ciljeva optimalizacije. Uzet će se u obzir m*akespan* (mjeri vrijeme potrebno da se izvrše svi poslovi), *tardiness* (odnosi se na negativne troškove tj. troškove koji nastaju samo u slučaju prekoračenja roka) i *idle time* (zbroj svih perioda kada je izvor raspoloživ, a nezaposlen).

Potrebno je razviti genetski algoritam, koji će primjenom standardnih operatora mutacije i križanja stvarati nove jedinke unutar generacije, ali će i omogućiti takav način selekcije koji će skratiti vrijeme optimalizacije. Kraće vrijeme optimalizacije povećava uporabljivost programskog rješenja i dodatno pomaže da se eksperimentiranjem pronađe najbolji raspored.

U radu razvijeni algoritmi biti će testirani na modelu sustava AGV vozila u kontejnerskim terminalima, kao sustav podrške odlučivanju pri određivanju rasporeda poslova. Testiranjem se želi pokazati da su Petrijeve mreže primjerene za proučavanu problematiku i da su metode evolucijskog računanja, posebice genetski algoritmi, primjereni za rješavanje problema rasporeda.

Realizacija navedenog cilja temeljena je na sljedećim hipotezama:

H1. *Sustav s diskretnim događajima moguće je optimalizirati u dinamičkom okruženju primjenom P-vremenskih Petrijevih mreža i višekriterijskog genetskog algoritma, s*

ciljem poboljšanja značajki sustava s ograničenim resursima, te predviđanja i sprječavanja pojava zastoja i konflikata.

H2. *Za potrebe genetskog algoritma kromosomi, koji predstavljaju niz okidanja, mogu se generirati iz vremenske Petrijeve mreže promatranog sustava.*

H3. *Sustav AGV vozila moguće je modelirati vremenskim Petrijevim mrežama, te razviti genetski algoritam za rješavanje više-ciljnog problema optimalizacije rasporeda kojim će se pridružiti operacije AGV vozilima, odrediti slijed operacija za svako vozilo, minimalizirati broja AGV vozila, optimalizirati ukupno vrijeme izvođenja svih poslova i optimalizirati ukupni troškovi.*

1.3. Motivacija

Ova disertacija se fokusira na primjeni PMGA algoritma koji integrira Petrijevu mrežu i genetski algoritam u modeliranju i određivanju rasporeda poslova u više-projektnom sustavu koji uključuje skup poslova, različite tipove resursa, od kojih su neki višeradni i ograničenog kapaciteta. Pokušao se pronaći takav model DES koji bi, s jedne strane, osigurao stabilan, siguran i optimalan rad u izvođenju poslova, a s druge strane, koji bi bio dovoljno jednostavan, lagan za implementaciju i općenito učinkovit u praktičnoj primjeni.

Kako ne postoji standardni alat za kreiranje modela DES, odabrane su PM. Iako, postoje i drugi grafičke tehnike modeliranja, koje su i bolje od PM obzirom na čitljivost i prezentaciju strukture, odabrane su upravo Petrijeve mreže i za modeliranje i za analizu fizičkih sustava koji su objekt istraživanja u ovoj disertaciji. Postoje najmanje četiri dobra razloga za njihov odabir: (1) PM su grafički jezik i njihova semantika se može lako matematički formulirati, što je pogodno za analizu sustava; (2) Za analizu PM-e postoji više različitih tehnika; (3) PM su pogodne za modeliranje sustava gdje se mnogi poslovi izvršavaju asinkrono i istodobno. Pogodne su za modeliranje i ispitivanje postojanosti konfliktnih stanja, kao i stanja zastoja; (4) Pružaju dovoljno informacija menadžerima koje im pomažu da ispitaju kašnjenja u sustavu kao i o njihovim uzrocima.

Za razliku od drugih pokušaja integracije PM i GA, u ovom radu odabrana je P-vremenska MRF_1 klasa PM koja se integrira s GA. Razlog za odabir MRF_1 je taj što se za ovu PM može primijeniti matrična metoda za proračun nadzornika sustava koja se lako kombinira s GA. Nadzornik upravlja dodjelom resursa i određuje pravila kojima se izbjegavaju konflikti i zastoji u sustavu. Isto tako, omogućava i praćenje stanja resursa u svakom trenutku te upravljanje vremenima. Drugi razlog je taj što, ne postoje istraživanja koja su se bavila integracijom takvih vrsta PM i GA za DES sustave.

1.4. Struktura rada

Rad je podijeljena u osam poglavlja, uključujući uvod i zaključak. U nastavku se ukratko opisuje sadržaj rada po poglavljima.

***Drugo poglavlje**: Sustavi s diskretnim događajima.* U ovom se poglavlju razmatra jedna od mogućih podjela sustava. Definiran je sustav s diskretnim događajima, te njegove značajke koje ga razlikuju od drugih sustava. Navedeni su pojednostavljeni primjeri realnih sustava (računalni sustavi, proizvodni sustav). Pokazano je kako se mogu definirati skupovi događaja i pripadni vektori stanja za prikazane primjere, što predstavlja osnovu za analizu sustava s diskretnim događajima. Na kraju su prikazani problemi i metode optimalizacije sustava s diskretnim događajima.

Treće poglavlje: *Petrijeve mreže.* U ovom poglavlju dan je opis Petrijevih mreža. Detaljno su opisana osnovna svojstva opće Petrijeve mreže. Prikazane su tehnike analize pomoću grafa dostupnih stanja, linearne algebre, redukcije i dekompozicije. Pored općih Petrijevih mreža, definirane su i proširene Petrijeve mreže kao što su obojene Petrijeve mreže CPM i vremenske Petrijeve mreže TPN.

Četvrto poglavlje: *Genetski algoritmi.* U ovom poglavlju detaljno su opisani genetski algoritmi, heuristička metoda optimaliziranja koja je zamišljena kao imitacija prirodne evolucije. Prikazan je mehanizam jednostavnog genetskog algoritma, što uključuje načine prikazivanja kromosoma, definiranje funkcije cilja ili funkcije dobrote i genetske operatore. Također, opisana je i izvedba višekriterijskog genetskog algoritma (MCGA).

Peto poglavlje: *Sustav za određivanje i optimalizaciju rasporeda poslova.* U ovom poglavlju opisan je opći problem rasporeda. Također, bit će opisan generator rasporeda, koji će se koristiti za pretvorbu kromosoma u fleksibilan raspored. Opisat će se višekriterijski genetski algoritam MCGA za rješavanje složenih problema raspoređivanja, uzimajući u obzir, istodobno, više kriterija raspoređivanja.

Šesto poglavlje: *Formalizacija integracije Petrijeve mreže i genetskog algoritma.* Ovo poglavlje, zajedno s poglavljem 7, je temeljni dio cijelog rada. U ovom poglavlju dana je definicija MRF klasa sustava i opis njegovog matričnog modela. Poseban naglasak je na MRF_1 klasi Petrijevih mreža koje su integrirane s parametriziranim genetskim algoritmom. Predstavljen je novi hibridni model za generiranje rasporeda poslova: GA, P-vremenska MRF_1 i procedura generiranja rasporeda poslova. Algoritam i adekvatna strategija upravljanja vrednovani su na sustavu prometa brodova morskim kanalima.

Sedmo poglavlje: M*odel rasporeda automatski upravljanih vozila u kontejnerskim lukama.* U ovom poglavlju detaljno je opisan problem rasporeda automatski upravljanih vozila (AGV-a) u automatiziranim kontejnerskim terminalima. Opisane su značajke resursa koji se rabe na terminalu. Kao studij slučaja uzet je kontejnerski terminal luke Koper, a kao sustav s diskretnim događajima promatra se i analizira proces ukrcaja/istovara kontejnera na/s broda i transport do skladišta ili do željeznice. Razvijen je i predložen PMGA model koji treba pronaći optimalan raspored poslova bez konflikata i zastoja. U ovom poglavlju izneseni su i analizirani rezultati testiranjem razvijenih algoritama.

Osmo poglavlje: *Zaključak.* U zaključku se kratko rezimiraju postignuti rezultati provedenih istraživanja u odnosu na postavljene hipoteze.

2. SUSTAVI S DISKRETNIM DOGAĐAJIMA

Znanstvenici i inženjeri koji se bave proučavanjem prirodnih, društvenih i tehničkih procesa koriste teoriju sustava kao metodu koja omogućava rješavanje cijelog niza različitih problema. Ako se promatra prirodni ili tehnički sustav, nužno je potrebno poznavati fizikalne i druge adekvatne zakonitosti sustava, te doći do tzv. matematičkog modela sustava. Matematički model sustava moguće je dobiti teoretskom analizom (modeliranje sustava) ili eksperimentalnom analizom (identifikacija sustava). Danas su razvijene razne tehnike koje omogućavaju da se tako modelirani sustav analizira, te da se u potpunosti dobije uvid u njegovo ponašanje. Na osnovi toga moguće je projektirati upravljanje sustavom.

Sustavi s diskretnim događajima su dinamički sustavi u kojima se promjene stanja sustava događaju kada se pojavi neki događaj. Neki od primjera sustava s diskretnim događajima su proizvodni sustavi, komunikacijske i računalne mreže, zračni, cestovni, pomorski prometni sustavi i dr. U ovim sustavima, glavni mehanizam dinamičnosti u uspješnom izvođenju zadataka proizlazi iz sinkronizacije, međusobnog isključivanja ili nadmetanja u uporabi zajedničkih resursa, koji se zahtijevaju pri rješavanju konfliktnih situacija i definiranju prioriteta, te svih vrsta problema poznatih pod zajedničkim nazivom problem rasporeda.

Cilj ovog poglavlja je definirati sustav s diskretnim događajima, identificirati njegove značajke koje ga razlikuju od drugih sustava, te istaknuti probleme i neke od metoda optimalizacije takvih sustava.

2.1. Definicije sustava

Definicija 2.1 Sustav (IEEE Standard Dictionary of Electrical and Electronic Terms)

Sustav (engl. *System*) se može definirati kao "kombinacija komponenata koje zajednički djeluju u svrhu obavljanja neke određene funkcije, a koja se ne može izvršiti niti jednom pojedinačnom komponentom".

Pri tome se podrazumijeva da postoji međusobna interakcija među komponentama sustava.

Definicija 2.2 Sustav (Bertalanffy, 1968[1])

"A system is an entity which maintains its existence through the mutual interaction of its parts..."

Sustav je "jedinka koja upravlja svojim postojanjem/opstankom kroz međusobno i uzajamno djelovanje svojih dijelova".

Sustav predstavlja "nešto realno" (automobil, pojačalo, ljudsko tijelo), a *model* predstavlja "apstrakciju realiteta" (skup matematičkih jednadžbi). Na temelju iskustva koje je

[1] Dostupno na http://en.wikipedia.org/wiki/Systems_theory, citirano iz: Ludwig von Bertalanffy: "*General System Theory Foundations, Development, Applications*", New York, George Braziller, 1968.

proizašlo iz opažanja, ljudi grade modele i stavljajući ih u različite uvjete, mogu predvidjeti buduće događaje. Kada se ovi uvjeti, koji su već testirani na modelu, pojave u stvarnom životu, moguće je znati što će biti ishod.

Sustavu je moguće pridružiti ulazne i izlazne varijable. Varijable sustava mogu biti kontinuirane i/ili diskretne. Ulazne varijable sustava se mogu prikazati kao stupčasti vektor $u(t)$, a izlazne varijable sustava kao stupčasti vektor $y(t)$.

$$u(t) = \left[u_1(t), \ldots, u_p(t)\right]^{\tau}, \tag{2-1}$$

$$y(t) = \left[y_1(t), \ldots, y_m(t)\right]^{\tau}, \tag{2-2}$$

gdje je:

$\left\{u_1(t), \ldots, u_p(t)\right\}$ - skup ulaznih varijabli,

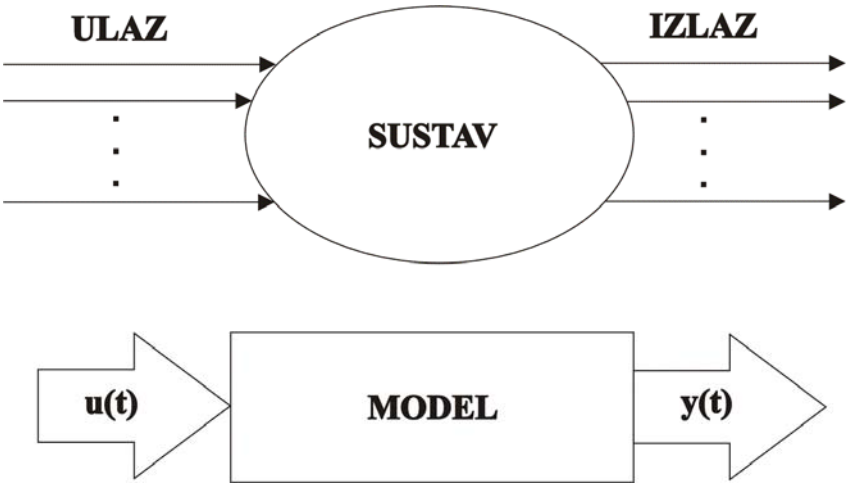
$\left\{y_1(t), \ldots, y_m(t)\right\}$ - skup izlaznih varijabli.

Na slici 2.1 prikazan je sustav sa svojim ulaznim i izlaznim varijablama te njegov pripadajući model.

Definicija 2.3 Stanje sustava

Stanje sustava u trenutku t_0 je informacija koja se zahtijeva u tom trenutku tako da je izlazna varijabla $\{y(t)\}$, za svako $t \geq t_0$, jednoznačno određena tom informacijom i ulaznom varijablom $\{u(t)\}$, za svako $t \geq t_0$.

Prostor stanja sustava se definira kao skup svih mogućih vrijednosti stanja koje sustav može poprimiti.



Slika 2.1. Sustav i njegov model

2.1.1. Podjela sustava

S obzirom da je u ovom radu naglasak na razmatranju sustava s diskretnim događajima, u ovom podpoglavlju je prikazan jedan od mogućih načina na koji se svi sustavi mogu podijeliti. Cilj ovakve podjele je definiranje osnovnih sličnosti i razlika sustava s diskretnim događajima i ostalih sustava. Ova podjela nije i jedino moguća, no ona dobro opisuje različite značajke pojedinih sustava i dobar je uvod u analizu sustava s diskretnim događajima.

Prema Cassandras i Lafortune [CAS99], sustavi se mogu podijeliti na (slika 2.2):

- statičke i dinamičke,
- vremenski promjenjive i vremenski nepromjenjive,
- linearne i nelinearne,
- kontinuirane i diskretne,
- diskretne sustave vođene vremenom i sustave s diskretnim događajima,
- stohastičke i determinističke.

U *statičkim* sustavima vrijednosti izlaznih varijabli ovise isključivo o trenutačnim vrijednostima ulaznih varijabli, dok za *dinamičke* sustave vrijedi da vrijednosti izlaznih varijabli ovise o trenutačnim i prošlim vrijednostima ulaznih varijabli. Za opisivanje ponašanja tih sustava najčešće se koriste diferencijalne jednadžbe.

Dinamički sustavi mogu biti vremenski promjenjivi ili vremenski nepromjenjivi.

Parametri *vremenski promjenjivih* sustava se vremenom mijenjaju, pa se i odziv takvih sustava na istu pobudu vremenom mijenja. Parametri *vremenski nepromjenjivih* sustava, za razliku od vremenski promjenjivih sustava, se ne mijenjaju vremenom, što znači da je odziv sustava uvijek isti za određenu pobudu, neovisno o starosti sustava. Vremenski nepromjenjivi sustavi se mogu podijeliti na linearne i nelinearne.

Za razliku od *nelinearnih* sustava, *linearni* sustavi zadovoljavaju uvjet superpozicije [KEZ04].

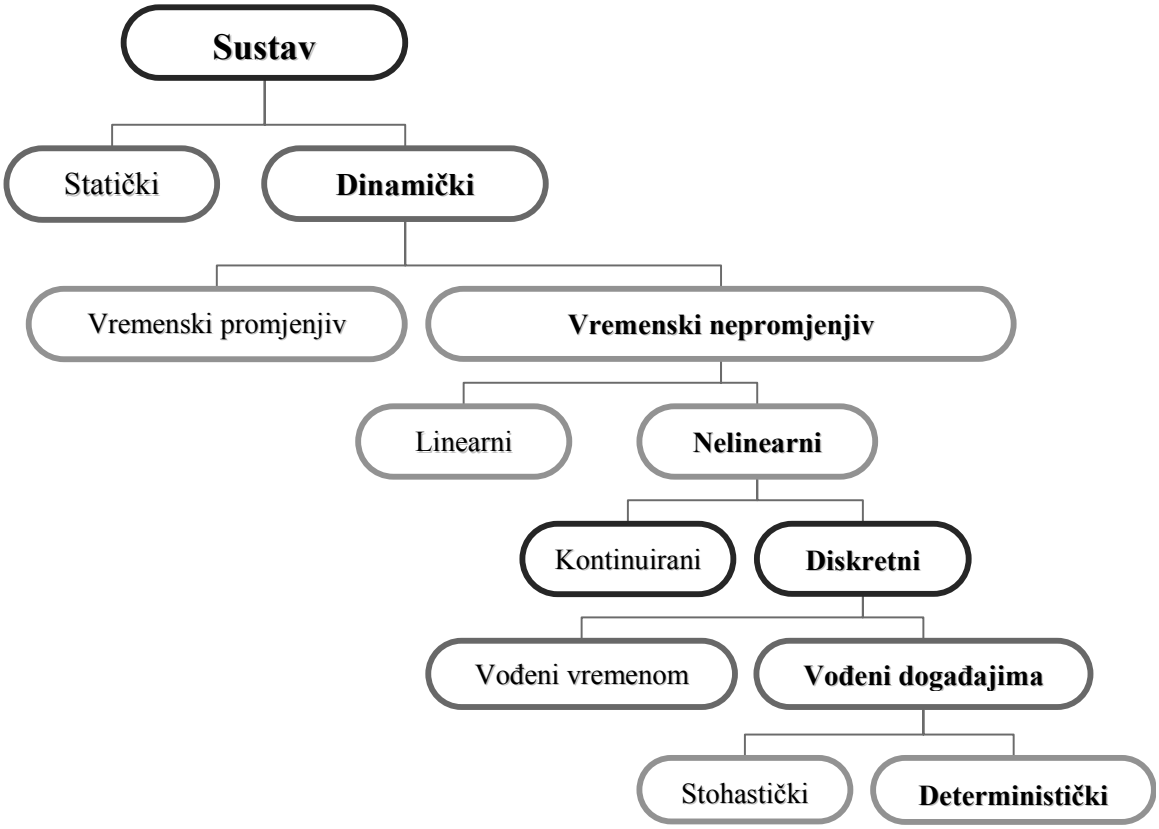
U *kontinuiranim* sustavima varijable stanja se mijenjaju kontinuirano u vremenu i mogu općenito imati bilo kakvu realnu vrijednost. Prostor stanja je kontinuiran i neprebrojiv, a alat za rješavanje ovih sustava su diferencijalne jednadžbe.

U *diskretnim* sustavima se stanje sustava mijenja samo u nekim vremenskim točkama. Te se promjene nazivaju *događaji*. Varijable stanja diskretnih sustava su elementi diskretnog skupa vrijednosti, primjerice skupa pozitivnih cijelih brojeva.

Diskretni sustavi se mogu podijeliti na sustave vođene vremenom i sustave s diskretnim događajima. Promjena stanja *diskretnih sustava vođenih vremenom* je sinkronizirana s unaprijed određenim vremenskim intervalom uzorkovanja, koji nastaje u postupku diskretizacije sustava. Promjena stanja između tih intervala nije moguća. Za *sustave s diskretnim događajima* je karakteristično da se promjene stanja u sustavu pojavljuju asinkrono, te da su uzrokovane jednim ili više međusobno neovisnih diskretnih događaja. Između dvaju diskretnih događaja nema *promjena* stanja u sustavu.

Deterministički sustavi su sustavi čije je ponašanje unaprijed predvidivo. *Stohastički* sustavi imaju barem jednu slučajnu izlaznu varijablu. Za te sustave je značajno njihovo

slučajno ponašanje, koje se ne može unaprijed predvidjeti. Promjene stanja takvih sustava opisuju se stohastičkim procesima i za analizu ovih sustava koristi se teorija vjerojatnosti.



Slika 2.2. Podjela sustava

2.2. Značajke sustava s diskretnim događajima

Većina sustava se može sagledati kao sustavi s diskretnim događajima. U ovu skupinu se svrstavaju sustavi koji su sastavljeni od konačnog broja resursa koje dijele više korisnika i svi doprinose postizanju nekog zajedničkog cilja (paralelno izvođenje, proizvodnja na automatskoj pokretnoj traci, prijenos komunikacijskih paketa i sl.). Primjer takvih sustava su računalni sustavi s poslovima/zadacima koji se natječu za resursima (CPU, memorijom,…) ili proizvodni sustavi s proizvodnim dijelovima i elementima koji se natječu za strojevima i robotima. Dinamičko ponašanje takvih sustava se ne mijenja s vremenom i najčešće ga je nemoguće opisati diferencijalnim jednadžbama. Takvi sustavi se nazivaju sustavi s diskretnim događajima.

Definicija 2.4 Sustav s diskretnim događajima

Sustav s diskretnim događajem (engl. *Discrete Event System*, pokrata: DES) se definira kao sustav u kojemu samo pojava asinkrono generiranih diskretnih događaja može uzrokovati promjene stanja sustava. Ponašanje takvih sustava je određeno diskretnim događajima iz okoline sustava.

Definicija 2.5 Događaj

Događaj (engl. *event*) se može definirati kao pojava koja izaziva prijelaz iz jednog stanja u neko drugo stanje sustava.

Događaj može virtualno predstavljati bilo što, a da se pojavljuje iznenada. Primjerice, događaj može odgovarati dolasku paketa u čvor komunikacijske mreže, dolazak korisnika, završetak zadatka ili kvar stroja u proizvodnom sustavu.

Važno je razlikovati modele sustave s diskretnim događajima od modela koji se temelje na diferencijalima i diferencijalnim jednadžbama. Kod modela temeljenih na diferencijalima i diferencijalnim jednadžbama važno je uočiti da se stanja mijenjaju s vremenom, a kod sustava s diskretnim događajima stanje sustava se mijenjaju pojavom događaja. Pridjev *"diskretan"* u izrazu "sustav s diskretnim događajima" ne znači da je "vrijeme diskretno" niti to nužno povlači da je "stanje diskretno" (varijable stanja mogu poprimiti kontinuirane vrijednosti). Zapravo, riječ *"diskretan"* označava činjenicu da je dinamičnost posljedica događaja koji se pojavljuje iznenada, a ne kontinuirano.

Ponekad se koristi naziv *Dinamički sustav s diskretnim događajima* (engl. *Discrete Event Dynamic System*, pokrata: DEDS) kako bi se istakla dinamička priroda sustava s diskretnim događajima.

Osnovne značajke sustava s diskretnim događajima su (MengChu, DiCesare, 1993 [KEZ04]):

- njihov prostor stanja je diskretan skup,
- pokretani su događajima (engl. *event-driven*),
- asinkronost (engl. *asynchronous*),
- slijedni odnos događaja (engl. s*equential relation*),
- istovremenost događaja (engl. c*oncurrency*),
- međusobno isključivanje događaja u sustavu (engl. *mutual exclusion*),
- nedeterminističnost događaja u sustavu (engl. n*on-determinism*),
- sukob između pojedinih događaja (engl. *conflict*),
- potpuni zastoji u sustavu (engl. *deadlock*).

Asinkronost je jedna od najvažnijih značajki sustava s diskretnim događajima i znači da događaji koji uzrokuju prijelaz iz jednog u drugo stanje nisu sinkronizirani s unaprijed određenim vremenskim intervalom uzorkovanja. *Slijedni odnos* događaja podrazumijeva da se događaji odvijaju jedan za drugim. *Istovremenost* pak podrazumijeva da se događaji mogu pojaviti u istom trenutku. Istovremenost isključuje slijedni odnos i obratno. *Međusobno isključivanje* pojedinih događaja u sustavu znači da pojava jednog događaja u potpunosti onemogućava pojavu nekog drugog događaja. *Nedeterminističnost* znači da se ne mogu točno predvidjeti trenuci pojave određenih događaja u sustavu, te da li će se uopće određeni događaji i dogoditi. *Sukob* u sustavu nastaje kada dva događaja zahtijevaju isti resurs sustava u istom trenutku. *Potpuni zastoj* je stanje sustava s diskretnim događajima u kojem više nije moguće pokrenuti niti jedan događaj. Postoji više razloga koji uzrokuju potpuni zastoj i osnovni zadatak projektanta sustava je projektirati nadzorni sustav koji će spriječiti tu pojavu.

Osnovni elementi DESs su:

- Diskretni prostor stanja, oznaka **X**,
- Diskretni skup događaja, oznaka *E*.

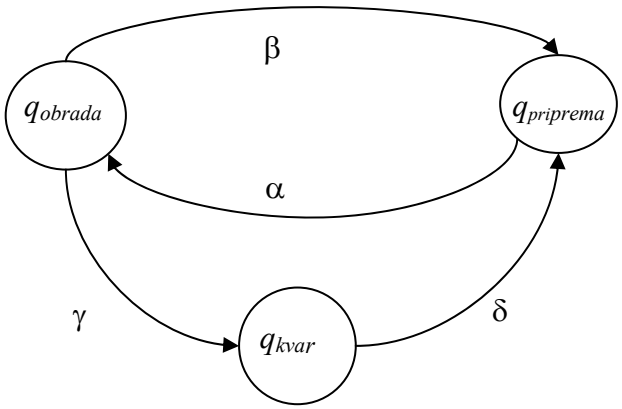
Primjer 2.1 (Sustav s diskretnim događajima)

Kao primjer sustava s diskretnim događajima može se razmatrati stroj (procesna jedinica) u proizvodnom sustavu. U određenom trenutku stroj može biti u samo jednom od tri stanja: priprema ($q_{priprema}$), zaposlen (q_{obrada}) i kvar stroja (q_{kvar}) i vrijedi da se prostor stanja ovog sustava može prikazati kao $\mathbf{X} = \{q_{priprema}; q_{obrada}; q_{kvar}\}$.

Skup od četiri diskretna događaja $E = \{\alpha,\beta,\gamma,\delta\}$, koji uzrokuju promjene u prostoru stanja, može se definirani kako slijedi.

Pretpostavimo da se na početku procesa obrade stroj priprema za rad i nalazi se stanju $q_{priprema}$. Element dolazi iz vanjskog svijeta (u grafu ulazni događaj je označen s α), sustav započinje s obradom i nalazi se u stanju zaposlenosti q_{obrada}. Kada je element obrađen (događaj β), stroj se ponovo vraća u stanje pripreme za rad $q_{priprema}$. U slučaju da se dogodi kvar na stroju za vrijeme obrade (događaj γ), stroj ide u stanje kvara q_{kvar}. Nakon što je stroj osposobljen za rad (događaj δ), ponovo se vraća u stanje pripreme $q_{priprema}$.

Na slici 2.3 prikazana je trajektorija u prostoru stanja. Vidljivo je da je prostor stanja diskretan i prebrojiv.



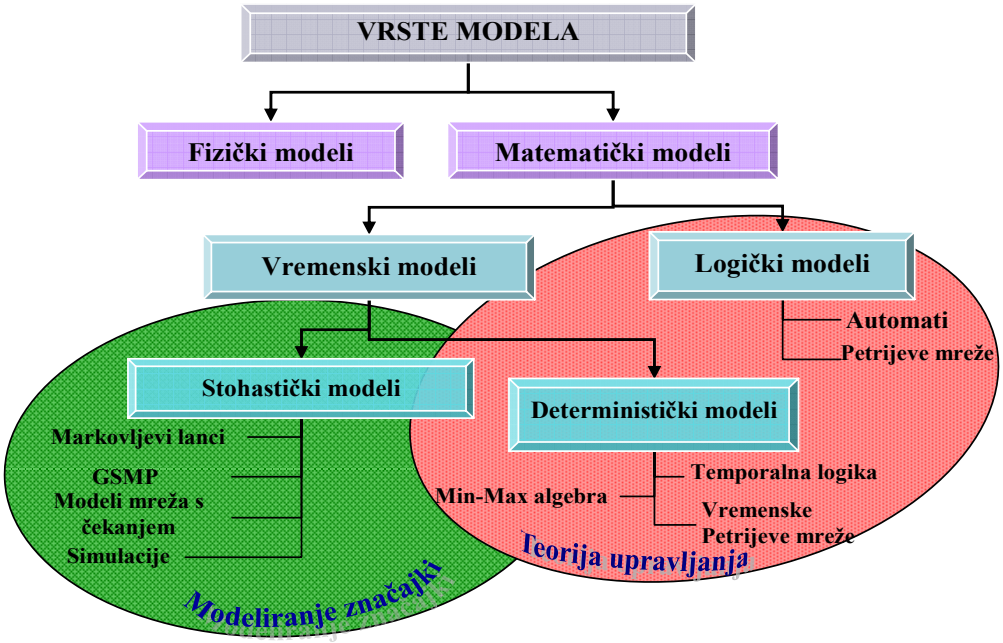
Slika 2.3. Trajektorija prostora stanja

■

Pristup teoriji sustava s diskretnim događajima može se podijeliti u dva ili tri smjera (slika 2.4):

- *Logički* pristup koji razmatra pojavljivanje događaja ili nemogućnost pojavljivanja događaja ("*zastoj*"); pristup koji razmatra pojavljivanje niza ovakvih događaja, a pri tome ne uzima u obzir precizno vrijeme pojavljivanja tih događaja.
- *Kvantitativni* pristup koji ističe *ocjenu značajki* (engl. *performance evaluation*) i *optimalizaciju značajki* (engl. *performance optimization*).

Razlika između bezvremenskih i vremenskih DES modela je jasna. Prva skupina modela naglašava niz stanja sustava s diskretnim događajima i pri tome se ne ističu ili u potpunosti zanemaruju značajke vremena za svako pojedino stanje. Ovakvi modeli su pogodni u teoriji upravljanja sustavom gdje je potrebno dobiti odgovore na logička pitanja, odnosno pitanja na koje se očekuje odgovor tipa "*da*" ili "*ne*", "*istina*" ili "*neistina*". Primjeri bezvremenskih modela su automati i Petrijeve mreže.

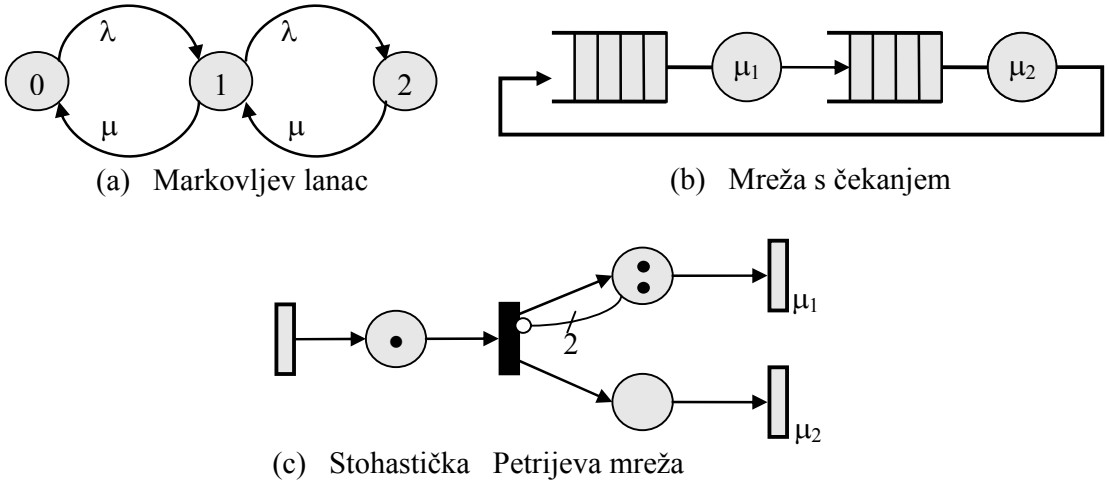


Slika 2.4. Modeli sustava s diskretnim događajima

Vremenski modeli ugrađuju vrijeme kao integralni dio modela i povoljni su za odgovaranje na pitanja kojima se modeliraju značajke sustava. Mogu biti deterministički i stohastički. Primjeri determinističkih vremenskih modela su vremenski automati, vremenske Petrijeve mreže, max-plus algebra (engl. *max-plus algebras*).

Stohastički modeli sustava s diskretnim događajima su korisni u različitim situacijama i sadrže informacije o događajima (pojavljivanje, redoslijed), točno vrijeme kada se događaj javlja, te statističke informacije (distribuciju vjerojatnosti nad skupom niza događaja). Među razvijenim stohastičkim modelima mogu se nabrojati sljedeći:

- Modeli Markovljevih lanaca (engl. *Markov Chains Models*) (slika 2.5 a); u ovu skupinu spadaju i stohastičke Petrijeve mreže (slika 2.5 c)
- Modeli mreža s čekanjem (engl. *Queuing Network Models*) (slika 2.5 b)
- Poopćeni semi-Markovljev proces (engl. *Generalized semi-Markov Process Models*); u ovu skupinu su uključeni i svi pokušaji koji se odnose na opće simulacijske jezike diskretnih stanja.



(a) Markovljev lanac (b) Mreža s čekanjem

(c) Stohastička Petrijeva mreža

Slika 2.5. Stohastički modeli sustava s diskretnim događajima

2.2.1. Zašto su potrebni sustavi s diskretnim događajima?

DES su korisni kada se promatrani sustav ne može potpuno opisati klasičnim modelima, primjerice diferencijalnim jednadžbama. Takvi sustavi sve više prevladavaju u području modeliranja zbog dva, niže opisana, razloga.

Kao prvo, složenost sustava, koje je čovjek izgradio (primjerice inženjerski sustavi), zahtijeva njihovo sagledavanje s različitih razina apstrakcije. Najčešće je nemoguće imati potpunu sliku cijelog sustava. Uobičajena praksa je definirati različite razine apstrakcije sustava. U takvom sagledavanju sustava s više razina apstrakcije, pogled s najniže razine dopušta detaljan opis dinamičnosti različitih komponenti sustava, dok se viša razina koristi za iskazivanje veza među komponentama. Dok se najniža razina najčešće opisuje diferencijalnim jednadžbama, viša razina uključuje dinamičnost različite prirode, primjerice donošenje logičkih odluka, diskretnu kontrolu operacija ili izmjenu načina rada. Zbog toga je prirodna, pa čak i neizbježna, uporaba događaja kako bi se istakla dinamičnost promatranog sustava.

Drugi razlog primjene DES-a je taj što su kod složenih sustava problemi nadzora i odlučivanja najčešće diskretni. Oni uključuju dodjeljivanje resursa, raspoređivanje, sinkronizaciju, preusmjeravanje i nadzor pristupa u komunikacijskim mrežama. Rješenju tih problema ne može se pristupiti uobičajenim modelima kao što su diferencijalne jednadžbe.

2.3. Primjeri sustava s diskretnim događajima

U ovom podpoglavlju su opisani pojednostavljeni primjeri realnih sustava. Prikazani primjeri su interpretirani na način koji omogućava njihovu analizu kao sustava s diskretnim događajima.

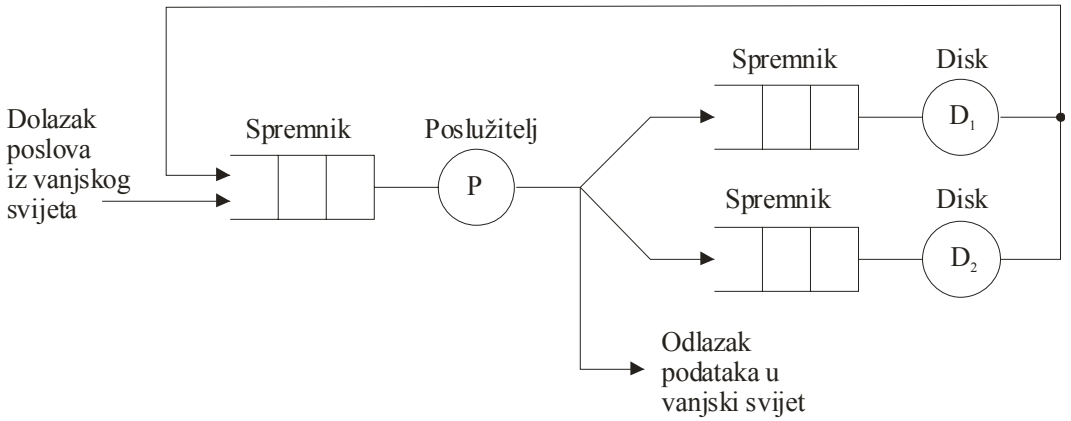
Primjer 2.2 (Računalni sustav [KEZ04]):

Pojednostavljena interpretacija računalnog sustava (slika 2.6), a sastoji se od sljedećih resursa:

poslužitelja P, diska D_1, diska D_2.

Svaki od resursa ima pridružen spremnik, jer resursi mogu obavljati samo jedan posao u jednom trenutku. Dolazak i odlazak poslova iz spremnika vrši se po FIFO (engl. *First-Come First Served*) načelu – opsluživanje po redoslijedu dolaska u sustav.

Poslovi koji se obrađuju u poslužitelju *P* čekaju u redu za obradu u spremniku i mogu doći iz vanjskog svijeta, diskova D_1 ili D_2. Nakon obrade u poslužitelju *P*, podaci se prosljeđuju u vanjski svijet ili ka jednom od diskova D_1 ili D_2, gdje također čekaju u redu čekanja za trajno pohranjivanje.



Slika 2.6. Pojednostavljena interpretacija računalnog sustava.

Skup događaja E sastoji se od dolazaka poslova ili podataka na obradu ili pohranjivanje u pojedini resurs te odlazaka poslova s pojedinog resursa:

$$E = \{dp, op, dd_1, od_1, dd_2, od_2\}, \tag{2-3}$$

gdje su događaji u skupu *E* definirani kao:

dp - dolazak posla na poslužitelj *P* ,
op - odlazak posla sa poslužitelja *P*,
dd_1 - dolazak podataka na D_1 iz poslužitelja *P*,
od_1 - odlazak podataka sa D_1 na poslužitelj *P*,
dd_2 - dolazak podataka na D_2 iz poslužitelja *P*,
od_2 - odlazak podataka sa D_2 na poslužitelj *P*.

Vektor stanja ovog sustava s diskretnim događajima može se definirati kao:

$$\mathbf{x} = \left[x_P, x_{D1}, x_{D2}\right]^{\mathrm{T}} \tag{2-4}$$

gdje su:

x_P, x_{D1}, x_{D2} - brojevi poslova ili blokova podataka koji čekaju u spremnicima,
T - oznaka za transponiranu matricu.

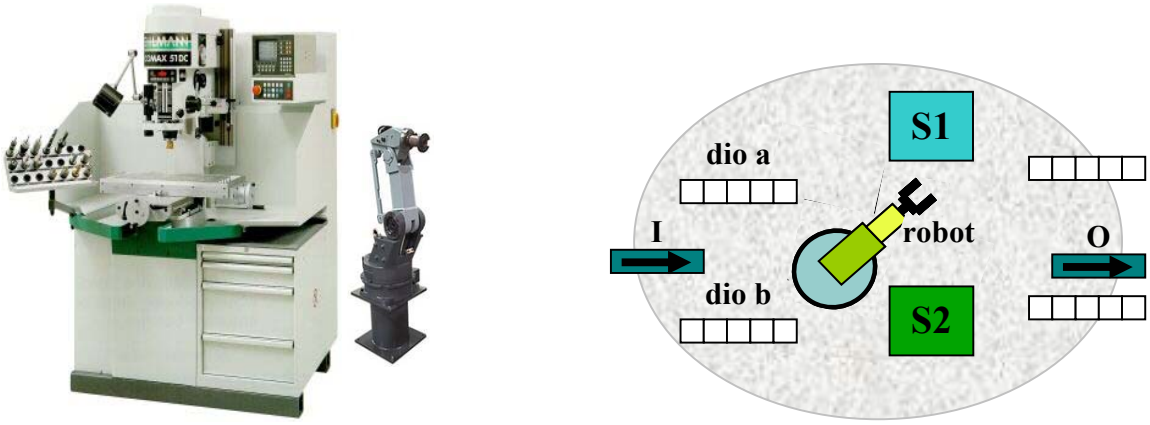
Prostor stanja predstavlja sva moguća stanja sustava i može se definirati kao:

$$\mathbf{X} = \left\{\left(x_P, x_{D1}, x_{D2}\right) : x_P, x_{D1}, x_{D2} \geq 0\right\} \tag{2-5}$$

■

Primjer 2.3 (Proizvodni sustav)

Dio robotiziranog proizvodnog sustava (slika 2.7) također se može prikazati kao sustav s diskretnim događajima. Sustav se sastoji od dva stroja koja obrađuju dijelove. Dijelovi se dobavljaju na obradu robotskom rukom. Dvije vrste dijelova, *a* i *b*, se obrađuju na sljedeći način. Oba dijela dovode se iz vanjskog svijeta pomoću ulazne pokretne trake. Ulaskom u sustav, dio *a* zahvaća robot i prebacuje ga do stroja S1. Kada je obrada završila, robot pomiče dio sa stroja na izlaznu pokretnu traku i isti se prosljeđuje u vanjski svijet. Pri dolasku, dio *b* se obrađuje na stroju S2. Nakon obrade se robotskom rukom prebacuje na izlaznu pokretnu traku.



Slika 2.7. Dio proizvodnog sustava iz primjera 2.3

Dok stanja stroja S1 (isto vrijedi i za stroj S2) mogu biti "*besposlen*" - *B* ili "u procesu"- *P*, situacija s robotom je drugačija jer on izvršava tri zadatka. Njegova stanja bi se mogla opisati kao "*raspoloživ*" - *R*, "micanje dijela *a* na stroj S1" - *M*, "pomicanje dijela *a* sa stroja S1" - 1, "pomicanje dijela *b* sa stroja S2" - 2.

Stanje sustava se može opisati s tri parametra, pri čemu se prvi odnosi na stanje robota, drugi na stanje stroja S1 i treći na stanje stroja S2. Prostor stanja **X** ovog sustava s diskretnim događajima može se definirati kao:

$$\mathbf{X} = \{(x_1, x_2, x_3) : x_1 \in \{R, M, 1, 2\}, x_2, x_3 \in \{B, P\}\}.$$

Skup događaja $E = \{\alpha, \beta, m, f, r, c\}$ se sastoji od onih događaja koji omogućuju prijelaz između stanja koji su gore navedeni. Opis tih događaja navedeni su u tablica 2.1.

Tablica 2.1. Događaji u proizvodnom sustavu iz primjera 2.3

Događaj	Opis
α	dolazak dijela *a* iz vanjskog svijeta
β	dolazak dijela *b* na stroj S2 (početak obrade)
m	početak obrade dijela *a* na stroju S1
f	početak zamjene dijela *b* sa stroja S2
r	završetak zamjene dijela *b* sa stroja S2
	završetak zamjene dijela *a* sa stroja S1
c	početak zamjene dijela *a* sa stroja S1

■

2.4. Problemi i metode optimalizacije sustava s diskretnim događajima

Sustavi s diskretnim događajima se proučavaju kako bi se shvatila izvedba i značajke sustava, te odredio najbolji način kako poboljšati njihovu izvedbu. Ovi sustavi su složeni i teški i za shvatiti ih i za uspješno djelovanje s njima. Zbog sve veće njihove općenitosti, fleksibilnosti i snage, simulacija sustava s diskretnim događajima je jedna od najčešće korištenih tehnika operacijskog istraživanja kojom se inženjeri i menadžeri služe pri analizi i dizajniranju složenih sustava kao što su telekomunikacijske mreže, zdravstveni sustavi, financijski sustavi ili fleksibilni proizvodni sustavi [GRO02]. *Simulacija* je proces izrade modela stvarnog sustava i provođenja eksperimenata s tim modelom. Simulacija se može koristiti kao:

- analitički alat za predviđanje ponašanja postojećeg sustava te radnog učinka (primjerice, određivanje broja proizvoda koji se po jedinici vremena izrade na nekom stroju);
- alat za testiranje i vrednovanje novog sustava u različitim okolnostima (primjerice, određivanje veličine spremnika koji se nalazi ispred svakog stroja kako bi se minimaliziralo vrijeme obrade).

Kako bi dizajniranje sustava bilo što učinkovitije, simulacija mora biti povezana s optimalizacijskim pristupom.

U tom slučaju simulacija je usmjerena na optimalizaciju određenih parametara simuliranog sustava s ciljem da se poboljšaju vrijednosti odabranih značajki funkcije cilja tj. optimalizacijskog kriterija. Takvi optimalizacijski problemi su uobičajeni kod simulacijskih modela proizvodnih sustava [BER01]. Primjerice, sljedeće aplikacije su već predstavljene:

- rješavanje problema alokacija spremnika, gdje je optimalizacija usmjerena na određivanju optimalnih veličina spremnika koji su postavljeni između radnih stanica [GRO02];
- dizajniranje sustava za rukovanje materijalom, koji se sastoji od velikog automatiziranog skladišta i vozila za dohvaćanje materijala, automatski upravljanih vozila, transportera i dizalica [GRO02];
- dizajniranje proizvodnih ćelija [HUA92].

Cilj povezivanja je izgraditi integrirani programski paket koji kombinira simulacijski program s optimalizacijskim algoritmima. Takav paket bi trebao skratiti vrijeme procesa, smanjiti cijenu i poboljšati kvalitetu određenog sustava. Naime, kod sustava s diskretnim događajima, postoji mnogo slučajeva gdje je moguće definirati i izmjeriti "cijenu" prijelaza iz jednog stanja u drugo. Pojam cijene se može opisati u smislu koliko je potrebno, primjerice vremena, snage ili sile, da se omogući prijelaz iz jednog stanja sustava u drugo, te koliki su troškovi pri tome. U tom slučaju problem optimalizacije sustava s diskretnim događajima se može formulirati: *pronaći optimalan niz prijelaza stanja sustava ili niz stanja koja poboljšavaju radni učinak sustava, uz minimalne troškove.*

Istraživanja i aplikacije u polju optimalizacija sustava s diskretnim događajima znatno je uznapredovalo u prošlom desetljeću. Pregled optimalizacijskih metoda za sustave s diskretnim događajima od 1988. godine dali su Swisher i drugi [SWI00]. Oni su proučavali

različite pristupe koji su bili usmjereni na poboljšavanje učinkovitosti optimalizacije sustava s diskretnim događajima.

Optimalizacijske tehnike koje su primjenjive na sustave s diskretnim događajima mogu se podijeliti ovisno o tome jesu li deterministički ulazni parametri kontinuirani ili diskretni. Osamdesetih godina dvadesetog stoljeća dominirale su metode optimalizacije sustava s diskretnim događajima s kontinuiranim ulaznim parametrima. Među njima dominirali su stohastički aproksimacijski algoritmi i tehnike procjene gradijenta. Stohastički aproksimacijski algoritmi su primjenjivi na optimalizacijske probleme u sustavima s diskretnim događajima samo ako postoji pogodna gradijentna metoda koja se svodi na pronalaženje minimuma funkcije iz nužnog i dovoljnog uvjeta za minimum funkcije, da je gradijent funkcije jednak nuli $\nabla f = 0$. Predloženo je nekoliko tehnika, a najčešća među njima je *analiza perturbacija* (engl. *Perturbation analysis*) [FU97]. Takve tehnike su kao optimalizacijske procedure bile ugrađene u stohastičke algoritme i kontrolirale su veličinu koraka uzetog u smjeru gradijenta pri svakoj iteraciji. Ne-gradijentne metode su bile alternativa gore navedenim metodama. One uključuju Nelder-Mead (simpleks) metodu i Hooke & Jeeves metodu [SWI00]).

Problemi optimalizacije sustava s diskretnim događajima obično imaju neke diskretne varijable, kao što su broj strojeva, broj vozila, broj ručnih uređaja, veličina spremnika, itd. Zbog toga, za optimalizaciju takvih sustava nisu pogodne metode kontinuiranog prostora stanja, kao što su stohastičke aproksimacije te se devedesetih godina 20. stoljeća istraživači okreću prema DES s diskretnim ulaznim parametrima. Ako je skup ulaznih parametara konačan i mali, onda su primjenjive statističke metode rangiranja i selekcije (Goldsman i Nelson (1994, 1998), vidi [SWI00]). Statistička optimalizacija se zasniva na izboru jednog rješenja iz velikog skupa alternativa i naglasak je na statističkom sagledavanju problema.

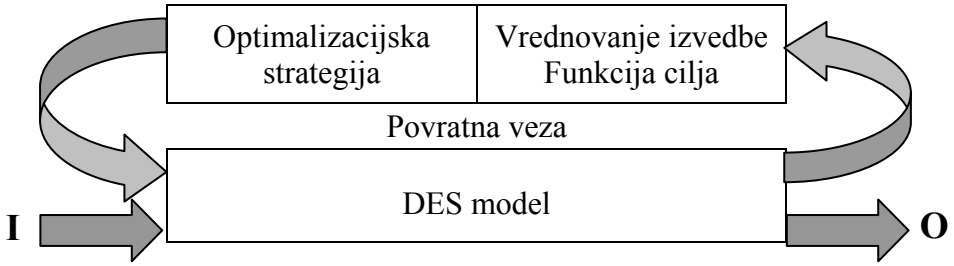
Drugi pristup je poznat pod nazivom *generativne* ili *konstruktivne* tehnike optimalizacije s naglaskom na tehnike slučajnog pretraživanja prostora rješenja. Kod ovih tehnika prvenstveno su važni zadani skup kriterija i dinamička ograničenja, preko kojih algoritam traži poboljšano rješenje i generira skup odluka. U ovu grupu spadaju oni problemi koji dopuštaju primjenu numeričkih metoda programiranja, kao što su dinamičko programiranje, cjelobrojno mješovito programiranje, cjelobrojno kvadratno programiranje. To su razne iterativne tehnike kojima se maksimalizira (minimalizira) funkciju cilja niza varijabli. Pri tome varijable funkcije cilja moraju zadovoljiti niz ograničenja. Ove metode su primjenjive ako je sustav s diskretnim događajima jednostavan, jer numeričke metode postaju vrlo složene ako se želi uzeti u obzir stvarno stanje sustava i optimalizacija po više raznih parametara.

Međutim, ako je skup ulaznih parametara beskonačan ili izrazito velik, onda je nepraktično potpuno pretraživanje prostora rješenja. Zato se u većini radova istražuju brze *heurističke* metode iz kombinatoričke optimalizacije, koje se temelje na evolucijskoj strategiji, kao što su simulirano kaljenje, tabu pretraživanje i genetski algoritmi [GEN97]. Ovo su općenite metode koje zahtijevaju razvijanje određenih algoritama i odabir parametara za rješavanje svakog problema posebno.

Heuristički algoritmi ne garantiraju nalaženje optimalnog rješenja, ali mogu brzo pronaći rješenje koje je blizu optimalnog. Tompkins i Azadivar (1995) (vidi [SWI00]) koriste genetske algoritme za rješavanje problema optimalizacije proizvodnih sustava, pri čemu su se algoritmi pokazali pogodnim pri optimiranju kvalitativnih ulaznih parametara. U [LAC01] uspoređuje se simulirano kaljenje i genetski algoritam s Nelder-Mead simpleks algoritmom za

analizu četiri različita slučaja iz industrijskog okruženje. Algoritmi su prilagođeni deterministčkoj optimalizaciji. Nelder-Mead simpleks algoritam se pokazao dobrim za male ili srednje probleme (do 12 ulaznih parametara), samo s numeričkim varijablama. Genetski algoritam se pokazao kao jako robusna metoda za rješavanje velikih problema i problema s ne-numeričkim varijablama.

Algoritmi pretraživanja na inteligentan način vode DES model do optimalnog rješenja (slika 2.8).



Slika 2.8. Iterativna optimalizacija

Kao što je vidljivo, optimalizacijski proces je iterativan proces između DES modela i optimalizacijske strategije. Korisnik unosi početne ulazne vrijednosti i inicira proces. Ove vrijednosti se šalju DES modelu, čijim izvođenjem se generiraju izlazne vrijednosti i vrednuje se izvedba sustava pomoću algoritma pretraživanja. Kakvoća izlaznih vrijednosti vodi algoritam ka odabiru novih ulaznih vrijednosti i proces se ponavlja. Generiranje izlaznih značajki sustava može se ostvariti nekim od modela koji su opisani u podpoglavlju 2.2.

Općenito, a u skladu s gore navedenim, pristupi optimalizaciji sustava s diskretnim događajima se mogu kategorizirati na sljedeći način:

- algoritmi koji se zasnivaju na gradijentnoj metodi (uglavnom za stohastičke optimalizacije);
- algoritmi koji se zasnivaju na matematičkom programiranju.

U ovom radu će se razmatrati deterministički modeli sustava s diskretnim događajima. Primjerice, za problem transporta dolasci i vrijeme servisiranja se mogu shvatiti kao determinističke veličine. Takvo shvaćanje ima smisla, jer se vremena mogu vrlo precizno predvidjeti iz praćenja prometa, vremenskog planiranja, obračunskih ugovora i slično. Slično vrijedi i za većinu proizvodnih sustava, gdje se dolasci poslova i vremena završetka poslova mogu predvidjeti iz rasporeda i dokumenata planiranja. Za optimalizaciju takvih sustava koristit će se genetski algoritmi, jer pripadaju skupini heurističkih algoritama koji ne zavise o problemu koji se rješava te se mogu primijeniti na determinističke probleme optimalizacije s bilo kakvim ograničenjima.

2.5. Sažetak poglavlja

U ovom poglavlju opisana je posebna klasa sustava, DES, koja je u proteklom desetljeću postala integralni dio našeg svijeta [CASS99]. Prije nego što su opisani takvi sustavi, ovo poglavlje započinje kratkim opisom pojma "sustav" i iznošenjem najvažnijih koncepata vezanih uz teoriju sustava (podpoglavlje 2.1).

Nakon toga prikazan je pregled klasifikacije sustava kako bi se uočile značajke DES, sustava koji je predmet istraživanja ovog rada. Dinamički sustavi s diskretnim događajima, ili samo sustavi s diskretnim događajima su sustavi gdje je prostor stanja diskretan. Kao rezultat asinkronog pojavljivanja događaja kroz određeno vrijeme, kod takvih sustava mijenjaju se jedino stanja sustava. Uobičajene diferencijalne jednadžbe nisu pogodne za opisivanje DES. Osnovni elementi svakog DES-a su

(a) prostor diskretnih stanja koji se uobičajeno označava s $\mathbf{X}$,

(b) skup diskretnih događaja koji se označava s E.

U ovom poglavlju opisane su fundamentalne značajke DES-a i opisana su dva realna sustava (računalni sustav i dio proizvodnog sustava) koji su ovdje sagledani kao sustavi s diskretnim događajima.

Kako bi dizajniranje DES-a bilo što učinkovitije, mora se povezati s optimalizacijom tog sustava, odnosno s optimalizacijom određenih parametara u sustavu u cilju poboljšanja vrijednosti odabranih značajki u sustavu. Zbog toga su na kraju poglavlja kratko opisane metode optimalizacije DES-a, kao i problemi koji su vezani za optimalizaciju takvih sustava.

3. PETRIJEVE MREŽE

Za Petrijeve mreže se može reći da imaju dugu povijest uporabe. Razlog za ovo nije samo njihovo uvođenje prije više do četrdeset godina u doktorskoj disertaciji *Kommunikation mit Automat* Carl Adam Petri-a (1962). Mnogo je važnija činjenica da se Petrijeve mreže temelje na formalnoj teoriji koja se stalno proučava i razvija. Također, za Petrijeve mreže postoje različiti alati kao i širok skup aplikacija u kojima se koriste [WAN07].

Petrijeve mreže su poopćenja usmjerenih grafova. Usmjerenim grafom se ne mogu prikazati jako složeni sustavi, kao primjerice sustavi u kojima postoje paralelne ili konfliktne operacije i zbog toga se koriste Petrijeve mreže. Osnovna namjena Petrijevih mreža je modeliranje, simulacija i analiza procesa i sustava s diskretnim događajima [PET81].

Kako bi jedan alat za modeliranje bio dostupan širokom krugu korisnika, od istraživača u laboratorijima do inženjera u praksi, potrebno je da alat za modeliranje ima pogodan način uporabe. Kod Petrijevih mreža, ova značajka se odlikuje u kombiniranju matematičkog opisa s grafičkim predstavljanjem dinamičkog ponašanja sustava. Pri tome se dopušta analitička analiza strukture i svojstava modeliranog sustava, primjerice postojanosti, analiza prostora stanja, dostupnosti. Petrijeve mreže su primjenjive u brojnim aplikacijskim domenama, primjerice računalnim mrežama, komunikacijskim sustavima, proizvodnim sustavima [ZUR94], upravljačkim sustavima [CAM93], kemiji, matematici pa čak i pravosudnim sustavima. Petrijevim mrežama mogu se prikazati stanja i događaji koji izazivaju prijelaze iz jednog u drugo stanje, te se mogu eksplicitno izraziti i uvjeti koji moraju biti zadovoljeni da bi došlo do prijelaza iz jednog stanja u drugo [GIR03].

Petrijeve mreže predstavljaju dinamički model sustava s diskretnim događajima, a koji se ne može dobiti Boolev-im izrazima kao kod stabla otkaza (engl. *fault tree*) ili klauzalnih grafova (engl. *clausal graph*). Formalizam Petrijevih mreža je značajno evoluirao u odnosu na svoje početno rješenje (Petri, 1962, [MUR89]). Da bi se podržao širok opseg primjena, razvijaju se različita proširenja osnovne definicije Petrijevih mreža. Neka od značajnijih proširenja su obojene Petrijeve mreže (engl. *Coloured Petri nets*) [JEN97], vremenske Petrijeve mreže (engl. *Timed Petri nets*) [WAN98], Petrijeve mreže sa sprječavajućim lukovima (engl. *Inhibitor arc Petri nets*), Petrijeve mreže s prioritetima (engl. *Priority Petri nets*), itd. Svaka od spomenutih Petrijevih mreža ima svoje značajke. U ovom poglavlju bit će obrađene opće Petrijeve mreže [MUR89, PET81], koje uostalom predstavljaju osnovu iz koje su izvedene i sve ostale vrste Petrijevih mreža. Pored općih Petrijevih mreža kratko se razmatraju obojene Petrijeve mreže i vremenske Petrijeve mreže.

3.1. Opće Petrijeve mreže

Kako su Petrijeve mreže posebna vrsta grafova, ovo poglavlje započinje s osnovnim pojmovima iz teorije grafova.

Graf se sastoji od dvije vrste elemenata, čvorovi i lukovi, te od mehanizma koji određuje kako su elementi međusobno povezani. U ovisnosti o načinu povezivanja, postoji nekoliko vrsta grafova.

Definicija 3.1 Jednostavni graf

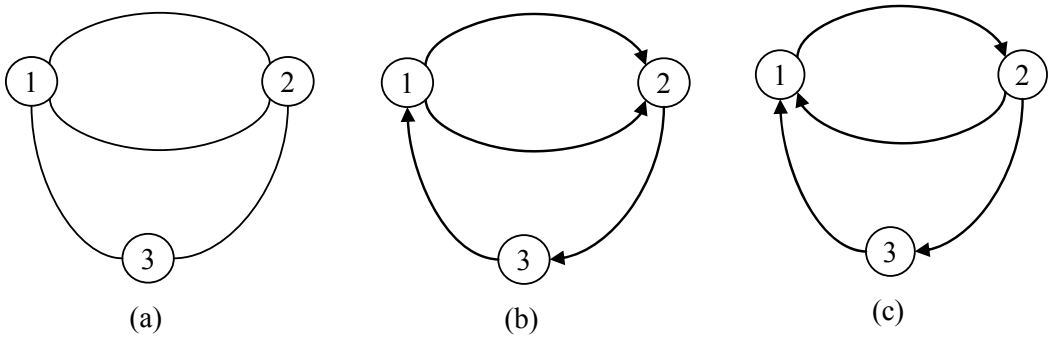
Jednostavni graf je trojka $G=(V,E,R)$, gdje je V neprazan, konačan *skup čvorova*, E skup *lukova,* odnosno skup simetričnih parova međusobno različitih čvorova u V. R je nerefleksivna simetrična relacija definirana na skupu V.

Relacija R je nerefleksivna što znači da elementi iz V nisu u relaciji sami sa sobom, odnosno $\forall a \in V,\ (a,a) \notin R$. Relacija R je simetrična što znači da ako je $(a,b) \in R,$ onda je i $(b,a) \in R$.

Ukoliko su lukovi predstavljeni uređenim parom čvorova, dobiveni graf se naziva *usmjereni graf.*

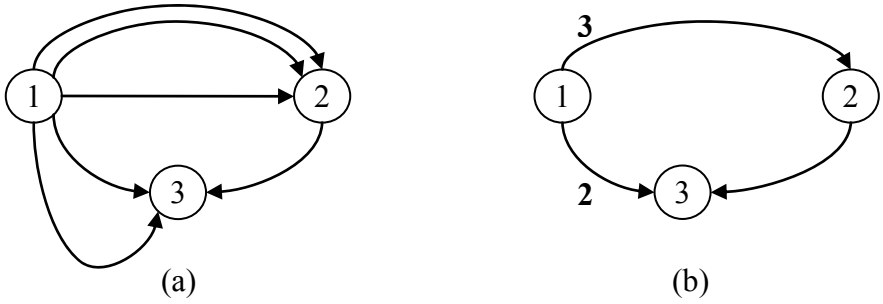
Definicija 3.2 Multigraf

Kada graf sadrži dva ili više paralelnih lukova, odnosno, lukove koji spajaju isti par čvorova i ako su usmjereni, imaju isti smjer, onda se takav graf naziva *multigraf* (slika 3.1b).



Slika 3.1 Primjeri grafova: (a) neusmjereni, (b) multigraf, (c) nije multigraf

Kako bi se pojednostavnilo grafičko prikazivanje paralelnih lukova, bolje ih je zamijeniti s jednim lukom kojemu je pridružen *faktor težine* luka $w\ (w \in N)$, koji označava broj paralelnih lukova. Ako je $w=1$, onda se lukovima ne pridružuje faktor težine (slika 3.2b).

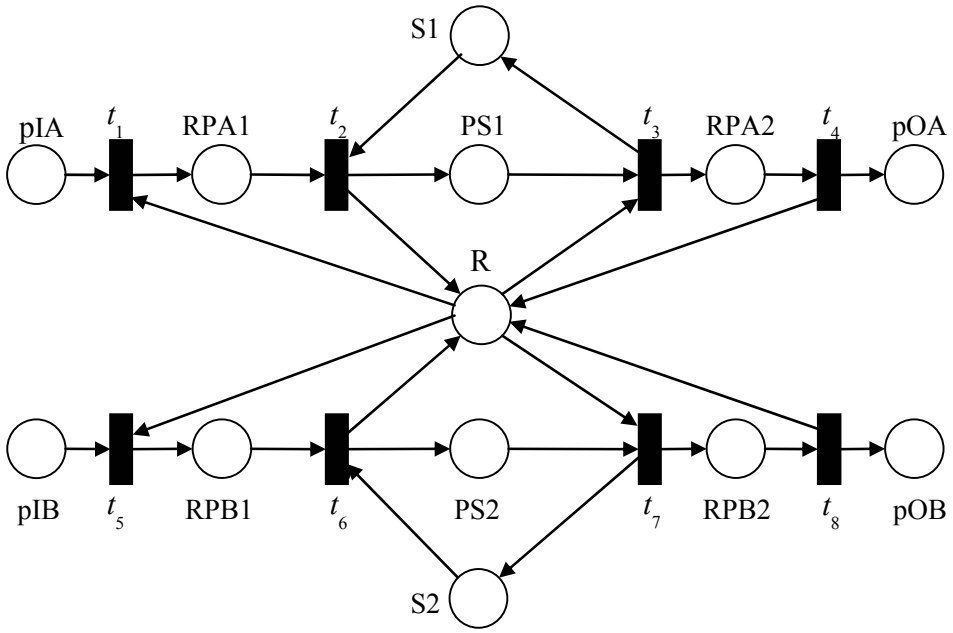


Slika 3.2 Primjer multigrafa: (a) bez navođenja težine lukova, (b) navođenjem težina lukova

Definicija 3.3 Bipartitni graf

Bipartitni graf je jednostavni graf u kojem se skup čvorova može podijeliti u dva skupa X i Y takva da svaki luk ima svojstvo da je jedan čvor u X, a drugi u Y. Ovakav graf se označava s $G=(X,Y,E)$.

Petrijeve mreže su *usmjereni bipartitni multigrafovi* određeni s četiri vrste objekata. Ti objekti su *mjesta* (engl. *places*), *prijelazi* (engl. *transitions*), *usmjereni lukovi* (engl. *directed arcs*) i *oznake* (engl. *tokens*). Mjesta i prijelazi predstavljaju čvorove Petrijeve mreže. Stanja, kao i određeni uvjeti u sustavima s diskretnim događajima, mogu se prikazati kao mjesta Petrijeve mreže, a događaji kao prijelazi Petrijeve mreže. Kako su Petrijeve mreže bipartitni grafovi, lukovi povezuju mjesta i prijelaze ili obrnuto, a nikada dva mjesta i/ ili dva prijelaza. Čvorovi i određena relacija njihovog povezivanja sačinjava *graf* Petrijeve mreže. Primjer grafa Petrijeve mreže prikazan je na slici 3.3. Uobičajeno je da se u Petrijevoj mreži mjesta označavaju kao krugovi, a prijelazi kao crno obojeni pravokutnik (ili deblja crta). Značenje simbola na grafu dano je u tablici 3.1.



Slika 3.3 Graf Petrijeve mreže proizvodnog sustava iz primjera 2.3

Tablica 3.1 Simboli za graf Petrijeve mreže sa slike 3.3

Simbol	Značenje	Simbol	Značenje
PIA	Dio *a* je raspoloživ	t_1	Prijelaz 1
PIB	Dio *b* je raspoloživ	t_2	Prijelaz 2
RPA1	Robot zahvaća dio *a*	t_3	Prijelaz 3
RPB1	Robot zahvaća dio *b*	t_4	Prijelaz 4
PS1	Stroj S1 obrađuje dio *a*	t_5	Prijelaz 5
PS2	Stroj S2 obrađuje dio *b*	t_6	Prijelaz 6
RPA2	Robot pomiče dio *a* sa stroja S1	t_7	Prijelaz 7
RPB2	Robot pomiče dio *b* sa stroja S2	t_8	Prijelaz 8
S1	Stroj S1 je slobodan		
S2	Stroj S2 je slobodan		
R	Robot je raspoloživ		
POA	Dio *a* se prosljeđuje u vanjski svijet		
POB	Dio *b* se prosljeđuje u vanjski svijet		

■

Definicija 3.4 Graf Petrijeve mreže (Peterson, 1981)

Graf Petrijeve mreže je dvostrani graf definiran kao uređena četvorka:

$$NE = (P, T, A, w) \tag{3-1}$$

gdje je:

$P = \{p_1, p_2, \ldots, p_n\}, n > 0;$ - konačni skup mjesta,

$T = \{t_1, t_2, \ldots, t_s\}, s > 0;$ - konačni skup prijelaza,

$I : P \times T \to \{0,1\}$ - *ulazna* funkcija – povezuje mjesta s prijelazima,

$O : T \times P \to \{0,1\}$ - *izlazna* funkcija – povezuje prijelaze s mjestima,

$A \subseteq (P \times T) \cup (T \times P)$ - konačni skup usmjerenih lukova,

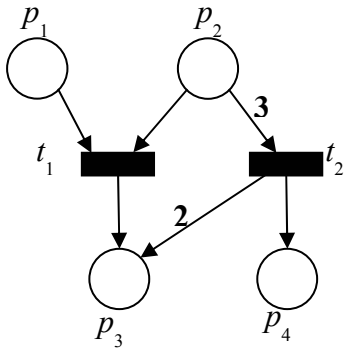
$w : A \to \{1, 2, 3, \ldots\}$ - težinska funkcija koja elementu iz skupa usmjerenih lukova pridružuje faktor težine luka,

$P \cap T = 0$ i $P \cup T \neq 0$.

Tipični luk označava se kao (p_i, t_i) ili (t_i, p_i).

Primjer 3.1

U ovom primjeru razmatra se graf Petrijeve mreže koji se sastoji od četiri mjesta i dva prijelaza (slika 3.4).



Slika 3.4 Graf Petrijeve mreže

Prema definiciji, graf Petrijeve mreže sadrži:

$$P = \{p_1, p_2, p_3, p_4\}; \quad T = \{t_1, t_2\};$$

$$I = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix}; \quad O = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 \end{bmatrix};$$

$$A = \{(p_1, t_1), (p_2, t_1), (p_2, t_2), (t_1, p_3), (t_2, p_3), (t_2, p_4)\}.$$

Težinska funkcija w pridjeljuje svakom od usmjerenih lukova faktor težine luka:

$$w(p_1, t_1) = 1,\ w(p_2, t_1) = 1,\ w(p_2, t_2) = 3,\ w(t_1, p_3) = 1,\ w(t_2, p_3) = 2,\ w(t_2, p_4) = 1.$$

■

Definicija 3.5 Skup ulaznih i izlaznih čvorova

Skup ulaznih čvorova $x \in P \cup T$ može se definirati kao:

$$I(x) = \{y \in P \cup T : (y,x) \in A\}, \tag{3-2}$$

gdje je:

$I(x)$ - skup ulaznih čvorova u čvor x.

Skup izlaznih čvorova $x \in P \cup T$ može se definirati kao:

$$O(x) = \{y \in P \cup T : (x,y) \in A\}, \tag{3-3}$$

gdje je:

$O(x)$ - skup izlaznih čvorova iz čvora *x*.

Čvor grafa *NE* za kojeg vrijedi da je $\bullet x = 0 = x \bullet$ naziva se *izoliranim* čvorom . Nadalje se podrazumijeva da graf *NE* nema izoliranih čvorova.

Za graf Petrijeve mreže *NE* se kaže da je *čist* (engl. *pure*) onda i samo onda ako vrijedi:

$$\forall x, y \in P \cup T : (x,y) \in A \Rightarrow (y,x) \notin A. \tag{3-4}$$

Relacija (3-4) znači da je graf *NE* čist ako ne postoji povratni luk na isti čvor. Ukoliko postoji npr. luk iz nekog mjesta ka određenom prijelazu, tada ne smije postojati povratni luk iz tog prijelaza nazad na isto mjesto.

Da bi se moglo grafički prikazati aktivnost određenih mjesta, potrebno je graf Petrijeve mreže proširiti na "opću Petrijevu mrežu".

Definicija 3.6 Opća Petrijeva mreža (Cassandras, Lafortune, 1999)

Opća Petrijeva mreža (u daljnjem tekstu Petrijeva mreža) je definirana kao petorka:

$$PM = (P,T,A,w,m_0), \tag{3-5}$$

gdje je:

P, T, A, w - jednako definirani kao kod grafa Petrijeve mreže (3–1),

$m_0 : P \to \{0,1,2,3,...\}$ - početno stanje Petrijeve mreže.

Značajka koja razlikuje Petrijeve mreže od općih grafova je funkcija *stanja* $m : P \to N = \{0,1,2,3,...\}$, koje svakom mjestu p Petrijeve mreže pridružuje ne-negativni cijeli broj. $m(p) = l$ se naziva *broj oznaka* i označava se kao *l* točaka unutar mjesta (kruga). Stanje Petrijeve mreže *m* definirano je stupčastim vektorom $m = [m(p_1), m(p_2), \ldots, m(p_n)]^{\mathrm{T}} \in N^n$, gdje je *n* broj mjesta u Petrijevoj mreži. *i*-to mjesto vektora *m* pokazuje broj oznaka u mjestu p_i, što se označava kao $m(p_i)$.

Početno stanje Petrijeve mreže m_0 je stupčasti vektor koji definira početni broj oznaka u mjestima Petrijeve mreže ($m_0 = [m_0(p_1), m_0(p_2), \ldots, m_0(p_n)]^T \in N^n$). Petrijeva mreža u početnom stanju definira se kao par (PM, m_0).

Svaki element *PM* ima svoje elemente koji ulaze u njega i izlaze iz njega. Tako mjesto može kao ulazne i izlazne elemente imati samo prijelaze, dok prijelazi za ulazne i izlazne elemente mogu imati samo mjesta. U mnogim slučajevima biti će interesantno znati ulazne i izlazne elemente nekog dijela *PM*. Stoga će se definirati operator $\bullet$ na način da, ako je $p \in P$ mjesto Petrijeve mreže, onda je $\bullet p$ skup svih prijelaza koji ulaze u mjesto *p*. Također, $p\bullet$ je skup svih prijelaza na izlazu iz mjesta $p \in P$. Analogno vrijedi i za prijelaze. Ako je $t \in T$ prijelaz onda je $\bullet t$ skup svih mjesta koji ulaze u prijelaz $t \in T$. Također, $t\bullet$ je skup svih mjesta na izlazu iz prijelaza $t \in T$.

S obzirom da Petrijeve mreže služe za modeliranje dinamičkog ponašanja sustava s diskretnim događajima, dinamika ponašanja mreža mjesto/prijelaz (engl. *place/transition nets)* dana je sljedećom definicijom.

Definicija 3.7 Omogućen prijelaz

Prijelaz $t_j \in T$ je *omogućen* iz stanja *m* onda i samo onda ako svako mjesto *p* koje ima luk prema prijelazu t_j sadrži najmanje toliko oznaka koliki je faktor težine $w(p, t_j)$ pridružen tom luku. Matematički, to se može izraziti sljedećom nejednadžbom:

$$\forall p \in \bullet t_j : m(p) \geq w(p, t_j) \qquad (3\text{-}6)$$

Definicija 3.8 Okidanje prijelaza

Ukoliko je prijelaz omogućen, tada je moguće "*okinuti*" (engl. *fire*) prijelaz. Okidanjem prijelaza dolazi do promjene stanja unutar Petrijeve mreže.

Kada se prijelaz okine:

- Smanjuje se broj oznaka za sva mjesta koja se nalaze ispred prijelaza za onoliko koliki je faktor težine ulaznog luka.
- Povećava se broj oznaka za sva mjesta koja se nalaze iza prijelaza za onoliko koliki je faktor težine izlaznog luka.

Mehanizam okidanja mreže formalno je definiran preko funkcije prijelaza stanja.

Definicija 3.9 Funkcija prijelaza stanja

Funkcija prijelaza stanja $f : N^n \times T \to N^n$ Petrijeve mreže definirana je za prijelaz $t_j \in T$ samo ako je on omogućen (zadovoljena relacija (3–6)). Ako je relacija (3–6) zadovoljena i ako je funkcija $f(m, t_j)$ definirana, tada se prijelaz može okinuti, te nastaje novo stanje Petrijeve mreže opisano vektorom $m' = f(m, t_j)$ za koji vrijedi:

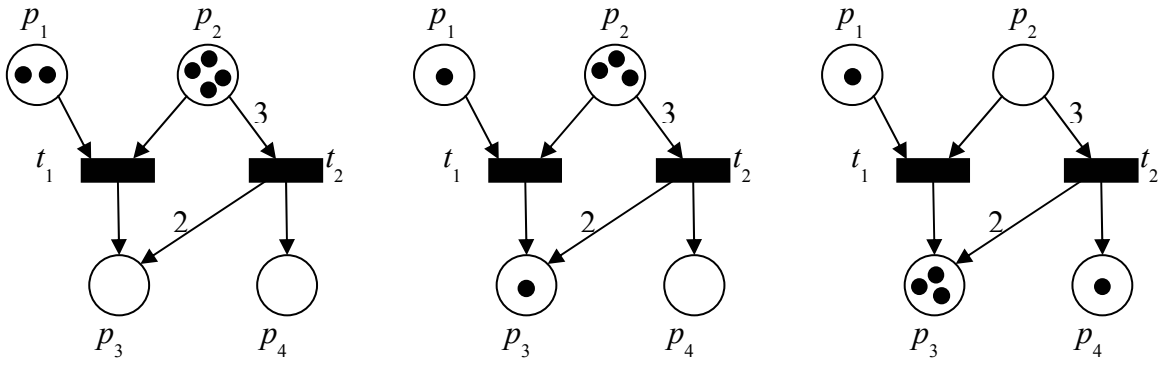
$$m'(p_i) = m(p_i) - w\left(p_i, t_j\right) + w\left(t_j, p_i\right), \; i = 1,...,n \tag{3-7}$$

Prijelaz iz stanja m u stanje m' okidanjem prijelaza t_j označava se kao $m\left[t_j > m'\right.$.

Moguća je i situacija kada je p_i i ulazno i izlazno mjesto u t_j (nije zadovoljena relacija (3-4)), u kojem slučaju se po relaciji (3-7) istovremeno odstranjuje $w(p_i, t_j)$ oznaka iz p_i i dodaje $w(t_j, p_i)$ novih oznaka nazad u p_i. Broj oznaka Petrijeve mreže može ali i ne mora biti isti prije i poslije okidanja prijelaza, što proizlazi iz relacije (3-7).

Primjer 3.2

Proces okidanja prijelaza na primjeru PM iz primjera 3.1 prikazan je na slici 3.5. Početno stanje mreže može se prikazati vektorom početnog stanja $m_0 = [2 \;\; 4 \;\; 0 \;\; 0]^T$ (slika 3.5a). Iz tog stanja je omogućen prijelaz t_1, jer je zadovoljena relacija $w(p_1, t_1) < m_0(p_1) = 2$ i $w(p_2, t_1) < m_0(p_2) = 4$. S druge strane, omogućen je i prijelaz t_2, jer je zadovoljena relacija $w(p_2, t_2) < m_0(p_2) = 4$. Ako se prijelaz t_1 okine, dolazi do odstranjivanja po jedne oznake iz mjesta p_1, p_2 i do dodavanja jedne oznake u mjesto p_3. Novo stanje mreže je $m_1 = [1 \;\; 3 \;\; 1 \;\; 0]^{\mathrm{T}}$ (slika 3.5b).



(a) stanje m_0 (b) stanje m_1 (c) stanje m_2

Slika 3.5 Okidanje prijelaza Petrijeve mreže

Iz stanja m_1 omogućeni su prijelazi t_1 i t_2. Po okidanju t_2 tri oznake se odstranjuje iz mjesta p_2, a 2 oznake se dodaju mjestu p_3 i jedna oznaka mjestu p_4. Novo stanje Petrijeve mreže je $m_2 = [1 \;\; 0 \;\; 3 \;\; 1]^{\mathrm{T}}$ (slika 3.5c). Iz ovog stanja više nije moguće okinuti niti jedan prijelaz u Petrijevoj mreži, pa kažemo da je mreža u stanju potpunog zastoja.

■

Definicija 3.10 Put Petrijeve mreže (Cassandras, Lafortune, 1999)

Put (engl. *path*) Petrijeve mreže je neprazan niz čvorova mreže $(x_1, x_2, ..., x_r)$ za koje vrijedi:

$$\forall i, 1 \le i \le r-1 : (x_i, x_{i+1}) \in A, \tag{3-8}$$

gdje je:

r - nenegativan cijeli broj.

Put se naziva *elementarnim* onda i samo onda ako su x_i različiti izuzev prvog i zadnjeg čvora x_1 i x_r. Petrijeva mreža se naziva strogo povezanom mrežom ako postoji put između bilo kojih čvorova mreže.

Krug (engl. *circle*) Petrijeve mreže je put $(x_1, x_2, ..., x_r)$ za koje vrijedi da je $x_1 = x_r$. Krug je elementaran ako je elementaran put koji ga sačinjava.

Pored općih Petrijevih mreža postoje i *obične (engl. simple) Petrijeve mreže* koje predstavljaju podvrstu općih Petrijevih mreža gdje je faktor težine svih lukova jednak 1 [MUR89].

Definicija 3.11 PT Petrijeva mreža

Petrijeve mreže mjesto/prijelaz (engl. *Place-transition nets,* pokrata: *PT mreže*) su također vrsta općih Petrijevih mreža kojima je faktor težine lukova od mjesta ka prijelazima jednak 1, dok faktori težine lukova od prijelaza ka mjestima mogu biti veći ili jednaki 1, što se matematički može izraziti kako slijedi:

Ako je $PM = (P, T, A, w, m_0)$ opća Petrijeva mreža, tada PM nazivamo PT Petrijevom mrežom ako vrijedi relacija:

$$\forall p \in P, \forall t \in T, \_ako \_ je \_(p,t) \in A \_tada \_ je \_ w(p,t) = 1. \tag{3-9}$$

PT mreže se označavaju s $PT = (P, T, A, w, m_0)$.

3.2. Značajke Petrijevih mreža

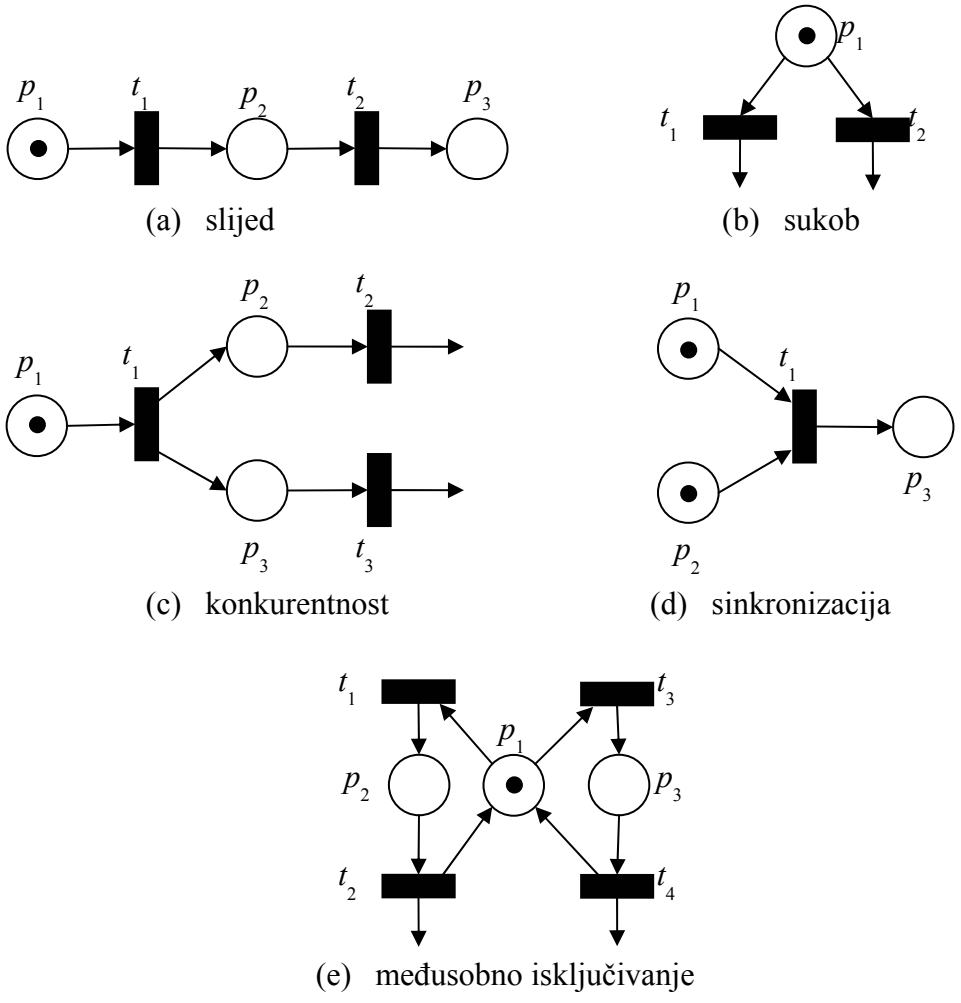
Uz osnovna svojstava Petrijevih mreža vežu se pojave nekih uobičajenih značajki izvođenja operacija, kao što su slijedno izvođenje, konkurentnost, sukobi i sinkronizacija.

1) *Slijedno izvođenje* (engl. *sequential execution*). Na slici 3.6a prikazano je slijedno izvođenje operacija. Prijelaz t_2 moguće je okinuti samo nakon okidanja prijelaza t_1. Time je određen kriterij prvenstava "t_2 nakon t_1". Ovakvi kriteriji prednosti su uobičajeni kod izvođenja dijelova dinamičkog sustava. Isto tako, ovakva konstrukcija Petrijeve mreže modelira uzročno-posljedične veze među operacijama.

2) *Sukob* (engl. *conflict*). Slika 3.6b prikazuje situaciju sukoba u Petrijevoj mreži, koja nastaje kada iz jednog mjesta izlazi više lukova prema više različitih prijelaza. U slučaju da je mjesto

p_1 označeno, omogućeni su prijelazi t_1 i t_2. Ako se okine prijelaz t_1, odstrani se oznaka iz mjesta p_1 i više se ne može omogućiti prijelaz t_2. Ukoliko su događaji pridruženi pojedinim prijelazima, onda se vidi da su ti događaji u sukobu, međusobno su konkurentni i ovisni. Pojava jednog od njih isključuje pojavu drugog, što može dovesti do nedeterminističkog ponašanja sustava.

3) *Konkurentnost ili razdvajanje* (engl. *concurrency*). Do konkurentnosti dolazi kada se na jedan slijed nastavljaju dva slijeda koji se odvijaju paralelno. Na slici 3.6c, prijelazi t_1 i t_2 su međusobno konkurentni.



Slika 3.6 Osnovni oblici Petrijeve mreže za prikaz značajki sustava: (a) slijed, (b) sukob, (c) istodobnost, (d) sinkronizacija, (e) međusobno isključivanje

4) *Sinkronizacija* (engl. *synchronization*). Prirodno je kod dinamičkih sustava da događaj zahtijeva mnogostruke resurse. Sinkronizacija resursa se može istaknuti prijelazima kao na slici 3.6d. Da bi se prijelaz t_1 omogućio, potrebno je da su oba mjesta p_1 i p_2 označena ili aktivna. Dakle, za okidanje prijelaza potrebna je međusobna usklađenost dvaju stanja.

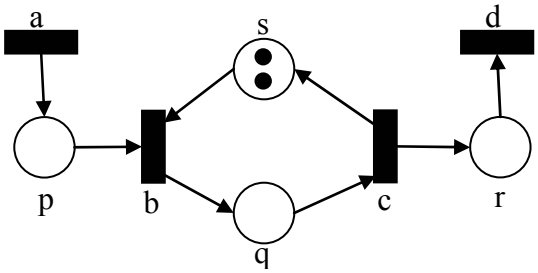
5) *Međusobno isključivanje* (engl. *mutually exclusive*). Dva procesa se međusobno isključuju ako se ne mogu istovremeno izvesti u skladu s ograničenjima na uporabu zajedničkih resursa.

Slika 3.6e prikazuje takvu situaciju. Primjerice, strojevi mogu dijeliti robot koji zahvaća element, dovodi ga na stroj i nakon obrade ga prebacuje sa stroja u vanjski svijet.

Petrijeve mreže su pogodne za modeliranje i analizu sustava s ograničenim resursima.

Primjer 3.3

Na primjeru reda čekanja s dva servera (slika 3.7) prikazan je sustav s ograničenim resursima. Prijelaz *a* označava dolazak klijenata, *b* i *c* označavaju početak i kraj posluživanja, a *d* označava odlazak klijenata. Mjesto *p* označava klijente koji čekaju da budu posluženi, mjesto *q* modelira klijente koji se upravo poslužuju. Klijent koji se poslužuje zauzeo je resurs, odnosno server. Mjesto *s* označava da su resursi slobodni. U početku su dva resursa (servera) raspoloživa. Kako bi okidanje prijelaza *b* bilo moguće, mora biti aktivno mjesto *p* (postoji klijent koji čeka) i u mjestu *s* mora biti oznaka koja označava da je server slobodan. Kada stigne prvi klijent, server postaje zaposlen. Ako drugi klijent stigne prije nego što je prvi poslužen, drugi server postaje zaposlen. Ako treći klijent stigne prije nego što su poslužena prva dva klijenta, on mora čekati sve dok jedan od servera ne postane slobodan.



Slika 3.7 Model reda čekanja s dva servera

Kao matematički alat, Petrijeve mreže posjeduju brojna svojstva, koja se mogu podijeliti u dvije grupe. U prvu grupu spadaju ona svojstva koja ovise o početnom stanju, a nazivaju se *svojstva ponašanja* (engl. *behavioral properties*). U drugu grupu spadaju ona svojstva koja ne ovise o početnom stanju, a poznata su kao *strukturna svojstva* (engl. *structural properties*) [BOG06].

Neka od važnijih svojstva Petrijevih mreža su [MUR89]:

- Osnovna svojstva
    - Dostupnost
    - Omeđenost
    - Sigurnost
    - Živost
    - Reverzibilnost
    - Prekrivenost
- Strukturna svojstva
    - Postojanost
    - Strukturna omeđenost
    - P i T postojanost

Definicija 3.12 Dostupnost (engl. *reachability*)

Za stanje m_j kaže se da je *dostupno* iz stanja m_i ako postoji niz okidanja prijelaza $\sigma_{ij} = t_m t_r \ldots t_p$ koji omogućava prijelaz iz stanja m_i u stanje m_j. To se označava kao $m_i \lfloor \sigma_{ij} > m_j$.

Skup svih *mogućih dostupnih stanja* (engl. *reachability set*), koji se mogu doseći iz početnog stanja m_0 označava se kao $R(m_0)$, a skup svih mogućih nizova okidanja koji su mogući iz stanja m_0 označavaju se kao $L(m_0)$. Budući da se ponekad može doći do određenog stanja preko različitih nizova okidanja, broj članova skupa $R(m_0)$ ne mora nužno biti isti broju članova skupa $L(m_0)$.

Problem dostupnosti u Petrijevoj mreži svodi se na određivanje niza okidanja prijelaza σ koji u Petrijevoj mreži osiguravaju da se stanje m_0 promjeni u stanje m_n. Tada vrijedi da je $m_n \in R(m_0)$ i da je $\sigma \in L(m_0)$. Ako se stanje m_n ne može doseći, tada je vjerojatno napravljena pogreška prilikom modeliranja sustava i Petrijeva mreža u tom slučaju ne predstavlja željeni realni sustav.

Definicija 3.13 Omeđenost (engl. *boundness*)

Za Petrijevu mrežu PM s početnim stanjem m_0 kaže se da je *k-omeđena* onda i samo onda ako postoji prirodni broj *k* takav da za svako dostupno stanje $m_n \in R(m_0)$, broj oznaka u bilom kojem mjestu ne prelazi broj *k,* odnosno

$$(\forall p \in P)(\forall m_n \in R(m_0))(\exists k \in N) \text{ takav da je } m(p) \le k \qquad (3\text{-}10)$$

PM je neomeđena ako je *k* beskonačno velik broj. Primjerice, PM prikazana slici 3.7 je neomeđena.

Definicija 3.14 Sigurnost (engl. *safeness*)

Ako je Petrijeva mreža s početnim stanjem m_0 k-omeđena i $k = 1$, onda za tu Petrijevu mrežu kažemo da je *sigurna*.

Broj oznaka u mjestima Petrijeve mreže obično predstavlja broj aktivnih operacija ili broj resursa u sustavu kao što su palete, roboti, brojači, registri i slično, kojih može biti do određenog maksimalnog broja. Ukoliko se osigura da je mreža ograničena, tada ne može doći do prevelikog broja oznaka u određenim mjestima, odnosno prekoračenja kapaciteta resursa, bez obzira koji je niz prijelaza okidan.

Definicija 3.15 Živost (engl. *livness*)

Za Petrijevu mrežu *PM* s početnim stanjem m_0 kaže se da je *živa* ako za bilo koje stanje m_n, koje je dosegnuto iz stanja m_0, postoji niz okidanja $\sigma = t_1 t_2 \ldots t_n$ koji iz stanja m_n može okinuti bilo koji prijelaz t_j, odnosno

$$\left(\forall t_j \in T\right)\left(\forall m_n \in R\left(m_0\right)\right)\left(\exists \sigma \in L\left(m_0\right)\right) \text{ tako da je } \left(m_0\left[\sigma > m_n\right)\right. \tag{3-11}$$

Koncept živosti je usko povezan s odsustvom zastoja u sustavu. To osigurava da živa mreža ne može doći u stanje zastoja bez obzira koji je niz okidanja $\sigma = t_1 t_2 ... t_n$ primijenjen.

Za prijelaz t_j u Petrijevoj mreži *PM* kaže se da je:

- "*mrtav*" ako ne postoji niti jedan član skupa $L\left(m_0\right)$ koji će okinuti prijelaz t_j,
- L_1 – živ (s potencijalnom mogućnošću okidanja) ako postoji barem jedan član skupa $L\left(m_0\right)$ koji ga može okinuti,
- L_2 – živ ako postoje određeni članovi skupa $L\left(m_0\right)$ koji ga mogu okinuti barem k puta, gdje je k bilo koji pozitivni cijeli broj,
- L_3 – živ ako postoji član skupa $L\left(m_0\right)$ koji ga može okinuti beskonačni broj puta,
- L_4 – živ ako je t_j živ za svaku oznaku $m_n \in R\left(m_0\right)$.

Za Petrijevu mrežu kaže se da je L_k*-živa* ako je svaki prijelaz u mreži L_k-živ (k = 0, 1, 2, 3, 4). Za prijelaz se kaže da je *striktno* L_k- živ ako je L_k- živ, ali ne i L_{k+1}-živ.

Vidljivo je da samo L_4 živa mreža zadovoljava definiciju živosti. Stoga se u daljnjem tekstu pod živom mrežom podrazumijeva upravo L_4 živa mreža. Model sustava s diskretnim događajima s L_4-živom Petrijevom mrežom nikada ne može doći u stanje zastoja, a to garantira da će se moći odviti svi modelirani procesi u mreži.

Za Petrijevu mrežu $\left(PM, m_0\right)$ kaže se da nema potpunog *zastoja* (engl. *deadlock-free*) onda i samo onda ako je:

$$\left(\forall m_n \in R\left(m_0\right)\right)\ \left(\exists t \in T\right) \text{ omogućen u } m_n \ . \tag{3-12}$$

Ako se mreža nalazi u stanju potpunog zastoja nije moguće okidanje niti jednog prijelaza.

Definicija 3.16 Reverzibilnost (engl. *reversibility*)

Redovito se zahtjeva od tehničkih sustava cikličko ponašanje, odnosno mogućnost povratka u početno stanje ili neko osnovno stanje. Petrijeva mreža *PM* je *reverzibilna* ako za svaku oznaku $m_i \in R\left(m_0\right)$ postoji niz okidanja σ_i takav da je početno stanje m_0 dostupno iz stanja m_i. Prema tome, u reverzibilnoj mreži uvijek postoji mogućnost vraćanja u početno stanje. U mnogim aplikacijama nije nužan povratak na početno stanje m_0, nego na neko drugo osnovno stanje m_h. Za stanje m_h se kaže da je *osnovno* ako za svaku oznaku $m_i \in R\left(m_0\right)$ vrijedi da je stanje m_h dostupno iz stanja m_i.

Kao primjer osnovnog stanja može se navesti stanje koje nastaje nakon iznenadnog kvara u sustavu. Tada sustav prestaje raditi po danom programu i vraća se u stanje iz kojega je moguć rad u izvanrednim okolnostima.

Definicija 3.17 Prekrivenost (engl. *coverability*)

Za stanje m_n u Petrijevoj mreži *PM* kaže se da je *prekriveno* ako postoji $m_n'(p)$ takav da je $m_n'(p) \geq m_n(p)$ za svaki p u mreži. Prekrivenost je povezana s L_1-živošću mreže.

Ako je $m_n \in R(m_0)$ minimalno stanje koje je potrebno da se omogući prijelaz t_j tada je prijelaz t_j mrtav (nije L_1-živ) onda i samo onda ako nije prekriven. To znači da je t_j živ samo ako je m_n prekriven.

3.3. Analiza Petrijevih mreža

U prethodnom poglavlju opisana su osnovna svojstva Petrijevih mreža. Međutim, samo modeliranje je od male koristi. Potrebno je analizirati modelirani sustav. Ova analiza bi trebala dati uvid u ponašanje modeliranog sustava.

Tri su osnovna pristupa analizi Petrijevih mreža [MUR89]:

1) analiza pomoću grafa dostupnih ili prekrivnih stanja,
2) analiza pomoću linearne algebre,
3) analiza pomoću redukcije i dekompozicije.

Analize pomoću grafa dostupnih ili prekrivnih stanja podrazumijeva pronalaženje svih dostupnih stanja ili svih prekrivenih stanja i daje odgovore na mnoga pitanja koja ovise o početnom stanju Petrijeve mreže. Ova se tehnika može, načelno, primijeniti na sve klase Petrijevih mreža, ali je ipak primjenjiva na relativno male mreže s ograničenim brojem dostupnih stanja zbog problema eksplozije stanja. S druge pak strane, analiza pomoću linearne algebre je uobičajenija i koristi se za određivanje strukturnih svojstava mreže.

3.3.1. Analiza pomoću grafa dostupnih ili prekrivnih stanja

Ako je Petrijeva mreža omeđena pomoću grafa dostupnih ili prekrivnih stanja moguće je analizirati osnovna svojstva Petrijevih mreža opisana u točki 3.1.1.

Ako je definirana Petrijeva mreža *PM* i početno stanje m_0, tada se može odrediti onoliko *novih* stanja u mreži koliki je broj omogućenih prijelaza iz svih tih stanja. Iz svakog se novog stanja ponovo mogu doseći nova stanja okidanjem novo omogućenih prijelaza. Na taj je način moguće dobiti strukturu grafa dostupnih stanja Petrijeve mreže ili grafa prekrivnih stanja Petrijeve mreže, koja se naziva *stablo dostupnih* ili *prekrivnih stanja* i prikazuje sva moguća dostupna ili prekrivna stanja Petrijeve mreže. Razlika između stabla dostupnih i prekrivnih stanja je u tome što prvo ima konačan broj stanja, a drugo beskonačan broj stanja.

Stablo dostupnih ili prekrivnih stanja izgleda kao okrenuto stablo, gdje je u korijenu početno stanje. Stablo dostupnih ili prekrivnih stanja sastoji se od čvorova, koji označavaju različita stanja mreže i lukova koji označavaju prijelaze između tih stanja. Kada se odredi stablo dostupnih ili prekrivnih stanja, iz njega se može dobiti graf dostupnih ili prekrivnih stanja mreže.

Graf dostupnih ili prekrivnih stanja Petrijeve mreže *PM* za koju je zadano početno stanje m_0 je označeni usmjereni graf koji je definiran kao:

$$G = (V, E), \tag{3-13}$$

gdje je:

V – skup čvorova (stanja),

E – skup lukova (prijelaza).

Svakom luku iz skupa E pridružuje se jedan prijelaz t_j Petrijeve mreže koji se može okinuti.

Put grafa stanja sastoji se od konačnog ili beskonačnog niza trojki definiranih kao:

$$(m, t_1, m_1), (m_1, t_2, m_2), ..., (m_{k-1}, t_k, m_k) = m[t_1, t_2, ..., t_k > \mathrm{m_k}, \tag{3-14}$$

gdje je :

m_1 - neposredno dostupno stanje iz m nakon okidanja t_1,

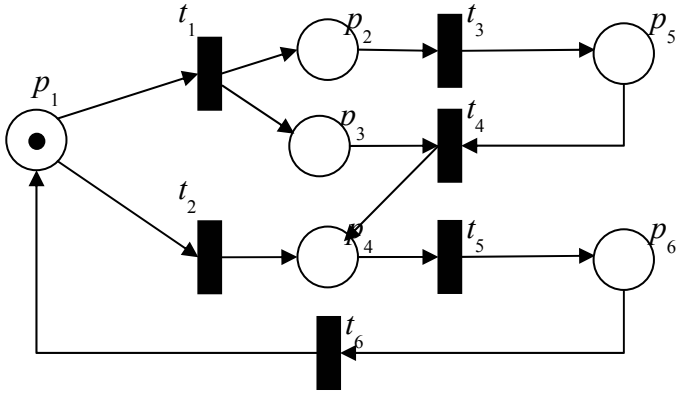
m_2 - neposredno dostupno stanje iz m_1 nakon okidanja t_2,

m_k - neposredno dostupno stanje iz m_{k-1} nakon okidanja t_k.

U primjeru 3.4 prikazana je konstrukcija stabla dostupnih stanja i pripadajućeg grafa dostupnih stanja, a u primjeru 3.5 prikazana je konstrukcija stabla prekrivnih stanja i grafa prekrivnih stanja.

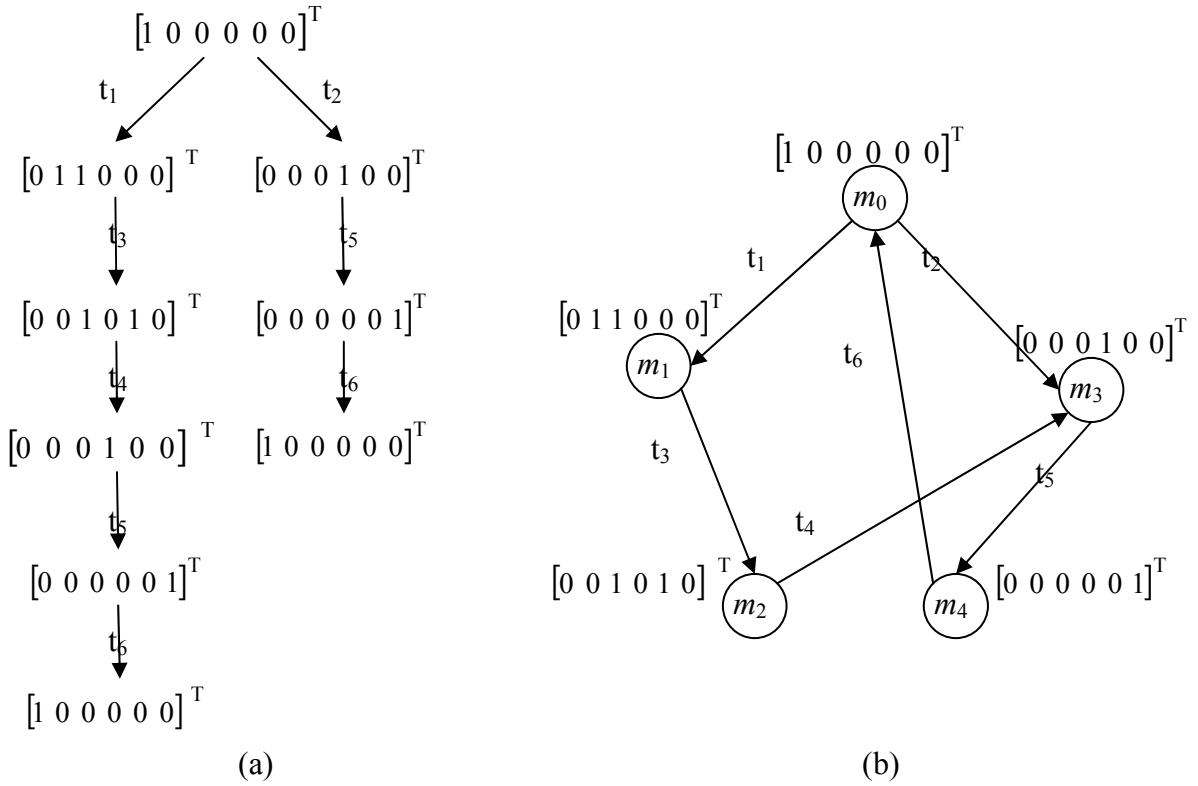
Primjer 3.4 (David, Alla, 1992):

Na slici 3.8 prikazana je Petrijeva mreža s početnim stanjem $m_0 = [1\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]^T$



Slika 3.8 Petrijeva mreža

Na slici 3.9a prikazano je pripadno stablo dostupnih stanja Petrijeve mreže sa slike 3.8, a na slici 3.9b prikazan je ekvivalentan graf dostupnih stanja. Iz slike 3.9a je vidljivo da postoji pet mogućih stanja m_0, m_1, m_2, m_3, m_4. Između pojedinih stanja prelazi se okidanjem odgovarajućih prijelaza koji su označeni pored lukova. Može se uočiti da je mreža sigurna (najveći broj oznaka u mjestima je jedan), živa (nema zastoja u mreži) i da je reverzibilna (nizovi okidanja prijelaza $t_1t_3t_4t_5t_6$ i $t_2t_5t_6$ vraćaju mrežu u početno stanje m_0).



Slika 3.9 (a) Stablo dostupnih stanja, (b) graf stanja Petrijeve mreže sa slike 3.8

■

Graf dostupnih stanja razlikuje se od stabla dostupnih stanja u činjenici da ne sadrži ponavljajuće čvorove te predstavlja daleko sažetiji prikaz skupa dostupnih stanja. No, graf dostupnih stanja ne mora imati konačan broj stanja, kao što je to u primjeru 3.4, već broj stanja može narasti do beskonačnosti, ukoliko Petrijeva mreža nije omeđena (u nekim mjestima je moguć beskonačno velik broj oznaka). Naravno, takav beskonačni graf dostupnih stanja je nemoguće analizirati, pa je stoga potrebno za takve slučajeve naći metodu konstruiranja konačnog grafa dostupnih stanja, koji se tada naziva grafom prekrivnih stanja.

Iz grafa dostupnih stanja Petrijeve mreže moguće je zaključiti sljedeće (Reisig i Rozenberg, 1998 [KEZ04]):

1) Petrijeva mreža (PM, m_0) je ograničena i prema tome je i $R(m_0)$ ograničen onda i samo onda ako ne postoji beskonačan broj u bilo kojem mjesta iz skupa V,
2) Petrijeva mreža (PM, m_0) je sigurna onda i samo onda ako se svi čvorovi iz skupa V sastoje samo od 0 ili 1,
3) Petrijeva mreža (PM, m_0) može doći u stanje potpunog zastoja onda i samo onda ako postoje čvorovi iz skupa V iz kojih ne ide barem jedan izlazni luk prema nekom drugom čvoru,
4) Petrijeva mreža (PM, m_0) je živa onda i samo onda ako za svaki čvor iz skupa V postoji put

$$(m,t_1,m_1),(m_1,t_2,m_2),\cdots(m_{k-1},t_k,m_k), \tag{3-15}$$

u kojemu niz $t_1,t_2,...,t_k$ sadrži sve prijelaze u mreži,

5) Petrijeva mreža (PM,m_0) je reverzibilna ako i samo ako postoji direktni put koji povezuje bilo koji čvor iz skupa V s početnim stanjem m_0.

3.3.2. Analiza pomoću linearne algebre

Kako se dinamičko ponašanje Petrijevih mreža ne može opisati diferencijalnim jednadžbama linearna algebra, koja se jednostavno može implementirati u računalne programe, ima veliko značenje u teoriji Petrijevih mreža. Zbog toga će se u ovom podpoglavlju prikazati način analize Petrijevih mreža pomoću matričnih jednadžbi. Rješivost ovih jednadžbi je prilično ograničena djelom zbog nedeterminističke prirode modela opisanih Petrijevim mrežama i zbog ograničenja da se rješenja mogu naći samo za pozitivne cijele brojeve.

Matematički opis procesa okidanja i prijelaza stanja u Petrijevoj mreži svodi se na određivanje relacije koja generira novo stanje $m_{k+1}=\left[m_{k+1}(p_1),m_{k+1}(p_2),\ldots,m_{k+1}(p_m)\right]^{\mathrm{T}}$, ako je poznato staro stanje $m_k=\left[m_k(p_1),m_k(p_2),\ldots,m_k(p_m)\right]^{\mathrm{T}}$ i određeni prijelazi t_j koji se okidaju.

Da bi se mogla napisati jednadžba stanja Petrijeve mreže, potrebno je definirati vektor okidanja u (*n*-dimenzionalni stupčasti vektor) koji ima oblik:

$$\mathrm{u}=\left[0\cdots011\cdots0\right]^{\mathrm{T}}, \tag{3-16}$$

gdje se može pojaviti jedinica na j-tom mjestu, $j\in\{1,...,n\}$, i koja pokazuje da se prijelazi t_j okidaju (n je ukupan broj prijelaza u mreži).

Pri svakom okidanju, dozvoljeno je samo jednom prijelazu da okine. To je izraženo sljedećom jednadžbom:

$$\sum_{i=1}^{n}\mathrm{u}_i(k)=d, \qquad k=1,2,\cdots d\ . \tag{3-17}$$

Pored toga, definira se i *matrica događanja* $\mathrm{A}=\left[a_{i,j}\right]$ (engl. *incidence matrix*) dimenzije $(m\times n)$, gdje je element matrice $a_{i,j}$ definiran kao:

$$a_{i,j}=w(t_j,p_i)-w(p_i,t_j), \tag{3-18}$$

gdje je:

$w(t_j,p_i)$ - faktor težine luka od prijelaza t_j do mjesta p_i,

$w(p_i,t_j)$ - faktor težine luka od mjesta p_i do prijelaza t_j,

m - ukupan broj mjesta u mreži,

n - ukupan broj prijelaza u mreži.

Matrica događanja A može se prikazati kao razlika dviju matrica:

$$\mathrm{A} = \mathrm{A}^{+} - \mathrm{A}^{-}, \tag{3-19}$$

gdje je:

$\mathrm{A}^{+} = \left[a_{i,j}{}^{+}\right]_{m\times n} = \left[w\left(p_i, t_j\right)\right]_{m\times n} : \left(P\times T\right)\to N$ - izlazna matrica događanja,

$\mathrm{A}^{-} = \left[a_{i,j}{}^{-}\right]_{m\times n} = \left[w\left(t_j, p_i\right)\right]_{m\times n} : \left(T\times N\right)\to N$ - ulazna matrica događanja.

Izlazna matrica događanja A^{+} definira izlazne lukove koji idu iz svih mjesta prema svim prijelazima mreže, dok ulazna matrica događanja A^{-} definira lukove koji idu iz svih prijelaza prema svim mjestima mreže.

Ako je poznato staro stanje m_k, matrica događanja A i vektor okidanja u promjene stanja se matematički definiraju primjenom sljedeće jednadžbe stanja koja daje kao rezultat novo stanje mreže m_{k+1}:

$$m_{k+1} = m_k + \mathrm{A}u\,. \tag{3-20}$$

Ako se promatra put Petrijeve mreže koji se sastoji iz niza stanja $\{x_o, x_1, ..., x_n\}$ koja su rezultat niza okidanja prijelaza σ (od kojih neki prijelazi mogu biti okidani i više puta), m-dimenzionalni vektor okidanja u se tada sastoji od *n* elemenata čiji *j-ti* element odgovara broju okidanja prijelaza t_j u nizu okidanja σ.

U sljedećem primjeru ilustrira se analiza Petrijeve mreže pomoću linearne algebre.

Primjer 3.5

Petrijeva mreža na slici 3.4 nalazi se u početnom stanju $m_0 = [2,4,0,0]$. Da bi se odredila nova stanja nakon okidanja pojedinih prijelaza, prvo je potrebno odrediti matricu događanja A koja se dobije analizom grafa Petrijeve mreže. U ovom je slučaju matrica događanja A:

$$\mathrm{A} = \begin{bmatrix} -1 & 0 \\ -1 & -3 \\ 1 & 2 \\ 0 & 1 \end{bmatrix}.$$

Matrica A ima četiri retka i dva stupca. Redci odgovaraju mjestima mreže, a stupci prijelazima u mreži. Elementi matrice A dobiju se primjenom jednadžbe (3-18). Na primjer element matrice $a_{2,1}$ dobije se:

$$a_{2,1} = w\left(t_1, p_2\right) - w\left(p_2, t_1\right) = 0 - 1 = -1\,.$$

Koristeći relaciju (3-20) može se primjerice dobiti stanje m_1 koje proizlazi iz stanja m_0 okidanjem prijelaza t_1:

$$m_1 = \begin{bmatrix} 2 & 4 & 0 & 0 \end{bmatrix}^T + \begin{bmatrix} -1 & 0 \\ -1 & -3 \\ 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix}^T = \begin{bmatrix} 2 & 4 & 0 & 0 \end{bmatrix}^T + \begin{bmatrix} -1 & -1 & 1 & 0 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 1 & 0 \end{bmatrix}^T$$

Na sličan način može se dobiti stanje m_2 iz stanja m_1, okidanjem prijelaza t_2:

$$m_2 = \begin{bmatrix} 1 & 3 & 1 & 0 \end{bmatrix}^T + \begin{bmatrix} -1 & 0 \\ -1 & -3 \\ 1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 & 1 & 0 \end{bmatrix}^T + \begin{bmatrix} 0 & -3 & 2 & 1 \end{bmatrix}^T = \begin{bmatrix} 1 & 0 & 3 & 1 \end{bmatrix}^T$$

■

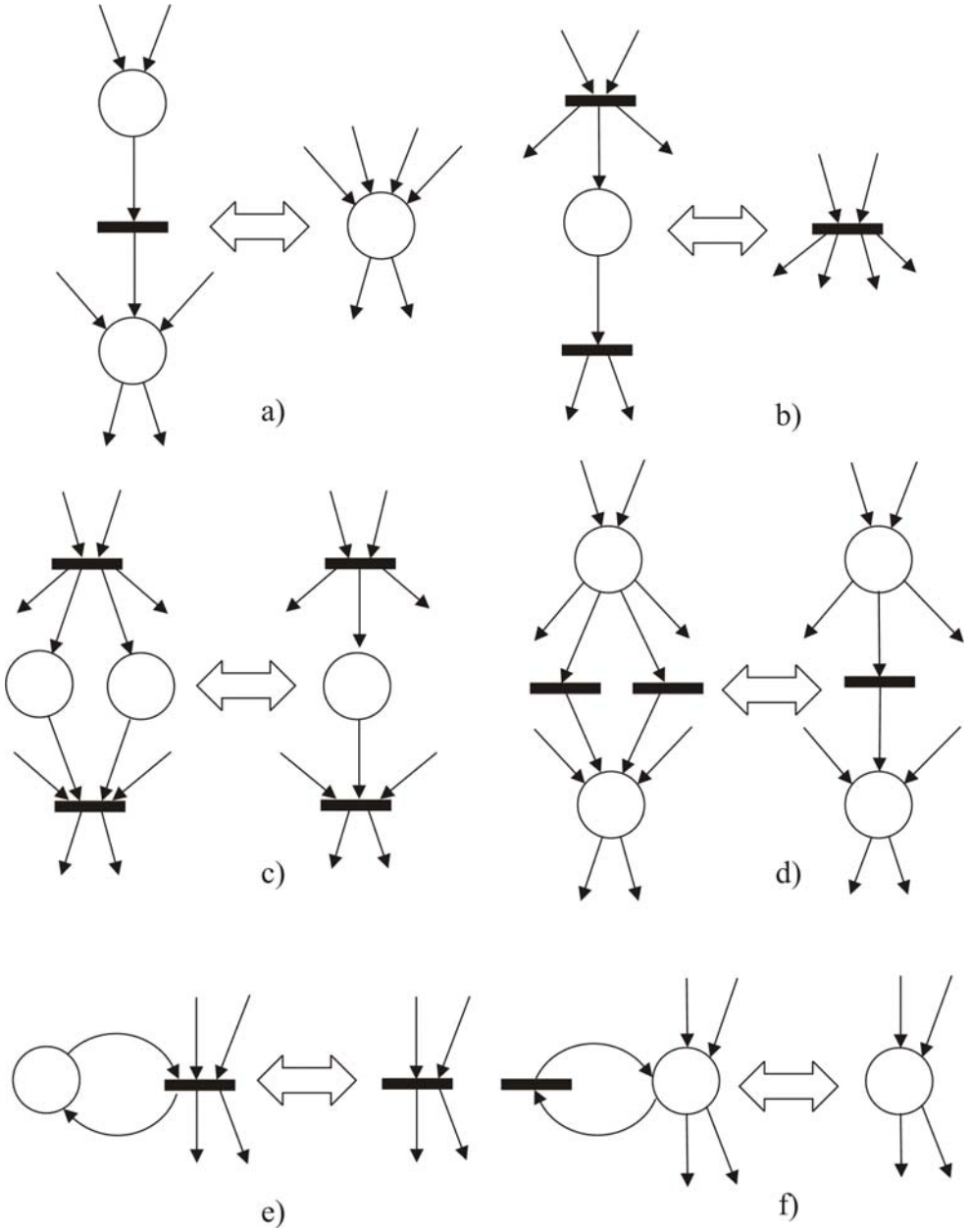
3.3.3. Analiza pomoću redukcije i dekompozicije

Veliki broj prijelaza i mjesta otežava proračun grafa dostupnih ili prekrivnih stanja mreže. Matrica događanja postaje preglomazna, a sve matematičke operacije s takvim matricama su teško izvodljive. Da bi se olakšala analiza velikih Petrijevih mreža, koje se susreću pri modeliranju složenih procesa, primjenjuje se postupak redukcije i dekompozicije veće Petrijeve mreže u manju mrežu koja je pogodnija za daljnju analizu. Pri tome je bitno da reducirana mreža zadrži sve značajke koje je imala originalna velika mreža.

Ako je definirana Petrijeva mreža (PM, m_0) prije redukcije i dekompozicije, i mreža nakon redukcije i dekompozicije (PM', m_0'), tada se može dokazati da je (PM', m_0') živa, sigurna i ograničena onda i samo onda ako je i (PM, m_0) živa, sigurna i ograničena [GIR03].

Petrijeva mreža nakon redukcije i dekompozicije zadržava svojstva živosti, sigurnosti i ograničenosti primjenom sljedećih transformacija (slika 3.10):

a) spajanje serijskih mjesta,
b) spajanje serijskih prijelaza,
c) spajanje paralelnih mjesta,
d) spajanje paralelnih prijelaza,
e) odstranjivanje mjesta s povratnom petljom,
f) odstranjivanje prijelaza s povratnom petljom.



Slika 3.10 Pravila za redukciju i dekompoziciju mreže.

Moguć je i obrnut put, tj. pretvaranje manjih Petrijevih mreža u veće. Takve se metode koriste pri sintezi Petrijevih mreža ili gradnji modela. Na takav se način od pojednostavljenog modela postupno izgrađuje složeniji model dodavanjem novih mjesta i prijelaza, uz uvažavanje prethodno navedenih pravila.

Analiza Petrijevih mreža pomoću grafa dostupnih ili prekrivnih stanja služi da bi se odredila osnovna svojstva manjih Petrijevih mreža i podobna je za pronalaženje stanja potpunih zastoja mreže. Međutim, za modele stvarnih sustava, koji su jako složeni, ove tehnike su prezahtjevne, osobito za analizu problema u svezi dostupnosti i živosti. Analiza Petrijevih mreža pomoću redukcije i dekompozicije služi da bi se velika Petrijeva mreža mogla pretvoriti u jednu jednostavniju. Kako je nova mreža manja i jednostavnija, moguće je uspješno primijeniti analizu pomoću grafa dostupnih te analizu pomoću linearne algebre.

3.4. Strukturna svojstva Petrijevih mreža

Strukturna svojstva Petrijevih mreža mogu se dobiti analizom grafa Petrijeve mreže, jer nije potrebno poznavati broj oznaka u pojedinim mjestima mreže, već samo način na koji su povezana pojedina mjesta i prijelazi. U grafu se mogu pronaći značajni skupovi mjesta i prijelaza, te se po tome može zaključiti kakva će biti osnovna svojstva Petrijeve mreže.

3.4.1. Postojanosti u Petrijevoj mreži

Kako su *postojanosti* (engl. *invariants*) u Petrijevoj mreži jedna od strukturnih svojstava Petrijevih mreža, predstavljaju važan alat u analizi Petrijevih mreža. Štoviše, analiza se može izvesti na bilo kojoj lokalnoj podmreži, bez razmatranja cijelog sustava. Isto tako, postojanost se može koristiti za provjeru valjanosti modela.

U teoriji Petrijeve mreže razlikuju se dvije vrste postojanosti: *P*-postojanost i *T*-postojanost.

_P_-postojanost

P-postojanost je n-dimenzinalni pozitivni cjelobrojni vektor x, gdje n predstavlja ukupni broj mjesta u Petrijevoj mreži. *I-ti* element vektora x je težinski faktor *i-tog* mjesta, za kojeg vrijedi da je broj oznaka mjesta pomnožen s određenim težinskim faktorom uvijek konstantan i neovisan o stanju u kojemu se mreža nalazi.

Štoviše, izrečena definicija za *P*-postojanost ima formalni oblik, iz kojeg se može izvesti algoritam za njeno određivanje.

Definicija 3.18

P-postojanost Petrijeve mreže $PM = (P,T,I,O)$, s n mjesta i matricom događanja A, je n-dimenzionalni pozitivan cjelobrojni vektor $x \in N_0^n$, (N_0 - skup pozitivnih cijelih brojeva), koji zadovoljava relaciju

$$\mathrm{A}^{\mathrm{T}} \cdot x = 0 . \tag{3-21}$$

Množeći jednadžbu (3-20) s x^{T} i primjenjujući jednadžbu (3-21) kako bi se eliminirao izraz $x^{\mathrm{T}}\mathrm{Au}$, dobije se sljedeća relacija:

$$x^{\mathrm{T}} \cdot m_{k+1} = x^{\mathrm{T}}\left(m_k + \mathrm{Au}\right) = x^{\mathrm{T}} \cdot m_k . \tag{3-22}$$

Izrazom (3-22) uspostavljena je veza između (3-21) i ispitivanog svojstva, koja se iskazuje teoremom:

Teorem 3.1

Vektor $x \in N_0^n$ je P-postojanost Petrijeve mreže (PM, m_0), ako i samo ako vrijedi

$$\forall m_0, m_k \in R(m_0), \qquad x^{\mathrm{T}} \cdot m_k = x^{\mathrm{T}} \cdot m_0 . \tag{3-23}$$

gdje je: m_k - neko od dostupnih stanja iz skupa $R(m_0)$,

m_0 - početno stanje mreže.

Iz relacije (3-22) vidi se da je zbroj oznaka svih mjesta koje pripadaju P-postojanosti uvijek konstantan za bilo koje dostupno stanje $R(m_0)$, uključivo i početno stanje m_0 Petrijeve mreže. Drugim riječima, promjene stanja u mreži ne utječu na broj oznaka u P-postojanosti.

Skup mjesta koji odgovaraju isključivo pozitivnim elementima P-postojanosti $\mathbf{x} \geq 0$, naziva se *podrška* (engl. *support*) P-postojanosti $\mathbf{x}$ i označava se $\|\mathbf{x}\|$. P–postojanost je *minimalna* onda i samo onda ako ne postoji P-postojanost $\mathbf{z} \neq \mathbf{x}$ za koju vrijedi $\|\mathbf{z}\| \subseteq \|\mathbf{x}\|$.

Definicija 3.19 *T*-postojanost

T-postojanost $\mathbf{y}$ je vektor čiji elementi označavaju broj okidanja pojedinih prijelaza mreže potrebnih da bi se mreža vratila u početno stanje.

T-postojanost je n- dimenzionalni pozitivni cjelobrojni vektor $\mathbf{y}$, gdje n predstavlja ukupni broj prijelaza u Petrijevoj mreži. T-postojanost mreže može se pronaći ako se nađe pozitivno cjelobrojno rješenje jednadžbe:

$$\mathrm{A}\mathbf{y} = 0. \qquad (3\text{-}24)$$

Pozitivni elementi T-postojanosti označavaju koliko puta se mora okinuti odgovarajući prijelaz, koji pripada slijedu prijelaza koji početno stanje m_0 ponovo vraća u stanje m_0.

Skup $\|\mathbf{y}\| = \{t \in T : \mathbf{y}(t) > 0\}$ je *podrška* T-postojanosti $\mathbf{y}$. Podrška predstavlja skup prijelaza koja odgovaraju pozitivnim elementima vektora $\mathbf{y}$. T–postojanost je *minimalna* onda i samo onda ako ne postoji T-postojanost $v \neq y$ za koju vrijedi $\|v\| \subseteq \|y\|$.

Ako postoji T-postojanost mreže i samo neki elementi T-postojanosti su jednaki 0, tada je mreža djelomično reverzibilna, jer samo neki od prijelaza sudjeluju u nizu okidanja σ koji mrežu vraćaju u početno stanje. Ukoliko su svi elementi T-postojanosti veći od 0, tada je mreža *reverzibilna*, jer svi prijelazi sudjeluju u nizu okidanja koji mrežu vraćaju u početno stanje.

3.4.2. Zastoji i zamke u Petrijevoj mreži

Kada se Petrijeva mreža nalazi u takvom stanju da neki prijelazi nikada više ne mogu okinuti, kažemo da je došlo do *potpunog zastoja* (engl. *deadlock*). Zastoj je veoma opasan kada se dogodi prije nego je sustav došao do ciljanog stanja. Ako se to dogodi, daljnji tijek operacija je onemogućen, sustav je izvan kontrole i operator ne može intervenirati. U nastavku bit će opisani sifoni i zamke [PET81, MUR89, BOG06, KEZ04], strukture Petrijeve mreže koje su važne za analizu osnovnih svojstava Petrijeve mreže, kao što su živost i potpuni zastoj.

Definicija 3.20 Sifoni

Sifon (engl. *siphon*) S je podskup mjesta Petrijeve mreže za koji vrijedi da svaki prijelaz, koji izlazi iz nekog mjesta u S, ujedno ulazi u neko od mjesta u S. Jednom kada se iz skupa S odstrane oznake tijekom okidanja prijelaza, skup trajno ostaje bez oznaka (prazan). Nakon što je sifon ostao prazan, neovisno o daljnjem tijeku okidanja u Petrijevoj mreži, sifon nikada više ne može dobiti niti jednu oznaku [PET81].

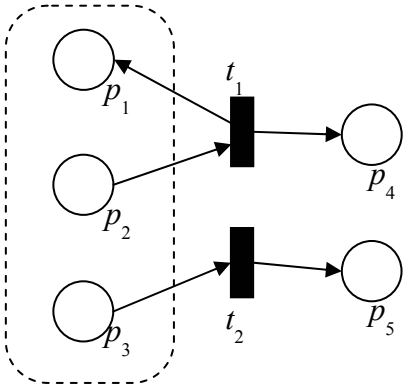
Formalno, neprazan skup mjesta S je sifon onda i samo onda ako vrijedi:

$$\bullet S \subseteq S \bullet, \tag{3-25}$$

gdje je:

$\bullet S$ - skup prijelaza iz kojih idu lukovi prema sifonu,

$S \bullet$ - skup prijelaza u koje idu lukovi iz sifona.



Slika 3.11 Primjer zastoja $S = \{p_1, p_2, p_3\}$, $\bullet S = \{t_1\}$, $S \bullet = \{t_1, t_2\}$.

Definicija 3.21 Zamka

Zamka (engl. *trap*) je skup mjesta za koje vrijedi da jednom kada dobije oznaku, ne mogu je više izgubiti. Oznaka ostaje trajno "zarobljena" unutar zamke [PET81].

Skup mjesta Q je zamka onda i samo onda ako vrijedi:

$$Q \bullet \subseteq \bullet Q. \tag{3-26}$$

gdje je:

$\bullet Q$ - skup prijelaza iz kojih idu lukovi prema zamki,

$Q \bullet$ - skup prijelaza u koje idu lukovi iz zamke.

Unija sifona (zamki) je također sifon (zamka). *Bazni sifon* (*bazna zamka*) je sifon (zamka) koji ne nastaje unijom drugih sifona (zamki). Stoga, svi zastoji (zamke) u Petrijevoj mreži, koji nisu bazni, mogu nastati unijom nekih baznih sifona (zamki).

Minimalni sifon (*minimalna zamka*) je sifon (zamka) koji u sebi ne sadrži neki drugi sifon (zamku). Svi minimalni sifoni (zamke) su ujedno i bazni sifoni (bazne zamke), dok svi bazni sifoni (zamke) ne moraju biti minimalni.

Nužan i dovoljan uvjet za živost većine klasa Petrijevih mreža jest da svi sifoni S u Petrijevoj mreži sadrže inicijalno označenu zamku (Commoner, 1972 [KEZ04]). Ukoliko svi sifoni nemaju inicijalno označenu zamku, što je redovito i slučaj, tada postoji vjerojatnost da se pojedini sifoni isprazne. To može izazvati stanje potpunog zastoja.

Iz tog razloga, za sprječavanje potpunog zastoja, nužno je prvo pronaći sve sifone u mreži, što će se razmatrati u slijedećoj točki. Nakon toga potrebno je osigurati da sifoni ostanu označeni za sva dostupna stanja $R(m_0)$. Tada Petrijeva mreža nikada ne može doći u stanje potpunog zastoja.

Do danas su objavljeni brojni algoritmi za traženje sifona u Petrijevim mrežama. Neki algoritmi se temelje na linearnim nejednadžbama, dok drugi koriste logička pravila ili algebarske jednadžbe [BOG06]. Ipak, nijedna od ovih metoda se ne može izravno primijeniti na sve klase Petrijevih mreža. Više informacija o algoritmima za traženje sifona može se pronaći u [BOG06] i [KEZ04].

3.5. Vremenske i obojene Petrijeve mreže

Opće Petrijeve mreže nisu pogodne za modeliranje mnogih sustava koji se mogu pronaći u proizvodnji, u komunikacijskim, fleksibilnim proizvodnim sustavima i informacijskim procesima. Petrijeve mreže, opisujući realne sustave, mogu biti jako složene i izrazito velike. Ponekad je i nemoguće ispravno modelirati ponašanje sustava. Kako bi se riješili ovi problemi mnogi autori predlažu proširenja osnovnog modela Petrijeve mreže.

Dvije su osnovne vrste proširenja: (i) proširenja koja povećavaju snagu modeliranja i (ii) proširenja koja korisniku samo olakšavaju izradu sažetijih i upravljivih modela. Primjeri proširenja koja ne povećavaju snagu modela Petrijeve mreže su višestruke strelice i mjesta s ograničenjima na kapacitet [MUR89]. S druge strane, postoje proširenja, primjerice s prioritetima [PET81], koja povećavaju snagu modeliranja.

3.5.1. Dodavanje boje

Obojene Petrijeve mreže (engl. C*oloured Petri nets*, pokrata: CPM) razvio je Kurt Jensen 1981. Ova proširenja općih Petrijevih mreža nastala su iz želje da se razvije jezik za modeliranje koji će biti, istovremeno, teorijski dobro utemeljen i koji će se moći u praksi primijeniti na opsežne i složene sustave na koje se nailazi u industrijskim projektima. CPM su nastale kombinacijom snage Petrijevih mreža sa snagom programskih jezika. Dok se Petrijevim mrežama može opisati sinkronizacija konkurentnih procesa, programskim jezicima moguće je opisati tipove podataka i manipulirati njihovim vrijednostima.

Kod CPM svakom mjestu je pridružen *tip* (engl. *type*), čime je određen tip podataka koje mjesto može sadržavati. Tijekom izvođenja CPM svako mjesto će sadržavati različiti broj oznaka. Svaka oznaka donosi vrijednost podatka koja pripada onom tipu koji je pridružen mjestu. Uobičajeno je da se tipove podataka pridruženih mjestu naziva *skup boja* (engl. *colour set*), a vrijednost oznake *boja oznake* (engl. *token colour*). [2] Sve oznake u pojedinom mjestu moraju pripadati istom skupu boja. Ovo je metaforička slika Petrijeve mreže gdje se oznake međusobno razlikuju i zato su "obojane", što je suprotno općim Petrijevim mrežama koje sadrže sve iste "crne" oznake.

Kod CPM tipovi podataka mogu biti složeni, primjerice zapis gdje je jedno od polja realno, drugo znakovni niz, dok je treći lista cijelih brojeva. Djelovanje CPM prikazano je prijelazima. Okidanjem prijelaza, oznake se pomiču s ulaznog mjesta i dodaju se izlaznim mjestima na isti način kao kod običnih Petrijevih mreža. Jedina razlika je u tome što je definirana funkcijska ovisnost između boje okinutog prijelaza i boja oznaka koje su uključene. Ulazni i izlazni lukovi prijelaza su označeni formalnim izrazima koji označavaju uvjete prijelaza. Kod ulaznih lukova formalni izraz određuje kombinacije boja oznaka ulaznih

[2] Analogija s višim programskim jezicima: boja oznake je vrijednost varijable, skup boja oznake je tip varijable.

mjesta koje se pomiču kada se prijelazi okinu, a kod izlaznih lukova definiraju boju novih oznaka koje se dodaju izlaznim mjestima. Dodavanjem boje svakoj oznaci i dodavanjem skupa boja svakom mjestu CPM, uporabom boja, vode ka sažetijem mrežnom modelu. Ova značajka je prikazana primjerom 3.6.

Primjer 3.6 Proizvodni sustav

U ovom primjeru razmatra se proizvodni sustav koji se sastoji od dva stroja M1 i M2, koji obrađuju tri vrste sirovina j1, j2, j3. Svaka vrsta sirovine prolazi obradu koja se može izvesti ili na stroju M1 ili M2. Nakon što je obrađena, sirovina se uklanja iz sustava, a u sustav se dovodi nova neobrađena sirovina istog tipa. Na slici 3.12 prikazan je model ovog sustava (neobojenom) Petrijevom mrežom [WAN07]. Mjesta i prijelazi u modelu definirani su kako slijedi:

p1 (p2): Stroj M1 (M2) je raspoloživ.

p3 (p4,p5): Sirovina tipa 1 (tipa 2, tipa3) je raspoloživa.

p6 (p7,p8): Stroj M1 obrađuje sirovinu tipa 1 (tipa 2, tipa3).

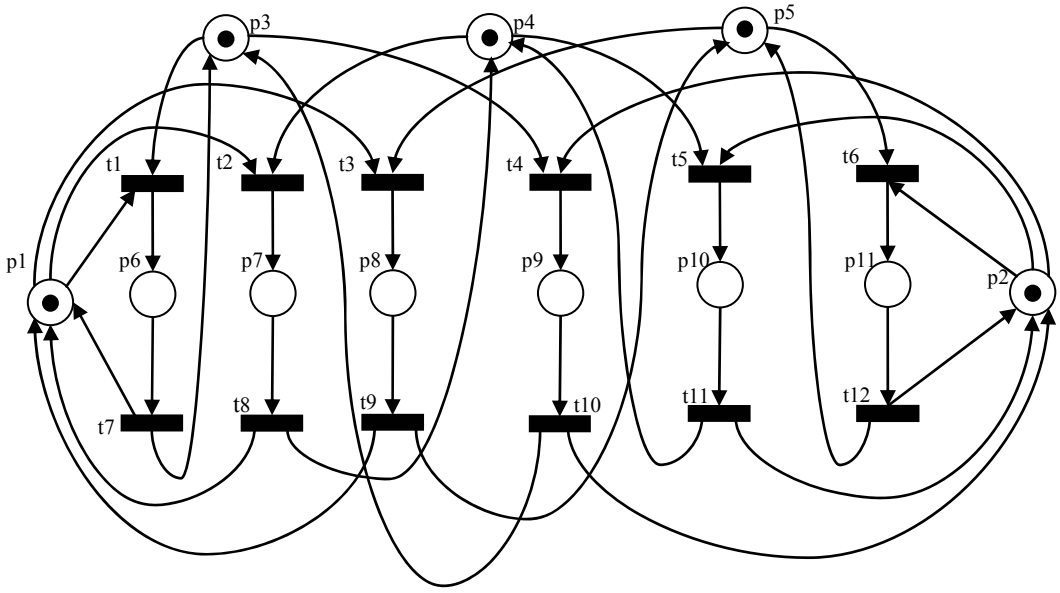
p9 (p10,p11): Stroj M2 obrađuje sirovinu tipa 1 (tipa 2, tipa3).

t1(t2,t3): M1 započinje obradu sirovine tipa 1 (tipa 2, tipa3).

t4(t5,t6): M2 započinje obradu sirovine tipa 1 (tipa 2, tipa3).

t7(t8,t9): M1 završava obradu sirovine tipa 1 (tipa 2, tipa3).

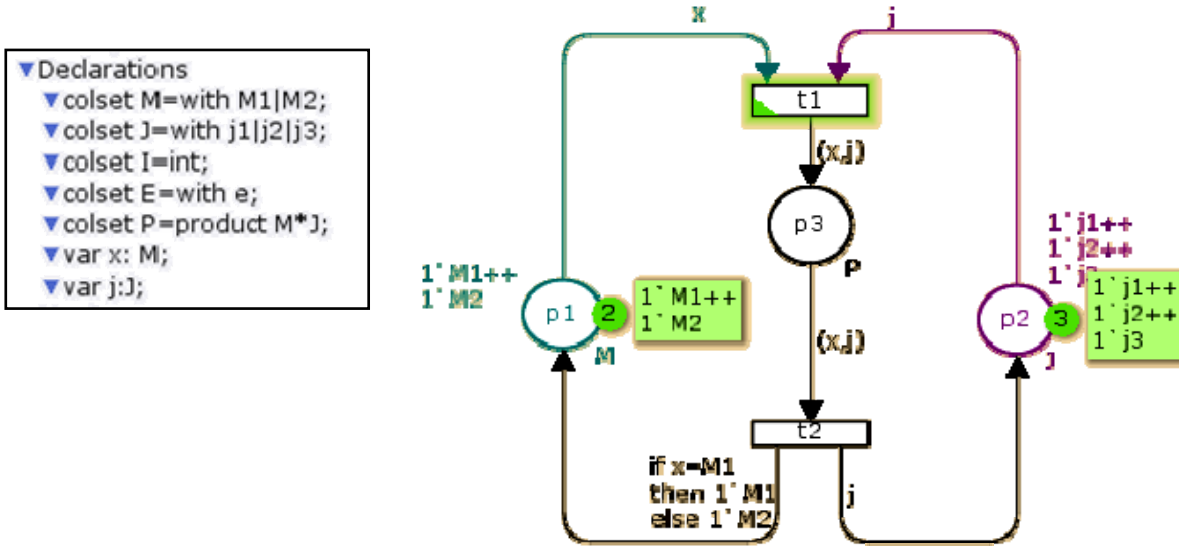
t10(t11,t12): M2 završava obradu sirovine tipa 1 (tipa 2, tipa3).



Slika 3.12 Petrijeva mreža jednostavnog proizvodnog sustava

Na slici 3.13 prikazan je model ovog sustava obojenom Petrijevom mrežom. Može se uočiti da kod modela s obojenom Petrijevom mrežom postoje tri mjesta i dva prijelaza u usporedbi na 11 mjesta i 12 prijelaza kod modela na slici 3.13. Značenje mjesta i prijelaza je zadano kako slijedi:

p1: strojevi su raspoloživi,
p2: sirovine su raspoložive,
p3: obrada se izvodi,
t1: početak obrade,
t2: kraj obrade.



Slika 3.13 CPM model proizvodnog sustava

U modelu se koriste tri skupa boja: M, J i P=M×J, pri čemu je $M=\{M1,M2\}$, $J=\{j1,j2,j3\}$. Boja svakog čvora je dana kako slijedi:

$$C(p1)=\{M1,M2\}$$
$$C(p2)=\{j1,j2,j3\}$$
$$C(p3)=M\times J$$
$$C(t1)=C(t2)=M\times J$$

■

CPM mogu se analizirati pomoću analize grafa dostupnih stanja. Međutim, takav graf može biti jako velik, čak i za male CPM. Druga mogućnost analize CPM je simulacija. Više informacija o CPM, metodama analize i praktičnoj primjeni CPM može se pronaći u [JEN97], [JEN98], [KRI98].

3.5.2. Dodavanje vremena

Dinamički modeli kontinuirano uzimaju u račun podatke koji se s vremenom mijenjaju. Moguće je u model Petrijeve mreže ugraditi vrijeme i na taj način je jednostavno modelirati evoluciju ponašanja sustava. Uvođenjem vremena u Petrijevu mrežu, potrebno je određenim operacijama u mreži pridružiti vrijeme trajanja (kašnjenje). Kada Petrijeva mreža sadrži vremensku varijablu, postaje *vremenska Petrijeva mreža* (engl. *Timed Petri net,* pokrata: TPN) [WAN98].

Definicija vremenske Petrijeve mreže sadrži detaljno navođenje:

- topološke strukture,
- označavanje strukture i
- pravila okidanja.

Topološka struktura vremenske Petrijeve mreže je ista kao i kod općih Petrijevih mreža. Označavanje vremenske Petrijeve mreže je pridruživanje brojčanih vrijednosti jednom ili više niže navedenih elemenata Petrijeve mreže:

- prijelazi,
- mjesta,
- lukovi koji povezuju mjesta i prijelaze.

Pravila okidanja se različito definiraju ovisno o načinu kako je Petrijeva mreža označena vremenskom varijablom. Pravila okidanja definirana za TPN kontroliraju proces micanja oznaka u mreži.

Osim po načinu označavanja, vremenske Petrijeve mreže mogu se podijeliti i po vrsti kašnjenja [WAN07]: (i) deterministčke vremenske Petrijeve mreže (engl. *Deterministic Timed Petri nets,* pokrata: DTPNs) i (ii) stohastičke vremenske Petrijeve mreže (engl. *Stochastic Timed Petri nets,* pokrata: STPNs).

Deterministčke vremenske Petrijeve mreže prvi je uveo Ramchandani, 1974 [WAN07]. Kod DTPN mreža vrijeme se uvodi na jedan od dva načina: vrijeme se dodaje mjestima (engl. *Place Timed Petri Nets,* pokrata: PTPN) ili se vrijeme dodaje prijelazima (engl. *Transition Timed Petri Nets,* pokrata: TTPN).

DTPN je šestorka [WAN98]:

$$DTPN = \left(P, T, A, w, m_0, \tau\right), \tag{3-27}$$

gdje je

$\left(P,T,A,w,m_0\right)$ - jednako definirani kao kod Petrijeve mreže (3-5).

$\tau: P(\text{ili } T) \to R^+$ - funkcija kašnjenja koja mjestima ili prijelazima pridružuje deterministčka (fiksna) vremena.

Drugi način je prirodniji jer prijelazi obično predstavljaju operacije u sustavu i operacije se izvode određeno vrijeme. Kod ovog pristupa vrijeme se razmatra na sljedeći način: oznake se pomiču iz ulaznih mjesta onda kada se okine prijelaz, ali da bi se nove oznake uvele u izlazna mjesta potrebno je da prođe određeno vrijeme u sustavu.

Kada se vrijeme pridružuje mjestima u Petrijevoj mreži, pri okidanju prijelaza nema odgode, ali oznaka koja se treba uvesti u izlazno mjesto mora sačekati određeno vrijeme prije nego se iskoristi kao ulaz za slijedeći prijelaz. Na ovaj način je vrijeme kašnjenja pridruženo mjestima i mjesta s vremenom kašnjenja različitim od nule se označavaju kao *vremenska mjesta*.

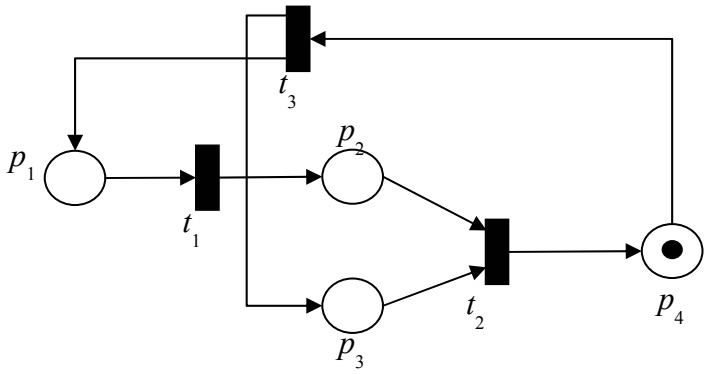
U [WAN98] je pokazano da su oba modela vremenskih PM ekvivalentna obzirom na snagu modeliranja. DTPN su primjenjive na posebnu skupinu sustava, koji se nazivaju mreže bez odluka (engl. *decision-free nets*). U strukturnom smislu, to znači da svako mjesto Petrijeve mreže mora imati najviše jedan ulazni luk i najviše jedan izlazni luk [WAN98].

Primjer 3.7

U ovom primjeru razmatrat će se PTPN prikazana na slici 3.14 sa sljedećim vremenima kašnjenja koja su pridružena mjestima:

$$\tau_1 = 1, \tau_2 = 5, \tau_3 = 1, \tau_4 = 3.$$

Početno stanje mreže je $m_0 = [0\ 0\ 0\ 1]$. Oznaka u mjestu p_4 je raspoloživa tek nakon vremenskog perioda od 3 sekunde. Nakon tog vremenskog perioda, prijelaz t_3 će se okinuti i mreža prelazi u novo stanje $m_1 = [1\ 0\ 1\ 0]$. Sada mjesto p_1 ima kašnjenje od 1 sekunde i kad prođe to vrijeme, novo stanje mreže je $m_2 = [0\ 1\ 1\ 0]$. Iako, sada mjesto p_3 sadrži oznaku, prijelaz t_2 se još neće okinuti, jer još uvijek oznaka u mjestu p_2 nije raspoloživa za okidanje. Pet sekundi nakon što je oznaka uvedena u mjesto p_2 moguće je okinuti prijelaz t_2 i mreža prelazi u novo stanje $m_3 = [0\ 0\ 0\ 1]$. Proces se nastavlja.



Slika 3.14 PTPN mreža za primjer 3.10

■

Stohastičke vremenske Petrijeve mreže su mreže kod kojih je vrijeme kašnjenja opisano probabilističkom razdiobom podataka. Kod STPN vrijeme kašnjenja se pridružuje prijelazima i obično je nekom razdiobom određeno vrijeme kada je moguće okinuti prijelaz. Odabir takve razdiobe je često težak i podložan greškama.

3.6. Sažetak poglavlja

U ovom poglavlju je definirana i detaljno opisana struktura, ponašanje, svojstva i metode analize općih Petrijevih mreža. Petrijeve mreže su grafički alat za modeliranje sustava, koji se temelji na jakom matematičkom formalizmu kojim je moguće modelirati, analizirati i provjeriti dinamičko ponašanje promatranog sustava. Mogu se primijeniti na različite aplikacijske domene, omogućuju modeliranje različitih realnih sustava, opisivanje konfliktnih i konkurentnih operacija. Pogodne su za opisivanje i analizu sustava čije su značajke asinkronost, distribuiranost, paralelizam i nedeterminiranost. Ono što ih čini posebno pogodnim alatom je istovremeno opisivanje mjesta (stanja) i prijelaza (događaja), što znači da je moguć istovremeni uvid i u uvjete za izvođenjem neke operacije i u njenu učinkovitost. Također, ukratko su opisane i neke od Petrijevih mreža više razine, kao što su obojene PM i vremenske PM.

Petrijeve mreže imaju i određene nedostatke [OSK99].

1) Čak ako su mjesta i prijelazi označeni nazivom odgovarajućih stanja i događaja, čitatelj mora poznavati osnove Petrijevih mreža kako bi mogao razumjeti model.
2) Petrijeve mreže mogu biti grafički jako nepregledne za male primjere. Postoji nekoliko prijedloga za hijerarhijsku kompoziciju Petrijevih mreža kako bi se djelomično savladao ovaj nedostatak, ali takve tehnike se uglavnom koriste u konstrukcijskoj fazi modeliranja.

4. GENETSKI ALGORITMI

Genetski algoritmi su adaptivne metode pretraživanja koji se zasnivaju na postavkama i mehanizmima prirodne selekcije i preživljavanja jedinki koje su najbolje prilagođene uvjetima koji vladaju u okolini. "*Genetski algoritmi su algoritmi pretraživanja koji se zasnivaju na mehanizmima prirodne selekcije i prirodne genetike*" [GOL89].

Genetske algoritme je predložio i razvio Johna H. Hollanda (1975). Cilj njegovog istraživanja je bio formalno proučavanje i točno objašnjenje fenomena prirodnog prilagođavanja jedinki u prirodi i razvijanje softvera koji bi sadržavao najvažnije mehanizme prirodnih sustava.

Dokazano je da su genetski algoritmi učinkovite tehnike pretraživanja za rješavanje klase problema koji su NP-kompletni[3]. "*U teoriji složenosti, klasa NP označava skup svih problema koji su rješivi u ne-deterministčkom polinomijalnom vremenu. Ovi problemi se smatraju "teški"' u smislu da nisu rješivi u deterministčkom polinomijalnom vremenu.*" (De Jong and Spears, 1989).

Tijekom nešto više od dva desetljeća, a posebno u posljednjih nekoliko godina, genetski algoritmi pokazali su se moćnim i u isto vrijeme općenitim alatom za rješavanje čitavog niza optimalizacijskih problema. Pogodni su za rješavanje problema koji se mogu opisati nekontinuiranom ili višekriterijskom funkcijom cilja. To se može objasniti njihovom jednostavnošću te doprinosu niza znanstvenika i inženjera koji su ih prilagodili velikom broju problema i povećali njihovu učinkovitost. Kako ne postavljaju zahtjeve za računanje derivacija funkcije cilja i ne zahtijevaju posebna znanja iz domene optimalizacijskog problema, nije potrebno algoritam prilagođavati pojedinoj funkciji. Na taj način je postignuta visoka robusnost algoritma.

Ovo poglavlje predstavlja pregled genetskog algoritma kao metode optimalizacije. U prvom podpoglavlju bit će općenito opisani evolucijski algoritmi. Drugi dio razmatra osnove jednostavnog genetskog algoritma i teoretsku podlogu kojom se objašnjava mogućnost upotrebe genetskog algoritma kao optimizacijske tehnike. U trećem podpoglavlju bit će opisana uporaba genetskog algoritma za višekriterijsku optimalizaciju.

4.1. Uvod u genetske algoritme

Mnogi optimalizacijski problemi su, po svojoj prirodi, jako složeni i teški da bi se riješili klasičnim optimalizacijskim metodama, jer je prostor rješenja problema najčešće prevelik da bi računalo moglo pretražiti sva rješenja u nekom razumnom vremenu. Od 60-tih godina 20. stoljeća brojni autori u imitaciji prirodnog evolucijskog procesa traže inspiraciju za osmišljavanjem novih računalnih algoritama. Prirodna evolucija neke vrste se može promatrati kao proces optimalizacije, potraga za jedinkom koja je najbolje prilagođena uvjetima koji vladaju u okolini. Prirodnom selekcijom se biraju jedinke koje će preživjeti i stvoriti potomstvo. Na taj način populacija napreduje i sve se bolje prilagođava okolini. Oponašanjem evolucijskih procesa, koji postoje u prirodi, nastala je grupa stohastičkih

[3] engl. NP-complete

optimalizacijskih metoda koji se nazivaju *evolucijski algoritmi (*engl. *Evolution Algorithms,* pokrata: EA).

Dijele se na četiri najvažnije skupine:

- *genetski algoritmi* (engl. *Genetic algorithm,* pokrata: GA),
- *genetsko programiranje* (engl. *Genetic programming,* pokrata: GP),
- *evolucijske strategije* (engl. *Evolution strategies*, pokrata: ES),
- *evolucijsko programiranje* (engl. *Evolutionary programming*, pokrata: EP).

Posljednjih godina intenzivno se razvijaju hibridni algoritmi nastali "križanjem" evolucijskih algoritama s drugim metodama umjetne inteligencije, kao što su neuronske mreže i neizrazita logika[4], ili s drugim heurističkim metodama kao što je lokalno pretraživanje i simulirano kaljenje (engl. *simulated annealing*).

Iako su razvijeni različiti evolucijski algoritmi, svi oni imaju neke općenite i temeljne značajke (slika 4.1):

(i) Evolucijski algoritmi djeluju na populaciju mogućih rješenja koristeći principe opstanka najboljih. Sve jedinke populacije su predstavljene određenom podatkovnom strukturom (broj, niz, matrica, stablo, itd.), koja predstavlja kodirano moguće zadanog problema, što je analogno genetskoj informaciji živog organizma.

(ii) U cilju vrednovanja prilagođenosti jedinki njihovom okruženju, mjera kvalitete ili dobrota mora biti pridružena svakoj jedinki. U skladu s vrijednostima funkcije dobrote, mehanizmom selekcije se osigurava tako da jedinke s najboljim svojstvima prenesu ta svojstva u sljedeću generaciju.

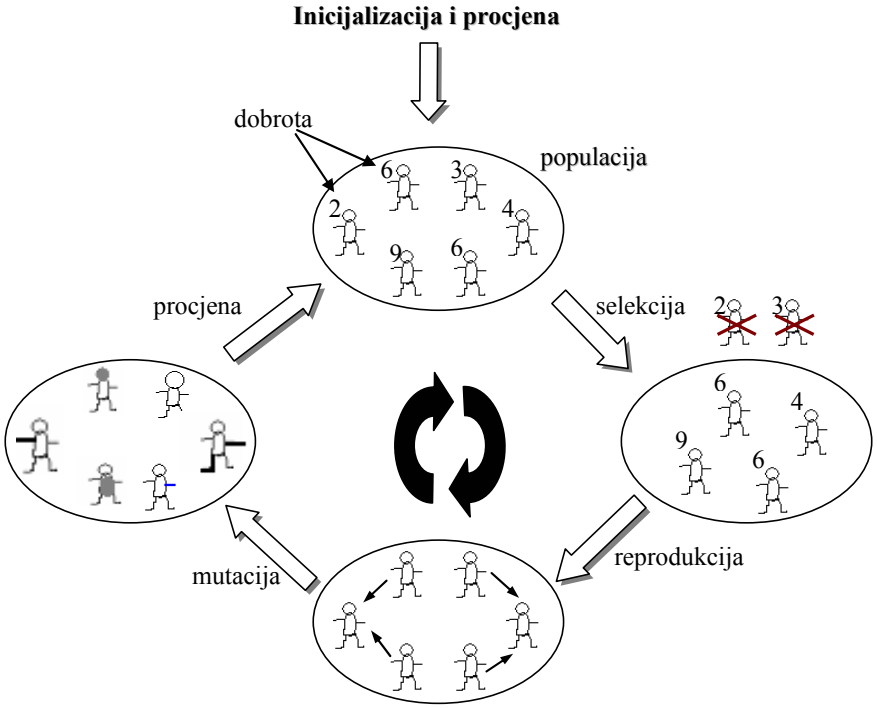
(iii) Potomstvo jedinki se kreira pomoću mutacije i rekombinacije. Mutacija odgovara pogrešnom samopreslikavanju jedinki, dok se rekombinacijom izmjenjuju informacije između dvije ili više jedinki.

Danas su genetski algoritmi najpoznatija vrsta evolucijskih algoritama. Najveći doprinos dali su Holland i Goldberg [GOL89] teoretskom razradom algoritama implementacije. Područje primjene genetskih algoritama uključuje veliki broj znanstvenih i inženjerskih problema. Evo nekih od tih primjera primjene gdje genetski algoritam služi za:

- rješavanje problema rasporeda [FAN94], primjerice planiranje rasporeda poslova (engl. *Job Scheduling*) [BIE90, BIE99, CRO95],
- raspoznavanje uzoraka (npr. izdvajanje slikovnih značajki, analizu uzoraka, satelitskih i rendgenskih slika [GOL94], kod imunoloških sustava),
- sintezu i dizajn (primjerice, računalnih mreža, elektromagnetskih sustava [COE04], zrakoplovnih sustava [CVE00], društvenih sustava, sustava za strojno učenje).

Paralelno s povećanjem primjene povećava se i opseg istraživanja rada i svojstava genetskih algoritama i pokušavaju se svesti njihovi elementi na neke teorijske osnove. Nažalost, rezultati postignuti na teorijskom području su dvojbeni, a genetski algoritmi ostaju i do danas u osnovi heurističke metode.

[4] engl. *Fuzzy Logic*



Slika 4.1 Inicijalizacija i iterativni ciklus evolucijskog procesa

4.1.1. Evolucijski proces

Evolucijski algoritam simulira evolucijski proces na populaciji jedinki u cilju pronalaženja najboljeg mogućeg aproksimacijskog rješenja zadanog problema optimalizacije. Proučavanju evolucije u prirodi najviše je doprinio britanski znanstvenik Charles Darwin u 19. stoljeću. On je u svom djelu "*Porijeklo vrsta*" postavio temelje današnje znanosti o biološkoj evoluciji.

Evolucija je neprekidan proces prilagođavanja živih bića na svoju okolinu, tj. na uvjete u kojima žive. U prirodi vlada nemilosrdna borba za opstanak u kojoj pobjeđuju najbolji, a loši odumiru. Da bi neka vrsta tijekom evolucije opstala, mora se prilagođavati uvjetima i okolini u kojoj živi, jer se i uvjeti i okolina mijenjaju. Primijetivši različitosti među jedinkama iste vrste, Darwin je dokazao da su sva živa bića prilagođeni potomci jednog ili nekoliko predaka. Priroda proizvodi jedinke s različitim osobinama: jedinke nasljeđuju svojstva od svojih roditelja uz poneku varijaciju (mutaciju). Dokazao je i zaključio da se prirodnom selekcijom jedinki reguliraju evolucijske promjene i veličina populacije. Svaka jedinka nastoji prenijeti svoje osobine svojim potomcima. Teorijom prirodne selekcije pretpostavlja se da će jedinke dobiti takva svojstva, kao što su otpornost na razne bolesti, sposobnost trčanja, itd., koja im pomažu da prežive u neprestanoj borbi za opstanak i reproduciraju se u okolini u kojoj žive. Jedinka s najboljim svojstvima imat će i najveću vjerojatnost preživljavanja. Tijekom vremena dobra svojstva će postati dominantna i proširit će se populacijom.

U evolucijskom procesu značajna su dva procesa: prirodna selekcija i spolna reprodukcija. *Selekcija* je proces kojim se vrši odabir jedinki iz populacije; one će preživjeti i dalje se reproducirati. Roditelji koji su se bolje prilagodili vjerojatnije će biti izabrani za produkciju djece. *Spolna reprodukcija* je proces kojim se osigurava da svaka nova jedinka

(dijete) od roditelja naslijedi neka genetska svojstva. Tijekom tog procesa događaju se križanje i mutacija koje pogoduju stvaranju različitosti u genetskom materijalu.

Evolucijski proces je znatno kvalitetniji kada u reprodukciji sudjeluju dva roditelja, nego u slučaju da sudjeluje samo jedan, kao kod nespolnog razmnožavanja. Evolucijski procesi kod organizama koji se razmnožavaju nespolno mogu se vršiti samo mutacijama. Svaka slijedeća generacija neke vrste mora pamtiti *dobra* genetska svojstva prethodne generacije, pronalaziti i mijenjati ta svojstva tako da ostanu dobra u neprekidno novim uvjetima.

Kodiranje u prirodi

Danas se pretpostavlja da su sva svojstva živog bića pohranjena u kromosomima. *Kromosomi* su lančaste tvorevine koje se nalaze u jezgri svake stanice. Djelić kromosoma u kojem je pohranjeno jedno svojstvo, naziva se *gen*. Geni posredno pridonose pogodnosti organizma određujući im osobine koje će im dati određene prednosti u oplodnji. Kromosomi dolaze uvijek u parovima: jedan kromosom je od oca, a drugi od majke. Dakle, za svako svojstvo postoje dva gena ili dvije informacije. Takav par gena koji sadrže informaciju za jedno svojstvo naziva se *alel*. U genetskom paru geni mogu biti ravnopravni ili neravnopravni, tako da je jedan dominantan, a drugi recesivan. U neravnopravnom paru dominantan gen određuje novo svojstvo, dok se uz ravnopravni par gena dobiva svojstvo koje je negdje između svojstava oca i majke.

U evolucijskom krugu teži se i tome da "loši" kromosomi i "loši" geni potpuno nestanu iz okoline, ostavljajući one "bolje" kromosome koji sadrže "bolje" gene koji će proizvesti "bolje" potomstvo. Dakle, jedinke koje imaju loša svojstva, imaju malu vjerojatnost preživljavanja u borbi za opstanak i one će najvjerojatnije odumrijeti, a zajedno s njima i loša svojstva. Dobra svojstva imaju veću vjerojatnost nasljeđivanja, odnosno prenošenja na slijedeću generaciju. Pojmovi "bolji" i "lošš" su relativni obzirom na populaciju.

Govoreći biološkim jezikom, *dobrota* organizma najčešće se mjeri u smislu uspješne oplodnje, a ne u smislu neke populacije. To znači, da pogodnost kromosoma zavisi o svakom aspektu vanjske okoline, koja uključuje pojedine detalje drugih kromosoma u populaciji.

4.1.2. Struktura genetskog algoritma

Osnove genetskih algoritama, odnosno jednostavni genetski algoritam (engl. *Simple Genetic Algorithm*, pokrata: SGA) opisao je Goldberg [GOL89]. Dodatna literatura je dana u [CVE00, GOL01, GOL02].

Gotovo svaki genetski algoritam ima strukturu prikazanu slikom 4.2.

```
Procedura Genetski_algoritam
   {  t = 1;
      kreiraj početnu populaciju potencijalnih rješenja P(0);
      ocijeni P(0);
      sve dok nije zadovoljen uvjet završetka evolucijskog procesa
              // npr. dobrota, broj generacija, vrijeme
             {
                 selektiraj P(t) iz P(t-1);
                 križaj jedinke iz P(t);
                 mutiraj jedinke iz P(t);
                 ocijeni P(t);
                 t = t + 1;   }
       }
```

Slika 4.2 Struktura genetskog algoritma

Genetski algoritam mora sadržavati pet komponenti kako bi se riješio zadani problem:

- predstavljanje kromosoma;
- način kako napraviti početnu populaciju;
- funkciju dobrote;
- reprodukcije;
- vrijednosti parametara.

Kratak opis navedenih komponenti je niže iznesen.

Predstavljanje kromosoma: Svi podaci koji obilježavaju jednu jedinku zapisani su u strukturi koja se naziva kromosom.

Definicija 4.1 Kromosom

Kromosom c je vektor duljine *n*. Formalno,

$$c = (c_i : 1 \le i \le n), \tag{4-1}$$

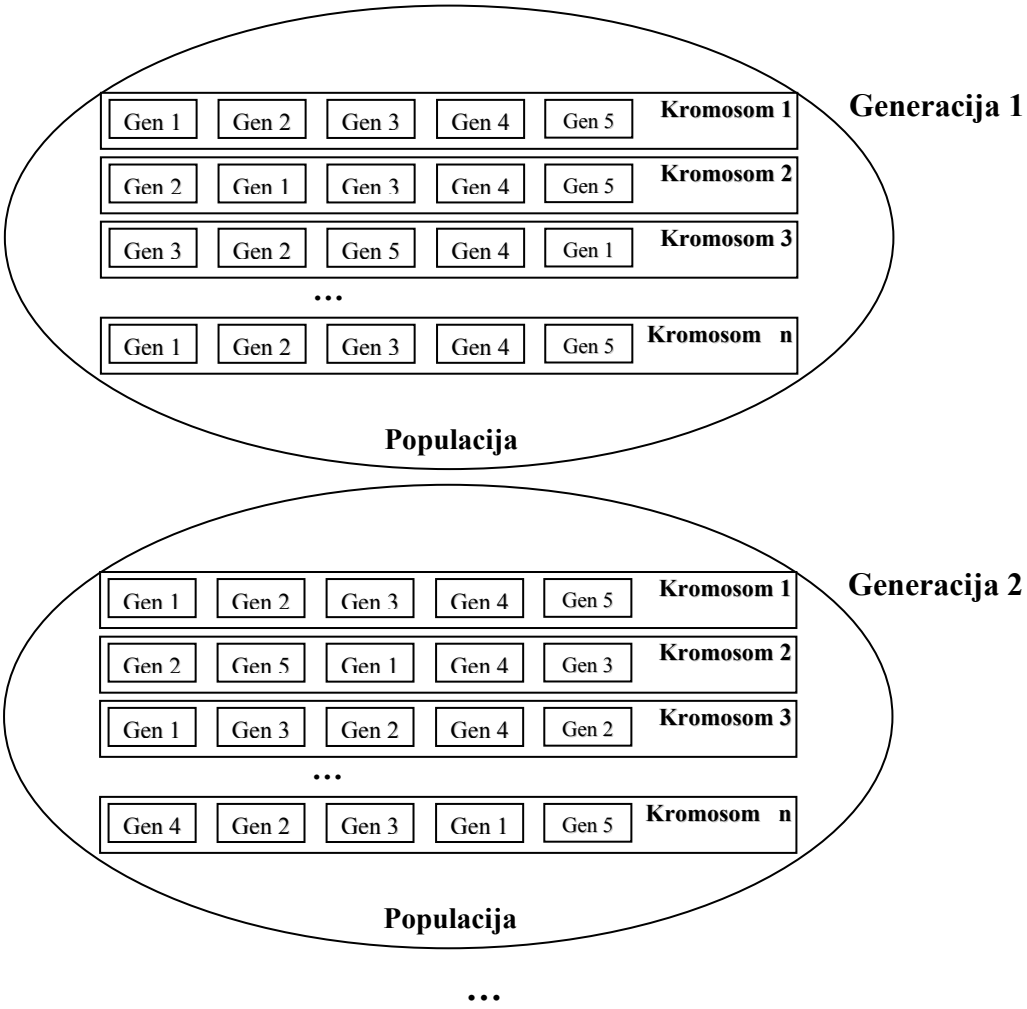
pri čemu se c_i naziva *gen*, a *n* je broj gena. Kromosom je apstrakcija biološkog DNA kromosoma.

Skup kromosoma se naziva *populacija*, a određeni kromosom u populaciji je *jedinka* populacije. Kromosomi se razvijaju kroz niz iteracija koje se zovu *generacije* (slika 4.3). U svakoj iteraciji genetski algoritam izvodi niz računanja nad trenutnom populacijom kako bi se kreirala nova populacija. Iako se jedinke populacije mijenjaju iz generacije u generaciju, veličina populacije i struktura kromosoma je stalna.

Za svaki genetski algoritam od ključne je važnosti kako je kromosom predstavljen. Zato je prvi korak u implementaciji GA određivanje funkcija kodiranja/dekodiranja kromosoma kao parametara problema. *Kodiranje* je postupak pretvorbe kromosoma u niz znakova određene abecede. Abeceda se može sastojati od binarnih znamenaka (0 i 1), cijelih brojeva, brojeva s pomičnim zarezom, simbola (A, B, C, D…), matrica itd. *Dekodiranje* je obrnuti postupak u kojemu niz simbola pretvaramo u parametar.

Ovisno o tome kakvi se simboli i strukture rabe za predstavljanje kromosoma, Gen & Cheng [GEN97] razlikuju sljedeće načini kodiranja:

- binarno kodiranje;
- kodiranje realnim brojevima;
- kodiranje permutiranjem;
- kodiranje općenitim strukturama podataka.



Slika 4.3 Gen, kromosom, populacija, generacija

Problemi mogu biti različite prirode, primjerice problem najkraćeg puta ili problem rasporeda. Moguće rješenje tih problema, zapisano u kromosomu, je *put* ili *raspored*. U prvom slučaju kromosom može biti jednodimenzionalno polje cijelih brojeva pri čemu svaki broj označava redni broj mjesta, a niz brojeva označavaju put od početnog mjesta do cilja. U drugom slučaju kromosom može biti dvodimenzionalno polje u obliku matrice.

Ako je problem višedimenzionalan, umjesto jednog broja koji je zapisan u kromosomu, treba biti polje brojeva veličine dimenzije problema. Općenito, kromosom može biti bilo kakva struktura podataka koja opisuje svojstva jedne jedinke. Za svaku strukturu podataka valja definirati genetske operatore tako da oni ne stvaraju nove jedinke koje

predstavljaju nemoguća rješenja, jer se time znatno umanjuje učinkovitost genetskog algoritma. Prikaz rješenja može bitno utjecati na učinkovitost genetskog algoritma, stoga je izbor prikaza kromosoma izuzetno značajan. Problem predstavljanja kromosoma je predmet mnogih znanstvenih istraživanja.

Inicijalizacija: Pri inicijalizaciji genetskog algoritma kreira se početna populacija jedinki, odnosno početni skup kandidata za rješenje problema. Početna populacija se najčešće odabire slučajno, ali se može i heuristički odabrati. To se mora pažljivo napraviti budući da genetski algoritmi mogu brzo konvergirati k lokalnom optimumu ako početna populacija sadrži jednu ili nekoliko, relativno, dobrih jedinki, ali ne optimalnih, koje postepeno prevladaju u populaciji. Navedena pojava se naziva *preuranjena konvergencija*. Primjerice, ako je problem naći maksimalnu vrijednost funkcije triju varijabli *x, y, z,* onda početni korak može biti kreiranje kolekcije slučajnih trojki (x_i,y_i,z_i), ako je to odabrana reprezentacija.

Funkcija dobrote: *Funkcija cilja* daje numeričku vrijednost kvalitete jedinke unutar domene problema. Funkcija cilja može biti matematički izraz s parametrima dekodiranog kromosoma jedinke, ali može biti rezultat ili niz rezultata dobivenih računalnom simulacijom.

Numerička vrijednost funkcije cilja za svaki kromosom koristi se kao mjera kvalitete, kojom se iskazuje koliko je taj kromosom blizu traženog rješenja. Ta se mjera kvalitete naziva *dobrota* (engl. *fitness*), a funkcija za određivanje dobrote se naziva *funkcija dobrote*.

Definicija 4.2 Funkcija dobrote

Funkcija dobrote ili funkcija *ocjene kvalitete*[5] jedinke je funkcija prilagođenosti i koristi se za transformaciju vrijednosti funkcije cilja u mjeru relativne dobrote. Dva su osnovna zahtjeva transformacije. Prvi je da je vrijednost dobrote nenegativan broj s najvećom vrijednošću danoj najboljem rješenju (jedinki). Drugi je zahtjev da su opseg i raspored vrijednosti dobrote takvi da se osigura ispravan rad odabranog mehanizma selekcije. Opći oblik funkcije dobrote je:

$$d(v) = g(f(v)) \qquad (4\text{-}2)$$

gdje je *g* funkcija koja transformira vrijednost funkcije cilja *f(v)* u skladu s prethodno navedenim zahtjevima.

Funkcija dobrote je pokazatelj koliko dobro se jedinka prilagodila okolini. Kod jednostavnih problema, kao što je pronalaženje minimuma neke funkcije, funkcija dobrote se može prikazati kao vrijednost koju ta funkcija ima za zadani kromosom. Kod složenijih problema dobrota se može izraziti kao, primjerice, vrijeme trajanja projekta, vrijeme koje je potrebno da se obavi simulacija sustava koji se optimalizira i slično. Funkcija dobrote je ključ za proces selekcije.

Selekcija nove populacije: Nova populacija se stvara iz stare populacije tako da se postupkom kojim se imitira prirodna selekcija odabiru jedinke koje će preživjeti u skladu s njihovom funkcijom dobrote.

Reprodukcija: Kako bi se stvorila nova populacija, novi kromosomi, koji se nazivaju "*djeca*", se stvaraju ili

[5]U literaturi se ova funkcija još naziva fitnes funkcija.

(a) kombiniranjem dva stara kromosoma ("*roditelja*") iz trenutne populacije primjenom operatora *križanja* (engl. *crossover*) ili

(b) primjenom unarnog operatora *mutacije* (engl. *mutation*) koji djeluju na genetskom materijalu jednog kromosoma.

Ideja križanja je da se dobra svojstva roditelja kombiniraju tako da proizvode djecu s boljim svojstvima od onih kod roditelja; ako je dijete s lošim svojstvima onda su mu male šanse da će kasnije biti izabran za reprodukciju. Takvim postupkom se iz generacije u generaciju postiže sve veća i veća prosječna dobrota populacije.

Zamjena populacija: Skup jedinki se mijenja izbacivanjem nekih ili svih jedinki iz postojeće generacije (obično počinje s onima koje imaju najmanju dobrotu) i njihovom zamjenom s jedinkama koje su dobivene u koraku reprodukcije. Tako upravo dobivena populacija postaje trenutna generacija.

Cijeli taj postupak (ocjena kvalitete, selekcija i primjena genetskih operatora) se ponavlja sve dok ne istekne vrijeme ili dok se ne zadovolji uvjet zaustavljanja algoritma (npr. funkcija dobrote 95% jedinki odstupa za manje od zadane vrijednosti ε).

Po čemu se genetski algoritmi razlikuju od drugih optimalizacijskih metoda i algoritama pretraživanja?

Obilježja genetskih algoritama po kojima se razlikuju od ostalih metoda optimiranja su [GOL02]:

- Postupak optimiranja je dugotrajan, tj. genetski algoritam troši znatno više procesorskog vremena od drugih optimalizacijskih metoda;
- Operatori djeluju nad populacijom rješenja, a ne nad jednim rješenjem;
- Genetski algoritmi djeluju u domeni kodiranih rješenja;
- Genetski algoritmi koriste stohastička, a ne deterministička načela;
- Informacija se prenosi iz generacije u generaciju;
- Rezultat optimiranja je dobro (gotovo optimalno ili optimalno) rješenje, tj. GA ne pronalazi optimalno rješenje sa stopostotnom sigurnošću;
- GA su jednostavni za implementaciju;
- GA su pogodni za paralelno izvođenje;
- GA su robusni u pogledu dimenzije problema.

Tri su osnovne prednosti genetskih algoritama nad tradicionalnim metodama:

1) Kako GA koriste informaciju dobivenu iz funkcije cilja (neovisno o derivabilnosti, kontinuiranosti i multimodalnosti), a ne bilo koje drugo vanjsko znanje iz domene problema, najveća prednost genetskih algoritama svakako je primjenjivost na velik broj različitih vrsta optimalizacijskih problema. Pomoću njih je moguće rješavati bilo koju funkciju cilja s bilo kojim ograničenjima (linearni ili nelinearni) definiranim nad diskretnim, kontinuiranim ili kombiniranim prostorom pretraživanja. Međutim, u mnogim slučajevima potrebno je uložiti značajan napor u pronalaženje prikladnog načina kodiranja parametara.

2) Klasične metode lokalno pretražuju konvergentno stepenastom procedurom, koja uspoređuje vrijednost susjednih točaka i postepeno se pomiču k relativno optimalnim točkama. GA koriste populaciju točaka u traženje rješenja, a ne jednu točku. Korištenje populacije točaka daje veću mogućnost izbjegavanja lokalnog ekstrema jer se istovremeno pretražuje veliki dio prostora rješenja.
3) GA koriste prelazna pravila zasnovana na vjerojatnosti, a ne deterministička pravila. Elementi izbora bazirani na vjerojatnosti omogućavaju fokusiranje na pretraživanje prostora s najvećom vjerojatnošću poboljšanja rješenja. Istovremeno se osigurava i pretraživanje ostalih područja. Ovo omogućava da se iz koraka u korak dobivaju sve bolja rješenja, ako već nije pronađen globalni optimum.

4.2. Mehanizam jednostavnog genetskog algoritma (SGA)

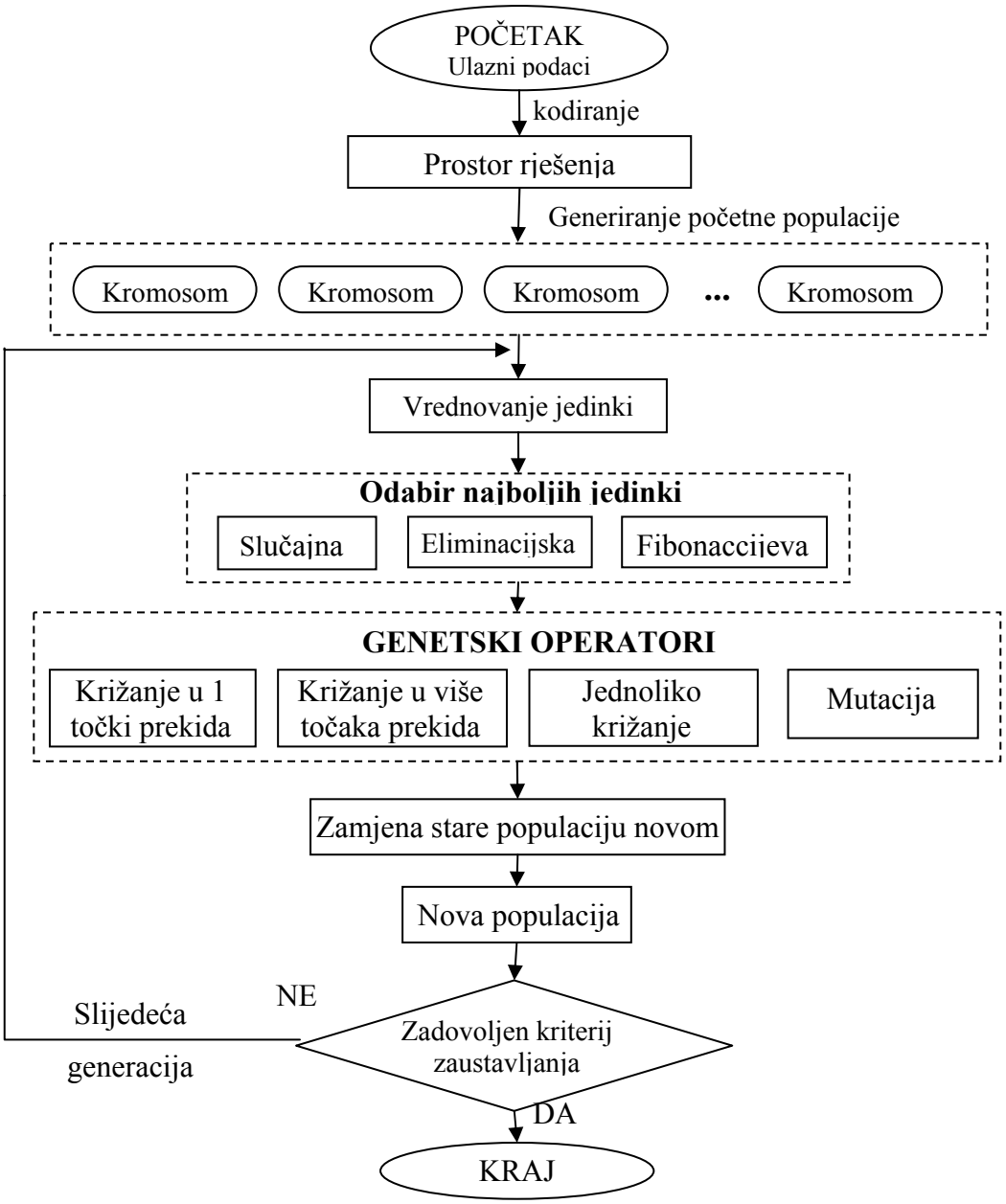
Genetski algoritam se temelji na tri modula, koji su poznati kao *produkcijski* modul, *evaluacijski* modul i *reprodukcijski* modul. Algoritam starta sa skupom mogućih rješenja (početna populacija) i kroz niz ponavljanja (iteracija) zamjenjuje trenutnu populaciju s novom. Kod implementacije GA osnovni je problem kako odabrati način kodiranja i dekodiranja kromosoma, čime se dobivaju ulazni podaci za funkciju dobrote. Reprodukcijski mehanizam bira roditelje i kombinira njihov genetski materijal uporabom operatora križanja kako bi se stvorilo potomstvo, koje se dalje lokalno mijenja operatorom mutacije. Svaki od ovih operatora ima svoje parametre, kao što su vjerojatnost križanja i mutacije, pravilo selekcije i veličina populacije. Ove vrijednosti uvelike utječu na to hoće li algoritam naći približno optimalno rješenje ili će naći točno rješenje. Slika 4.4 prikazuje dijagram toka SGA.

4.2.1. Kodiranje i dekodiranje rješenja

Dva najčešća pristupa predstavljanja kromosoma su izravno i neizravno. Kod izravnog kodiranja, kromosom potpuno predstavlja rješenje. Kod neizravnog kodiranja kromosom sadrži podatke koji se koriste da se dobije rješenje. Niz koji sadrži binarne ili numeričke vrijednosti naziva se *genotip*[6], dok se rješenje, dekodirano iz niza, naziva *fenotip*[7].

Fenotip jedinke je njegova vrijednost u domeni nad kojom se funkcija dobrote definira. Ovo je analogno značajkama kao što su težina, broj članova, duljina staze, itd. Genotip jedinke je *reprezentacija* fenotipa jedinke što je analogno genetskom nizu sadržanom u biološkom kromosomu, kojeg računalo pohranjuje i nad kojim genetski algoritam manipulira. Dakle, fenotip se *kodira* u genotip, obično kao niz bitova ili kao neka druga struktura podataka.

[6] engl. the *genotypic* level
[7] engl. the *phenotypic* level



Slika 4.4 Blok dijagram genetskog algoritma

4.2.1.1. Prikaz rješenja s pomoću prirodnog binarnog koda

Binarno kodiranje je, kao dio klasičnog genetskog algoritma, pokazalo da daje najbolje rezultate za one probleme gdje se rješenje prirodno preslikava u niz nula i jedinica.

Kod klasičnog binarnog kodiranja [BAC97], za prikaz kromosoma koristi se bitovni vektor (slika 4.5), pri čemu broj bitova odgovara broju alela, odnosno broju svojstava koje jedinka može imati. Dužina vektor N označava broj bitova, odnosno broj jedinica ili nula u jednom kromosomu. U takav vektor je moguće zapisati 2^N različitih kombinacija nula i

jedinica, koje odgovaraju jednom od 2^N različitih rješenja prostora pretraživanja. Za računala ovo je prirodni način kodiranja.

Kromosom A	1	0	1	0	0	0	1	0	1	0
Kromosom B	0	0	1	0	1	0	0	1	1	0

Slika 4.5 Binarno prikazani kromosomi

Funkcija dobrote problema koji se rješava može imati jednu ili više varijabli. Ako je to funkcija jedne varijable, onda cijeli binarni niz kromosoma odgovara varijabli rješenja. U slučaju da se radi o funkciji s više varijabli (tablica 4.1) tada se kromosom promatra kao niz sastavljen od podnizova i svaki odgovara jednoj varijabli (slika 4.6).

Tablica 4.1 Primjer binarnog kodiranja: kreiranje kromosoma s četiri varijable [BAC97]

Varijable	Tip varijable	Domena varijable x_i	Broj alela	Veličina podniza
x_1	Kontinuirana	$[0,10]$	$2^{10} = 1024$	10
x_2	Kontinuirana	$[0,10]$	$2^{10} = 1024$	10
x_3	Diskretna	$\{10;12.5,15;17.5\}$	$2^2 = 4$	2
x_4	Cjelobrojna	$\{0;1\}$	$2^1 = 1$	1

0	1	1	0	1	1	1	0	1	0	0	1	0	1	1	0	1	0	0	1	1	0	1

S_1 S_2 S_3 S_4

Slika 4.6 Primjer kromosoma za prikaz jedinke s četiri varijable

■

4.2.1.2. Prikaz Grayevim kodom

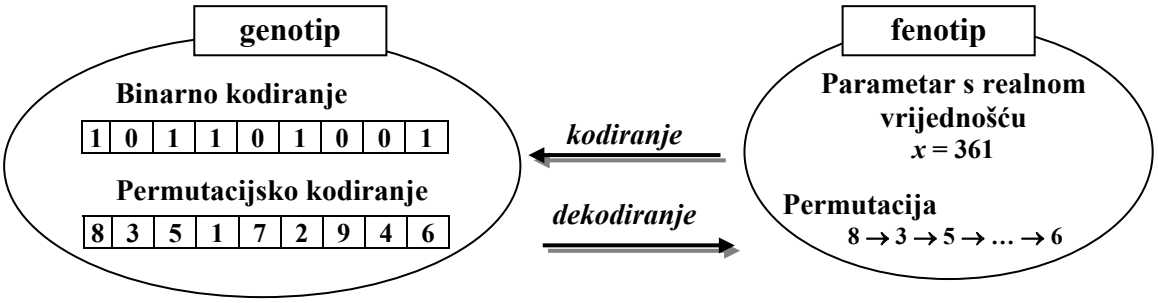
Binarno kodiranje je zbog svoje jednostavnosti pogodno za implementaciju, ali ima i jedan veliki nedostatak. Naime, opće je prihvaćeno da bi kodiranje trebalo što je moguće bolje odražavati ponašanje varijabli. Primjerice, male promjene vrijednosti varijable trebale bi uzrokovati i male promjene u genetskom kodu. Međutim, to nije slučaj kod binarnog kodiranja, gdje novonastali aleli mogu imati potpuno drugačije kromosome. Drugim riječima, ako je genetski algoritam u nekom koraku pronašao jedno *dobro* rješenje za b=011111111_2, a optimum se postiže za b=100000000_2 treba promijeniti svih devet bitova. Kako bi se ispravio taj nedostatak, za kodiranje brojeva koristi se Grayev kod. Gray-ev kod se razlikuje od binarnog koda u tome što se kromosomi predstavljaju kao nizovi nula i jedinica.

Značajke Grayevog koda su:

- Grayev kod predstavlja svaki broj u nizu cijelih brojeva $\{0,1,...,2^n\text{-}1\}$ kao binarni niz duljine n;
- Susjedni cijeli brojevi imaju u Grayevom kodu reprezentaciju koja se razlikuje u samo jednom bitu.

4.2.1.3. Permutacijsko kodiranje

Permutacijsko kodiranje se rabi za sekvencijalne probleme kao što su problemi rasporeda i problemi trgovačkog putnika. Za takve probleme prirodnije je kromosome prikazati permutacijskim nizovima brojeva nego binarnim nizovima. [MUR97] prikazuje primjere nizova i za binarno kodiranje i za permutacijsko kodiranje. Binarni nizovi se najčešće dekodiraju kao parametri s cjelobrojnim ili realnim vrijednostima. Permutacijski nizovi se sastoje od brojeva "1" do "*n*" pri čemu svaki broj odgovara nekom poslu ili zadatku kod problema rasporeda ili odgovara jednom mjestu kod rješavanja problema trgovačkog putnika. "*n*" je ukupan broj poslova ili mjesta. Pri rješavanju problema rasporeda, poslovi se obrađuju u skladu s njihovim redoslijedom u permutaciji ili kad se rješava problem trgovačkog putnika, mjesta se obilaze u skladu s njihovim redoslijedom u permutaciji.



Slika 4.7 Genotipi i fenotipi

4.2.2. Vrednovanje funkcije dobrote

Nakon svake generacije, svako moguće rješenje u populaciji se ocjenjuje kroz funkciju dobrote. Genetski algoritmi traže niz s boljom vrijednosti dobrote među svim genotipima. Potencijalna rješenja koja predstavljaju bolja rješenja problema postižu i veće vrijednosti funkcije dobrote. Naime, funkcija dobrote bi trebala imati najveću vrijednost za optimalno rješenje. Rješenja s manjom dobrotom se brišu iz populacije. Ovaj proces se naziva "*preživljavanje najboljih*".

Za zadani optimizacijski problem najveću poteškoću predstavlja definiranje funkcije dobrote, koja treba vjerno odražavati problem koji se rješava.

4.2.3. Genetski operatori

Pri izradi genetskog algoritma najzanimljiviji i najznačajniji su genetski operatori, koji izgrađuju potencijalno rješenje. Priprema operatora je teška i obično zahtijeva eksperimentiranje, kako bi se odlučilo koju vrstu operatora koristiti za rješavanje određenog problema.

4.2.3.1. Operatori križanja

Križanje je najznačajniji operator u genetskim algoritmima. Operator križanja kombinira elemente roditeljskih kromosoma u jedan ili više novih kromosoma koji se nazivaju *djeca* (engl. *Offsprings*). Budući da dolazi do miješanja genetskog materijala, djeca

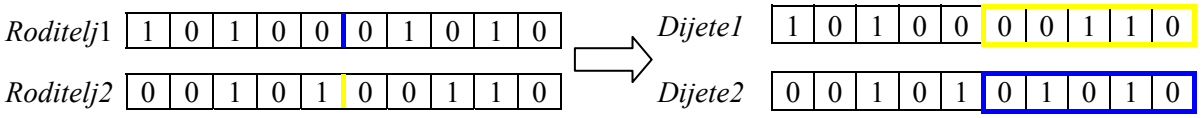
nasljeđuju svojstva svojih roditelja, pa ako su roditelji dobri, velika je vjerojatnost da će i djeca biti dobra, ako ne i bolja od svojih roditelja. Postoji više vrsta operatora križanja, a u ovom poglavlju bit će prikazani neki od njih.

Kod prikaza kromosoma binarnim ili Gray kodom ili kodom s fiksnom točkom, tri vrste operatora križanja su implementirane:

- *Križanje s jednom točkom prekida* (engl. *One-point crossover*)

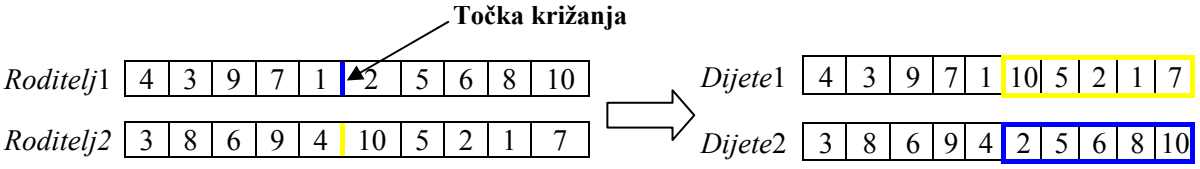
U postupak križanja ulazi se s dvije jedinke izabrane u postupku selekcije i izabire se uniformno slučajno broj k u rasponu od 1 do $Nbit-1$ koji predstavlja mjesto prekida binarnog niza. *Nbit* je broj bitova od kojih se sastoji binarni niz jedinke. Nove dvije jedinke se dobiju na način da se razmijeni genetski materijal oko točke prekida (slika 4.8).

Primjerice, ako je točka prekida k=5:



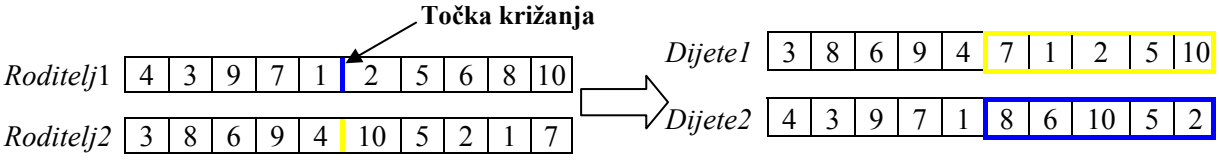
Slika 4.8 Primjer križanje s jednom točkom prekida

Ako se operator križanja u jednoj točki primjeni na permutacijske nizove, operator generira dijete koje ne udovoljava zahtjevu da svaki niz mora biti permutacija svih elemenata. Slika 4.9 prikazuje djecu generiranu standardnim križanjem u jednoj točki. Niti jedno dijete sa slike ne predstavlja ispravnu permutaciju.



Slika 4.9 Primjer križanja u jednoj točki za permutacijske nizove

U kromosomu *Dijete*1, elementi "1" i "7" se dva puta pojavljuju, dok se elementi "6" i "8" ne pojavljuju. Također, kromosom Dijete2 je neispravna permutacija, jer niz ne sadrži sve elemente. Zbog toga je potreban operator križanja koji generira dobre kromosome za probleme permutacija. Jedan od takvih operatora križanja je operator *križanja poretka u jednoj točki* (engl. *One-point order crossover,* pokrata: OX). Ovakvo križanje prikazano je na slici 4.10. Točka križanja se slučajno odabere. Skup elementa iz prvih dijelova kromosoma se međusobno zamijene. Ostali elementi jednog kromosoma, a koji nisu uključeni u prvom dijelu tog kromosoma, se upisuju u redoslijedu kako se pojavljuju u tom nizu. Ovim operatorom, oba djeteta su ispravni nizovi elemenata koji udovoljavaju zahtjevu permutacijskih problema. Na slici 4.10 novi kromosom *Dijete*1 nasljeđuje prvi dio kromosoma *Roditelj*2, a elementi zadnjeg dijela kromosoma *Roditelj*2 su naslijeđeni u zadnji dio kromosoma *Dijete*1, u redoslijedu njihovog pojavljivanja u kromosomu *Roditelj*1.



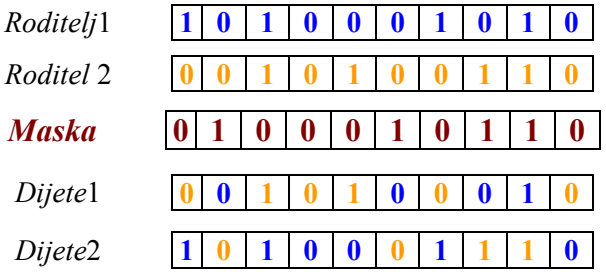
Slika 4.10 Primjer križanja u jednoj točki za permutacijske nizove

- *Križanje s prekidom u više točaka* (engl. *N-point crossover*)

Procedura je slična kao kod križanja s jednom točkom prekida, osim što se bira m točaka prekida k_i iz skupa $[1,2,\ldots,Nbit-1]$, gdje je $Nbit$ broj bitova od kojih se sastoji binarni niz jedinke. Skup točaka prekida je bez dupliciranih točaka i sortiran po veličini. Zatim se međusobno izmijene binarni nizovi između neparne točke prekida i slijedeće točke prekida(ili kraja binarnog niza).

- *Uniformno križanje* (engl. *Uniform crossover*)

Uniformno križanje je generalizacija križanja u više točaka. Prvo se slučajno generira binarni niz iste duljine kao kromosom, koji se naziva *maska križanja*. Zatim se kreiraju dvoje djece i to tako da se svaki gen potomka kreira kopiranjem odgovarajućeg gena jednog od roditelja koji je izabran prema masci križanja (slika 4.11).



Slika 4.11 Primjer uniformnog križanja

Kad je u masci na određenoj poziciji 1 onda se za prvo dijete kopira gen iz prvog roditelja, a kad je 0 kopira se iz drugog roditelja. Za drugo dijete je obrnuto. Za svaki par roditelja generira se nova maska. Prema ovome potomci sadrže mješavinu gena roditelja.

Ako se radi o realnom kodiranju kromosoma, operatori križanja se drukčije definiraju od operatora koji su gore opisani. Ako su $x^{(i)} = \left\{x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}\right\}$ roditelji, postoji nekoliko operatora križanja kojim se mogu dobiti djeca oblika $y^{(i)} = \left\{y_1^{(i)}, y_2^{(i)}, \ldots, y_n^{(i)}\right\}$.

- *Simulirano binarno križanje* (engl. *simulated binary crossover*) predložio je K.Deb 1995, citirano u [CVE00]. Da bi se kreiralo dvoje djece $y^{(1)}$ i $y^{(2)}$ od dvoje roditelja $x^{(1)}$ i $x^{(2)}$, primjenjuje se sljedeća tehnika:
  - Kreirati slučajan broj između 0 i 1;
  - Izračunati sljedeće parametre:

$$\bar{\beta} = \begin{cases} (\alpha u)^{1/(\eta_c+1)} & \text{ako je} \quad u \le \alpha^{-1} \\ \left(\frac{1}{2-\alpha u}\right)^{1/(\eta_c+1)} & \text{inače,} \end{cases} \tag{4-3}$$

gdje je:

$$\alpha = 2 - \beta^{-(\eta_c+1)}, \tag{4-4}$$

$$\beta = 1 + \frac{2}{y^{(2)} - y^{(1)}} min\left[\left(x^{(1)} - x^{l}, \left(x^{u} - x^{(2)}\right)\right)\right] \tag{4-5}$$

U jednadžbama (4-3) – (4-5), pretpostavlja se da je $x^{(1)} < x^{(2)}$ (za $x^{(1)} > x^{(2)}$ jednostavno je promijeniti gore navedene jednadžbe). x^{l} i x^{u} su donja i gornja granica varijable, a η_c je korisnički definiran parametar (uobičajena vrijednost je $\eta_c = 1$).

- Djeca se računaju kako slijedi:

$$y^{(1)} = 0.5 \cdot \left[\left(x^{(2)} + x^{(1)}\right) - \bar{\beta}\left|x^{(2)} - x^{(2)}\right|\right] \tag{4-6}$$

$$y^{(2)} = 0.5 \cdot \left[\left(x^{(2)} + x^{(1)}\right) + \bar{\beta}\left|x^{(2)} - x^{(2)}\right|\right] \tag{4-7}$$

- *Diskretna rekombinacija* (engl. *Discrete recombination*)

U ovom slučaju vrijedi $y_j \in \left\{x_j^{(1)}, x_j^{(2)}\right\}$ za svaki $1 \le j \le n$. Ovo odgovara standardnom uniformnom križanju kod binarnog koda.

- *Intermedijarna rekombinacija* (engl. *Intermediate recombination*)

Dijete se kreira u skladu sa sljedećim pravilom:

$$y_j^{(1)} = x_j^{(1)} + \alpha\left(x_j^{(2)} - x_j^{(1)}\right), \tag{4-8}$$

gdje je

α - uniformno slučajno odabran parametar za kojeg vrijedi $0 \le \alpha \le 1$ i $x_j \le y_j$.

Svaka varijabla u djeteta je rezultat kombiniranja varijabli od roditelja u skladu s izrazom (4-8) s novom odabranom vrijednosti parametra α za svaki par roditeljskih gena.

4.2.3.2. Operator mutacije

Mutacije imaju dvojnu ulogu pri djelovanja genetskih algoritama. S jedne strane brinu se o raznovrsnosti u populaciji (mijenja gene jedinki u populaciji), a s druge strane djeluje kao operator pretraživanja. Mutacija je operator koji obično djeluje na jednom kromosomu i sastoji se od slučajne promjene jednog ili više gena. Takav operator mutacije se može sagledati kao prijelaz iz trenutnog rješenja u njegovo susjedno rješenje u algoritmu lokalnog pretraživanja. Važno je znati da se operatori mutacije za binarno kodirana rješenja razlikuju od onih za permutacijsko kodiranje. U ovom poglavlju bit će opisani operatori mutacije za oba načina kodiranja.

A) Mutacija za binarni niz

Za jedinke koje su binarno kodirane mutacija znači promjenu određenog bita iz 0 u 1 ili obrnuto. Bit jedinke će se promijeniti s vjerojatnošću p_{mut} za koju se uzima vrijednost iz intervala <0, 1>. Ako vjerojatnost mutacije teži ka 1, tada se genetski algoritam pretvara u postupak slučajne pretrage prostora rješenja, a ako vjerojatnost mutacije teži ka nuli, manje se unosi "svježe" genetske informacije u populaciju, tako da postoji velika vjerojatnost da će genetski algoritam stati u nekom lokalnom optimumu. Optimalna vrijednost parametra mutacije treba biti takva da njena vrijednost osigurava pretraživanje određenog područja susjedstva rješenja. Mühlenbein i Schlierkamp – Voosen (1993), citirano u [MUD02] pokazali su da je odabir vrijednosti vjerojatnosti mutacije jednog bita približno:

$$p_{mut} = \frac{1}{N} bita \qquad (4\text{-}9)$$

Na sljedećoj slici prikazan je primjer mutacije potomka gdje je mutacija nastala na četvrtom bitu.

Potomak	0	1	1	1	0	0	1	1	0	0	0
Mutirani potomak	0	1	1	0	0	0	1	1	0	0	0

Slika 4.12. Primjer mutacije jedinke [MUD02]

B) Mutacija za permutacijski niz

Operator mutacije za permutacijski niz trebao bi djelovati tako da kreiraju novi niz elemenata koji udovoljavaju permutacijskim problemima. Najčešće se koriste mutacija promjene položaja gena i mutacija pomaka.

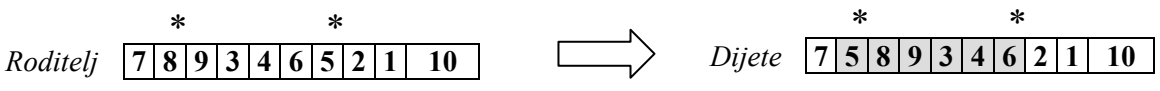
- *Mutacija promjene položaja gena*

Dva slučajno odabrana gena u kromosomu zamjene međusobno mjesta. Slika 4.13 prikazuje primjer ove mutacije.

Slika 4.13 Primjer mutacija zamjenom položaja gena

Mutacija pomaka (*engl*. Shift mutation)

Kod ove mutacije, gen prve pozicije se pomiče na mjesto druge pozicije kako je prikazano na slici 4.14. Dvije pozicije se slučajno odabiru. Gen s druge odabrane pozicije se postavi ispred gena s prve pozicije.



Slika 4.14 Primjer mutacija pomaka gena

4.2.4. Selekcija u GA

U prirodi, svaka jedinka mora biti dovoljno jaka kako bi opstala u okolini te pronašla partnera za reprodukciju i stvaranje potomstva. U genetskom algoritmu, ovaj prirodni mehanizam je simuliran operatorom selekcije. Obzirom na vrijednosti funkcije dobrote, selekcijom se odabiru jedinke koje će sudjelovati u reprodukciji. Jedinke s većom dobrotom imaju veću vjerojatnost da budu odabrane od onih s manjom dobrotom. Na taj način se dobri geni ili dobri genetski materijal sačuvaju i prenose na slijedeću populaciju, a loši odumiru. Kaže se da se mehanizmom selekcije jedinke izabiru u bazen parenja (engl. *mating pool*). Kako su u bazenu za parenje one jedinke čiji geni se nasljeđuju u sljedeću generaciju, poželjno je da bazen za parenje sadržava "dobre" jedinke. Broj pojavljivanja jedinke u bazenu za parenje utječe na mogući broj potomaka iste.

Da bi se mjerio učinak pojedinih metoda selekcije u okviru GA uvedeni su slijedeći pojmovi [GOL89]:

Definicija 4.3 Selekcijski pritisak (engl. *selection pressure*)

Selekcijski pritisak je vjerojatnost izbora najbolje jedinke u usporedbi s prosječnom vjerojatnosti selekcije za sve jedinke. Drugim riječima, selekcijski pritisak je sposobnost preživljavanja jedinki. Neka vjerojatnosti selekcije dvije jedinke s oznakama i i j iznose $p(i)$ i $p(j)$. Neka je jedinka s oznakom i bolja od one s oznakom j, tj. neka je $dobrota(i)>dobrota(j)$, odnosno $p(i)>p(j)$. Selekcijski pritisak je veći što je kvocijent $p(i)/p(j)$ veći ili što je veća razlika $p(i)-p(j)$. Selekcija ima veliki selekcijski pritisak ukoliko s velikom vjerojatnošću prenosi bolje jedinke u iduću iteraciju, odnosno s velikom vjerojatnošću eliminira jedinke ispod prosječne dobrote. Selekcijski pritisak utječe na kvalitetu rješenja. Prema Cantú-Paz (1999) (citirano u [GOL01]), genetski algoritmi s velikim selekcijskim pritiskom pronalaze prije rješenje, tj. brže konvergiraju žrtvujući kvalitetu dobivenog rješenja, jer brža konvergencija uzrokuje veću vjerojatnost zaglavljivanja u lokalnom optimumu. S druge strane, ako je selekcijski pritisak premali, nepotrebno se troši vrijeme na beskorisne iteracije jer je konvergencija u tom slučaju prespora.

Selekcijom preživljavaju bolje jedinke i očekuje se da prosječna vrijednost preživjelih jedinki bude veća od prosječne vrijednosti cijele populacije. Posljedica selekcijskog pritiska je *selekcijska razlika* između preživjelih jedinki i cijele populacije [GOL01].

Definicija 4.4 Selekcijska razlika (engl. *selection differential*)

Selekcijska razlika s je razlika prosječne vrijednosti dobrote preživjelih jedinki $\overline{d_p}$ i prosječne vrijednosti dobrote svih jedinki $\overline{d}$ u svakoj iteraciji t:

$$s(t) = \overline{d_p}(t) - \overline{d}(t) \tag{4-10}$$

Prosječna vrijednost dobrote svih jedinki koje čine populaciju računa se prema izrazu:

$$\overline{d}(t) = \frac{1}{N}\sum_{i=1}^{N} d_i(t). \tag{4-11}$$

Definicija 4.5 Pristranost (engl**. *bias*)**

Pristranost je apsolutna razlika između normalizirane dobrote jedinke i očekivane vjerojatnosti reprodukcije iste jedinke.

Definicija 4.6 Raznovrsnost (engl. ***spread*)**:

Raznovrsnost je opseg mogućih vrijednosti broja potomaka neke jedinke. *Gubitak raznovrsnosti* je udio, odnosno, postotak jedinki u populaciji koje nisu izabrane za reprodukciju. To je postotak jedinki koje nisu selektirane za reprodukciju.

Definicija 4.7 Intenzitet selekcije

Neka je $\sigma(t)$ standardna devijacija svih dobrota u populaciji u generaciji t:

$$\sigma(t) = \sqrt{\frac{\sum_{i=1}^{N}\left[dobrota_i(t) - \overline{dobrota}(t)\right]^2}{N}}. \tag{4-12}$$

Uz pretpostavku da dobrota populacije $dobrota_i$ ima normalnu razdiobu $N\left[\overline{dobrota}(t), \sigma(t)\right]$ u generaciji t, selekcijska razlika je proporcionalna standardnoj devijaciji populacije:

$$s(t) = s_I \cdot \sigma(t), \tag{4-13}$$

pri čemu se faktor $s_I = s(t)/\sigma(t)$ naziva *selekcijskim intenzitetom* [GOL01].

Definicija 4.8 Varijanca selekcije

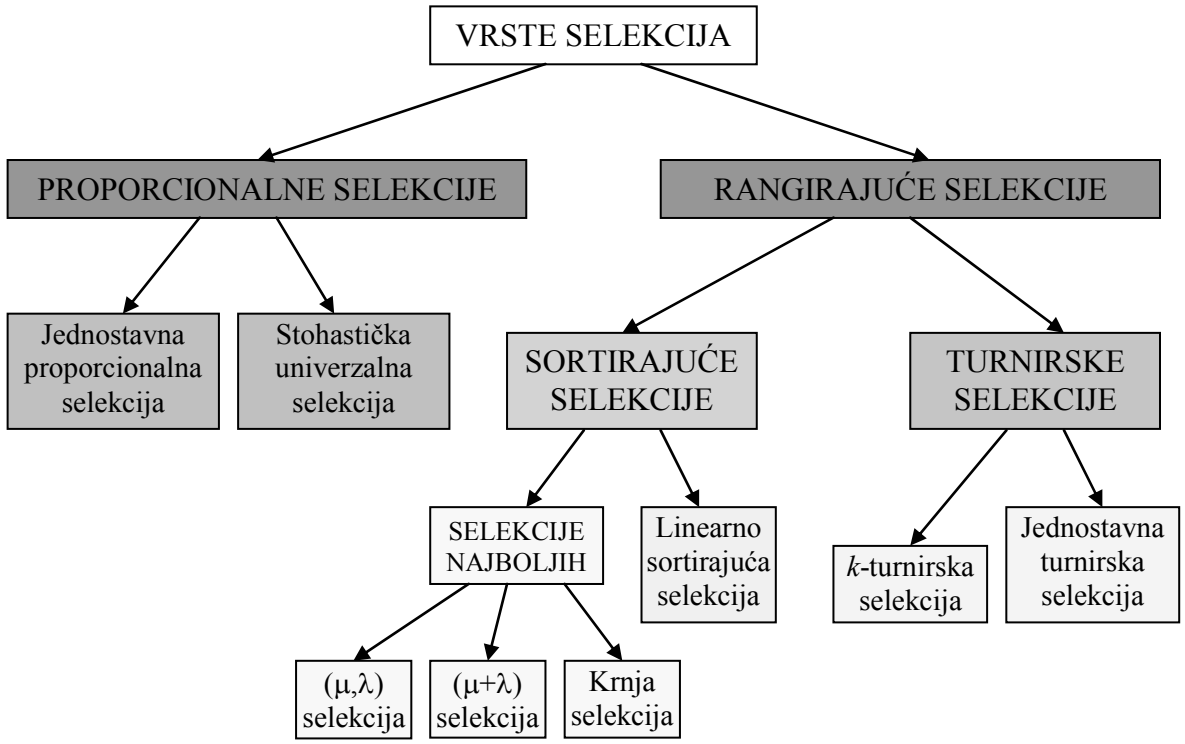
Varijanca selekcije je očekivana varijanca distribucije dobrote u populaciji nakon primjene metoda selekcije na populaciju s normalnom Gausovom razdiobom.

U literaturi su, različiti tipovi selekcijskih shema predloženi, testirani i uspoređivani. Različite selekcijske sheme imaju različiti selekcijski pritisak što može utjecati na rezultat pretraživanja genetskog algoritma. Preveliki selekcijski pritisak može dovesti genetski algoritam do preuranjene konvergencije, dok s premalim pritiskom proces evolucije može biti neučinkovit.

Postupci selekcije se dijele na dvije osnovne grupe: *generacijska* selekcija i *eliminacijska selekcija*. Prema vrsti ugrađene selekcije i genetski algoritmi se dijele na generacijske i eliminacijske. Kod generacijskog genetskog algoritma procesom selekcije se iz stare generacije stvara nova, a u slučaju eliminacijskog genetskog algoritma se "rupa" u populaciji nastala eliminacijom loših jedinki popunjava novim jedinkama.

Ovisno o metodi odabira boljih jedinki kod generacijskih selekcija, odnosno loših jedinki kod eliminacijskih selekcija, postupci selekcije se dijele na *proporcionalne* i *rangirajuće* (Cantú-Paz, 1999., citirano u [GOL01]). Selekcije se razlikuju prema načinu određivanja vrijednosti vjerojatnosti selekcije određene jedinke. Proporcionalne selekcije odabiru jedinke s vjerojatnošću koja je proporcionalna dobroti jedinke, odnosno vjerojatnost selekcije ovisi o kvocijentu dobrote jedinke i prosječne dobrote populacije. Broj jedinki s određenim svojstvom u sljedećoj iteraciji je proporcionalan kvocijentu prosječne dobrote tih jedinki i prosječne dobrote populacije.

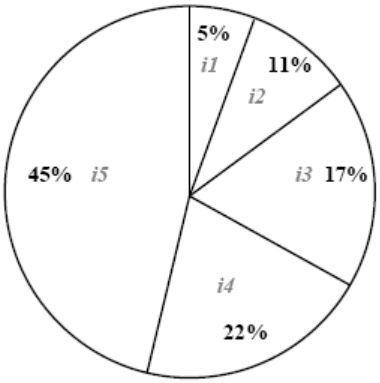
Rangirajuće selekcije odabiru jedinke s vjerojatnošću koja ovisi o položaju jedinke u poretku jedinki sortiranih po dobroti. Rangirajuće selekcije se dijele na *sortirajuće* i *turnirske* selekcije. Sortirajuće selekcije su: linearno sortirajuća selekcija, μ-λ selekcije i krnja selekcija. Turnirske selekcije se dijele prema broju jedinki koje sudjeluju u *turniru*. Na slici 4.15 prikazana je podjela selekcija na proporcionalne i rangirajuće, te na njihove podvrste.



Slika 4.15 Vrste selekcija [GOL01]

Među selekcijskim shemama najčešće se koriste sljedeće sheme:

- **Selekcija pomoću kotača ruleta** (engl. *the roulette wheel selection*) je najpoznatija i najčešće korištena selekcijska shema. Ova shema radi baš kao i kotač ruleta ili rotirajući kotač, pri čemu je svakoj jedinki pridružen isječak na točku duljine proporcionalne njenoj dobroti. Slika 4.16 prikazuje primjer kotača ruleta za pet jedinki i_1 do i_5. Postoci ukazuju na vjerojatnost izbora jedinki. Stoga, vjerojatnost jedinke s većom dobrotom da će biti odabrana za reprodukciju je veća od vjerojatnosti jedinke s manjom dobrotom.



Slika 4.16 Primjer selekcije pomoću kotača ruleta

Neka je N_{pop} broj jedinki svake populacije u GA, a $\Psi = \{x_1, x_2, \ldots, x_{N_{pop}}\}$ su jedinke iz trenutne generacije. Svako rješenje x_i se odabire kao roditelj u skladu s vjerojatnošću odabira $P(x_i)$, koja se definira kako slijedi:

$$P(x_i) = \frac{d(x_i)}{\sum_{j=1}^{N_{pop}} d(x_j)}, \quad \text{za} \quad i = 1,2,\ldots,N_{pop} \tag{4-14}$$

gdje je $d(\cdot)$ funkcija dobrote za rješenje x.

Proces selekcije se zasniva na rotiranju kotača onoliko puta kolika je veličina populacije.

- **Turnirska selekcija** (engl. *tournament*); u općem obliku turnirske selekcije iz cijele populacije izabire se broj jedinki u opsegu od 2 do ukupnog broja jedinki te se najbolja jedinka bira za roditelja. Najčešći oblik turnirske selekcije je binarna turnirska selekcija gdje se biraju dvije jedinke. Izabrana veličina turnira određuje intenzitet selekcije.

veličina turnira	1	2	3	5	10	30
intenzitet selekcije	0	0.56	0.85	1.15	1.53	2.04

Slika 4.17 Relacija između turnira i inteziteta selekcije [MUD02]

4.2.5. Kombiniranje operatora

Kombiniranjem operatora moguće je postići bolje značajke djelovanja genetskog algoritma u odnosu kad se djeluje samo s jednom vrstom operatora. Tako je za određeni primjer potrebno izvesti testiranje nad skupom podataka s opisanim operatorima, a zatim najbolji tip križanja kombinirati s mutacijom, koja se namjerava implementirati.

Nadalje, moguće je kombinaciju operatora još poboljšati mijenjanjem relativne uporabe pojedinog operatora. Takovo rješenje pri djelovanju operatoru, obično pri operatoru križanja, u početku brzo konvergira u smjeru konačnog rješenja, ali se potom uskoro

stabilizira na nekoj vrijednosti. Mutacija, koja je korištena, brine o raznovrsnosti u populaciji i u smjer rješenja ide postepeno. Stabilizira se tek pri kraju djelovanja genetskog algoritma. Najbolje je operator križanja upotrijebiti u prvom dijelu djelovanja genetskog algoritma, a mutaciju u drugom dijelu.

Neki autori su u kombiniranju operatora krenuli korak naprijed. Oni su predložili algoritam koji mjeri uspješnost djelovanja operatora pri svakoj promjeni populacije, tako da je dinamično mijenjanje vjerojatnosti uporabe operatora u skladu s njegovom trenutnom uspješnošću i mogućnošću postizanja boljeg rješenja (Davis, 1989. citirano u [BER01]).

4.3. Višekriterijski genetski algoritam (MCGA)

U praksi se često nailazi na probleme koje je potrebno istodobno optimalizirati po različitim kriterijima. Naime, da bi se vrednovala kvaliteta mogućeg rješenja nužno je uzeti u obzir nekoliko kriterija koji su po svojoj prirodi različiti i najčešće su međusobno proturječni. To znači, da poboljšanje rješenja po jednom kriteriju uzrokuje njegovo pogoršanje po drugim kriterijima. Tada se ne dobije samo jedno optimalno rješenje, nego više njih koji se nazivaju Pareto optimalna fronta.

Radi pomanjkanja metoda, primjernih za višekriterijsku optimalizaciju, u prošlosti su se problemi višekriterijske optimalizacije rješavali klasičnim metodama, tako da bi se problem pretvorio u jednokriterijski i riješio bi se klasičnim pristupom. Tek s evolucijskim algoritmima su se višekriterijski optimalizacijski problemi tretirali tako, da se tražilo više rješenja koji su aproksimirali Pareto optimalnu frontu.

Danas je višekriterijska optimalizacija, vrlo zanimljiva tema i za istraživače i za inženjere. Naime, još uvijek nema opće prihvaćene definicije "*optimuma*" kao kod jednokriterijske optimalizacije, što čini izrazito teškim uspoređivati jednu metodu sa drugom.

4.3.1. Uvod u višekriterijski optimalizaciju

Kako se iz samog naziva može zaključiti, metode višekriterijske optimalizacije koriste se za istodobnu optimalizaciju više funkcija cilja (kriterijskih funkcija)[8]. Nadalje, postoje i problemi gdje se istodobno mora zadovoljiti i više ograničenja te se takvi problemi definiraju kao *višekriterijski optimalizacijski problemi*. Svaka kriterijska funkcija može imati različito mjerilo i može imati složeno međudjelovanje s drugim kriterijima funkcijama.

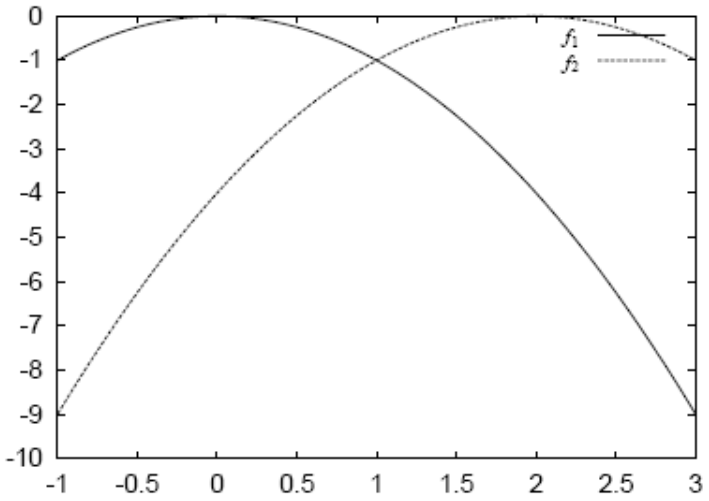
Primjer 4.1 (Cvetković, 2000.)

Klasičan primjer je Schaffer-ova funkcija (1984) prikazana na slici 4.18:

$$F(x) = (f_1(x), f_2(x)) = \left(-x^2, -(x-2)^2\right) \qquad (4\text{-}15)$$

Vidljivo je da funkcija f_1 postiže maksimum za $x = 0$, dok funkcija f_2 postiže maksimum za $x = 2$, stoga ne postoji jedinstvena točka u kojoj obje funkcije postižu svoj maksimum.

[8] U literaturi se umjesto pojma kriterijska funkcija koriste i pojmovi kriterij, objektna funkcija ili objekt.



Slika 4.18 Graf funkcije (4-15) [CVE00]

■

Kod ovakvih problema korisnik nikada nije zadovoljan pronalaženjem jednog rješenja koje je optimalno obzirom na jedan kriterij.

Naime, kod višekriterijske optimalizacije, ne može se govoriti o najboljem (*dominantom*) niti optimalnom rješenju u klasičnom smislu, nego o rješenju (može ih biti i više) koje, na neki način, najbolje zadovoljava postavljene kriterije. Ovakvo optimalno rješenje ima poseban naziv, *Pareto-optimalno rješenje*. Pareto optimalnost je nazvana po ekonomistu Vilfredo Pareto (1896), koji je prepoznao problem višekriterijske optimalizacije. Koncept Pareto-optimalnosti (dominantnosti) je veoma važan kod višekriterijske optimalizacije i bit će objašnjen u sljedećem potpoglavlju.

Optimalnost se definira kako slijedi:

Rješenje je Pareto optimalno ako ne postoji drugo rješenje koje dominira nad njim.

To znači ako niti jedno drugo rješenje, prema svim kriterijima, nema bolje ili jednake vrijednosti funkcije cilja od zadanog rješenja onda ono pripada Pareto skupu. Stoga je ideja da *rješenje optimalizacije višekriterijskog problema obično nije jedno rješenje već skup rješenja, Pareto skup.* Isto tako, dobro rješenje u višekriterijskoj optimalizaciji se može definirati kao ono koje nije dominatno.

Ako je velika razlika među optimalnim rješenjima koje odgovaraju različitim kriterijima, onda se kaže da su kriteriji međusobno *konfliktni*. Primjerice, kod proizvodnih sustava glavni cilj optimalizacije je proizvesti proizvod u što kraćem vremenu uz maksimalnu uporabu strojnih/ljudskih resursa. Štoviše, kako raste broj poslova to je teže pronaći dobro rješenje u nekom razumnom vremenu.

4.3.2. Osnove definicije višekriterijske optimalizacije

Pojmovi koji su nezaobilazni u procesu višekriterijskog odlučivanja jesu ciljevi ili kriteriji, ograničenja i atributi [VUČ07].

Kriteriji ili ciljevi predstavljaju željeni pravac promjena stanja od strane onoga ili onih koji o njemu odlučuju. Pravac promjena može biti maksimiziranje ili minimiziranje, ali i održavanje stanja na određenom stupnju koji se može kvantificirati (naravno, odgovarajućim transformacijama ova tri pravca mogu se svesti na problem maksimiziranja, te se kao takvi najčešće i promatraju).

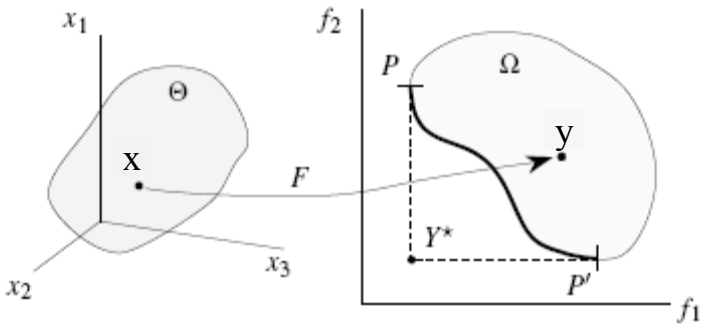
Izbor kriterija ovisi prije svega o dobrom razumijevanju problema, tj. njegovu modeliranju. Pristup modeliranju može biti putem detaljnoga pregleda prikaza sličnih problema i stanja, analitičke studije problema ili čak i uporabe iskustava. Primjeri ciljeva jesu ostvarenje što veće dobiti, visoka kvaliteta proizvoda/usluga, visoka iskoristivost resursa, očuvanje i unaprjeđivanje kvalitete okoliša, zadovoljstvo klijenata, bitno smanjenje mogućnosti prirodnih katastrofa, itd.

Atributi opisuju značajke i kvalitetu parametara postupaka u procesima optimalizacije. Zadatak atributa jest da omogući procjenu zadovoljavanja postavljenih ciljeva. Za svaki postavljeni atribut (npr. očuvanje prirode, zadovoljavajuće komunikacije i kvalitete komunalne infrastrukture i sl.) veže se određeni indikator (koncentracija zagađenja, dužina cesta, pokrivenost vodoopskrbom i odvodnjom itd.) i formiraju se odgovarajući rasponi kojima pripadaju referentne vrijednosti.

Varijable odlučivanja jesu nepoznate veličine koje treba odrediti u zadanom problemu optimalizacije. Uglavnom su predstavljene ne-negativnim veličinama, a često su neprekidne pa i ograničene. Primjerice, količina vode potrebne za vodoopskrbu; ova vrijednost nije diskretna već kontinuirana, ograničena ukupnom dostupnošću vode (npr. zapreminom akumulacije) i ne-negativna je.

Ograničenja izražavaju međusobnu ovisnost između varijabli odlučivanja i parametara, te naravno stanja sustava. Mogu biti izražena kako jednakostima koje opisuju zahtijevano stanja ravnoteže, tako i nejednakostima (koje najčešće opisuju ograničenja resursa), pa čak i mjerama vjerojatnosti. Može se dogoditi da neki cilj postane ograničenje, ali i obratno, jer crta razgraničenja ciljeva i ograničenja nije uvijek jasno povučena.

U skladu s onim navedenim u poglavlju 4.3.1, sljedeće definicije su važne za višekriterijsku optimalizaciju.



Slika 4.19 Prostor varijabli i prostor kriterija [LEW02]

Neka je zadano preslikavanje $F:\Theta\to\Omega$, gdje je $\Theta\subset R^N$ *N*-dimenzionalan prostor pretraživanja (odlučivanja), $\Omega\subset R^n$ *n*-dimenzionalan prostor cilja (kriterija), $y\in\Omega$ je vektor od *n* kriterija. To je prikazano slikom 4.19, pri čemu je *N*=3 i *n*=2.

Ako je zadatak optimalizirati ili f_1 ili f_2, dovoljno je rabiti neku jednokriterijsku metodu optimalizacije. Onda će, za ovaj primjer, traženje rješenja završiti ili u točki P ili P'. Međutim, cilj višekriterijske optimalizacije je optimalizirati istovremeno obje funkcije.

Definicija 4.9 Višekriterijska optimalizacija

Problem *višekriterijske* optimalizacije se definira kao problem traženja vektora varijabli koji minimalizira vektorsku funkciju, čiji su elementi kriterijske funkcije. Bez gubitka općenitosti, optimalizacija je ograničena na minimalizaciju svih kriterijskih funkcija, budući se svaki problem određivanja maksimuma funkcije f može prevesti u problem traženja minimuma funkcije na sljedeći način:

$$max(f(x)) = -min(-f(x)) \tag{4-16}$$

U matematičkom obliku problem višekriterijske optimalizacije glasio bi:

Odrediti optimum vektora kriterijskih funkcija

$$min \quad F(x) = [f_1(x), f_2(x), \ldots, f_n(x)] \tag{4-17}$$

gdje je $x = (x_1, x_2, \ldots, x_N) \in \Theta$ vektorska varijabla odlučivanja koja zadovoljava odgovarajuće uvjete – ograničenja:

$$G_j(x) \le 0, \quad j = 1,2,\ldots,m; \quad m \le M \tag{4-18}$$

$$H_k(x) = 0, \quad k = m+1,\ldots,M \tag{4-19}$$

N je pojedinačni broj varijabli odlučivanja, *n* je broj odabranih kriterijskih funkcija, a M je broj poznatih ograničenja koja moraju biti zadovoljena. Skalarna funkciju $f_i(x)$ je element vektora kriterija i označava *i*-ti kriterij. Funkcijski vektori $G(x)$ i $H(x)$ označavaju ograničenja u obliku nejednakosti, odnosno jednakosti.

Kako su komponentne kriterijske funkcije $f_i(x)$ obično međusobno konfliktne, što znači da željeno minimiziranje ili maksimiziranje jedne od njih često vodi neželjenom – suprotnom reagiranju neke druge kriterijske funkcije, to se pod optimumom vektorske funkcije ovdje podrazumijeva takva vrijednost $F(x)$ koja nije inferiorna u odnosu na bilo koju drugu $F(x')$, a koja u odgovarajućem smislu (koji treba posebno definirati) zadovoljava donositelja odluke. Dakle, osim u rijetkim slučajevima, takvo rješenje ne predstavlja jednovremeni pretpostavljeni maksimum svih pojedinačnih kriterijskih funkcija $f_i(x)$.

Neka se razmatra problem s m funkcija cilja i $m>1$. Za bilo koja dva rješenja $x, x^* \in F$ vrijedi ili jedno dominira nad drugim ili niti jedno ne dominira (*F* je moguća domena).

Glavna poteškoća višekriterijske optimalizacije leži u usporedbi rješenja. Po definiciji jedno rješenje je bolje od drugog ako su vrijednosti svih funkcija cilja za prvo rješenje bolje nego za drugo rješenje. Također, kaže se da je prvo rješenje *prevladava* (*dominira*) nad drugim.

Definicija 4.10 Dominantnost

Za rješenje $a \in \Theta$ se kaže da *dominira nad* rješenjem $b \in \Theta$ ($a \preccurlyeq b$) ako i samo ako vrijede sljedeći zahtjevi:

(i) Rješenje a nije lošije[9] od b u svim kriterijskim funkcijama, tj.

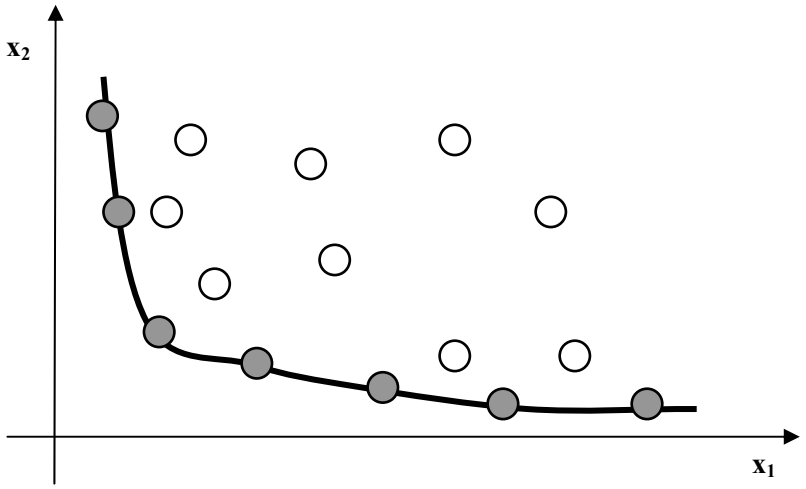
$$f_i(a) \le f_i(\mathrm{b}) \quad \text{za} \quad \forall\, i = 1,\ldots,n \tag{4-20}$$

(ii) Rješenje a je strogo bolje od b u barem jednoj kriterijskoj funkciji, tj.

$$f_j(a) < f_j(b), \quad \text{za barem jedan} \quad j \in \{1,\ldots,n\} \tag{4-21}$$

Ako nema drugih rješenja koja dominiraju nad zadanim rješenjem onda se takvo rješenje smatra *optimalnim*. Ali u skladu sa proturječnim značajkama kriterijskih funkcija, obično ne postoji jedno jedinstveno optimalno rješenje. Moguće je, odvojeno, poboljšati značajke za najmanje jednu funkciju cilja (ali ne za sve) za zadano rješenje, što obično povlači opadanje vrijednosti preostalih funkcija (ili barem jedne od njih). Tako, nekoliko različitih rješenja se mogu razmatrati kao optimalna rješenja, jer niti jedno od njih ne dominira nad drugima. Takva rješenja se nazivaju *Pareto-optimalna*, a skup svih Pareto-optimalnih rješenja se naziva *Pareto-fronta* i predstavlja granicu između prostora koji sadržava dominantna rješenja i prostora gdje nema takvih rješenja.

Za bi-kriterijski prostor (kriteriji x_1 i x_2) Pareto-fronta se prikazuje krivuljom. Slika 4.20 prikazuje primjer gdje sve točke predstavljaju Pareto-optimalna rješenja dok su dominantna rješenja prikazana bijelim točkama.

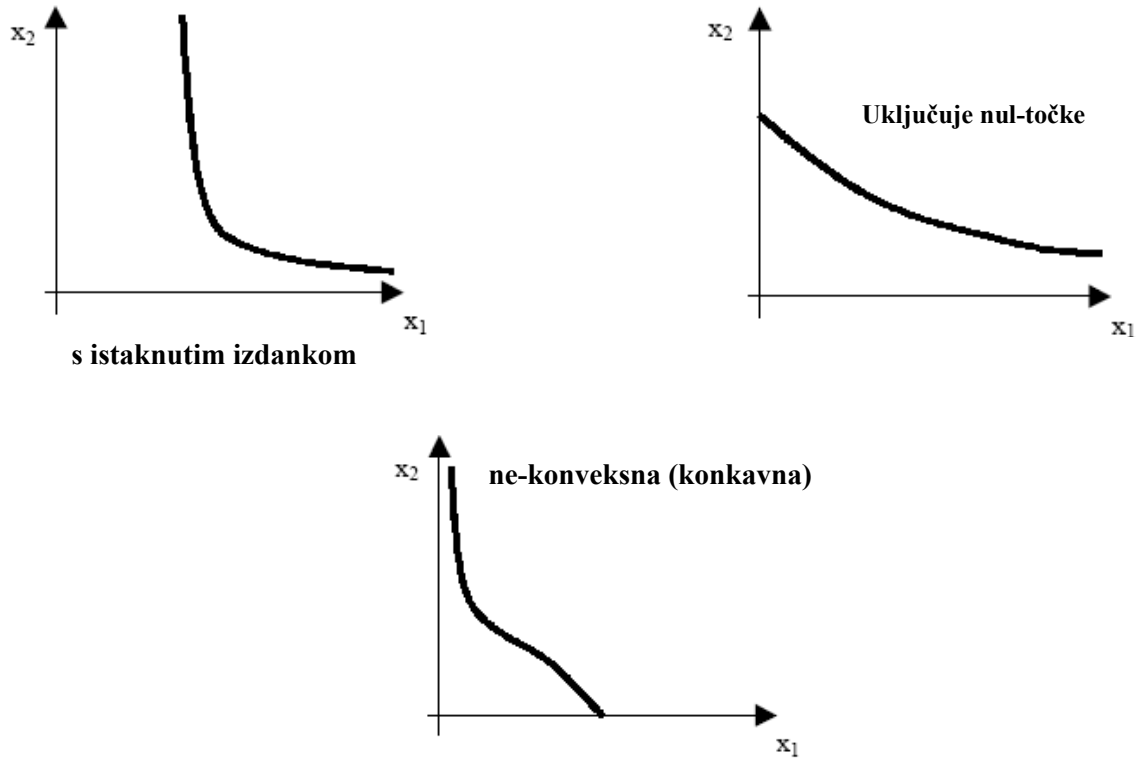


Slika 4.20 Pareto-optimalna rješenja sačinjavaju Pareto-frontu

[9] Operator $\prec$ označava lošije, a $\succ$ označava bolje.

Oblik krivulje ovisi o prirodi problema [BYK03]. Primjerice, moguće je da se teži k nekom kriteriju i u tom slučaju krivulja bi sjekla koordinatnu os ili bi krivulja bila konkavna (slika 4.21).

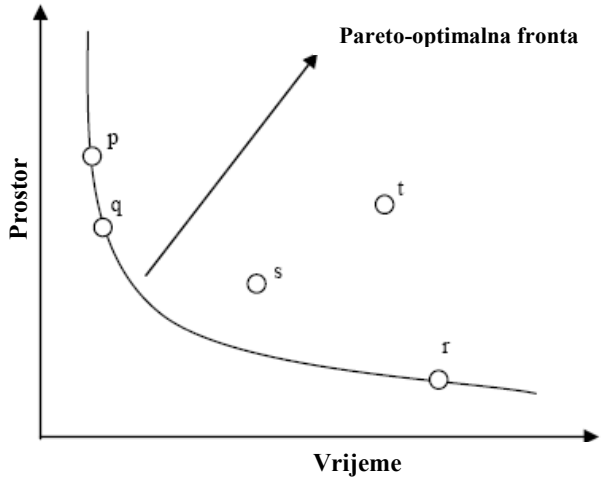
Općenito je ili nemoguće ili preteško analitički izraziti krivulju Pareto-fronte.



Slika 4.21 Različiti mogući oblici Pareto-fronte

Primjer 4.2

U ovom primjeru prikazat će se Pareto optimalno rješenje (slika 4.22) s vremenskom i prostornom složenosti algoritma. U ovom se problemu treba minimalizirati i vrijeme i prostor.



Slika 4.22 Pareto-optimalno rješenje

Točka 'p' predstavlja rješenje, koje ima minimalno vrijeme, ali je prostorna složenost previsoka. S druge strane točka 'r' predstavlja rješenje s visokim vremenom, ali s minimalnom prostornom složenošću. Ako se gledaju obje funkcije dobrote, niti jedno rješenje nije optimalno. Stoga se u ovom slučaju ne može reći da je rješenje 'p' bolje od rješenja 'r'. Naime, postoji mnogo rješenja kao 'q', koje isto tako pripada Pareto-optimalnom skupu i ne mogu se poredati rješenja prema nekoj vrijednosti metrike, ako se razmatraju obje funkcije dobrote. Sa slike 4.22 je vidljivo da postoje rješenja koja ne pripadaju Pareto-optimalnom skupu, kao što je rješenje 't'. Očito je zašto 't' ne pripada Pareto-optimalnom skupu.

■

Rješenje praktičnog problema može biti ograničeno brojnim restrikcijama varijable odlučivanja. Ograničenja obično pripadaju jednoj od dvije različite kategorije:

- *Ograničenja domene* izražavaju područje definicije funkcije dobrote. U upravljačkim sustavima, zatvorena petlja stabilnosti sustava može se navesti kao primjer ograničenja domene, jer većina mjerenja nisu definirana za nestabilne sustave.
- *Podešena ograničenja* (engl. *Preference constraints*) izriču daljnja ograničenja na rješenje problema u skladu sa znanjem na višoj razini. Primjerice, zadana granica stabilnosti izražava (subjektivne) odrednice projektanta.

4.3.3. Genetski algoritmi za višekriterijske probleme

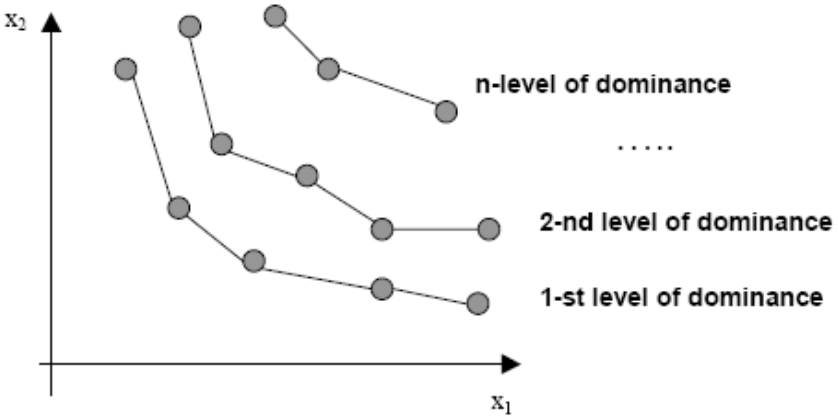
Ovi tipovi problema često se rješavaju metodom težinskih suma, metoda ometanja (engl. *Perturbation method*), ciljanim programiranjem (engl. *Goal programming*), Tchybeshev metodom, metodom minmax i drugim metodama [TOD97]. Ovo su klasične optimalizacijske metode koje, u najboljem slučaju, pronađu samo jedno optimalno rješenje u jednoj simulaciji.

Ideja uporabe genetskih algoritama za višekriterijske probleme nije nešto novo. Štoviše, genetski algoritam je bio prva heuristička metoda koja se primijenila za višekriterijsku optimalizaciju [BYK03]. U ovom potpoglavlju razmotrit će se različiti višekriterijski genetski algoritmi.

Obično su se ovakvi problemi rješavali uporabom težinske sume (engl. *Weighted sum of objectives*). Ova metoda daje samo jedno rješenje i ne upućuje na bilo kakav kompromis.

Prvo višekriterijsko proširenje genetskog algoritma razvio je David Schaffer (1984) [BYK03] i nazvao ga je *vektorski evaluiran genetski algoritam* (engl. *Vector Evaluated Genetic Algorithm*, pokrata: VEGA). Schaffer je koristio poseban mehanizam selekcije koji iz populacije odabire n podgrupa od N/n jedinki na osnovi njihovih ispunjavanja svakog od n kriterija (pretpostavljajući da je N veličina populacije). Zatim bi se ove podgrupe izmiješale tvoreći novu populaciju veličine N, na koju bi se primijenili genetski operatori.

1989 godine Goldberg [GOL89] je prvi primijenio ideju ne-dominantnog rangiranja. Ova metoda uključuje, prvo, pronalaženje svih Pareto optimalnih točaka unutar populacije i njima se pridruži najveći rang (rang 1) (slika 4.23). Zatim se isto ponovi na preostalim jedinkama populacije, pronađu se ne-dominantne jedinke i pridružio im je rang 2. Na taj način, rangira se cijela populacija. Umjesto dobrote koristio je dodijeljene rangove za vrednovanje rješenja u postupku selekcije.



Slika 4.23 Goldbergova metoda rangiranja populacije [BYK03]

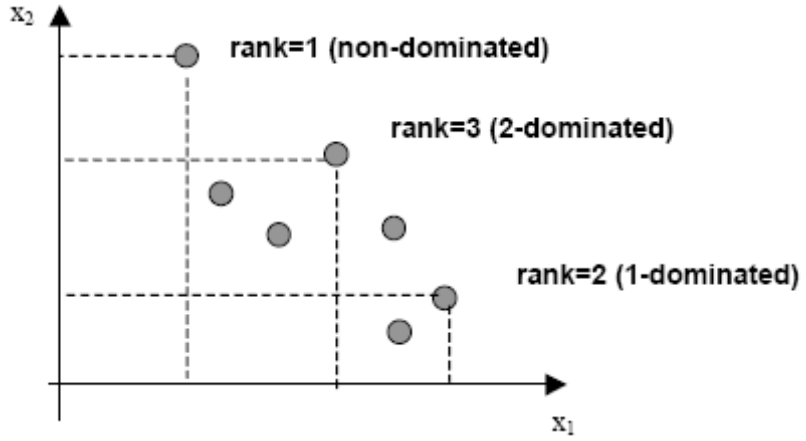
Kasnijom analizom značajki VEGA algoritma, Richardson i ost.(1989) pokazali su da je prosječni utjecaj svake funkcije dobrote na konačno rješenje proporcionalan veličini odgovarajuće pod-populacije. Stoga, veličina pod-populacije može imati isto značenje kao težine u slučaju kada se pri donošenju odluke razmatraju različiti kriteriji.

Algoritam ne-dominantnog rangiranja pripada skupini algoritama koji se nazivaju Pareto-tehnike. Ove metode dodjeljuju dobrotu svakoj jedinki prema relaciji njene dominantnosti u odnosu na ostale jedinke populacije. U svakoj generaciji, jedinke se provjeravaju jesu li ne-dominantne i tako se korak po korak približavaju Pareto-fronti. Glavna prednost ovih metoda je neovisnost o obliku Pareto-fronte. Dva su osnovna nedostatka:

- *Genetsko skretanje*: rješenja teže skupnom sakupljanju u jednom dijelu Pareto-fronte, dok drugi dio ostaje prazan.
- *Preuranjena konvergencija*: kako bi poboljšalo rješenje algoritam se zaustavlja prije prave Pareto-fronte.

Jedan od ciljeva višekriterijske optimalizacije je generirati i sačuvati različiti opseg Pareto-optimalnih točaka. Kako bi se to osiguralo rabe se tehnike koje koriste izvorni jednokriterijski GA. Goldberg [GOL87] je predložio *razmjenu dobrota (engl. Fitness sharing*) kako bi se prevladalo genetsko skretanje. Ako je nekoliko jedinki okupljeno u grupu (na osnovi udaljenosti između njih u kriterijskom prostoru), onda jedinke "dijele" svoje dobrote. Dobrota svake jedinke se podijeli s veličinom grupe. Na ovaj način grupa djeluje kao jedna jedinka. Ova metoda je zajednička svim Pareto tehnikama.

Drukčiji način rangiranja predložili su 1993. g. Fonseca i Fleming [FON93] koji su razvili *Više-objektni genetski algoritam* (engl. *Multi-Objective Genetic Algorithm*, pokrata: MOGA). Procedura rangiranja MOGA algoritma prikazana je sljedećom slikom.



Slika 4.24 Rangiranje kod MOGA algoritma [FON93]

Kod ove sheme rangiranja svakoj jedinki (rješenju) se dodjeljuje rang u skladu s brojem jedinki koje dominiraju nad njom. Na ovaj način ne-dominantna rješenja dobivaju rang 1. Rješenja, kojim dominira samo jedno rješenje dobiva rang 2 itd. Ovakav način rangiranja reducira genetska skretanja jer jedinke zajedno grupirane imaju manju vjerojatnost preživljavanja (jer nad njima dominira više jedinki).

1996. Fleming i Shaw su predstavili početnu studiju primjene ove tehnike probleme rasporeda [SF96a]. Ovaj algoritam je dao visoko kvalitetna rješenja koja se ne bi mogla postići primjenom tehnika težinskih suma.

Istovremeno s razvojem MOGA algoritma, razvila se i druga inačica Pareto-optimalizacijskog pristupa koja se zove algoritam NPGA (engl. *Niched Pareto Genetic Algorithm*). Razvili su ga Horn i Nafpliotis 1994 godine [HOR94]. U NPGA algoritmu primjenjuje se binarna verzija turnirske selekcije koja se naziva “Pareto dominantan turnir”. Nadmoć između dva rješenja je određena njihovom dominantnošću nad slučajno odabranim skupom rješenja. Jedinka s manjim brojem dominantnih točaka preživljava. Ako dva rješenja imaju jednaki broj dominantnih točaka, prioritet se daje onom rješenju s manjim brojem "susjednih" (okupljeni u istu grupu) jedinki. Ovaj mehanizam može značajno skratiti vrijeme računanja ne-dominantnih rješenja. Ipak, za dobru učinkovitost algoritma zahtijeva se razumno velika populacija, jer veći broj jedinki će povećati i pristranost prema jačim rješenjima.

Treća metoda, algoritam NSGA (engl. *Non-dominated Sorting Genetic Algorithm*) [SRI94] vraća se osnovnom Goldberg-ovom algoritmu. Autori predlažu vrednovanje rezultata pomoću njegova tzv. *dummy penalty*, koji se računa na osnovi ranga jedinke i srednje udaljenosti od drugih rješenja.

Obećavajuća ideja kako poboljšati učinkovitost višekriterijskih evolucijskih algoritama sastoji se od kategorizacije roditelja za rekombinaciju (engl. *mating restriction*). To je opisano u [GOL89] prvi je primijenio Allenson 1992 godine unutar algoritma VEGA (citirano u [BYK03]). Za bi-objektan slučaj, autor je koristio dvije vrste jedinki (“spolove”). Spol (nasumično dodijeljen pri porodu) je određivao cilj kojim je jedinka trebala biti ocijenjena. Operator križanja se mogao primijeniti na jedinke različitih spolova. Po autorovom mišljenju, takva "bioraznolikost" može pomoći da se naprave kompromisna rješenja. Autor je pretpostavio da ako takva mogućnost postoji u prirodi onda je vrijedno istražiti je u algoritmu. Lis i Eiben su 1996 godine poopćili ovaj pristup. Predložili su uporabu

nekoliko spolova (broj spolova je jednak broju funkcija cilja) i poseban operator križanja koji bi djelovao na više roditelja. Algoritam je testiran na ne-konveksnim i diskretnim problemima i rezultati su pokazali dobru pokrivenost Pareto-fronte.

David Todd (1997) je u svojoj doktorskoj disertaciji "*Multiple Criteria Genetic Algorithms in Engineering Design and Operation*" analizirao primjenu GA na višekriterijske probleme u inženjerskom projektiranju. Razvio je MCGA algoritam (engl. *Multiple Criteria Genetic Algorithm*) koji dopušta istovremeno rješavanje i problema maksimuma i minimuma kroz nekoliko kriterija. Ovaj algoritam će biti opisan u sljedećem poglavlju pri rješavanju složenih rasporeda poslova.

4.4. Sažetak poglavlja

U ovom poglavlju opisan je osnovni mehanizam GA i neki od osnovnih operatora i tehnika koje se koriste u području genetskih algoritama. Opisan je jednostavni GA s jednom funkcijom cilja. Također opisan je i operator križanja za realno kodiranje kromosoma. Na kraju je opisan i višekriterijski GA za više-ciljnu optimalizaciju. Dan je pregled nekih mehanizama višekriterijskog GA. Uporabe li se genetski algoritmi za rješavanje teških optimizacijskih problema s velikim brojem jedinki u populaciji i velikim brojem gena u kromosomu te s velikim brojem kriterija koji se moraju zadovoljiti, evolucijski postupak može trajati satima pa čak i danima. U nastavku ovog rada uvest će se i neka poboljšanja GA u vidu poboljšanja nekih genetskih operatora i uvođenjem dinamičke veličine populacije.

Poboljšavanjem mogućnosti računala nastoje se sve teži problemi riješiti u što je moguće kraćem vremenu.

5. ODREĐIVANJE I OPTIMALIZACIJA RASPOREDA POSLOVA

Uz računalne i proizvodne sustave, problem određivanja rasporeda može se primijeniti na različita područja uključujući transport, zdravstvo i poljoprivredu. Prema Collins Concise English Dictionary (1992)[10], *raspored* je "plan procedura nekog projekta", "popis proizvoda" ili "popis zadataka koji se moraju izvesti". *Raspoređivanje* (engl. *Scheduling*) je postupak izrade rasporeda.

U realnom okruženju, problem određivanje rasporeda je reaktivan proces, gdje nazočnost informacija u stvarnom vremenu stalno traži ponovno promišljanje i reviziju već određenog rasporeda. Algoritmi za određivanje rasporeda kojima se postižu dobra ili optimalna rješenja i koji se mogu jednostavno i učinkovito prilagoditi na zastoje u funkcioniranju sustava su, u većini slučajeva, poželjniji od onih koji postižu optimalni rezultat ali se ne mogu prilagoditi situacijama pojave zastoja.

Kako većina problema rasporeda spada u skupinu NP-teški[11], izrada algoritama za rješavanje takvih problema postao je izazov i za znanstvenike i za istraživače. Povijesno gledano, problemi određivanja rasporeda rješavali su se primjenom nekog od sljedećih pristupa [TOD97]:

- Egzaktne metode: Giffler & Thompson (1960), Brucker i ost. (1994) i Williamson i ost. (1997), Ignall & Schrage [citirano u MUR97];
- Metode grananja i granica (engl. *Branch and bound*): Lageweg i ost. (1977), Carlier & Pinson (1989, 1990), Sabuncuoglu & Bayiz (1999);
- Heurističke procedure koje se zasnivaju na pravilima prioriteta: French (1982), Gray & Hoesada (1991).

U ovom poglavlju bit će detaljno opisan opći problem rasporeda te kako se može primijeniti GA na rješavanje problema rasporeda. Kako je prirodna evolucija proces stalnog prilagođavanja jedinki okolini u kojoj živi, onda ima smisla razmatrati genetske algoritme kao dobar način rješavanja dinamičkih problema rasporeda.

Kada se vrednuje kvalitete rješenja ili rasporeda, kod mnogih problema rasporeda (primjerice, raspored strojeva, raspored zaposlenika, raspored predavanja, itd.) potrebno je istovremeno razmatrati više kriterija. Ti kriteriji se specificiraju ovisno o problemu koji se rješava. Primjerice, kod rješavanja problema rasporeda poslova u proizvodnom sustavu može se tražiti minimalna cijena i vrijeme te maksimalna iskoristivost resursa. Posljednjih godina razvijeni su brojni algoritmi za rješavanje problema višekriterijskog rasporeda. U tom smislu opisat će se MCGA-višekriterijski genetski algoritam. Za razliku od standardnog genetskog algoritma, MCGA pronalazi skup rješenja koji tvore područje kompromisa između postavljenih kriterija pretraživanja. Iz tog skupa rješenja, upravitelj projekta će odabrati jedno i implementirati ga. Mnogo više o MCGA algoritmu može se pronaći u [TOD97, TOD98].

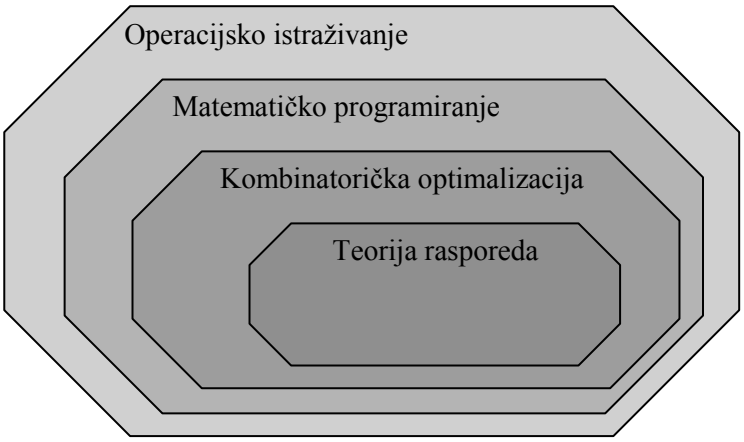
---

[10] Collins (1992). Collins Concise English Dictionary (3 ed.). Glasgow: HarperCollins (citirano u [UET01]).

[11] Klasa NP problema: valjanost nekog rješenja može se verificirati u polinomijalnom vremenu; Problem X je NP-kompletan ako rješava sve ostale NP probleme-"teški".

5.1. Teorija rasporeda

Kao izdvojeni dio Operacijskog istraživanja, teorija rasporeda se pojavila početkom pedesetih godina 20-tog stoljeća.



Slika 5.1 Teorija rasporeda

Općenito, *problem određivanja rasporeda j*e razvrstavanje zadanog skupa entiteta (poslovi, ljudi, vozila, resursi, ispiti, predavanja, sastanci, itd.), a da se pri tome zadovolje postavljeni kriteriji i ograničenja te se ostvare postavljeni ciljevi. Određivanja rasporeda spada u skupinu problema gdje se vrijeme, prostor i ograničeni resursi najčešće uzimaju u obzir pri rješavanju zadanog problema. Ograničenja su odnosi između entiteta i mogu se podijeliti na "meka" i "čvrsta" ograničenja. *Čvrsta ograničenja* (engl. *hard constraints*) se ne smiju prekršiti u bilo kojim okolnostima. Rješenja koja zadovoljavaju takva ograničenja nazivaju se *moguća rješenja* (engl. *feasible solutions*). Poželjno je da se zadovolji što je moguće više *mekih* ograničenja (engl. *soft constraints*). Međutim, ako se jedno od tih ograničenja prekrši, primjenjuju se penali, a rješenje se još uvijek uzima kao moguće rješenje. U praksi, na određivanje rasporeda se gleda kao na problem određivanja bilo kojeg mogućeg rasporeda ili kao optimalizacijski problem za kojeg se traži najbolji raspored. Pod najboljim rješenjem smatra se ono s najmanjim penalima (ako nisu zadovoljena meka ograničenja).

Većina problema određivanja rasporeda pripada skupini problema kombinatoričke optimalizacije i ako se dobro riješe mogu značajno skratiti vrijeme i troškove u sustavu. Dva su osnovna razloga zbog čega se proučavaju problemi rasporeda: raznolikost postojećih aplikacija i matematički interes za izradom odgovarajućeg modela.

Mnogi problemi određivanja rasporeda su po svojoj prirodi višekriterijski. Naime, istovremeno se želi postići nekoliko ciljeva [FAN94, MUR97, TOD97]. Primjeri takvih ciljeva su: optimalizirati uporabu raspoloživih resursa, zadovoljiti sklonosti ljudskih resursa, minimalizirati kašnjenja naloga, maksimalizirati poštovanje propisa i još mnogi drugi. Ovi problemi su se godinama rješavali na način da su se višestruki ciljevi združili u jednu skalarnu vrijednost pomoću funkcija težinskih suma, u skladu s postavljenim uvjetima i ograničenjima [BYK03, GHO04, MED03]. Kod mnogih višekriterijskih problema određivanja rasporeda, u procesu odlučivanja poželjno je predstaviti različita kompromisna rješenja kako bi se mogao odabrati najprimjereniji raspored. Iako bi se to moglo postići tehnikama pretraživanja i to više puta uz različite uvjete, zanimljivije je primjena Pareto-optimalnih tehnika na rješavanje

višekriterijskog problema rasporeda. Posljednjih godina, razvijeni su brojni algoritmi Pareto optimalizacije, jer se problemi višekriterijskog određivanja rasporeda mogu pronaći u bilo kojoj aplikacijskoj domeni.

U ovom poglavlju razmatrat će se problem određivanja rasporeda poslova. Treba izvršiti određeni broj *poslova* koji se sastoje od danog niza *operacija*, a uz uporabu određenog broja resursa. Problemi rasporeda poslova dolaze iz industrijskog okruženja, ali se mogu primijeniti i u drugim okruženjima kao što su fleksibilni proizvodni sustavi, transportni sustavi, informacijski sustavi, financijski sustavi, zdravstveni sustavi itd. Općenito, problem određivanja rasporeda može se primijeniti na različite kombinatoričke optimalizacijske probleme kao što su [BER01]:

- *Određivanje najprimjerenijeg rasporeda poslova* i pri tome se razlikuju dva tipa: *izbor toka poslova* (engl. *flow–shop*) i *izbor poslova* (engl. *job–shop*). Kod *izbora toka poslova* mora se izvršiti skup različitih poslova uz uporabu skupa resursa i pri tome je redoslijed uporabe resursa za svaki posao isti. Kod drugog tipa problema, *izbor poslova,* redoslijed uporabe resursa kod različitih poslova može biti različit.
- *Problem određivanja rasporeda resursa/osoblja* odnosi se na dodjeljivanje resursa (u smislu kvantitete i koje operacije će obavljati) zadacima. Primjerice, u slučaju određivanja rasporeda zaposlenika, broj zaposlenika koji posjeduju određene vještine ne smije biti veći nego što je broj zaposlenih u organizaciji, inače se moraju novi zaposliti, što povlači dodatne troškove. Dakle, cilj je rasporediti operacije tako da se smanje troškovi [TOD97]. U slučaju strojeva cilj bi trebao biti regulacija potrošnje energije ili nekih drugih resursa [BIE90].
- Kod masovne proizvodnje, *problem usklađivanja proizvodnje na pokretnoj traci* (engl. *The Assembly Line Balancing*): Problem uključuje skup od *n* stanica. Na svakoj stanici se izvodi skup operacija i onda se prosljeđuju sljedećoj stanici. Problem je dodijeliti operacije stanicama na takav način da se proizvodnja odvija bez zastoja [AND01].
- Planiranje potrebnih materijala (engl. *Material Requirements Planning*).
- Planiranje potrebnih kapaciteta (engl. *Capacity Requirements Planning*), tijekom kojeg se određuje kojim redoslijedom se koji resurs/radnik angažira radi izvođenja pojedinih operacija.

5.1.1. Osnovni pojmovi iz teorije rasporeda

U ovom podpoglavlju navedeni su (po abecedi) osnovni i najčešće, u literaturi, korišteni pojmovi iz teorije rasporeda.

Disjunktivno ograničenje (engl. *Disjunctive Constraint*): Ograničenje između para operacija kojim se ne dopušta preklapanje operacija u određenom trenutku. Najčešći razlog je nedovoljno velik kapacitet resursa da istodobno opsluži obje operacije.

Dopušteno odstupanje (engl. *Allowance*): Označava razliku između krajnjeg roka i trenutka kada poslovi postaju raspoloživi.

Kapacitet (engl. *Capacity*): Mjera kojom se izražava koliko se neki resurs može koristiti. Uobičajeni primjer navođenja kapaciteta je broj proizvoda koji su raspoloživi resursu u određenom vremenskom periodu.

Makespan: Pogledati objašnjenje za *Ukupno vrijeme izvođenja*.

Međuspremnik (engl. *Buffer*): Resurs koji služi kao postrojenje za spremanje jednog ili više proizvoda između dviju operacija.

Međusobno isključivanje (engl. *Mutual Exclusion*): Višeradni resurs može istovremeno obavljati samo jedan zadatak (jedna operacija isključuje druge).

Niz (engl. *Sequence*): Predstavlja redoslijed operacija na svakom resursu.

Operacija (engl. *Activity*): Sveobuhvatni naziv za određenu fazu izvođenja posla. Najčešće se naziv operacija odnosi na proces obrade (koji još nije završen) proizvoda na stroju, ali isto tako i neke druge aktivnosti kao transport, "setiranje" resursa itd. mogu se smatrati kao zasebne operacije. Da bi se izvršile, operacije zahtijevaju resurse koji im moraju biti raspoloživi. Katkad se pojam *zadatak* rabi umjesto pojma operacija.

Posao (engl. *Job*): Odnosi se na proces proizvodnje određenog proizvoda ili na obavljanje određenih usluga. Posao je sastavljen od operacija koje se ne bi smjele preklapati.

Predpražnjenje (engl. *No Preemption*): Resurs ne može prekidati posao koji radi i započeti novi, sve dok prethodni posao nije završen.

Prekid poslova (engl. *Job Preemption*): Ovo je postupak privremenog ili potpunog zaustavljanja izvođenja jednog posla u korist drugog. Prekinuti posao kasnije se nastavlja izvoditi. U većini slučajeva ovo se ne dozvoljava. Obično se obavlja kada iskrsne posao visokog prioriteta ili kada na čvoru na kojem se izvodi prvi posao postoji rezervacija.

Protok vremena (engl. *Flow Time*)**:** Označava razliku između vremena završetka posla i vremena otpuštanja.

Resurs (engl. *Resource*): Materijal, novac, stroj, robot, ljudska snaga, odnosno bilo koje sredstvo potrebno da se obavi određeni posao nazivaju se resursi. Resursi su veoma važni za određivanje rasporeda poslova jer oni određuju tip problema koji se rješava. Ako je barem jedan resurs ograničen, onda se problem naziva određivanje rasporeda poslova s ograničenim resursima.

Rezerva (engl. *Slack*): Razlika između preostalog vremena do krajnjeg roka i preostalog vremena izvođenja posla u određenom trenutku. Negativna vrijednost ove veličine znači da će posao, neizbježno, prekoračiti krajnji rok.

Stroj (engl. *Machine*): Najčešći oblik resursa.

Ukupno vrijeme izvođenja (engl. *Total Production Time*): Označava vrijeme potrebno da se izvrše sve operacije. Ako su operacije grupirane u poslove, onda se ukupno vrijeme izvođenja odnosi na ukupno vrijeme potrebno da se završi i posljednji posao.

Uranjenost (engl. *Earliness*): Ako je posao završen prije roka, ova ne-negativna veličina jednaka je razlici krajnjeg roka i trenutka izvršavanja poslova, inače je jednaka nuli.

Uvjeti prednosti na poslove (engl. *Precedence Constraints*): Primjerice, ako posao *Pi* ima prednost nad poslom *Pj* , to znači da izvođenje posla *Pj* ne smije započeti prije izvođenja posla *Pi* ($Pi < Pj$).

Zadatak (engl. *Task*): Vidjeti pojam operacija.

Zastoj (engl. *Deadlock*): Situacija kada dva ili više proizvoda čekaju zbog ograničenog ili nedovoljnog prostora u međuspremniku. Ako niti jedan posao ne može nastaviti svoje izvođenje onda je riječ o *globalnom zastoju*.

5.1.2. Problem rasporeda poslova

Većina problema određivanja optimalnog rasporeda poslova ne može se u praksi riješiti klasičnim tehnikama operacijskog istraživanja, kao što je cjelobrojno programiranje, i jedan je od najtežih kombinatoričkih problema. Vrijeme izračuna globalnog optimuma eksponencijalno raste s povećanjem broja poslova, što čini problem pronalaženja optimalnog rasporeda poslova za projekt s ograničenim sredstvima NP teškim problemom. To je jedan od razloga što je ovaj problem postao subjekt brojnih istraživanja.

Općeniti problem raspoređivanja uključuje dodjeljivanje svakog od N poslova jednom od M resursa i onda je prostor mogućih rješenja jednak M^N, što povlači eksponencijalni rast kao funkcije zavisne o broju poslova i resursa.

Pri rješavanju problema rasporeda poslova potrebno je uzeti u obzir tip procesa za kojeg se pravi raspored. Potrebno je utvrditi, odnosno izdvojiti različite poslove u procesu, koliko se poslova može istodobno obavljati na određenom mjestu, te odrediti značajke procesa koji predstavlja problem rasporeda poslova. Problem se može definirati kako slijedi.

Definicija 5.1 Problem određivanja rasporeda poslova

Tipični *problem rasporeda poslova* (engl. *Job-Shop Scheduling Problem*, pokrata: JSP) se definira kako slijedi:

(i) Postoji skup R od m resursa i skup P od n različitih poslova;

(ii) Svaki posao $P_j \in P, j = 1,...,n$ se sastoji od određenog skupa operacija $t_k[j],\ 1 \le k \le n_j$;

(iii) Svakoj operaciji je zadano vrijeme izvođenja i pridružen određeni resurs $r(t) \in \{1,2,\ldots,m\}$ tako da vrijedi $r(t_k[j]) \neq r(t_h[j])$ za svaki $Pj \in P$ i $1 \le k,h \le n_j,\ k \neq h$;

(iv) Operacije se izvode u slijedu po unaprijed određenom redoslijedu ;

(v) Operacije se ne smiju prekidati;

(vi) Svaki resurs smije izvoditi samo jednu operaciju u određenom trenutku.

Rješavanje problema rasporeda poslova je složen zadatak zbog niže navedenih razloga:

- *Ograničenost resursa*: Ako se u procesu istodobno obavlja više od jednog posla, to dovodi do ograničenosti resursa pri obradi u cijelom procesu;
- *Vremenska ograničenost*: Osnovna vremenska ograničenost odnosi se na radnu smjenu, radni dan, radni mjesec itd. Pored osnovnih ograničenja pojavljuju se posebna ograničenja korisnika, primjerice, vrijeme popravka određenog stroja, nedostupnost potrebnog radnika u određenom trenutku ili ograničenost vremena izvođenja nekog posla (npr. od 7h do 15h);
- *Prioriteti*: Poslovi imaju određene prioritete glede zahtjeva klijenata, njegovih boniteta ili zahtjeva samog procesa (npr. hitan nalog, kvar potrebnog stroja, preventivni popravci). U slučaju gdje je nemoguće rasporediti sve naloge unutar raspoloživog vremena, potrebno je osigurati dodatno vrijeme (produljeni radni dan, radna subota) ili izbaciti naloge s najnižim prioritetima;

- *Prethodnost*: Određeni posao ne može biti izvršeni, dok se ne završe svi poslovi koji su mu prethodili. U tom slučaju, prvo se rasporede oni poslovi koji su nužni za nastavak procesa, a zatim oni poslovi koje proces nastavlja.

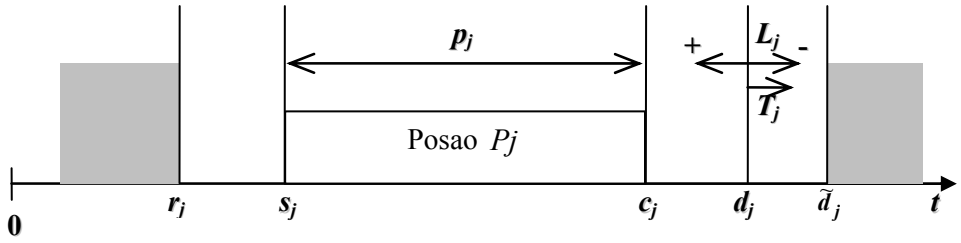
Dakle, u ovisnosti o uvjetima, definiraju se različiti problemi rasporeda. Općenita shema po kojoj se ti problemi klasificiraju ima format:

$$\alpha \mid \beta \mid \gamma, \Gamma \mid \delta$$

- α predstavlja broj poslova
- β predstavlja broj resursa
- γ ukazuje na redoslijed resursa kod poslova (F – izbor slijeda poslova, G – izbor poslova)
- Γ ukazuje na neku specifičnost (npr. *sequence dependent*)
- δ predstavlja kriterij optimalnosti.

Svaki posao $P_j \in P, j = 1,...n$ je određen sljedećim parametrima (slika 5.2).

- **Vrijeme izvođenja** (engl. *Processing Time*), p_j, je vrijeme potrebno za izvršavanje posla. Najčešće je to vrijeme fiksno, a može ovisiti o resursima koji se rabe.
- **Raspoloživo vrijeme** r_j (engl. *Ready Time*, *Release Date*) je vrijeme otpuštanja, odnosno vrijeme kada posao ili zadatak postaje raspoloživ.
- **Dogovoreni rok** (engl. *Deadline*), $\tilde{d}_j$, označava vremenski rok do kada posao mora biti završen, inače se pretpostavlja da je raspored pogrešan.
- **Termin** (engl. *Due Date*), d_j, označava vremenski rok do kada poslovi moraju biti izvršeni. Termin je ograničenje koje se može prekoračiti, ali su to onda dodatni tzv. *kazneni troškovi*.
- **Početak izvršenja poslova** (engl. S*tarting Time*), s_j, označava početak izvršavanja posla. Može se govoriti o početku izvršavanja cijelog posla, a pri tome se misli na početak izvršavanja njegove prve operacije.
- **Trenutak izvršenja poslova** (*engl*. *Completion Time*), c_j, je vrijeme kada je posao završen. Isto tako, ovaj pojam se može odnositi i na operaciju.
- **Vrijeme setiranja** (engl. *Set up Time*) δj, je vrijeme potrebno za "postavljanje" ili "setiranje" resursa kako bi resurs počeo izvršavati određeni posao.
- **Vrijeme kašnjenja** (engl. *Tardiness*), T_j : Ako je vrijeme potrebno da se izvrše poslovi veće od zadanog roka, vrijeme kašnjenja se računa kao razlika između vremena izvođenja poslova i zadanog roka. Inače je vrijeme kašnjenja jednako nuli. Optimalizacija temeljena na vremenu kašnjenja se rabi u slučajevima gdje je nepoželjno kašnjenje s isporukom proizvoda.
- **Kašnjenje** (engl. *Lateness*), L_j, se definira se kao razlika trenutka izvršavanja poslova i termina. Ova veličina je negativna ako je posao završen prije definiranog roka.



Slika 5.2 Grafički prikaz parametara posla $Pj \in J$

5.1.2.1. Klasifikacija problema rasporeda poslova

Ako su vrijeme izvođenja i svi ostali parametri, za svaki posao, poznati bez neizvjesnosti i fiksni onda se za problem određivanja rasporeda kaže da je *deterministički.* Kod problema koji se svrstavaju u *ne-determinističke (stohastičke)* neki ili svi parametri su neizvjesni.

Druga podjela je na *statičke* i *dinamičke* probleme. Kod statičkih problema rasporeda poznati su svi poslovi ili još općenitije, operacije koje se moraju izvesti unutar nekog planiranog vremena. Problem rasporeda poslova se može sagledati kao tablicu vremena $s_{j,k}$ (početaka izvođenja *k*-te operacije posla P_j) obzirom na raspored poslova po resursima. Trenutak izvršenja operacije je rezultat zbroja početka izvođenja operacije i vremena izvođenja operacije $c_{j,k} = \left(s_{j,k} + p_{j,k}\right)$. Najranije vrijeme kada može započeti izvođenje operacije određeno je kao veći broj između trenutka izvršenja prethodne operacije istog posla P_j i trenutka izvršenja prethodne operacije na istom resursu. Dakle, početak izvođenja operacije $o_{j,k}$ se računa sljedećim izrazom

$$s_{j,k} = max\left(s_{j,k-1} + p_{j,k-1}, s_{h,l} + p_{h,l}\right). \qquad (5\text{-}1)$$

Trenutak izvršenja posla P_j određen je trenutkom izvršenja njegove posljednje operacije.

Kod dinamičkog problema rasporeda, mogu se dodati novi poslovi (operacije) u toku vremenskog perioda određenog rasporedom. Dinamički problemi su po svojoj prirodi ne-deterministički, ali to ne povlači da su statički problemi i deterministički.

Početak izvođenja početne operacije posla P_j se odredi kao veći broj između trenutka r_j kada posao P_j postaje raspoloživ i trenutka izvršenja operacije $o_{h,l}$, koja je prethodno izvršena na istom resursu,

$$s_{j,1} = max\left(r_j, s_{h,l} + p_{h,l}\right) \qquad (5\text{-}2)$$

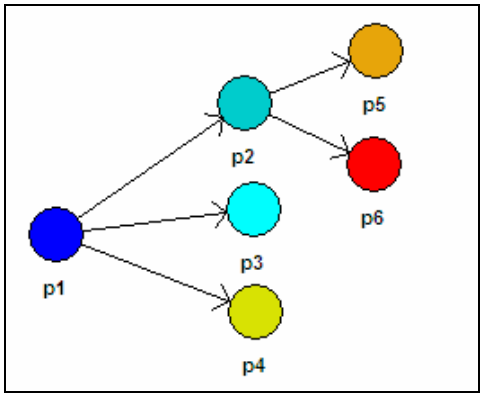
Za operacije koje nisu početne operacije, početak izvođenja se računa u skladu s izrazom (5-1). Uvijek kada se razmatraju i raspoloživa vremena pri rješavanju problema rasporeda poslova riječ je o dinamičkom problemu rasporeda poslova.

Problemi rasporeda se često kategoriziraju u sljedeće klase:

- Raspored, kod kojeg niti jedna operacija ne može započeti ranije, a da se pri tome ne promijeni redoslijed obrada operacija, naziva se *poluaktivan* (engl. *semiactive*);
- Raspored, kod kojeg niti jedna operacija ne može započeti ranije, bez kašnjenja barem jedne od preostalih operacija, naziva se *aktivan* (engl. *active*);
- Raspored se naziva raspored *bez kašnjenja* (engl. *non-delay*) ako nema resursa koji je u stanju mirovanja dok određeni posao/operacija čeka na izvođenje. Optimalan raspored ne mora nužno biti raspored bez kašnjenja.

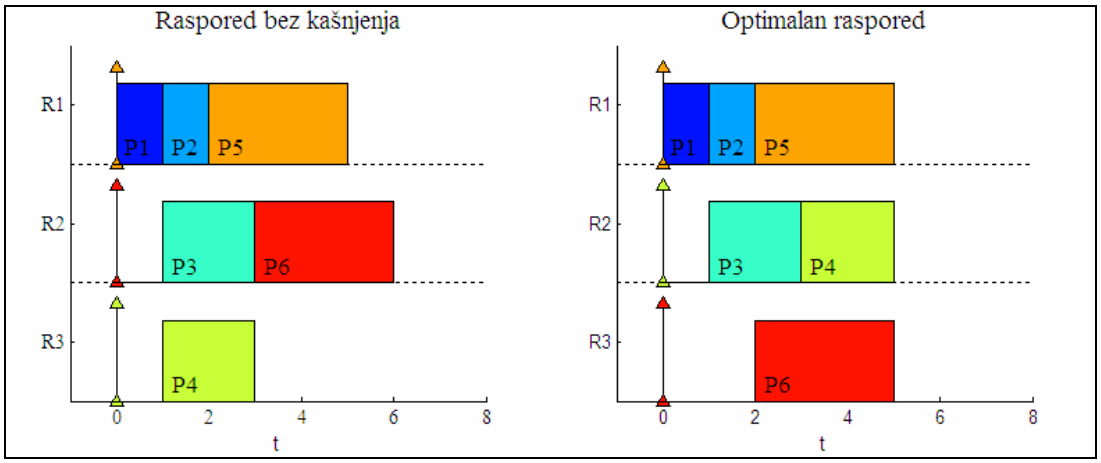
Primjer 5.1 Raspored bez kašnjenja

U ovom primjeru razmatrat će se raspored sa tri resursa i šest poslova P*i* (i = 1, …, 6). (Pretpostavka je da resursi izvode poslove, a ne operacije i svaki posao se može obaviti u bilo kojem resursu.) Vrijeme izvođenja poslova zadano je nizom $p = (1,1,2,2,3,3)$, gdje je p_i broj vremenskih jedinica potreban za izvođenje posla P*i*. Poslovi P2, P3, P4 ne mogu započeti prije nego završi posao P1, a poslovi P5 i P6 nakon što završi posao P2. Slika 5.3(a) prikazuje graf opisanog problema, pri čemu čvorovi predstavljaju poslove. (Svaki posao je predstavljen drugom bojom.) Slika 5.3(b) prikazuje Gannt dijagrame [12] rasporeda bez kašnjenja i optimalnog rasporeda. Može se uočiti da kod rasporeda bez kašnjenja vrijeme potrebno za izvođenje svih poslova iznosi 6 vremenskih jedinica, dok kod optimalnog rasporeda ono iznosi 5 vremenskih jedinica.



(a)

[12] Gantt dijagram je dijagram prikazan vodoravnim stupićima koji grafički prikazuje odnos među različitim operacijama /poslovima: X-os predstavlja vrijeme, Y-os predstavlja resurse, a boja ili stil uzorak mogu se koristiti kako bi se istakli pojedini poslovi.



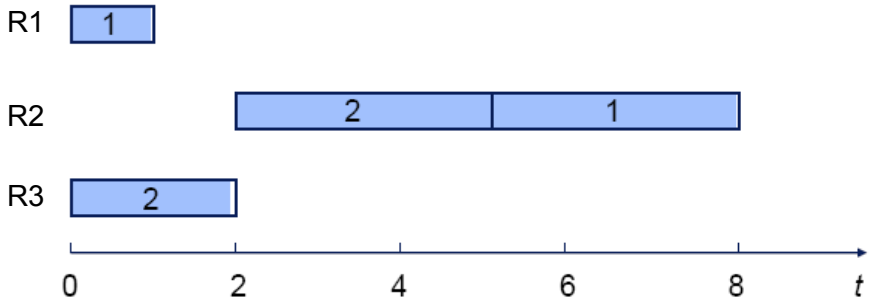
(b)

Slika 5.3 Rasporeda bez kašnjenja $P3|prec|c_{max}$: (a) Opis problema, (b) Gannt dijagrami

■

Primjer 5.2 Aktivan raspored

U ovom primjeru razmatrat će se raspored sa tri resursa i dva posla. Za izvođenje posao P1 treba 1 jedinicu vremena na resursu R1 i 3 jedinice vremena na resursu R2. Vrijeme izvođenja posla P2 na resursu R2 iznosi 3 jedinice vremena, a na resursu R2 2 jedinice vremena. I posao P1 i posao P2 moraju završiti izvođenje na resursu R2.



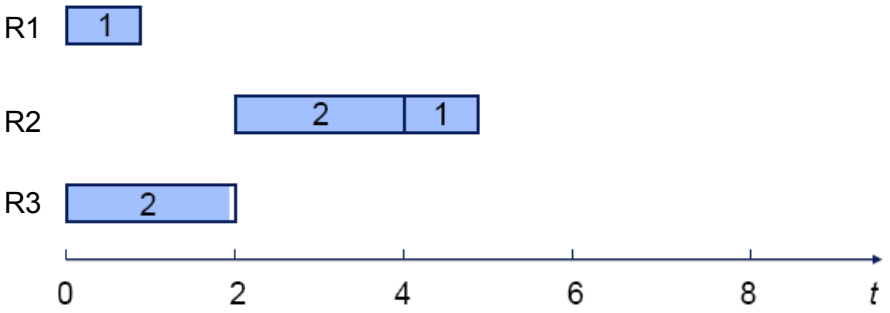
Slika 5.4 Ganttov dijagram mogućeg rješenja za problem izvođenja iz primjera 5.2

Ovaj raspored je aktivan jer kad bi se zamijenio redoslijed poslova P2 i P1 na resursu R2 odgodilo bi izvođenje posla P2. Raspored nije bez kašnjenja i nije optimalan. Resurs R2 je u stanju mirovanja sve do trenutka t=2 dok istovremeno postoji posao koji čeka na izvođenje u trenutku t=1.

■

Primjer 5.3 Poluaktivan raspored

Kao i u prethodnom primjeru i u ovom primjeru razmatrat će se raspored sa tri resursa i dva posla. Vremena izvođenja posla P1 na resursima R1 i R2 su jednaka i iznose 1 jedinicu vremena. Vremena izvođenja posla P2 na resursima R2 i R3 su jednaka i iznose 2 jedinice vremena.



Slika 5.5 Ganttov dijagram mogućeg rješenja za problem izvođenja iz primjera 5.3

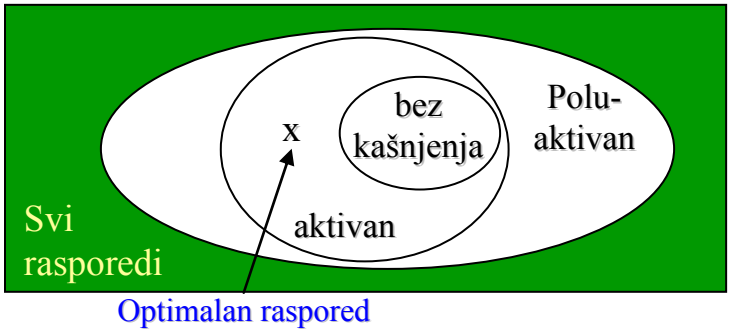
Razmatra se raspored kod kojeg se posao P2 izvodi na resursu R2 prije posla P1. To povlači da posao P2 započinje svoje izvođenje na resursu R2 u trenutku t=2, a posao P1 započinje svoje izvođenje na resursu R2 u trenutku t=4. Ovaj raspored je polu-aktivan. Nije aktivan raspored budući da se posao P1 može izvesti na resursu R2 bez kašnjenja u izvođenju posla P2 na resursu R2.

■

Vrijedi sljedeće:

- Svaki raspored bez kašnjenja je aktivan.
- Svaki aktivan raspored je poluaktivan.
- Ako je funkcija cilja regularna[13], skup aktivnih rasporeda sadrži optimalan raspored.

Odnos između aktivnog i poluaktivnog rasporeda te rasporeda bez kašnjenja prikazan je Venn-ovim dijagramima na slici 5.6.



Slika 5.6 Klasifikacija rasporeda

5.1.2.2. Konflikti među resursima i zastoji

Konflikti nastaju kada dva posla zahtijevaju uslugu istog resursa u istom radnom ciklusu. Primjerice, može se dogoditi da operacija učitavanja treba zbrajalo kako bi se izračunala memorijska adresa s koje se podatak učitava, dok se isto vrijeme izvodi operacija zbrajanja.

[13] regularan = ne smanjuje se obzirom na vrijeme završetka izvođenja

Ako skup resursa R dijele dva posla $P1$ i $P2$ i ako se izvodi posao $P1$ koji je zauzeo resurse, onda posao $P2$ mora čekati da posao $P1$ završi i oslobodi resurse. Ako poslovi $P1$ i $P2$ dijele resurs $R1$ to se označava $P1 \otimes P2 = R1$.

Sljedeća definicija opisuje konflikt između poslova *Pi* i *Pj* za zajedničkim resursom.

Definicija 5.2 Konflikt

Neka postoji skup R od m resursa i skup P od n različitih poslova. Za bilo koja dva posla $Pi, Pj \in P \ (Pi \neq Pj)$, Pi je u *konfliktu* s Pj za resursom $R1$ ako i samo ako vrijedi:

(i) $Pi \otimes Pj = R1$;

(ii) vremenski intervali $\left[T_{početak}(Pi), T_{kraj}(Pi)\right]$ i $\left[T_{početak}(Pj), T_{kraj}(Pj)\right]$ se poklapaju.

Uvjeti za postojanje konflikta među poslovima Pi, Pj s ograničenim resursima su:

(i) dva posla Pi, Pj dijele resurse;

(ii) vremena izvođenja poslova Pi, Pj se poklapaju.

Ako za dva posla vrijedi $P1 \otimes P2 = R1$, ali im se vrijeme izvođenja ne poklapa, primjerice, $\left[T_{kraj}(Pi) \leq T_{početak}(Pj)\right]$, Pj počinje s izvođenjem nakon što završi Pi, onda među njima ne postoji konflikt. Poslovi Pi, Pj koji su u konfliktu za resursima označavaju se $Pi \bowtie Pj$.

Definicija 5.3 Postojanost na resurse (engl. *Resource Consistency*)

Neka je R skup od m resursa i P skup od n različitih poslova. Sustav je *postojan* na resurse ako i samo ako za bilo koja dva posla $P_i, P_j \in P$ vrijedi da nisu u konfliktu, odnosno ne postoje $Pi, Pj \in P$ takvi da vrijedi $Pi \bowtie Pj$.

Zastoj (engl. *deadlock*) je jedan od najčešćih problema koji se mogu javiti u sustavima s višeradnim resursima, gdje se istovremeno odvija više procesa. Fleksibilni proizvodni sustav s autonomnim vozilima je jedan primjer takvog sustava. Primjeri zastoja iz svakodnevnog života su npr. čepovi u prometu koji se stvore za vrijeme velikih gužvi. Zastoj je situacija kada neki od automobila mora voziti unazad da bi svi ostali mogli nastaviti vožnju. Kada se govori o računalima zastoji su vrlo čest problem u operacijskim sustavima. Primjerice, ako dvije operacije zauzmu cijelu radnu memoriju, a nijedna od njih još nije završila i obje trebaju još memorije za završetak. Uz zastoj se veže i pojam neodređenog odgađanja izvršenja zadataka (engl. *indefinite postponement, livelock*). Kao primjer može se navesti još jedna situacija iz računalnog svijeta kada operacije visokog prioriteta drže procesor zauzetim 100% vremena, čime se uskraćuje izvođenje operacija nižeg prioriteta.

Dakle, zastoj je situacija u kojoj dvije ili više operacija čekaju jedna na drugu. Operacije čekaju da neka od drugih operacija završi, ali ni jedna ne može završiti, zato što neka druga operacija zauzima resurse koji su joj potrebni za izvođenje. To je situacija u kojoj se operacije same ne mogu osloboditi zastoja.

Resursi su sredstva, tj. objekti dodijeljeni procesima. Resursi mogu biti npr. strojevi za obradu ili ako govorimo o računalnim procesima mogu biti računalne komponente kao što su memorija, pisači, tvrdi diskovi itd. Resursi mogu biti:

- Zaposjedljivi (engl. *preemptable*);
- Nezaposjedljivi (engl. *nonpreemptable*).

Uvjeti za zastoj

Postoje 4 uvjeta i sva četiri moraju nužno biti zadovoljena da bi sustav došao u zastoj. Ako neki od ovih uvjeta nije zadovoljen opasnost od zastoja ne postoji. Teoretski se zbog toga zastoj može spriječiti nezadovoljavanjem nekog od ova 4 uvjeta, međutim često je u praksi to teško izvesti. Uvjeti su sljedeći:

- Uzajamna isključivost;
- Držanje i čekanje;
- Postojanje nezaposjedljivih resursa;
- Kružno čekanje.

Uzajamna isključivost znači da više procesa koristi isti resurs i to ne nužno istovremeno. Kada bi svaki proces imao svoje resurse, koje drugi procesi ne bi nikada smjeli koristiti (pa čak ni kada su ti resursi slobodni), ne bi postojala opasnost od zastoja, međutim u realnim sustavima to gotovo nikada nije zadovoljeno jer bi tada iskoristivost resursa i učinkovitost sustava bili vrlo mali.

Držanje i čekanje znači da proces drži neki resurs i čeka da se oslobode ostali resursi potrebni za izvršavanje zadatka. Ova situacija se može spriječiti tako da proces unaprijed alocira sve resurse koje treba za izvršenje zadatka ili ako je već neke resurse alocirao da ih otpusti i potom ponovno zatraži zajedno s onima koji su mu ranije nedostajali. Oba načina prevencije situacije *drži i čekaj* su vrlo neučinkovita i lako mogu dovesti do neodređenog odgađanja izvršenja zadatka, a često ih i nije moguće izvesti (npr. otpuštanje već alociranih resursa).

Kako nezaposjedljiv resurs ne može biti jednostavno oduzet procesu koji ga već koristi, to je problem ako taj isti resurs traži neki drugi proces. Potencijalno rješenje je da svaki posao/operacija polaže isključiva prava na resurse koje koristi, međutim u stvarnosti to nije moguće ostvariti, jer bi to značilo da npr. za svaki proces koji se odvija u našem računalu moramo imati posebnu pločicu radne memorije.

Kružno čekanje je situacija kada dva ili više poslova čekaju na resurs koji drži netko drugi u lancu poslova. Kružno čekanje nužan je preduvjet za zastoj sustava, ali nije nužno da će ga i izazvati. Ako resursi u kružnom čekanju prijeđu u stanje kružnog blokiranja, tada je zastoj sustava neminovan [LEE07]. *Kružno blokiranje* je situacija u kojoj niti jedan resurs, koji pripada kružnom blokiranju, više ne može postati dostupnim - *umrtvljeni* resursi.

Moguće rješenje bi bilo postojanje posebnih resursa za svaki posao, ali to je neekonomično u stvarnom svijetu. Rješenje ovog problema je ostvareno algoritmima upravljanja resursima koji prepoznaju kritične situacije u sustavu i zabranjuju dodjeljivanje novih resursa, dok se kritična situacija ne riješi. Iako ni ovo nisu jednostavni problemi, može se reći da se cirkularno čekanje može jednostavnije spriječiti nego prethodna 3 uvjeta.

5.1.3. Optimalizacija rasporeda poslova

Određivanje rasporeda je u svojoj osnovi određivanje redoslijeda operacija za obavljanje određenog posla, koji zahtijeva određeno vrijeme i određene resurse. Cilj optimalizacije je pronaći redoslijed operacija za izvršavanjem svih poslova, a pri tome da se zadovolji jedan ili više kriterija (primjerice, minimalno prosječno protočno vrijeme svih poslova ili minimalno prosječno kašnjenje) i ograničenja.

Kad je napravljen raspored za određenu listu poslova, potrebno je pronaći takav raspored, koji predstavlja dobar raspored. Pri tomu se može primijeniti GA. Dobar raspored je onaj koji zadovoljava sve kriterije i istodobno je dovoljno dobar, da ga je moguće primijeniti u stvarnosti (primjerice, u proizvodnji). *Optimalizacija* je proces traženja boljeg rasporeda od dobrog uvažavajući kriterije i ograničenja. Optimalan raspored poslova je najbolje moguće rješenje određenog problema.

Sljedećim primjerom je prikazan jedan problem rasporeda.

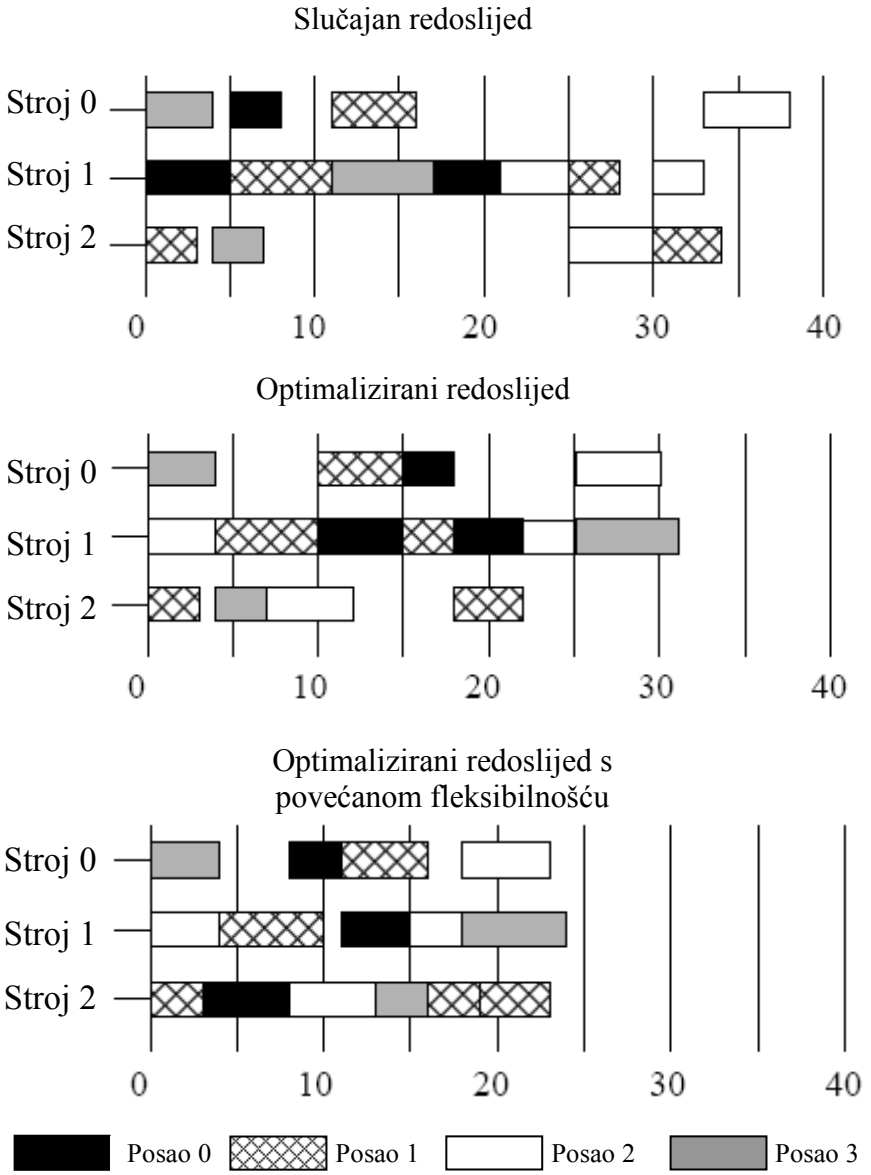
Primjer 5.4 Problem rasporeda poslova u proizvodnom sustavu [TOD97]

Tablica 5.1 prikazuje primjer proizvodnog sustava i raspored četiri posla na tri resursa. U ovom slučaju resursi su strojevi S0, S1, S2. Izraz Si,j označava da je stroj Si (i=0,1,2) potreban za izvođenje operacije, a broj j koji slijedi označava broj vremenskih jedinica koliko traje izvođenje te operacije.

Tablica 5.1 Problem rasporeda poslova

Strojevi S0, S1, S2	Operacija 0	Operacija 1	Operacija 2	Operacija 3	Operacija 4
Posao 0	S1, 5	S0, 3	S1, 4		
Posao 1	S2, 3	S1, 6	S0, 5	S1, 3	S2, 4
Posao 2	S0, 4	S2, 3	S1, 6		
Posao 3	S1, 4	S2, 5	S1, 3	S0, 5	

Redoslijed poslova na svakom stroju je veoma važan, što je i prikazano slikom 5.7. Prvi slučaj je slučajno generiran i ukupno vrijeme izvođenja svih poslova iznosi 38 vremenskih jedinica. Značenje optimalizacije jasno se može uočiti u drugom primjeru. Vrijeme izvođenja svih poslova može se smanjiti na 31 vremensku jedinicu. Ovaj utjecaj optimalizacije dobiva daleko više na važnosti što je problem veći. Posljednji primjer pokazuje što se može postići ako se poveća fleksibilnost strojeva u tom smislu da se na istoj razini posao može izvoditi na više od jednog stroja. U ovom primjeru, stroju 2 se dopušta da obavlja poslove stroja 1 kao da su njegovi. Na ovaj način vrijeme izvođenja poslova je reducirano na 24 vremenske jedinice.



Slika 5.7 Utjecaj redoslijeda poslova

■

5.1.3.1. Kriteriji optimalnosti

Raspored je određen s početkom izvršenja poslova s_i, $i = 1 \ldots n$ i trenutkom završetka poslova c_i, $i = 1 \ldots n$. Neka $c_{max} = max\{c_1, \ldots, c_n\}$ označava *vrijeme potrebno da se izvrše svi poslovi* (engl. *schedule time, total production time, makespan*). Pri određivanju rasporeda poslova često se zahtijeva minimalizacija funkcije c_{max}. Ova funkcija se optimalizira skraćivanjem vremenskog perioda potrebnog kako bi se izvršili svi poslovi. Rješenje ovog tipa problema generira vremenski kritičnu stazu (engl. *time-critical path*).

Kao kriterij optimalnosti temeljen na *vremenu završetka poslova* može se uzeti i težinska suma vremena završetaka poslova (engl. *weighted sum of completion times),* odnosno

$$min \sum_{i=1}^{n} w_i c_i . \tag{5-3}$$

Prosječno vrijeme završetka poslova (engl. *average completion time*)

$$min \frac{1}{n} \sum_{i=1}^{n} c_i . \tag{5-4}$$

U dinamičkom modelu c_{max} ovisi o trenutku kada je raspoloživ posljednji posao koji je izvršen. Stoga se c_{max} ne može uzeti kao odgovarajuća mjera kvalitete modela rasporeda. Zato se definira tzv. *protok vremena* (engl. *flow time*) kao $F_i = c_i - r_i$, odnosno prosječni protok vremena $\overline{F} = \frac{1}{n} \sum_{i=1}^{n} \left(c_i - r_i \right)$.

Kriteriji optimalnosti temeljeni na *protoku vremena* su sljedeći:

$$\min F_{\max}, \ \min \sum_{i=1}^{n} F_i, \ \min \sum_{i=1}^{n} w_i F_i, \ \min \frac{1}{n} \sum_{i=1}^{n} F_i \tag{5-5}$$

Ako se posao ne izvršava, a na raspolaganju je, kaže se da posao čeka, pa se definira vrijeme čekanja (engl. *waiting time*) kao $W_i = c_i - \left(r_i + \sum_{j=1}^{m} p_{ij} \right)$.

Kriteriji optimalnosti temeljeni na *vremenu čekanja* su sljedeći:

$$min W_{max}, \ min \sum_{i=1}^{n} W_i , \ min \sum_{i=1}^{n} w_i W_i , \ min \frac{1}{n} \sum_{i=1}^{n} W_i \tag{5-6}$$

Minimalizacija cijene

Kod ovakvih tipova optimalizacije problema rasporeda poslova, cijene nastale zbog troškova obavljanja poslova, uporabe resursa i dodatnih troškova zbog preuranjenost/kašnjenja poslova, nastoje se minimalizirati. Ovakva objektna funkcija se temelji na mjerenju vremena potrebnog da se posao obavi.

Slijede neki primjeri kako se može zadati objektna funkcija kod ovakvog tipa raspoređivanja poslova.

- Kriteriji optimalnosti temeljeni na *kašnjenju*:

Ako su zadani i rokovi izvršenja poslova d_i, definira se tzv. kašnjenje (engl. *lateness*) $L_i = C_i - d_i$. U tom slučaju definira se neka od sljedećih objektnih funkcija:

$$min\, L_{max},\ min \sum_{i=1}^{n} L_i,\ min \sum_{i=1}^{n} w_i L_i,\ min \frac{1}{n} \sum_{i=1}^{n} L_i \tag{5-7}$$

Jasno, ako je $C_i < d_i$, onda je $L_i = C_i - d_i < 0$. Dakle, dozvoljeni su negativni troškovi. Pozitivni troškovi ($L_i > 0$) nastaju samo u slučaju prekoračenja roka d_i. U tu se svrhu definiraju tzv. troškovi kašnjenja (engl. *Tardiness)* kao $T_i = max(0, L_i)$.

- Kriteriji optimalnosti temeljeni na *tardiness*:

$$min\, T_{max},\ min \sum_{i=1}^{n} T_i,\ min \sum_{i=1}^{n} w_i T_i,\ min \frac{1}{n} \sum_{i=1}^{n} T_i \tag{5-8}$$

- Kriteriji optimalnosti temeljeni na *vremenu početka poslova:*

$$min \sum_{i=1}^{n} s_i,\ min \sum_{i=1}^{n} w_i s_i,\ min \frac{1}{n} \sum_{i=1}^{n} s_i \tag{5-9}$$

- Kriteriji optimalnosti temeljeni na *vremenu setiranja resursa:*

Kako bi se modeliralo neraspoloživost resursa uvedeno je inicijalno vrijeme "setiranja" δ_j za stroj. Vrijeme "setiranja" δ_j ukazuje na najraniji trenutak kada je stroj M_j ponovo raspoloživ. Računa se kao

$$\delta_j = max\left(\max_{1 \le i \le n} \{ s_{i,k} + p_{i,k} \mid s_{i,k} < s_1 \}, s_1 \right) \tag{5-10}$$

Za posao *i* koji se obavljaju na resursu *k* vrijeme početka izvođenja se računa kao

$$s_{i,k} = max\left(r_i, \delta_{\mu_i(k)} \right) \tag{5-11}$$

U tom slučaju definira se neka od sljedećih objektnih funkcija:

$$min\, \delta_{max},\ min \sum_{k} \delta_k,\ min \sum_{k} w_k \delta_k, \tag{5-12}$$

Maksimalizacija kvalitete

Maksimalizacija kvalitete obavljanja poslova je jedan od važnijih ciljeva za upravljanje sustavima. To je razlog zašto se maksimalizacija kvalitete, također, uzima kao funkcija cilja u problemima određivanja rasporeda poslova.

5.1.3.2. Pravila prioriteta za određivanje rasporeda

Pravila prioriteta se proučavaju preko 30 godina. Strategija kojom se odabire posao među onima koji čekaju na izvođenje svaki put kada je resurs u stanju mirovanja naziva se *pravilo prioriteta.* Prioritet se može odrediti na temelju nekog od sljedećih pravila:

- FCFS (*first-come, first served*) – spontano, ako nema posebnih pravila;
- LCFS (*last-come, first served*) – ako se prispjeli predmeti za obradu slažu jedan povrh drugoga;
- DDATE (*earliest due date*) – prvo poslovi čiji termin najranije dospijeva;
- CUSTPR (*highest customer priority*) – prvo poslovi za prioritetne kupce;
- SETUP (*similar setup*) – najprije slični poslovi za koje treba najmanje podešavanja strojeva;
- SPT (*shortest processing time*) – prioritet poslovima koji najkraće traju;
- LPT (*longest processing time*) – prioritet poslovima koji najdulje traju.

Prioritet DDATE može imati varijante:

– SLACK (*minimum slack*) – prioritet imaju oni poslovi koji imaju manju vremensku rezervu:

SLACK=(datum dospijeća – današnji datum) – (preostalo vrijeme obrade)

– CR (*smallest critical ratio*) – prioritet imaju poslovi s manjim odnosom preostalog vremena do dospijeća i preostalog vremena obrade:

CR=(datum dospijeća-današnji datum)/(preostalo vrijeme obrade)

Ako je CR>1, onda je posao u pretjecanju

Ako je CR<1, onda je posao u kašnjenju

Ako je CR=1, onda se posao odvija pravovremeno.

Uporaba pravila prioriteta ima određene prednosti kao što je jednostavna implementacija, pronalaženje prihvatljivo dobrog rješenja u relativnom kratkom vremenu, brzina. Nedostaci pravila prioriteta su ograničena primjena u praksi te mogućnost pronalaženja nepredvidljivo lošeg rješenja. Kombiniranjem osnovnih pravila može se značajno poboljšati izvođenje.

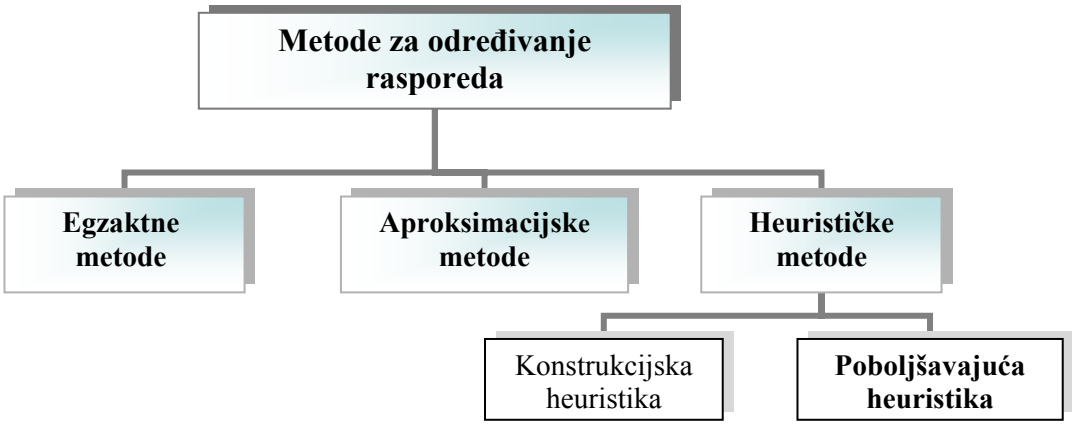
5.1.4. Postojeće metode za rješavanje problema rasporeda poslova

Različiti analitički modeli i optimalizacijski algoritmi su razvijeni za rješavanje statičkih i deterministčkih problema rasporeda [BRU94, BRU95]. U teoriji je moguće odrediti optimalan raspored za statičke ili dinamičke deterministčke probleme rasporeda. U praksi, većina stvarnih problema rasporeda su presloženi da bi se matematički formulirali. Teško je odrediti optimalan raspored poslova ako je uključeno mnogo poslova i resursa (primjerice, 100 poslova i 10 strojeva). Višestruka nelinearnost i kombinatorička povezanost čine problem rasporeda nerješivim klasičnim matematičkim metodama optimalizacije [BER01]. Naime, razvijeno je nekoliko metoda i sve one se mogu svrstati u tri značajne skupine (slika 5.8).

- *Egzaktne optimalizacijske metode* kao što su grafovi, dinamičko programiranje, metoda grananja i granica, su pokazale svoju ograničenost jer zahtijevaju značajno vrijeme

računanja kako bi postigle optimalno rješenje. Naime, samo za manje probleme (primjerice, broj poslova manji od 20 i broj strojeva manji od 10) moguće je odrediti optimalne rasporede primjenom neke od navedenih metoda [BRU94].

- *Aproksimacijske metode*, primjerice metode apsolutne aproksimacije, relativne aproksimacije, polinomijalne aproksimacijske sheme, pronalaze rješenja za koja se jamči da odstupaju od stvarnog optimuma u određenom postotku.
- *Heurističke metode* su se pokazale dobrom alternativom za rješavanje velikih problema, koje osiguravaju vrijeme izvođenja od svega nekoliko minuta. Kako se posljednjih godina stalno poboljšavala snaga procesora elektroničkih računala, to je omogućilo primjenu heurističkih pristupa, koji se temelje na uporabi lokalnog pretraživanja pri rješavanju problema optimalizacije rasporeda poslova.
  - Konstrukcijska heuristika je algoritam koji gradi rješenje prema određenim konstrukcijskim pravilima (primjerice pohlepni algoritam, engl. *Greedy Algorithm*).
  - Poboljšavajući algoritmi odabiru rješenje i usavršavaju ga kroz niz iteracija (primjerice algoritam simuliranog kaljenja).



Slika 5.8 **P**ostojeće metode za određivanje rasporeda

Osman & Potts (citirano u [MUR97]) su predložili uporabu simuliranog kaljenja. Taillard (1994), Nowicki & Smutnicki (1996) (citirano u [MUR97]) su predložili tabu heurističko pretraživanje. U posljednje vrijeme sve više autora primjenjuju genetske algoritme [DAV85, GOL89] na kombinatoričke probleme kao što su problemi trgovačkog putnika (primjerice Fang [FAN94]) i problemi rasporeda (primjerice, Biegel & Davern [BIE90], Croce [CRO95]). Pregled i usporedba ovih metoda za rješavanje deterministčkih problema rasporeda može se pronaći u [BRU95].

Uglavnom, cilj svih ovih procedura je minimalizirati ukupno vrijeme izvođenja poslova, ukupno vrijeme od početka prvog posla do završetka posljednjeg. Međutim, kod problema rasporeda poslova postoje i neki drugi važni faktori kao što su dodjela resursa, ograničenje na dopušteni broj operacija koje su u procesu, završetak poslova koje je određeno postavljenim rokovima, veze i odnosi među operacijama te konflikti za zajedničkim resursima. Ako se nastoje optimalizirati kombinacije ovih faktora doći će se do balansiranja rješenja u skladu s kriterijem.

5.2. Implementacija genetskog algoritma za određivanje rasporeda

Za izgradnju optimalizacijskog algoritma, koji implementira GA, moraju se definirati četiri komponente, koje su nužne za djelovanje GA. Potrebno je odrediti sintaksu kromosoma, interpretaciju i vrednovanje kromosoma, kao i skup genetskih operatora, koji operiraju nad kromosomima.

5.2.1. Sintaksa kromosoma

Vrlo važno pitanje pri izgradnji genetskog algoritma za rješenje problema rasporeda poslova je kako predstaviti rješenja problema zajedno s odgovarajućim genetskim operatorima, tako da bi svi kromosomi, koji se generiraju ili u inicijalnoj fazi ili u evolucijskom procesu, predstavljali mogući raspored. Ovo je vrlo važan korak genetskog algoritma koji utječe i na sve ostale korake genetskog algoritma. Kromosomi se mogu predstaviti kao niz binarnih, cijelih ili realnih brojeva. Svi načini kodiranja mogu se podijeliti u izravne i neizravne načine kodiranje. Pri izravnom kodiranju koriste se nizovi znakova koji izravno predstavljaju raspored i istog optimaliziraju. Kod neizravnog kodiranja u nizovima su kodirana pravila raspoređivanja koja se rabe za određivanje rasporeda, dakle razvijaju se bolja pravila. Posljednjih godina, uobičajeni načini predstavljanja kromosoma su sljedeći:

Izravno kodiranje:

1) ***Kodiranje koje se zasniva na operacijama*** – Ovaj način kodira raspored kao niz operacija i svaki gen u kromosomu predstavlja jednu operaciju. Redoslijed operacija određen je redoslijedom gena u nizu. Uporaba permutacijskog niza je jedan od načina kako se mogu imenovati operacije.

Primjer 5.5 Problem rasporeda tri posla na tri stroja

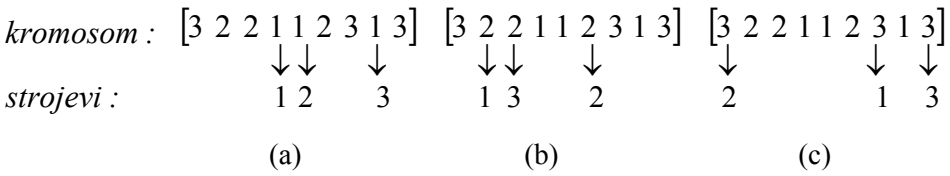
U ovom primjeru razmatra se problem rasporeda tri posla na tri stroja predstavljen tablicom 5.2.

Tablica 5.2 Problem rasporeda tri posla na tri stroja

Strojevi S0, S1, S2 / Poslovi	Operacije		
	O1	O2	O3
P1	S1, 3	S2, 3	S3, 2
P2	S1, 1	S3, 5	S2, 3
P3	S2, 3	S1, 2	S3, 3

Neka je kromosom zadan kao niz $[3\ 2\ 2\ 1\ 1\ 2\ 3\ 1\ 3]$, pri čemu 1 označava posao P1, 2 posao P2 i 3 posao P3. Kako svaki posao ima tri operacije, svaki broj (posao) pojavljuje se točno tri puta u kromosomu. Primjerice, u kromosomu se tri puta pojavljuje 2, što označava tri operacije od posla P2. Prvi put simbol 2 označava prvu operaciju posla P2 koja će se izvesti na stroju S1, drugi put 2 odgovara drugoj operaciji od P2 koja će se izvesti na stroju S3 i treći put 2 označava treću operaciju posla P2 koja će se izvesti na stroju S2. Može se zapaziti da su sve operacije posla P2 imenovane istim simbolom 2 i tumače se prema redoslijedu

pojavljivanja simbola 2 u nizu promatranog kromosoma. Slika 5.9 prikazuje odgovarajuće relacije operacija poslova i strojeva.



Slika 5.9 Operacije poslova i odgovarajući strojevi:
(a) za posao P1, (b) za posao P2, (c) za posao P3

Prema tim relacijama može se izvesti odgovarajuća lista strojeva, primjerice [2 1 3 1 2 2 1 3 3] (slika 5.10). Može se uočiti da redoslijed izvođenja operacija za stroj S1 glasi 2-1-3 (označeno zasjenjenim ćelijama), za stroj S2 3-1-2 i za stroj S3 2-1-3. Iz toga se može odrediti mogući raspored poslova kako je grafički prikazano (Gantt dijagram) na slici 5.11[14]. O_{ij} označava j-tu operaciju posla Pi.

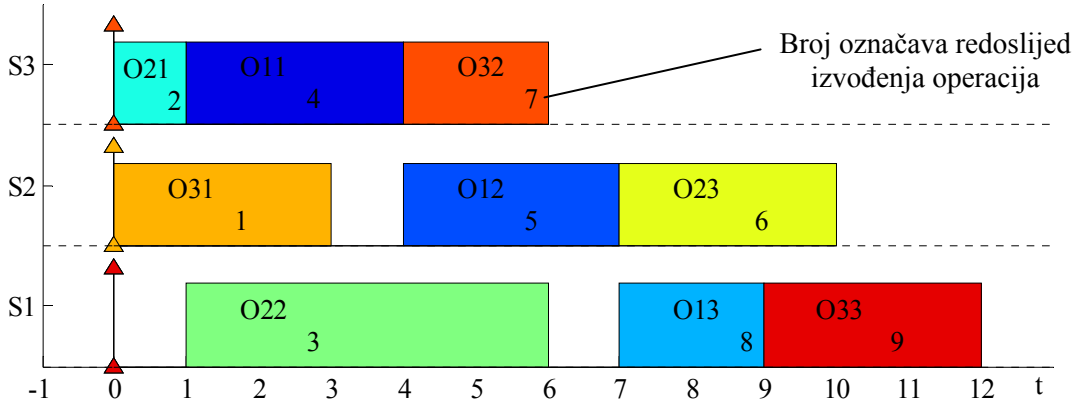
kromosom : [3 2 2 1 1 2 3 1 3]

strojevi : [2 1 3 1 2 2 1 3 3]

Slika 5.10 Redoslijed izvođenja poslova na strojevima

■

2) ***Kodiranje koje se zasniva na poslovima*** – Kod ovog načina predstavljanja kromosoma, svaki gen predstavlja posao i raspored se odredi prema nizu gena (poslova) u kromosomu. Nakon što je određen prvi gen/posao u nizu dodaje se drugi posao u niz, u cilju da se postigne što bolje vrijeme obrade poslova, i tako se nastavlja za sve poslove u sustavu.

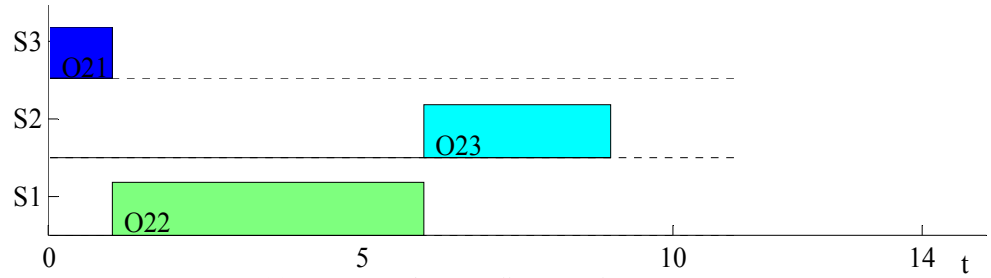


Slika 5.11 Mogući raspored poslova

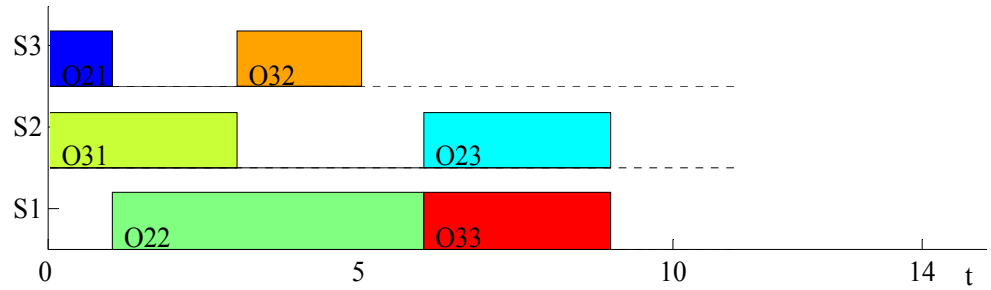
[14] Modeli primjera u ovom poglavlju implementirani su kao Matlab funkcije.

Primjer 5.6

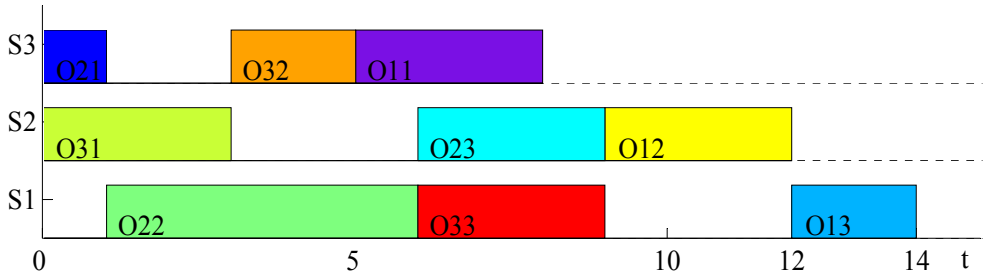
U ovom primjeru razmatrat će se isti problem koji je zadan tablicom 5.2. Primjerice, neka je kromosom dan kao niz $[2\ 3\ 1]$. Prvi posao koji će se izvesti je P2. Redoslijed operacija za posao P2 je određen kako je prikazano slikom 5.12(a). Nakon toga izvodi se posao P3. Redoslijed operacija posla P3 po strojevima je $[S_2\ S_1\ S_3]$ i vrijeme rada svakog stroja je $[3\ 2\ 3]$ vremenskih jedinica.



(a) Raspored operacija za prvi posao P2



(b) Raspored operacija za drugi posao P3



(c) Raspored operacija za treći posao P1

Slika 5.12 Raspored poslova

Redoslijed svake operacije se odredi tako da se postigne najbolje moguće vrijeme izvođenja onog stroja kojeg pojedina operacija zahtijeva (slika 5.12(b)). Na kraju se odredi raspored posla P1 kao što je prikazano slikom 5.12 (c).

■

3) ***Par poslova koji su u relaciji*** – U ovom slučaju koristi se binarna matrica kako bi se za par poslova odredilo koji ima prednost izvođenja na zadanom resursu.

4) ***Kodiranje slučajnih ključeva*** – Ovaj način predstavljanja kromosoma kodira rješenja kao slučajne brojeve. Broj se sastoji iz dva dijela: (i) cjelobrojni dio, koji predstavlja broj resursa; (ii) razlomljeni dio koji sadrži niz poslova koji se izvode na tom resursu.

Neizravno kodiranje

1) ***Kodiranje koje se zasniva na pravilima prioriteta*** – Kod ovog načina kodiranja svaki kromosom je predstavljen kao niz pravila za dodjeljivanje poslova. Raspored se odredi uporabom heurističkih algoritama za određivanje prioriteta dodjeljivanja na temelju niza pravila prioriteta. Genetski algoritmi se rabe za razvijanje onih kromosoma koji će pripomoći u stvaranju boljih nizova pravila prioriteta.

Primjer 5.7

I u ovom primjeru razmatrat će se problem koji je zadan tablicom 5.2 i četiri pravila prioriteta zadana tablicom 5.3. Primjerice, neka je kromosom dan kao niz $\left[1\ 2\ 2\ 1\ 4\ 4\ 2\ 1\ 3\right]$, pri čemu 1 označava SPT pravilo, 2 LPT pravilo, 3 MWR pravilo i 4 LWR pravilo.

Tablica 5.3 Odabrana pravila prioriteta

#Pravila	Pravilo	Opis
1	SPT	Izbor operacije s najkraćim vremenom izvođenja.
2	LPT	Izbor operacije s najdužim vremenom izvođenja.
3	MWR	Izbor operacije za posao s najviše preostalog vremena izvođenja.
4	LWR	Izbor operacije za posao s najmanje preostalog vremena izvođenja.

U početnom koraku se odredi:

$S_1 = \{o_{111}, o_{211}, o_{312}\}$, gdje je S_t skup raspoređenih operacija u iteraciji *t*,

$\phi_1^* = min\{3,1,3\} = 1$, gdje je $\phi_t = min_{i \in S_t}\{\phi_i\}$ najkraće vrijeme za koje bi operacija $i \in S_t$ trebala biti završena,

$m^* = 1$, gdje je stroj na kojem se treba izvršiti operacija ϕ_t^*,

$C_1 = \{o_{111}, o_{211}\}$, gdje je C_t skup operacija $i \in S_t$ koje zahtijevaju stroj m^* u iteraciji *t*.

Dakle operacije o_{111} i o_{211} konkuriraju za stroj m_1. Kako je prvi gen u zadanu kromosomu 1 to znači SPT pravilo prioriteta, onda će operacija o_{211} biti dodijeljena stroju m_1. Nakon obnavljanja podataka vrijedi:

$S_2 = \{o_{111}, o_{223}, o_{312}\}$

$\phi_2^* = min\{4,6,3\} = 3$

$m^* = 2$

$C_2 = \{o_{312}\}$.

Operacija o_{312} biti će dodijeljena stroju m_2. Nakon obnavljanja podataka vrijedi:

$S_3 = \{o_{111}, o_{223}, o_{321}\}$

$\phi_3^* = min\{4,6,3\} = 3$

$m^* = 1$

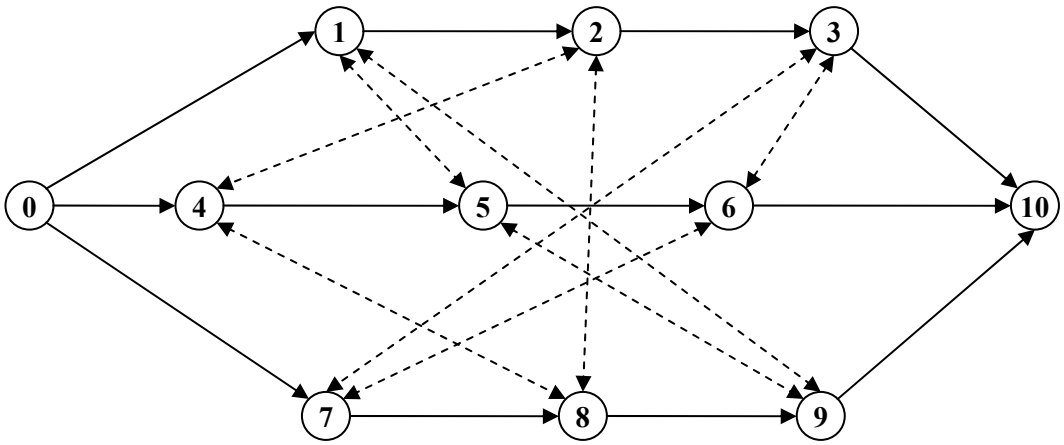
$C_3 = \{o_{111}, o_{321}\}$.

Sada operacije o_{111} i o_{321} konkuriraju za stroj m_1. Kako je treći gen u zadanom kromosomu 2 (LPT pravilo prioriteta) onda će operacija o_{111} biti dodijeljena stroju m_1. Ovi koraci se ponavljaju sve dok se ne deducira cijeli raspored iz zadanog kromosoma. ■

2) Disjunktivni graf – Kod ovog načina problem rasporeda poslova se predstavlja disjunktivnim grafom. Slika 5.13 prikazuje primjer disjunktivnog grafa za raspodjelu tri posla na tri resursa. Slika 5.14 prikazuje kromosom koji se sastoji od binarnog niza koji odgovara uređenoj listi disjunktivnih lukova, pri čemu e_{ij} označava disjunktivni luk između para čvorova (i, j) i definira se kako slijedi:

$$e_{ij} = \begin{cases} 1, & \text{odgovara smjeru disjunktivnog luka od čvora } j \text{ do čvora } i \\ 0, & \text{odgovara smjeru disjunktivnog luka od čvora } i \text{ do čvora } j \end{cases}$$

Kod ovakvog načina kodiranja, kromosom se ne koristi za prikaz rasporeda već ukazuje na pogodniji luk između para čvorova (i, j). Kada se za vrijeme dedukcije rasporeda pojavi konflikt između dva čvora (operacije) za jednim resursom, koristi se odgovarajući bit kromosoma kako bi se odredio redoslijed izvođenja te dvije operacije, tj. odredi se orijentacija luka između ta dva čvora.



Slika 5.13 Disjunktivni graf za tri posla na tri resursa

uređena lista disjunktivnih lukova	e_{15}	e_{19}	e_{59}	e_{24}	e_{28}	e_{48}	e_{36}	e_{37}	e_{67}
kromosom	0	0	1	1	0	0	0	1	1

Slika 5.14 Kodiranje disjunktivnog grafa

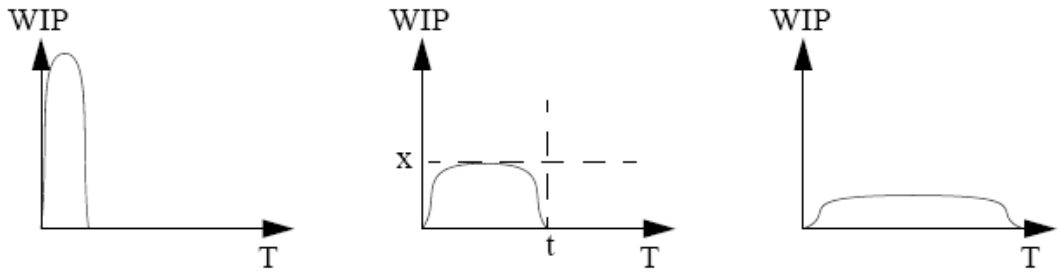
3) Kodiranje pomoću sklonosne liste (engl. *Preference List*) – Niz se formira kao *m* pod-kromosoma, po jedan za svaki resurs. Svaki pod-kromosom je duljine *n* i predstavlja sklonosnu listu za resurs. Ova lista se rabi unutar simulacije kako bi se za dani resurs odabrao posao među onima koji čekaju na izvođenje.

4) Kodiranje resursa – Kromosom se kodira kao niz resursa i iz njega se deducira raspored pomoću heurističkog algoritma za pomicanje uskog grla (engl. *Shifting Bottleneck Heuristic*), kojeg su predložili Adams, Bals i Zawack (citirano u [GEN97]). Kod ovog načina genetski algoritam se rabi za kreiranje niza resursa koji pružaju najkraći raspored primjenom heurističkih algoritama za rješavanje uskog grla.

5.3. Višekriterijski problem rasporeda

U prethodnom potpoglavlju su navedene neke od metoda za rješavanje problema rasporeda. Cilj takvih procedura je uglavnom minimalizirati vrijeme od početka izvođenja prvog posla do završetka posljednjeg. Međutim, postoje i drugi važni faktori koje bi trebalo uzeti u obzir pri određivanju rasporeda poslova. Određene alokacije resursa, limiti na dopušteni broj poslova u procesu, krajnji rokovi za izvođenje poslova, međudjelovanje između poslova kao i konflikti pri dodjeljivanju resursa su samo neki od primjera. Ako se pokuša optimalizirati kombinacija ovih faktora završiti će se sa skupom kompromisnih rješenja i svako od njih optimalizira određeni kriterij.

Primjerice, ako se promatra radionica ograničenog radnog prostora, to povlači i ograničavanje količine poslova koja je u obradi (engl, *Work in Progress*, pokrata: WIP). Trebalo bi minimalizirati vrijeme potrebno za izvođenje skupa poslova. Slika 5.15 [TOD97] prikazuje skup kompromisnih rješenja za tri različita slučaja optimalizacije. Moguće je ili minimalizirati vrijeme potrebno za izvođenje skupa poslova, ali se maksimalizira količina poslova koja je u obradi (slučaj 1) ili minimalizirati najveću razinu količine poslova u obradi, ali je dulje vrijeme izvođenje tih poslova (slučaj 3). Sam korisnik mora odabrati rješenje ovisno o tome što mu je značajnije. Ipak najpoželjnije rješenje je kompromisno rješenje (slučaj 2). Naime, umjesto da se optimalizira samo po jednom kriteriju, primjerice što kraće vrijeme izvođenja, odredi se maksimalna količina poslova koji mogu biti u obradi (označeno kao razina x). Optimalizacijom se pronađe minimalno vrijeme izvođenja poslova t za taj slučaj. I to rješenje predstavlja kompromis između rješenja (1) i rješenja (3).

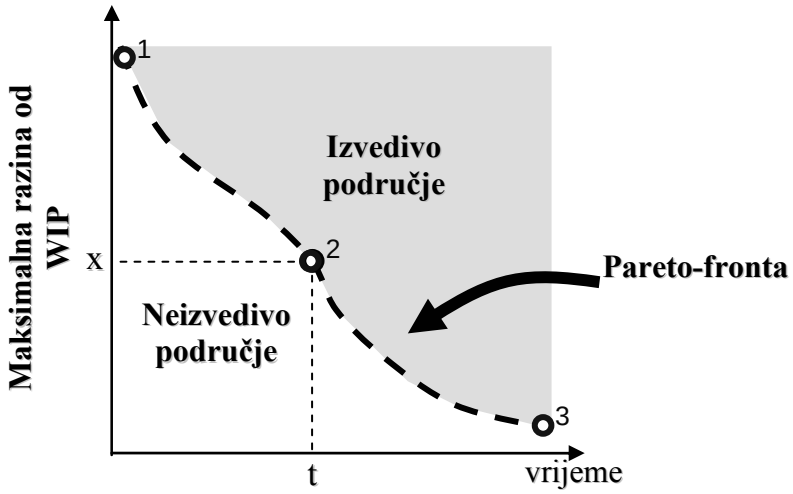


(1) minimalni makespan (2) kompromisno rješenje (3) minimalni WIP

Slika 5.15 Vrijeme (T) u odnosu na količinu posla u obradi (WIP) za tri primjera kompromisnih rješenja

Kompromis je jasno prikazan Pareto krivuljom (slika 5.16). Rješenje problema će biti zatvoreno unutar *izvedivog područja* (engl. *feasible region*) koja je određeno ograničenjima sustava i postavljenim uvjetima. Stoga je cilj rješavanja problema rasporeda u

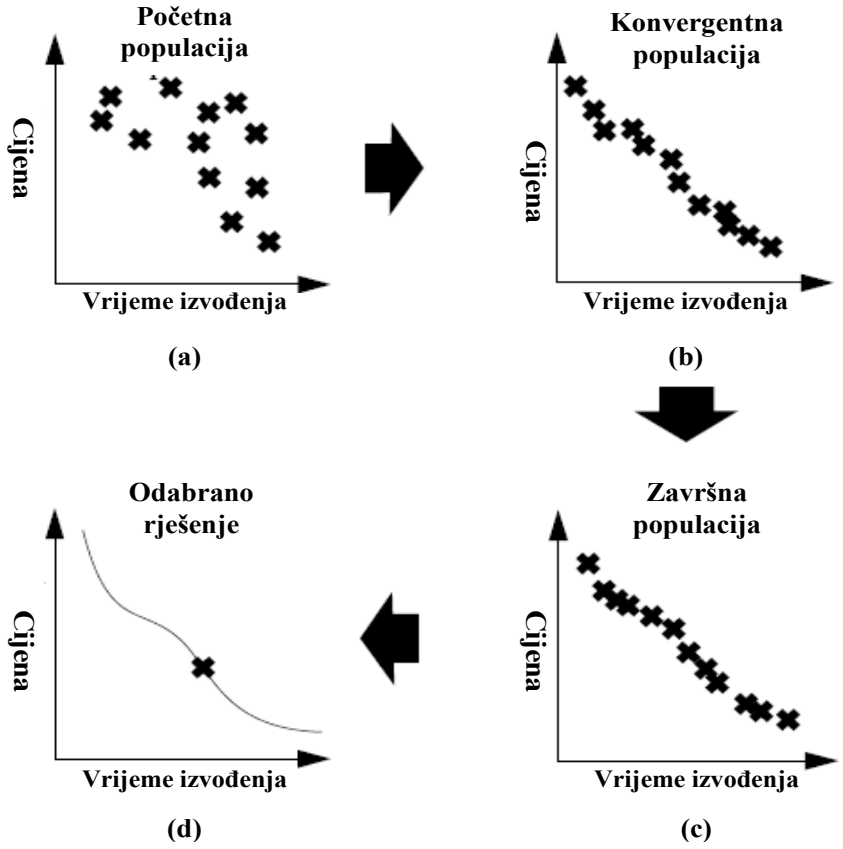
višekriterijskom kontekstu odrediti kompromisnu granicu ili Pareto-frontu, što se može učiniti uporabom višekriterijskog GA. Ova granica se može uporabiti za odabir rješenja kojim se postiže najbolja ravnoteža (balans) kriterija za zadani skup okolnosti. Najpoželjnija ravnoteža će zasigurno biti određena prioritetima u problemu.



Slika 5.16 Odnos poslova koji su u procesu WIP i vremena

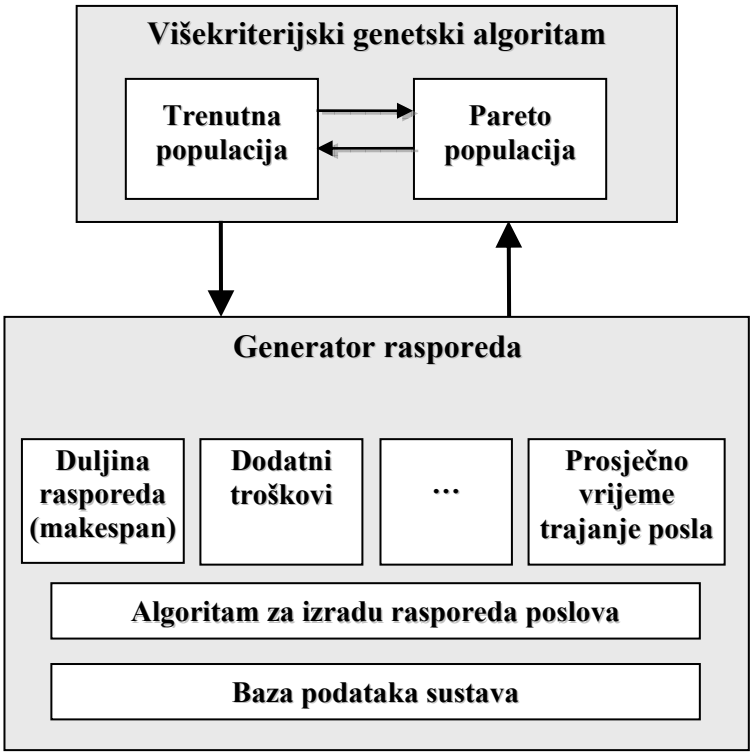
5.3.1. Višekriterijski genetski algoritam za određivanje rasporeda poslova

Genetski algoritam opisan u ovom podpoglavlju je proširena verzija jednostavnog genetskog algoritma, nazvan MCGA *višekriterijski genetski algoritam*. MCGA generira početnu populaciju jedinki slučajno. Svaka jedinka predstavlja jedno moguće rješenje problema koji se rješava. Jedinke se mogu odrediti pomoću različitih kriterija koji su određeni ciljevima promatranog problema. U svakom trenutku, ovi kriteriji se odvojeno održavaju i tako omogućuju istovremeno maksimiziranje i minimiziranje odvojenih kriterija, primjerice, maksimalizaciju dobiti uz istovremenu minimalizaciju ukupnog vremena potrebnog za izvođenje svih poslova (slika 5.17). Algoritam započinje sa slučajnom razdiobom rješenja (slika 5.17a). Umjesto traženja samo jednog rješenja, MCGA pronalazi skup rješenja koji definiraju područje kompromisa (slika 5.17b) među ovim različitim kriterijima pretraživanja. Ova rješenja tvore Pareto skup. GA nastavlja s iteracijama sve dok se ne postigne prihvatljivo područje rješenja (slika 5.17c). Rješenje, koje najbolje odgovara pretpostavkama onoga koji odlučuje, će se izdvojiti (slika 5.17d) i dalje će se analizirati ili će se implementirati na odgovarajući način. Detaljan opis tehničkih značajki MCGA algoritma mogu se pronaći u [TOD97, TOD98].



Slika 5.17 Razvoj područja kompromisnih rješenja

MCGA algoritam se može podijeliti u dva, međusobno povezana, dijela kako je prikazano slikom 5.18. Prvi dio je, prethodno opisan, višekriterijski genetski algoritam, dok je drugi dio nazvan generator rasporeda koji se, ovisno o aplikacijskoj domeni, može ukloniti ili promijeniti. Umjesto da se minimalizira samo jedna funkcija cilja, kao što je vrijeme izvođenja svih poslova, ovaj algoritam može se može primijeniti, istovremeno, na više funkcija cilja u cilju razvijanja takvih rasporeda poslova koji bi bili optimalni obzirom na različite kombinacije postavljenih kriterija. Isto tako, s MCGA je moguće razviti takve rasporede koje će uzeti u obzir i neraspoloživost resursa, vrijeme "setiranja" pri promjeni načina rada te ograničenja na vrijeme početka i vrijeme završetka pojedinog procesa. Kako bi se poboljšala fleksibilnost primjene, s MCGA je moguće razmatrati ne samo ograničenja na pravo prvenstva među pojedinim fazama posla, nego i prioritete među poslovima.



Slika 5.18 Struktura MCGA sustava

Zbog složenosti, problem rasporeda se ne može prikazati samo jednom tablicom u kojoj će se prikazati i poslovi, resursi, operacije i vremena kao što je prikazano tablicom 5.2. Za potpuni opis problema rasporeda poslova potrebno je nekoliko tablica koje se odnose na različite aspekte opisa sustava i koje su niže opisane.

Matrica prioriteta (engl. *Precedence Matrix*): Ova matrica definira poslove (procese) i njihove unutarnje odnose.

Matrica učinkovitosti resursa (engl. *Resource Efficiency Matrix*): Ova matrica sadrži popis svih resursa i pojedinosti o učinkovitosti resursa ovisno o tipu operacije. Za problem rasporeda proizvodnih poslova, resursi se odnose na pojedine strojeve (npr. Stroj 1, Stroj 2, itd.), dok tipovi operacija opisuju radnju (npr. ispitivanje naprezanja, tehničko crtanje, itd.). Elementi matrice predstavljaju relativnu učinkovitost svakog resursa, za svaki, unaprijed određeni, tip operacije. Ova matrica se stvara na organizacijskoj razini i s vremenom se može ažurirati. Resursi se mogu dodavati ili brisati, dok se učinkovitosti ažuriraju kao rezultat održavanja sustava, prakse i iskustva.

Matrica operacija (engl. *Operation Matrix*): Matrica operacija sadrži pregled svih poslova zajedno s pojedinostima operacija koje se tim poslom obavljaju. Zbog fleksibilnosti, svaki se posao može razložiti na niz faza omogućujući prikaz varijabilnih alokacija resursa za vrijeme obavljanja posla.

Ograničenja (engl. *Constraint Data*): U MCGA simulator rasporeda mogu se dodati i vremenski periodi, za vrijeme kojih se poslovi ne mogu rasporediti, kao i vremena početka i završetka pojedinog posla.

Primjer 5.8 Složeni problem rasporeda [TOD98]

U ovom primjeru razmatra se problem rasporeda pet poslova na četiri stroja. Poslovi se sastoje od određenog broja operacija (maksimalno pet) i izvođenje poslova se odvija u fazama.

Kako bi se opisao problem nužnu je definirati nekoliko varijabli: *Broj resursa, Broj operacija, Broj poslova, Maksimalan broj faza, Maksimalan broj zavisnosti i Maksimalan vremenski period za održavanje.* Također, postoji varijabla koja pokazuje je li dozvoljeno dijeljenje poslova. Dijeljenje poslova se javlja onda dok se posao izvodi na resursu, a planirano održavanje resursa prekida započeti posao. Posao se ponovo pokreće nakon održavanja. Ako se dijeljenje ne dopušta predviđeni posao neće se započeti sve dok se ne završi servisiranje.

Tablica 5.4 prikazuje učinkovitost strojeva kada izvode različite operacije. Kada je učinkovitost 1.0 znači da stroj izvodi operaciju u standardno definiranom vremenu. Vremenska oznaka može biti bilo koje trajanje u stvarnom vremenu (1 minuta, 10 minuta, 1 sat, itd). Kada je učinkovitost veća od 1 operacija se izvodi brže dok strojevi gdje je učinkovitost manja od 1 sporije izvode operaciju. Strojevi s učinkovitošću 0 ne mogu izvoditi operaciju. Primjerice, stroj S0 može izvesti operaciju Op0 sa 100% učinkovitosti i Op3 sa 50% učinkovitosti, ali ne može izvesti ni jednu drugu operaciju.

Tablica 5.4 Učinkovitosti rada strojeva

Strojevi	Operacije					
	Op0	Op1	Op2	Op3	Op4	Op5
S0	1.0	0	0	0.5	0	0
S1	0	0.5	0	0	0.8	0
S2	0	1.0	0.8	0	0	0.8
S3	0.5	0.8	0	0.3	0	1.0

Vrijeme održavanja strojeva prikazano je tablicom 5.5. Ako stroj zahtijeva ugradnju novih dijelova ili se treba očistiti onda mu se dodijeli procedura održavanja. Za svaki stroj naznačen je broj perioda i za svaki od tih perioda određen je vremenski interval. Primjerice, stroj S3 ima 2 perioda kada nije raspoloživ, u intervalu od 20 do 35 vremenske jedinice i od 60 do 64 vremenske jedinice.

Tablica 5.5 Vrijeme održavanja strojeva

	Br.	Interval 0	Interval1	Interval2	Interval3
S0	2	50-60	100-110		
S1	1	20-25	0		
S2	4	10-15	25-30	34-37	50-52
S3	2	20-35	60-64		

Ako je stroj u mogućnosti izvesti nekoliko različitih tipova operacija prijelaz među operacijama zahtijeva određeno vrijeme setiranja. Ova vremena su prikazana tablicom 5.6. Trenutna operacija je lijevo, a sljedeća operacija se očita s vrha. Primjerice, za prijelaz s operacije Op3 na operaciju Op4 trebat će 2 vremenske jedinice kako bi se ostvario taj prijelaz.

Tablica 5.6 Vrijeme setiranja strojeva pri prijelazu među operacijama

	Op0	Op1	Op2	Op3	Op4	Op5
Op0	0	4	2	2	3	1
Op1	3	0	1	5	4	2
Op2	3	2	0	2	3	1
Op3	4	3	2	0	2	1
Op4	3	5	2	1	0	2
Op5	5	4	4	1	2	0

Svaki posao se izvršava kroz slijed operacija koje se izvode jedna za drugom. Izvođenje pojedine operacije nazvat će se pojedinom fazom izvođenja posla. U tablici 5.7 za svaki posao prikazane su faze njegovog izvođenja. Svaka faza ima redni broj i definirana je s dva broja. Prvi broj je tip operacije koja se mora izvršiti u toj fazi. Drugi broj je standardno vrijeme koje se zahtijeva tom fazom.

Tablica 5.7 Prikaz izvođenja poslova po fazama

Poslovi	Br.	Faza0	Faza1	Faza2	Faza3	Faza4
P0	2	2,4	4,10			
P1	4	1,6	2,3	5,8	0,3	
P2	3	4,5	1,3	2,8		
P3	4	0,7	2,7	4,3	5,1	
P4	5	1,4	0,5	1,7	3,9	4,6

Ako se zajedno sagledaju tablice 5.4 i 5.7 onda se može prikazati utjecaj učinkovitosti. Primjerice, neka je iz tablice 5.7 odabran posao P3 – faza 0. Ova faza zahtijeva izvođenje operacije 0. Iz tablice 5.4 vidljivo je da strojevi S0 i S3 mogu izvoditi ovu operaciju. Ako je odabran stroj S3 onda ovaj stroj izvodi operaciju s učinkovitošću od 0.5. Ukupno vrijeme potrebno za izvođenje ove operacije dano je sljedećim izrazom:

$$Vrijeme = \frac{Uobičajeno\ vrijeme}{Učinkovitost}$$

Stoga će na stroju S3, posao P3-faza 0 trebati 7/0.5 = 14 vremenskih jedinica. Slično, posao P3-faza 0 će trebati 7 vremenskih jedinica na stroju S1, jer je učinkovitost jednaka 1.

■

MCGA algoritam pretraživanja, koristi standardne GA strukture uz određene preinake. To je prikazano slikom 5.19 i niže je objašnjeno.

```
Kreiranje populacije.
Vrednovanje populacije po svim kriterijima.
Rangiranje populacije na osnovi dominantnosti.
Obnavljanje Pareto populacije.
Pretvorba dobrote.
Dijeljenje dobrote.
Selekcija:
        ○       Elitistička strategija.
        ○       Pareto strategija.
        ○       Strategija ruletnog kotača.
Izvođenje heurističkog križanja.
Izvođenje mutacije.
Vrednovanje populacije po svim kriterijima.
Vraćanje na 3 sve dok se ne zadovolji  uvjet za
    kraj.
Izlazna konfiguracija.
```

Slika 5.19 MCGA algoritam

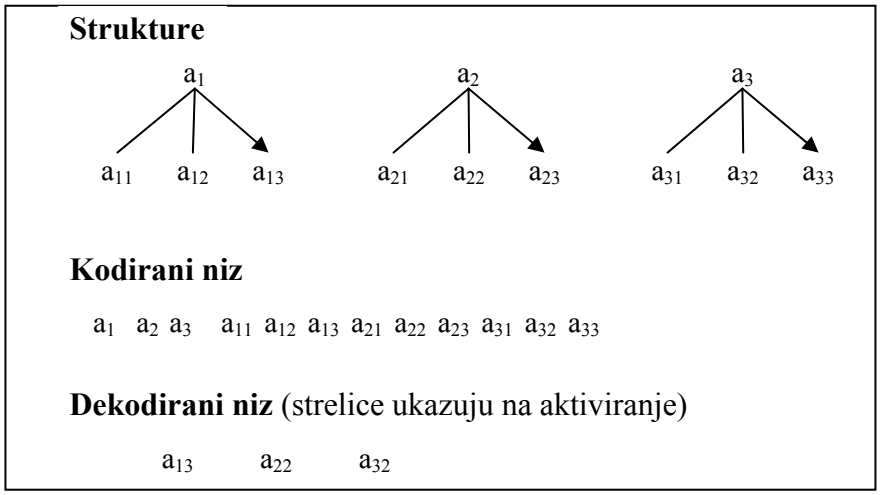
1) **Kreiranje populacije:** Prema prirodi problema rasporeda poslova nije moguća uporaba niza koji je po resursima podijeljen na podnizove od kojih svaki sadrži po prioritetima listu faza, fiksne duljine. To je zbog toga što moguća razmjena faza među resursima može uzrokovati da, iako su nizovi stalne duljine, granica među dijelovima po resursima se može mijenjati kako se gen pomiče unutar niza. Isto tako, ovo može uzrokovati pojavu duplih nizova u toku križanja. U MCGA sustavu niz se dijeli na particije po resursima uz istovremeno zadržavanje višestrukih kopija onih faza koje se pojavljuju kod nekoliko resursa. Uporabom grupe aktivacijskih gena koji se umeću na početka niza omogućeno je aktiviranje/isključivanje duplih faza unutar niza. Ova tehnika je razvijena iz ideja strukturiranog GA[15].

Selektivno aktiviranje gena

Tehnika strukturiranog GA se temelji na binarnom kodiranju. Ipak, ako se koncept proširi i na cjelobrojno kodiranje onda bi se trebali inicirati različiti tipovi aktivacijskih gena. Niz gena je isti kao i prije s tom razlikom da sada osim sposobnosti uključivanja i isključivanja skupa gena, gornja razina gena može odabrati koje gene će aktivirati (slika 5.20).

Primjerice, ako su a_1 i a_2 aktivirani, u vrednovanju funkcije cilja trebali bi se koristiti geni a_{11} a_{12} a_{13} a_{21} a_{22} a_{23}. Neaktivni dijelovi niza su neutralni i dalje se prenose kao zaliha genetskog materijala. Za vrijeme mutacije i križanja bilo koje promjene koje mijenjaju gornju razinu gena uzrokuju promjene u aktivnom skupu. To je veoma učinkovito jer to povlači i promjenu više gena unutar aktivnog niza u samo jednom ciklusu. Ovo omogućava bržu prilagodbu na promjene u okruženju.

[15] Dasgupta D., McGregor, D.: "*Structured Genetic Algorithms: The Model and First Results",* Department of Computer Science, University of Strathclyde, UK, 1992. (citiranu u [TOD98])



Slika 5.20 Aktiviranje gena

Primjer 5.9 Konstrukcija kromosoma [TOD98]

U cilju određivanja rasporeda poslova na pojedinim strojevima, može se primijeniti tehnika selektivne aktivacije gena. Tehnika će se pokazati na primjeru pomoću podataka prikazanih u tablicama 5.8 i 5.9. Svaka faza posla se definira pomoću operacije koja se treba izvršiti u toj fazi. Operacije se mogu zamisliti kao posebni zadaci (primjerice, bušenje ili obrada glodalicom) i obično je slučaj da jednu operaciju može obaviti više od jednog stroja.

Tablica 5.8 Učinkovitosti rada strojeva

Strojevi	Operacije					
	Op0	Op1	Op2	Op3	Op4	Op5
S0	1	0	0	1	1	0
S1	0	1	0	1	0	1
S2	0	0	1	0	1	1

Tablica 5.9 Faza izvođenja poslova po poslovima

Posao	Broj faza	Faza0	Faza1	Faza2	Faza3	Faza4
P0	3	1,5	4,3	1,4		
P1	5	2,3	1,6	3,5	1,3	4,4
P2	3	0,4	2,3	1,6		
P3	4	5,4	2,5	1,3	0,5	

Kromosom se gradi od n+1 odvojenih segmenta, pri čemu n označava broj strojeva i dodan je jedan poseban segment za skup aktivacijskih gena. Svaki od tih aktivacijskih gena trebao bi odgovarati jednom od izabranog stroja (faze koje može izvoditi više od jednog stroja). Stoga, u ovom primjeru, postoji 4 aktivacijskih gena. To je zbog toga što se samo operacije Op3,Op4 i Op5 mogu izvršavati na više od jednom stroju, dok su ostale operacije ograničene na jedan stroj (tablica 5.8). U svakom od strojnih segmenata postoji gen za svaku

od faza koje bi se trebale izvršiti na tom stroju, uključujući one koje se mogu pojaviti na dva ili više strojeva. Stoga se niz dobije ulančavanjem:

Odabir strojeva – stroj S0 - stroj S1 - ..... - stroj Si - ..... - stroj Sn

Za ovaj primjer vrijedi:

Odabir strojeva = P0faza1, P1faza2, P1faza4, P3faza0

stroj S0 = P0faza1, P1faza2, P1faza4, P2faza0, P3faza3

stroj S1 = P0faza0, P0faza2, P1faza1, P1faza2, P1faza3, P2faza2, P3faza0, P3faza2

stroj S2 = P0faza1, P1faza0, P1faza4, P2faza1, P3faza0, P3faza1

Kako bi se ovo kodiralo, u kromosomu na koji se može primijeniti MCGA potrebno je poduzeti nekoliko koraka. Kao prvo, dio koji se odnosi na izbor stroja dobije se ulančavanjem odabira strojeva u rastućem poretku po fazama poslova. U ovom primjeru postoji četiri izbora za stroj: P0faza1, P1faza2, P1faza4, P3faza0. To znači da postoje četiri cjelobrojna aktivacijska gena u dijelu kromosoma koji se odnosi na odabir stroja. Kod prvog gena, P0faza1, operacija koja se treba izvesti je Op4. Po tablici 5.8 ova operacija se može izvršiti ili na stroju S0 ili na stroju S2. Stoga, vrijednost tog gena može poprimiti vrijednost ili 0 ili 2.

P1faza2, operacija Op3 se može izvršiti ili na stroju S0 ili S1, zato se ovaj gen ograničava na vrijednost 0 ili 1. Postupak se nastavlja za preostala dva gena u dijelu kromosoma koji se odnosi na odabir stroja.

Konstrukcija kromosoma se nastavlja za svaki dio koji se odnosi na strojeve. Primjerice, za stroj S0 postoji 5 faza po poslovima pa će ovaj dio kromosoma sadržavati 5 gena. Svaki posao se jednostavno prikaže brojem posla kojeg izvodi, a pri tome se ne uzima broj faze. Za S0 ovaj dio kromosoma bi sadržavao niz 0,1,1,2,3. Dozvoljava se bilo koja permutacija tih pet gena, primjerice 1,2,0,1,3 ili 3,1,2,1,0. Redoslijed poslova tvori listu prioriteta za taj stroj tako da prvi posao ima najveći prioritet.

U slučaju za stroj S1 niz bi izgledao kao 0,0,1,1,1,2,3,3. U nizu se 1 pojavljuje tri puta, ali po redoslijedu da bi se moglo započeti s P1faza2, P1faza1 bi već trebala završiti. Zato, faza 1 mora biti prva. Ovakvo kodiranje se nastavlja za sve strojeve dok se ne završi cijeli niz. Početni niz se dobije automatski iz skupa ulaznih podataka.

Kako bi se prikazao postupak dekodiranja kao primjer uzet će se kromosom iz gornjeg primjera. Kromosom se prvo podijeli na dio koji se odnosi na odabir stroja i dijelove koji se odnose na strojeve.

Odabir stroja	S0	S1	S2
0, 0, 0, 2	3, 1, 2, 1, 0	2, 1, 1, 0, 3, 3, 0, 1	3, 1, 0, 1, 3, 2

Niz gena koji se odnose na izbor stroja rabi se za aktiviranje odgovarajućeg posla u svakoj od lista strojeva. U svim ostalim listama ti poslovi će se ukloniti. Primjerice, prvi gen u listi odabira stroja je posao P0 faza 1. Ovaj posao može ići ili na stroj S0 ili na stroj S2. U ovom slučaju odabran je stroj S0, stoga se posao P0 faza 1 mora ukloniti sa stroja S2. U nizu gena stroja S2 postoji samo jedan posao P0 pa se taj uklanja. Ako se više od jedne faza istog posla mora izvršiti na nekom stroju onda se koristi implicitni redoslijed poslova. Primjerice, drugi gen u dijelu za odabir stroja odnosi se na P1 faza2. Stroj S0 je odabran i zato se ovaj

posao uklanja iz liste za S1. Može se uočiti da stroj S1 ima tri gena čija je vrijednost 1. Prema tablicama 5.8 i 5.9, P1 faza2 je druga moguća faza posla P1 koju izvodi stroj S1, prva je P1faza1. Stoga se druga jedinica uklanja iz liste za stroj S1. Ako se ovo nastavi za sve gene iz dijela za odabir stroja lista poslova po strojevima bi glasila:

Lista za stroj S0: 3, 1, 2, 1, 0

Lista za stroj S1: 2, 1, 0, 3, 0, 1

Lista za stroj S2: 3, 1, 3, 2.

■

2) **Vrednovanje populacije po svim kriterijima:** Svaki niz generirane populacije se pošalje u generator rasporeda modeliranog problema i preslika se u oblik koji koristi generator rasporeda. Niz se vrednuje po svakom kriteriju i rezultati se vraćaju u MCGA.

3) **Rangiranje populacije pomoću dominantnost:** Umjesto da se radi sa samo jednim kriterijem, što je uobičajeno za jednostavni genetski algoritam, MCGA radi istovremeno sa više kriterija. Cilj nije pronaći samo jedno rješenje već generirati skup rješenja raspoređenih duž Pareto fronte. Također, MCGA izbjegava uporabu kombinacije kriterija u obliku težinskih suma, što vodi ka jedinstveno poželjnijim rješenjima.

4) **Obnavljanje Pareto populacije**: U svakoj generaciji Pareto populacija se obnavlja s novim ne-dominantnim rješenjima trenutne populacije. Populacija se rangira i bilo koji duplikat ili dominantno rješenje se uklanja. Ova populacija se, također, kasnije koristi pri vrednovanju dobrote, u procedurama selekcije i križanja.

5) **Pretvorba dobrote:** Rang bi trebao sadržavati vrijednost između 1 (ne-dominantno) i veličine populacije (potpuno dominantno). Ova vrijednost se može izravno konvertirati u mjeru dobrote pomoću jednadžbe:

$$d = \left(P + 1 - R(i)\right)^a ,$$

gdje je

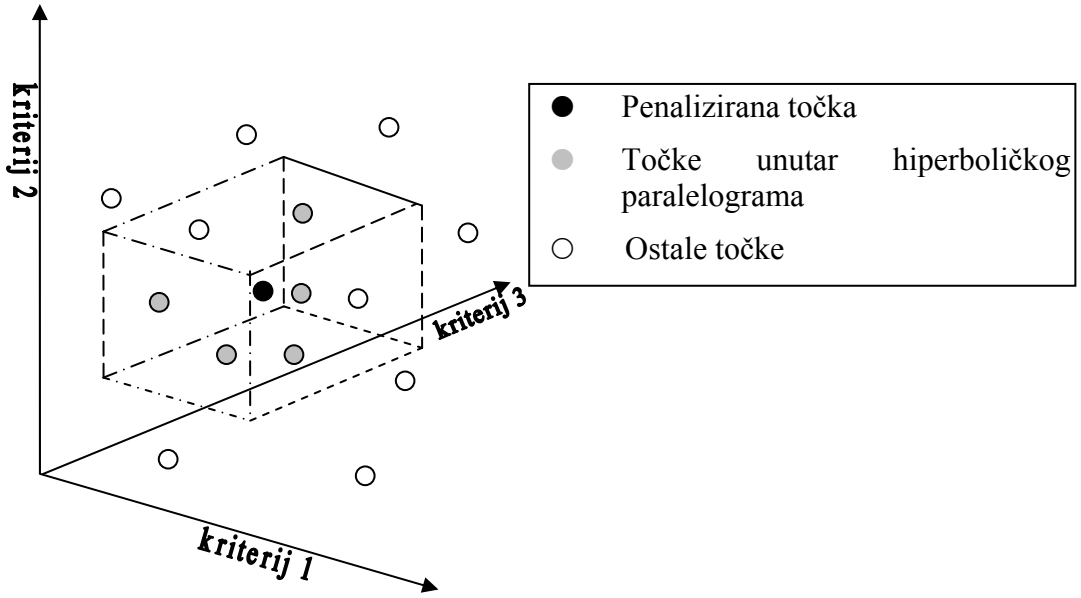
(i) P - veličina populacije,

(ii) $R(i)$ – rang i-te jedinke,

(iii) a – kontrolni faktor koji regulira pritisak selekcije.

6) **Dijeljenje dobrote:** Kada se radi s višekriterijskim genetskim algoritmom genetsko odstupanje (engl. *drift*) može uzrokovati gomilanje rješenja u samo jednom dijelu Pareto površine. Ovo se može izbjeći uporabom izmijenjenog načina razdiobe fitnesa, kojeg je prvi uveo Goldberg (1987). Ideja dodjeljivanja je da se iznos penalizacije dodaje jedinkama populacije koje se okupljaju jedna blizu druge, odnosno unutar područja ili udubine[16] unaprijed određene veličine. Ovo penalizira ugnježđivanje i potiče populaciju da se mijenja. U višekriterijskom prostoru isti postupak se primjenjuje na sve kriterije. Ovo je ekvivalentno kreiranju hiperboličkog paralelograma u prostoru kriterija okolo zadane točke i penaliziranju te točke razmjerno broju točaka koje se nalaze unutar paralelograma (slika 5.21).

[16] Takve skupine predstavljaju povoljna područja prostora pretraživanja i nazivaju se udubljenja (engl. niches).



Slika 5.21 Razdioba dobrote u višekriterijskom prostoru

Ova procedura se izvodi na svim točkama. Najvažniji faktor u implementaciji dijeljenja dobrote je odabir dobre veličine udubljenja (duljina svake od stranica paralelograma). Poželjno je postići ravnomjernu raspodjelu točaka duž Pareto površine. Kod nekih metoda [FON93] predlaže se određivanje veličine udubljenja na osnovi najvećih i najmanjih vrijednosti korištenih kriterija. Kako bilo, umjesto uporabe unaprijed utvrđene vrijednosti udubljenja koja se odredi na početku, može se vrijednost udubljenja automatski ažurirati. Dijeljenje dobrote samo jednom koristi Pareto populaciju. Uporabom ove populacije odredi se najveća i najmanja vrijednost svakog kriterija i veličina udubljenja svakog kriterija se odredi oduzimanjem minimuma od maksimuma i dijeljenjem s veličinom populacije.

7) Selekcija: Selekcija se izvodi kroz tri koraka.

Korak 1 - Pareto jedinke unutar populacije prolaze izravno kroz bazen za parenje.

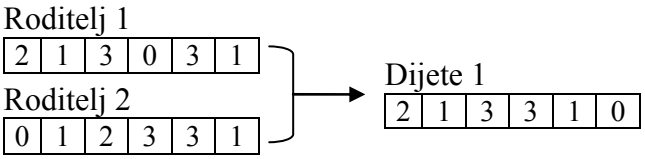
Korak 2 - U cilju promoviranja generacije novih Pareto jedinki i očuvanja raznolikosti u bazen za parenje ubaci se slučajna selekcija jedinki iz Pareto populacije.

Korak 3 - Ostatak bazena za parenje se popuni jedinkama iz trenutne populacije selekcijom pomoću kotača.

8) Operatori križanja i mutacije

Operator križanja se ne može primijeniti na cijeli niz budući je niz podijeljen na nekoliko dijelova. Zato se križanje primjenjuje na svaki pojedini dio kodiranog niza. Dio kromosoma, koji se odnosi na izbora resursa, križa se primjenom jednostavnog križanja u dvije točke (podpoglavlje 4.2.3.1). To znači da se odaberu dvije slučajne točke u dijelu za izbor resursa i geni roditeljskih kromosoma između tih točaka se zamijene. Ovo je valjano ako taj dio nije uređen po redoslijedu.

Unutar lista poslova po strojevima nije moguća primjena operatora križanja u dvije točke, jer bi to dovelo do ponavljanja gena (poslova). Zato se treba primijeniti takav operator križanja koji će osigurati da se poslovi ne ponavljaju i da se poslovi ne uklone iz kromosoma u toku križanja. Postupak je prikazan sljedećom slikom.



Slika 5.22 Križanje dijelova niza koji se odnose na resurse

Uporabom Roditelja 1 proces započinje poslom 2. Nakon toga, odrede se poslovi koji slijede u oba roditelje, pronađeni su 1 i 3 (gen 2 i gen 5 u oba kromosoma) i slučajno se odabere jedan od njih. Ako je odabran posao 1, poslovi koji slijede su 3 i 2 (gen3). Posao 2 se pojavljuje samo jednom i već je iskorišten, stoga se mora iskoristiti posao 3. Poslovi koji slijede nakon posla 3 u roditeljskim nizovima su poslovi 0 i 3, i 3 se slučajno odabere. Posao koji slijedi nakon drugog posla 3 je posao 1 u oba slučaja stoga se on odabere. Posao 0 je prvi posao Roditelja 1 koji nije odabran stoga se on dodaje na kraj. Ovaj proces se nastavlja od prvog posla u Roditelju 2 kako bi se kreiralo drugo dijete. Ovaj operator pokušava uzeti u obzir poslove koji neposredno slijede te isto tako ukupan redoslijed pokušavajući, u slučaju konflikta, odabrati onaj posao koji najprije započinje. Ako se pojave dva potpuno jednaka niza, djeca će biti potpuna jednaka roditeljima.

Mutacija je daleko jednostavniji operator. I mutacija se izvodi na izdvojenim dijelovima niza. U dijelu gdje se biraju resursi, ako se odabere gen onda se on slučajno resetira da aktivira neki drugi resurs. Primjerice, ako postoji izbor između tri resursa 0, 2 i 3 za određeni posao i resurs koji je upravo odabran je 2 to će promijeniti ili 0 ili 3. U dijelovima kromosoma koji predstavljaju listu poslova na resursima gen odabran za mutiranje bit će jednostavno zamijenjen drugim slučajno odabranim genom iz tog istog dijela.

U skladu s kombinatoričkom složenošću problema malo lokalno pretraživanje koje se pokreće ponaša se kao inteligentan operator mutacije. Ovo pretraživanje odabere prvi posao, a onda se drugi slučajno odabere i njih dva se zamijene. Ako ova zamjena poboljšava barem jedan od postavljenih kriterija rasporeda bez gubitka u bilo kojem drugom onda se zamjena prihvaća. Ako to nije slučaj onda se poslovi ponovo zamjene, vrate se na prvobitna mjesta i drugi posao se slučajno odabere de se zamijeni s prvim. Ovaj postupak se nastavlja 5 puta i ako dobra zamjena nije pronađena onda se kromosom ostavlja nepromijenjen. Ovakve zamjene se primjenjuju unutar svake sekcije resursa.

Križanje s ograničenjem: Ovaj operator se mora ograničiti uporabom iste tehnike korištene kod dijeljenja dobrote. Bilo koja jedinka može se pariti samo s jedinkama na rubovima okolnog hiperboličkog paralelograma. Ograničenje se uvodi kako bi se spriječilo međusobno križanje onih jedinke koje su u prostoru kriterija daleko izdvojene. Izvođenjem takvog parenja mogu se uništiti osobine oba kromosoma. Čak i s ovakvim ograničenjem operator križanja može izazvati greške u kromosomima. Procedura popravljanja se koristi kako bi se otkrili i uklonili duplikati.

9) **Vrednovanje populacije po svim kriterijima:** Nakon ovih operatora nova populacija se ponovo ocjenjuje.

10) **Povratak na korak 3** sve dok se ne zadovolji uvjet za kraj.

11) **Izlazni rezultati:** Lista Pareto optimalnih rješenja se prikaže. Iz te liste izdvojit će se ona rješenja koja su najbliža zahtjevima korisnika. To se može napraviti primjenom nekog od alata za više-atributsko odlučivanje (engl. *multi-attribute decision making*, pokrata: MADM) [TOD97.] Metoda se temelji na atributima koji služe u procesu ocjene ili evaluacije pojedinog kriterija. Veći broj atributa karakterizira pojedinu akciju (alternativu), a biraju se na temelju izabranih kriterija od strane donositelja odluke.

5.3.1.1. Generator rasporeda

Generator rasporeda uzima nizove koje je MCGA dao kao rezultat, dekodira ih u liste pogodnih resursa. Zatim se te liste koriste za kreiranje rasporeda. Za zadani niz uvijek će se generirati isti raspored. Generator započinje tako da sat postavi na nulu. Nakon toga započinje se petlja koja završava kada su završeni svi poslovi. Ta petlja započinje pregledavanjem svakog resursa kako bi se ustanovilo trenutno stanje, ili je stanje "setiranja", stanje servisiranja, radno stanje ili resurs miruje. Ako je resurs završio posao, posao je označen kao završen, bilježi se njegovo vrijeme završetka i resurs je postavljen u stanje mirovanja. Ako resurs izvršava posao i nalazi se ili u stanju "setiranja" ili se servisira, vrijeme potrebno za izvođenjem ovog zadatka se umanjuje za 1 vremensku jedinicu. Ako resurs obavlja posao onda se vrijeme iskoristivosti resursa uveća za 1.

U skladu s tim, razmatra se lista kako bi se za svaki resurs, koji je u stanju mirovanja, pronašao posao najvećeg prioriteta koji je raspoloživ, a još se nije izvršio.

Definicija 5.4 Raspoloživost posla [TOD98]

Posao je *raspoloživ* ako vrijedi:

(i) Sve faze tog posla koje su prethodile su završene;
(ii) Sve ovisnosti tog posla su zadovoljene;
(iii) Resurs je "setiran" za taj posao;
(iv) Ako nije dopušteno razlaganje tog posla, posao se mora završiti prije početka servisiranja resursa.

U slučajevima (i), (ii) i (iv) ispituje se sljedeći posao sa pogodne liste resursa kako bi se ustanovilo je li posao raspoloživ. U slučaju (iii) resurs započinje sa svojim "setiranjem" kako bi se pripremio za izvođenjem operacije tog posla. Vrijeme potrebno za "setiranje" se odredi na osnovi operacije koja se upravo izvodi i operacije koju zahtijeva novi posao i koja se nalazi u tablici vremena za "setiranje" (tablica 5.6). Ako na listi nema raspoloživih poslova, resurs se postavlja u stanje mirovanja. Ako se pronađe raspoloživ posao onda se izračuna vrijeme koje se zahtijeva da se obavi posao i posao se preda resursu. Nakon toga sat je uvećan za jedan i proces se vraća na početak petlje.

Kada su završili svi poslovi, napušta se, gore opisana, petlja i onda se mogu izračunati različiti statistički pokazatelji kao što su duljina izvođenja poslova, kazneni troškovi, cijena,

prosječno vrijeme izvođenja po poslu, iskoristivost resursa, itd. Ovo se proslijedi natrag u MCGA radi analize.

5.4. Sažetak poglavlja

Ovo poglavlje detaljno opisuje problem rasporeda poslova, problem koji će se u narednim poglavljima ove disertacije pokušati riješiti. Opisan je i kroz primjere argumentiran općenit problem rasporeda poslova. Predstavljeni su različiti tipovi rasporeda poslova. Prikazana je i implementacija jednostavnog GA, kao najčešća metoda za rješavanje problema rasporeda. Opisan je način kreiranja kromosoma, te realizacija genetskih operatora selekcije, mutacije i križanja.

Također, opisan je i problem više-kriterijskog određivanja rasporeda poslova. Predstavljen je MCGA algoritam za koji se pokazalo da ima potencijala u više-kriterijskoj optimalizaciji procesa [TOD97]. Procedura MCGA algoritma, kojom se generira raspored poslova, dopušta istodobnu i minimalizaciju i maksimalizaciju obzirom na različite kriterije optimalizacije.

Za mnoge aplikacije, općeniti model problema rasporeda poslova nije dovoljno detaljan i ne prikazuje realno te probleme. Naime, u definiciju problema rasporeda treba ugraditi i druge značajke, kao što su projekti, višeradni resursi, povezanost poslova, vrijeme setiranja i poslova i resursa, kapaciteti resursa. U tom kontekstu, u sljedećem poglavlju razmatrat će se i definirati složeniji raspored poslova koji će uključiti navedene značajke u cilju da se omogući traženje optimalnog rasporeda poslova obzirom na različite kriterije i različite zahtjeve korisnika.

6. FORMALIZACIJA INTEGRACIJE PETRIJEVE MREŽE I GENETSKOG ALGORITMA

U ovom poglavlju predložit će se metodologija integracije Petrijeve mreže i genetskog algoritma za modeliranje i određivanje rasporeda poslova u sustavima s diskretnim događajima. Mnoga su istraživanja provedena kako bi se integrirala simulacija i modeli odlučivanja.

Prva istraživanja kako integrirati DES simulaciju i optimalizacijski model učinjena su u FMS domeni, kako bi se riješio problem rasporeda. Najčešće, problem integracije nije shvaćen u potpunosti, što je osobito važno kad se razvijaju jako složeni modeli.

Među razvijenim modelima bilo je i pokušaja povezivanja PM s metodama pretraživanja koje se temelje na umjetnoj inteligenciji, kao mehanizma rasuđivanja pri rješavanju problema rasporeda u proizvodnim sustavima. Svaki od pristupa započinje modeliranjem sustava Petrijevom mrežom. Zatim se primjenjuje heuristička metoda pretraživanja, kao što je ograničeno širinsko pretraživanje (engl. *beam search*) ili pretraživanje A* algoritmom [LEE94] za određivanje optimalnog rješenja u PM. Rješavanje problema pretraživanja metodom grananja i granica proučavali su Chen, Yu, Zhang [CHE93] i Lloyd, Yu, Konstas [LLO95]. Onaga, Silva i Watanebe [ONA91] su određivali raspored poslova u sustavu modeliranom Petrijevom mrežom primjenom linearnog programiranja.

Sadašnja istraživanja usmjerena su na uporabu meta-heurističkih metoda kao što su genetski algoritmi, simulirano kaljenje, tabu pretraživanje (engl. *tabu search*) ili na uporabu hibridnih metoda koje uključuju jedan ili više navedenih pristupa rješenju problema.

Rješavanjem problema rasporeda uporabom GA bavili su se brojni istraživači, uključujući Davis [DAV85], Biegel i Davern [BIE90], Bierwirth [BIE95], Croce [CRO95], Dorndorf i Pesch [DOR95]. Chen i ost. [CHE01] predlažu model razvijen obojenom Petrijevom mrežom za proizvodnju poluvodičkih sklopova i uporabu GA za traženje najpovoljnijih pravila dodjeljivanja određene grupe strojeva pojedinim operacijama.

U prethodnim pristupima modeliranju sustava Petrijevim mrežama, mjesta su predstavljala resurse i uvjete, dok su prijelazi predstavljali operacije.

Za razliku od gore navedenog klasičnog pristupa, u ovom radu će se i operacije i resursi prikazati mjestima. Koristit će se $\mathbf{MRF_1}$ *klasa P-vremenske Petrijeve mreže (vrijeme će se pridružiti mjestima). Sustav će se modelirati sa disjunktnim skupom resursa, poslova i kontrolnih mjesta.*

U modelu integracije, potrebno je analizirati strukturna svojstva DES, kao što su postojanje konflikata i zastoja. Konflikti i zastoji u sustavu mogu uzrokovati nepotrebne troškove, nedovoljnu uporabu skupih i važnih resursa ili dulje vrijeme izvođenja procesa. Kako u sustavima s diskretnim događajima, distribucija resursa može dovesti do konfliktnih situacija i zastoja, mnogi istraživači su razvili algoritme za određivanje rasporeda poslova bez zastoja. Međutim, mali broj istraživača se bavi strukturnom analizom pri razvijanju svog integracijskog modela.

Gang i Wu (2004) [GAN04] su predložili algoritam za određivanje rasporeda bez zastoja koji kombinira GA i pretraživanja grafa dostupnih stanja Petrijeve mreže. Međutim, ako se radi o složenijem sustavu analiza je presložena.

Uporabu pravila za rješavanje konfliktnih situacija u sustavu modeliranom Petrijevom mrežom (engl. *conflict resolution rules*) proučavali su Camurri, Franchi, Gandolfo i Zaccaria (1993) [CAM93], Huang i Chang (1992) [HUA92], Takamura i Hatono (1991) [TAK91]. Cavory i ostali (2005) [CAV95] predložili su rješenje problema kružnog rasporeda poslova s linearnim ograničenjima. Autori su koristili PM za modeliranje proizvodnog sustava, a linearna ograničenja su iskoristili za određivanje prioriteta između pojedinih poslova. Također, pomoću GA razvili su i algoritam za rješavanje konfliktnih situacija. Dakle, kod ovog pristupa svako ograničenje se mora ugraditi u GA, što ima za posljedicu jako opsežan programski kod razvijene aplikacije.

U ovom radu će se primjenom matrične algebre analizirati strukturna svojstva PM – kružna čekanja među resursima, P-invarijanta, kritični sifon, kritični podsustav, ključni resursi, itd. Prva i druga razina zastoja može se izbjeći ograničavanjem broja oznaka u kritičnim podsustavima i osiguravanjem da ključni resurs ne bude i posljednji raspoloživi resurs u sustavu.

U skladu s potrebom dodjele resursa poslovima u sustavu na vrijeme te izbjegavanja konflikata i zastoja, ovo istraživanje primjenjuje integrirani model P-vremenske Petrijeve mreže i genetskog algoritma za simuliranje i optimiranje sustava s diskretnim događajima. Na taj način generira se raspored poslova koji minimalizira ukupno vrijeme izvođenja svih poslova u sustavu (makespan) i pri tome se osigurava raspoloživost resursa.

Kako je nizom okidanja prijelaza vremenske Petrijeve mreže moguće odredit raspored poslova u sustavu bez konflikata i zastoja, genetskim algoritmom se mora odrediti vrijeme početka i završetka svakog posla kako bi ukupno vrijeme izvođenja bilo što kraće.

Dok većina autora koristi direktnu reprezentaciju kromosoma u kojima slijed prijelaza u PM odgovara kromosomu [CAV98], Chen i ost. [CHE01] koristili su indirektnu reprezentaciju kromosoma, tj. geni su odgovarali pravilima određivanja prioriteta.

U ovom je poglavlju opisana integracija PM i GA (u nastavku PMGA) koja će se primijenit u području upravljanja i optimiranja projektima s ograničenim resursima. Danas se mnoštvo poslova organizira u obliku različitih projekata. Kao primjer upravljanja projektima uzet je sustav prometa morskim kanalima (pokrata: SPMK). Pomorski promet je važan aspekt ekonomije svake pomorske države, koji zahtijeva sigurno i učinkovito upravljanje. Važan problem u upravljanju pomorskim prometnim sustavom je uspostaviti ravnotežu između brodovlasnika koji zahtijevaju brze usluge svojih brodova i ekonomičnu uporabu dodijeljenih resursa. Kako je SPMK velik, a resursi skupi, poželjno ih je iskoristiti što je moguće bolje.

PMGA modelom moguće je sagledati cjelokupnu situaciju u sustavu u cilju donošenja što boljih strateških odluka. Sustav SPMK će se modelirati MRF klasom Petrijevih mreža. Također, *bit će razvijen algoritam koji će poboljšati i optimalizirati sustav i pri tome spriječiti zastoje. Problem koji se rješava odnosi se na određivanje složenog rasporeda poslova tako da se minimalizira ukupno vrijeme izvođenja svih poslova, kao i ukupna cijena, koja se odnosi na resurse.*

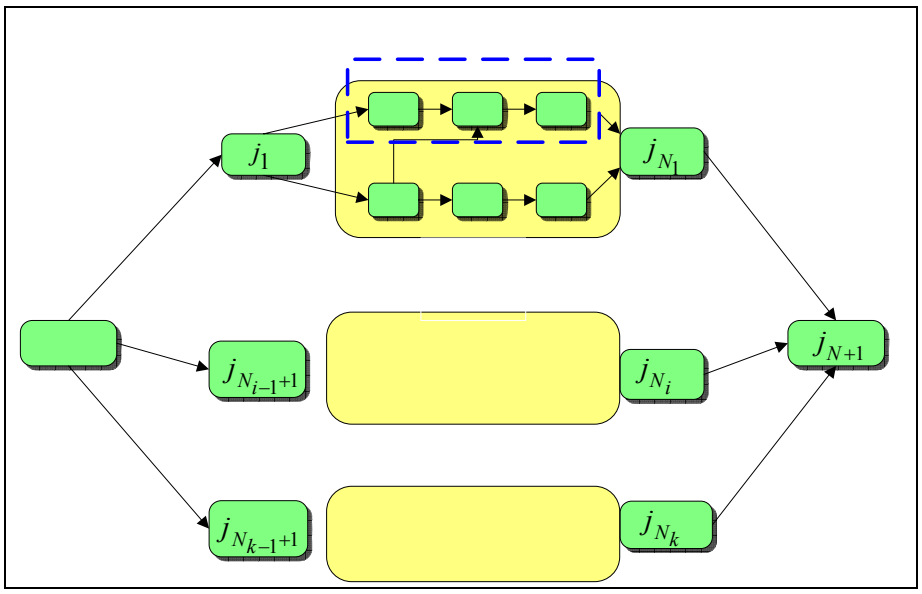
Predloženi model razvit će se i vrednovati računalnim programom u Matlab okruženju.

6.1. Opis problema i konceptualni model

Danas se mnoštvo poslova organizira u obliku različitih projekata. Projekt se sastoji od određenog broja poslova. Svaki posao traje određeno vrijeme i izvodi se pomoću određenog resursa.

Većina projekata nema dovoljan broj resursa na raspolaganju. Najčešći je problem nedostatak educirane i kvalitetne radne snage, ali također nedostatna novčana sredstva. Što se tiče određivanja rasporeda poslova kod više projekata, treba se ustvrditi koji su resursi kritični kako bi se smanjila mogućnost pojave konflikta i kako bi ukupno vrijeme izvođenja svih poslova bilo što kraće.

Problem i konceptualni model više-projektnog sustava može se opisati kao mreža projekata (slika 6.1). Problem se sastoji od $\mathcal{P}$ projekata. S različitim vremenima nastupanja, trajanja i završetka, *projekt* je privremeni, ciljno usmjeren skup poslova koji se obavljaju kako bi se napravio neki proizvod, usluga ili neki drugi rezultat. Realizacija poslova povezana je s uporabom resursa.



Slika 6.1 Mrežni plan primjera više-projektnog sustava

Dakle, svaki projekt sastoji od skupa poslova $\{j_{N_{i-1}+1},\ldots,j_{N_i}\}$ i skupa resursa $R=\{r_1,r_2,\ldots,r_{\mathcal{K}}\}$, gdje je $\mathcal{K}$ ukupan broj resursa u sustavu ($|R|=\mathcal{K}$). Poslovi $j_{N_{i-1}+1}$ i j_{N_i} nazivaju se *prividni* ili fiktivni poslovi (engl. *dummy jobs*) i predstavljaju početni i završni posao *i*-tog projekta. Ta dva posla traju 0 vremenskih jedinica i ne trebaju resurse. Svi ostali poslovi imaju točno određen početak i kraj, definiran rezultat i zahtijevaju barem jedan resurs. Skup $\mathcal{J}$ sadrži sve projektne poslove, kao i početni (engl. *source*) i završni (engl. *sink*) posao. Kritični elementi projekta su: resursi (ljudi, strojevi, materijal, oprema), vrijeme (predviđeno za dovršenje projekta) i sredstva (na raspolaganju za projekt). Rezultati projekta su svojstva proizvoda koji se isporučuje (engl. *features*).

Cilj svakog projekta ostvaruje se slijedom poslova kojima je točno definirano konačan rezultat, koje je predviđeno vrijeme izvođenja te koji se resursi rabe ili sudjeluju u radu. Svi poslovi raspoređene su u vremenski slijed ovisno o povezanosti rezultata rada te optimiranog rasporeda kako bi se skratilo ukupno vrijeme rada na projektu.

Međusobni odnos poslova određen je s dvije vrste ograničenja:

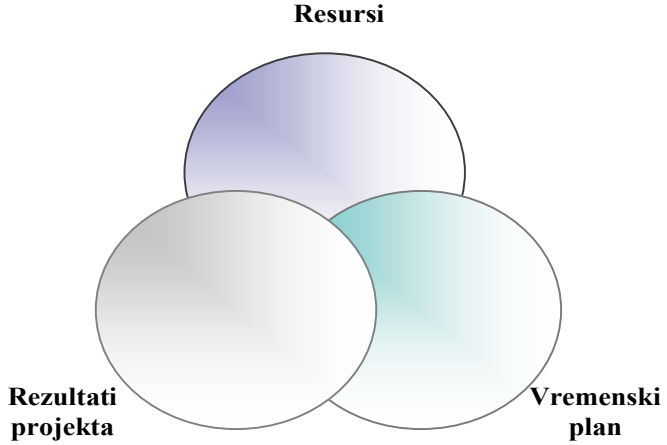
(1) *Ograničenje slijeda poslova* (engl. *precedence constraints*) – Posao $j \in \mathcal{J}$ ne može započeti prije nego završe svi poslovi, P_j, koji mu prethode.

(2) *Dostupnost resursa*: Svaki posao *j* za svoje izvođenje zahtijeva određeni resurs koji je ograničenog kapaciteta i izvođenje posla može započeti samo ako je taj resurs dostupan.

Tijekom izvođenja poslu *j* je potrebno $r_{j,k}$ jedinica resursa $r_k \in R$ u svakoj jedinici vremena za vrijeme njegova trajanja d_j. Resurs $r_k \in R,\ k = 1,...,\mathcal{K}$ ima ograničeni kapacitet $RD(k)$ za vrijeme izvođenja svih projekata i C_d je cijena resursa po jedinici raspoloživog kapaciteta. Pretpostavka je da su parametri d_j, $r_{j,k}$ i $RD(k)$ deterministički, cijeli, pozitivni brojevi. Za početni i završni posao vrijedi $d_0 = d_{n+1} = 0$ i $r_{0,k} = d_{n+1,k} = 0, \forall r_k \in R,\ k = 1,...,\mathcal{K}$.

Nakon što posao j_i napusti resurs i prije nego što sljedeći posao j_j zauzme taj isti resurs, potrebno je resurs "setirati", a to traje $S_{j_i j_j}$ vr.jed. $S_{0 j_j}$ je vrijeme setiranja koje se zahtijeva prije izvođenja posla j_j ako je to prvi posao dodijeljen resursu.

Kupci i dobavljači sudjeluju u projektu kako bi ostvarili različite, najčešće materijalne ciljeve. Kupci najčešće žele za svoj novac dobiti što je više moguće, a dobavljači žele isporučiti projekt s minimalnim ulaganjima, u što kraćem vremenu i s maksimalnom zaradom. Upravo su razlike između želja kupaca i dobavljača često uzrok kašnjenja i propadanja projekata. Dakle, upravljanje projektima je složen proces primjene znanja, vještina i alata te donošenja odluka kako bi projektni poslovi dali očekivani rezultat. Voditelj projekta mora uspješno balansirati trokutom koji uključuje resurse, rezultate projekta (primjerice, cijenu) i vremenski plan (vremenski tijek pojedinog projekta) [FRA06]. Trokut je prikazan Viennovim dijagramom (slika 6.2).



Slika 6.2 Trokut *resursi-rezultati-vrijeme*

Cilj optimalizacije je pronaći optimalno rješenje za jednu stranu trokuta. Uz pretpostavku da su rezultati i vremenski rokovi unaprijed zadani i nepromjenljivi, uz zadani raspored i kapacitete resursa pokušat će se pronaći optimalan projektni raspored, određen vektorom početaka poslova $s = (s_1, s_2, \cdots)$, tako da se minimalizira/maksimalizira neka mjera kvalitete rezultata (primjerice minimalizira se ukupno vrijeme trajanja projekta – makespan (oznaka c_{max}), a minimiziraju se kašnjenja).

Neka je F_j vrijeme završetka posla $j \in \mathcal{J}$. Raspored se može predstaviti vektorom vremena kada su poslovi završeni $(F_1, F_2, \ldots, F_N)$. Neka je $A(t)$ skup poslova koji se izvode u trenutku t. Tada se *formalni model* može definirati kako slijedi:

$$min\,(c_{max}) = min(f\,(F_1, F_2, \ldots, F_N)) \tag{6-1}$$

Uz ograničenja:

$$s_u + d_u + S^k{}_{uv} \le s_v, \quad u,v \in J \;\; v \in P_u, \;\; r_k \in R, \;\; k = 1,\ldots,\mathcal{K} \tag{6-2}$$

$$\sum_{j \in A(t)} r_{j,k} \le RD(k), \quad r_k \in R, \;\; k = 1,\ldots,\mathcal{K}, \;\; t \ge 0, \tag{6-3}$$

$$F_{N_{i-1}+1} \ge ERT_i, \quad i = 1,2,\ldots,\mathcal{P}, \tag{6-4}$$

$$s_{Ni} < DD_i, \quad i = 1,2,\ldots,\mathcal{P}, \tag{6-5}$$

$$F_j \ge 0, \;\; RD(k) \ge 0, \;\; j = 1,2,\ldots,N, \;\; r_k \in R, \;\; k = 1,\ldots,\mathcal{K}, \tag{6-6}$$

gdje je

ERT_i najraniji početak (engl. *Earliest Release Time*) za projekt *i*
DD_i krajnji rok (engl. *Due Date*) za izvođenje *i*-tog projekta.

Izraz (6-1) predstavlja funkciju cilja koju treba optimalizirati. Uvjet (6-2) definira ograničenje slijeda poslova za bilo koji par poslova (u,v) određenog projekta, pri čemu posao *u* neposredno prethodi poslu *v*. Ograničenje uključuje i vrijeme setiranja resursa: početak izvođenja posla $v \in \mathcal{J}$ može biti tek nakon isteka vremena "setiranja" $S^k{}_{uv}$ resursa $r_k \in R$ od trenutka kada ga je prethodni posao napustio. Izraz (6-3) ograničava, obzirom na kapacitet resursa, zahtjev za resursom od strane posla koji je započeo u trenutku *t*. Ograničenje (6-4) se odnosi na početak projekta. Ovo ograničenje je implicitno zadano relacijom (6-7) koja definira početak prvog posla svakog projekta tek nakon što je projekt raspoloživ,

$$s_{N_{i-1}+1} \ge ERT_i, \quad i = 1,\ldots,\mathcal{P}. \tag{6-7}$$

Izraz (6-5) osigurava da završni posao *i*-tog projekta započne prije zadanog roka kada cijeli projekt mora biti završen, dok izraz (6-6) ograničava vremena završetka poslova i kapacitete resursa ne-negativne vrijednosti.

Dakle problem optimalizacije rasporeda poslova u više-projektnom sustavu spada u više-kriterijske probleme rasporeda poslova i prema općoj shemi po kojoj se klasificiraju rasporedi (vidjeti podpoglavlje 5.1.1.), problem se može označiti kao

$$\alpha,p \,|\, \beta \,|\, \gamma, \Gamma \,|\, c_{max}$$

- α predstavlja broj poslova,
- p predstavlja broj projekata
- β predstavlja broj resursa
- γ ukazuje na redoslijed resursa kod poslova
- Γ ukazuje na neku specifičnost (npr. *sequence dependent*).

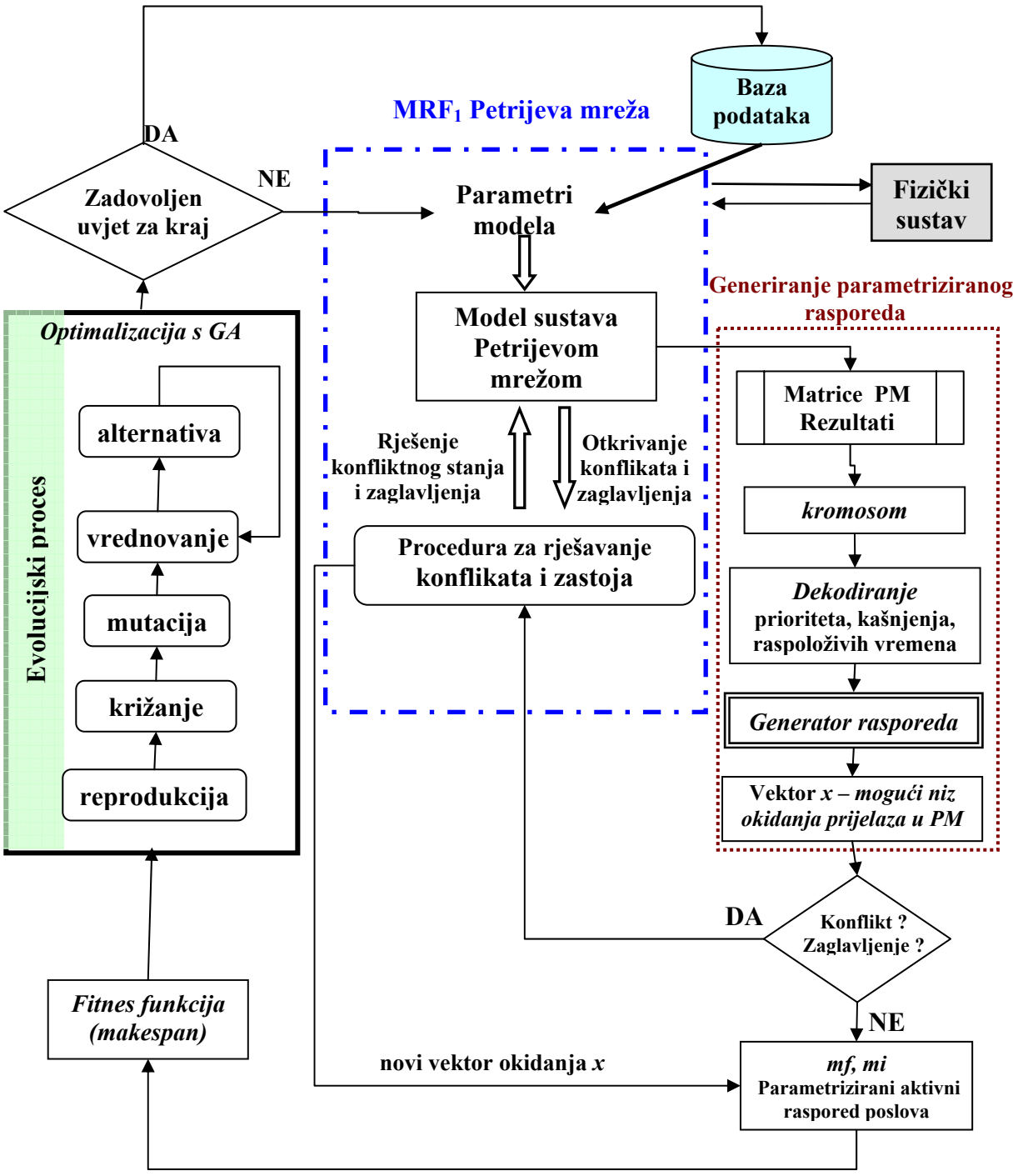
6.2. Novi pristup u rješavanju problema rasporeda poslova

Novi pristup u rješavanju problema rasporeda projektnih zadataka integrira MRF_1 klasu Petrijevih mreža i genetski algoritam, pokrata: PMGA, (slika 6.3). Cilj je pokazati primjenjivost PMGA na pronalaženju učinkovitog rasporeda poslova u više-projektnom sustavu sa više resursa.

Što MRF_1 Petrijeve mreže omogućuju upravitelju projekta?

MRF_1 nude brojne pogodnosti za upravitelja projekta koje se mogu sažeti kako slijedi:

1) Pomoću MRF_1 Petrijeve mreže može se modelirati i matrično analizirati sustav s diskretnim događajima, pri čemu se mnogi poslovi izvode asinkrono i konkurentski. Moguće je modelirati konkurentnost i konflikte, a postojanost zastoja u sustavu se primjenom i/ili matrične algebre jednostavno detektiraju.
2) Upravitelju projekta pružaju informacije o kašnjenjima u projektu.
3) MRF_1 PM su dinamička reprezentacija sustava i stoga su pogodne za on-line nadzor sustava. Iz postojećih matrica MRF_1 PM moguće je prikazati uzajamnu ovisnost među resursima, parcijalnu alokaciju i zamjene resursa, iskoristivost resursa, kao i međusobno isključivanje.



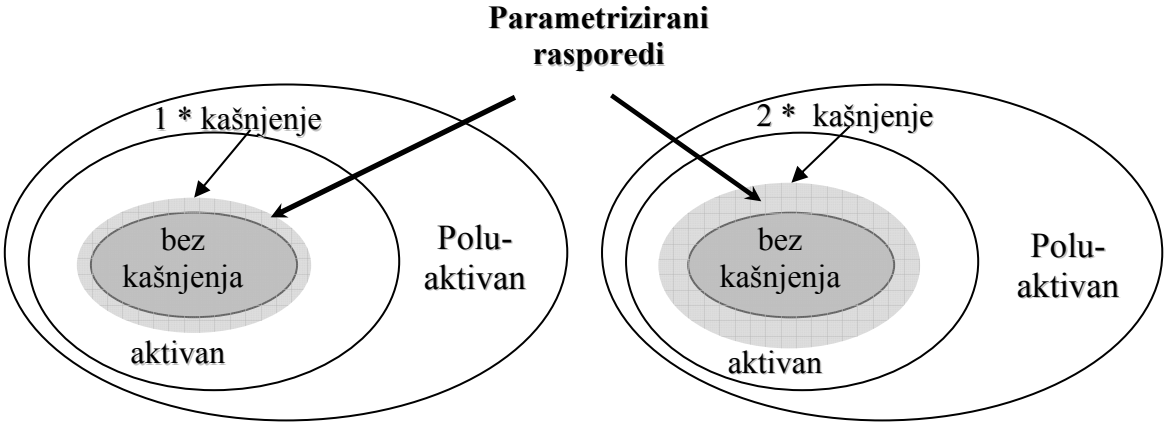
Slika 6.3 Arhitektura novog pristupa integracije Petrijeve mreže i genetskog algoritma

Genetski algoritam

GA je tehnika pretraživanja jako velikog prostora rješenja za globalnim optimumom. Kako je opisano u podpoglavlju 5.1.2.1, skup aktivnih rasporeda sadrži optimalan raspored (slika 6.4). Obično je skup aktivnih rasporeda jako velik i sadrži veliki broj rasporeda s relativno velikim kašnjenjima, što povlači prilično loša svojstva rezultata izvođenja poslova. Stoga, kontroliranjem vremena kašnjenja moguće je smanjiti ili povećati prostor mogućih

rješenja. Ako je maksimalno kašnjenje nula onda se prostor mogućih rješenja izjednačio sa prostorom rješenja (rasporeda) bez kašnjenja. U slučaju kad je vrijeme kašnjenja "beskonačno", tada je prostor rješenja jednak prostoru aktivnih rasporeda. Kako bi se smanjilo vrijeme kašnjenja, a time i prostor pretraživanja mogućih rješenja, u ovom modelu će se primijeniti koncept parametriziranog aktivnog rasporeda kao što je to predloženo u [GON05].

Slika 6.4 prikazuje gdje se nalazi skup paremetriziranih aktivnih rasporeda u odnosu na ostale načine raspoređivanja poslova. Ovisno o maksimalno dopustivom kašnjenju, moguće je smanjiti ili povećati prostor rješenja. Ako je vrijednost maksimalnog dopustivog kašnjenja jednaka nuli onda se prostor rješenja svodi na rasporede bez kašnjenja.



Slika 6.4 Parametrizirani aktivni raspored

GA je odgovoran za vrednovanje kromosoma koji predstavlja prioritete projekata, kašnjenja poslova te vremena raspoloživosti projekata, odnosno resursa. Zadatak je optimalizirati vremena izvođenja projekta i kašnjenja u sustavu. Nadalje, svaki kromosom mora proći kroz dvije faze:

(a) dekodiranje prioriteta, kašnjenja i vremena kada su poslovi i resursi raspoloživi;

(b) određivanje rasporeda projektnih zadataka.

6.3. FPM Petrijeva mreža

Za modeliranje pomorskog prometnog sustava, u ovom poglavlju koristit će se Petrijeva mreža za sustave proizvodnih linija (engl. *Flowline System Petri Net*, pokrata: FPM). FPM je podklasa P/N Petrijevih mreža koja je prvenstveno namijenjena analizi sustava koji pripadaju klasi nazvanoj višeprolazne proizvodne linije (engl. *multiple reentrant flowlines*, pokrata: MRF).

Neka je Π skup različitih objekata (proizvoda ili usluga) koji prolaze kroz MRF sustava i nad svakim objektom $k \in \Pi$ izvodi se strogo određen niz poslova $J^k = \left\{ J_1^k, J_2^k, \ldots, J_{N_k}^k \right\}$ sa barem jednim iskorištenim resursom za svaki posao. Dakle, u sustavu nema izbora poslova. Neka je R skup resursa u sustavu.

Kod FPM mreže skup mjesta P je podijeljen kako slijedi

$$P = R \cup J \cup J_{in} \cup J_{out} \tag{6-8}$$

gdje R, J, J_{in}, J_{out} označavaju skupove, pojedinačno, raspoložive resurse, skup poslova koje su u toku, objekte koji pristižu u sustav, objekte koji su završeni. Skup prijelaza T se može prikazati kao

$$T = \bigcup_{k \in \Pi} T^k \text{ , gdje je } T^k = \left\{ t_1^k, t_2^k, \ldots, t_{L_k+1}^k \right\} \text{ i } t_i^k = \bullet J_i^k = J_{i-1}^k \bullet, \text{ za } i \notin \{1, N_k\}.$$

Vrijedi $t_1^k = \bullet J_1^k = J_{in}^k \bullet$ i $t_{N_k+1}^k = \bullet J_{out}^k = J_{N_k}^k \bullet$.

Za prijelaz *t* se kaže da je posao (resurs) *omogućen* (raspoloživ) ako je $m(\bullet t \cap J) > 0$ i $m(\bullet t \cap R) > 0$. Za svaki $r \in R$, skup poslova koji koriste resurs r označava se kao $J(r)$ i petlja resursa kao $L(r) = r \cup J(r)$. $R(J_i^k)$ označava resurse koje traži posao J_i^k.

Sustav opisan u ovom poglavlju pripada klasi MRF_1 sustava.

Definicija 6.1 Klasa MRF_1 sustava

MRF_1 je podklasa MRF sustava opisana FPM mrežom, tako da vrijedi:

a) Sustav nema samostalnih petlji, odnosno $\forall p \in P, \quad \bullet p \cap p \bullet \neq \varnothing$;

b) U sustavu nema izbora poslova, $\forall J_i^k \in J, \ \left| J_i^k \bullet \right| = 1$;

c) U sustavu nema poslova sklapanja, $\forall t_i^k \in J, \ \left| \bullet t_i^k \cap J \right| \leq 1$ [17];

d) Jedan posao traži jedan i samo jedan resurs, $\forall J_i^k \in J, \ \left| R(J_i^k) \right| = 1$;

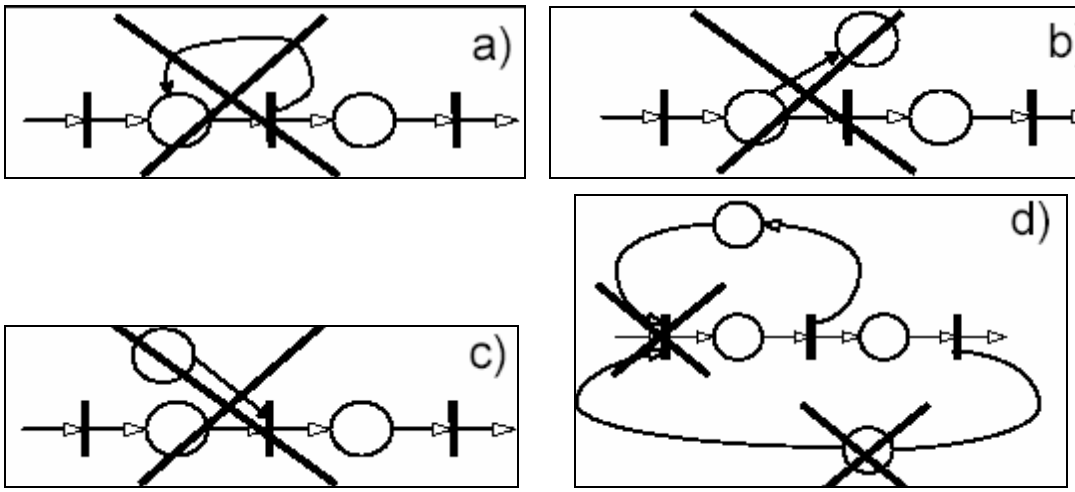
e) Definiran je ulaz i izlaz iz sustava, odnosno, $\forall k \in \Pi, \ t_1^k \bullet \cap P \setminus J = \varnothing$ i $\varnothing$

f) Ne postoje dva posla, koji slijede jedan za drugim, a traže isti resurs, $\forall J_i^k \in J, \ R(J_i^k) \neq R(J_{i+1}^k)$;

g) Kako su u sustavu određeni poslovi potpuno identični i zahtijevaju iste resurse, znači da postoje višeradni[18] resursi, koji se definiraju kao $r \in R,$ ako vrijedi $\left| J(r) \right| \geq 1$.

Na slici 6.5 grafički su objašnjeni uvjeti a, b, c, d.

[17] Operator $\bullet$ je opisan u podpoglavlju 3.1 i definira ulazne i izlazne elemente nekog dijela PM.
[18] U sustavu mogu postojati dva podskupa resursa: jednoradni i višeradni.



Slika 6.5 MRF_1 tip FPM Petrijeve mreže

Višeradni resursi u MRF sustavima vode do konflikata. Naime, višeradni resursi predstavljaju problem jer pogreška u dodjeli resursa može dovesti do zastoja u sustavu.

Definicija 6.2 Konflikt

Konflikt (engl. *Conflict*) je situacija kada okidanje jednog prijelaza onemogućuje okidanje jednog ili više drugih prijelaza.

Konfliktnim prijelazima potrebno je upravljati tako da se uvedu kontrolna mjesta koja će se povezati sa svakim konfliktnim prijelazom. Upravljanjem brojem oznaka u kontrolnim mjestima mora se omogućiti okidanje samo jednog prijelaza.

6.4. Matrični model MRF_1 Petrijeve mreže

Strukturna svojstva MRF_1 mreže mogu se očitati iz matrica sustava. Matrični model omogućuje i strožu analizu MRF_1, uključujući kružna čekanja, sifone i zastoje.

Postoje dva skupa matrica $\mathbf{F_u}, \mathbf{F_y}, \mathbf{F_v}, \mathbf{F_r}$ i $\mathbf{S_u}, \mathbf{S_v}, \mathbf{S_r}, \mathbf{S_y}$. Matrice $\mathbf{F}$ opisuju uvjete koji se trebaju zadovoljiti prije nego se okine prijelaz, dok su matrice $\mathbf{S}$ važne za događanja nakon što se prijelazi okinu. Broj redaka u matricama $\mathbf{F_u}, \mathbf{F_v}, \mathbf{F_r}, \mathbf{F_y}$ definira broj prijelaza, dok broj stupaca definira broj ulaznih mjesta, poslove, resurse i izlazna mjesta. Broj stupaca u matricama $\mathbf{S_u}, \mathbf{S_v}, \mathbf{S_r}, \mathbf{S_y}$ definira broj prijelaza, dok broj redaka definira broj ulaznih mjesta, poslove, resurse i izlazna mjesta. Ulazna matrica $\mathbf{F_u}$ definira izlazne lukove iz ulaznih mjesta, dok izlazna matrica $\mathbf{S_y}$ definira ulazne lukove u izlazna mjesta. Kako se pretpostavlja da su ulazna mjesta bez ulaznih prijelaza i izlazna mjesta bez izlaznih prijelaza, matrice $\mathbf{F_y}$ i $\mathbf{S_u}$ su nul-matrice, $\mathbf{F_y} = \mathbf{S_u} = [0]$.

$\mathbf{F_v}$ je matrica slijeda poslova i njeni elementi su

$$F_v(i,j) = \begin{cases} 1, & \text{ako se treba izvršiti posao } j \text{ kao neposredni prethodnik posla } i \\ 0, & \text{inače.} \end{cases}$$

$\mathbf{F_r}$ je matrica potrebnih resursa (engl, *resource requirements matrix*). Iz matrice $\mathbf{F_r}$ program razumije koji je resurs potreban za okidanje kojeg prijelaza (za izvođenje kojeg posla).

Elementi matrice su

$$F_r(i,j) = \begin{cases} 1, & \text{ako je resurs } j \text{ potreban za izvođenje posla } i \\ 0, & \text{inače.} \end{cases}$$

$\mathbf{S_v}$ je matrica pokrenutih poslova (engl. *job start matrix*) i njeni elementi su

$$S_v(i,j) = \begin{cases} 1, & \text{ako posao } i \text{ treba započeti kada se zadovolje svi zahtijevi ili logički uvjet } x_j \\ 0, & \text{inače.} \end{cases}$$

Dakle, vrijednost '1' u matricama $\mathbf{F_v}$ i $\mathbf{S_v}$ predstavljaju lukove koji povezuju prijelaza i mjesta pridruženi poslovima koje izvršavaju resursi.

Vrijednost '1' u matrici raspoloživih resursa (engl. *resource release matrix*) $\mathbf{S_r}$ definira povezanost između PM prijelaza i mjesta koja sadrže oznake kada je resurs raspoloživ.

Matrični model MRF_1 mreže može se opisati sljedećim jednadžbama [TAC97]:

- *Matrični model jednadžbe stanja* DES:
$$\overline{x} = F_v \cdot \overline{v}_c + F_r \cdot \overline{r}_c + F_u \cdot \overline{u} + F_D \cdot \overline{u}_D \qquad (6\text{-}9)$$
- *Početna jednadžba:*
$$v_s = S_v \cdot x\,, \qquad (6\text{-}10)$$
- *Jednadžba raspoloživosti resursa:*
$$r_s = S_r \cdot x\,, \qquad (6\text{-}11)$$
- *Izlazna jednadžba:*
$$y = S_y \cdot x\,. \qquad (6\text{-}12)$$

Ovo su logičke jednadžbe gdje operator '+' označava logički *'ili'* operator ili operaciju disjunkcije, a znak '$\cdot$' označava logički *'i'* operator ili operaciju konjunkcije, a gornja povlaka '$\overline{\ }$' označava logički 'ne' operator ili negaciju.

Vektor v_c predstavlja stanje izlaza i ako je *i*-ti element vektora jednak 1 znači da je završen *i*-ti posao. Zapis '1' u vektoru r_c predstavlja trenutno raspoloživ resurs.

Jednadžba (6-9), analogno matričnoj diferencijalnoj jednadžbi $\dot{x} = Ax + Bu$ u teoriji upravljanja sustavima, provjerava uvjete koji se moraju zadovoljiti kako bi se izvršio sljedeći posao u sustavu. Na osnovi tih uvjeta, spremljenih u logički vektor *x*, jednadžba početka izvođenja posla (6-10) računa koji se poslovi aktiviraju i mogu započeti. Zapis '1' u vektoru

v_s označava koji posao se počeo izvršavati. Jednadžba (6-11), prema završenim poslovima, računa koji su resursi raspoloživi. Zapis '1' u vektoru r_s označava koji resurs je raspoloživ.

Indeksi *c* i *s* su preuzeti iz engleskog govornog područja i označavaju, redom, *complete*[19] ili *current*[20], odnosno *start*. Iz sustava jednadžbi (6-9) - (6-12) mogu se definirati ulazna i izlazna matrica događanja, **I** i **O**:

$$\mathbf{I} = \left[\mathbf{F_u} \quad \mathbf{F_v} \quad \mathbf{F_r} \quad \mathbf{F_y} \right] = \mathbf{F} \tag{6-13}$$

$$\mathbf{O} = \left[\mathbf{S_u^T} \quad \mathbf{S_v^T} \quad \mathbf{S_r^T} \quad \mathbf{S_y^T} \right] = \mathbf{S^T}. \tag{6-14}$$

6.4.1. Izbjegavanje zastoja u MRF1

Kako je navedeno u točki 5.1.2.2, kružno čekanje je nužan preduvjet za zastoj sustava. Nužan, ali ne i dovoljan uvjet za postojanje kružnog čekanja jest postojanje barem jednog višeradnog resursa [BOG06].

Iz kružnog čekanja MRF_1 mreža se može naći u stanju zastoja koji se zove *kružno blokiranje* CB (vidjeti dodatak D3, definicija 0.3).

Izbjegavanje CB je nužno, ali općenito nije dovoljno za izbjegavanje zastoja u sustavu. Kako bi se izbjegli zastoji u MRF_1 sustavu prvo je potrebno izbjeći uvjete za CB, koji su usko povezani sa kritičnim sifonom i kritičnom zamkom (definicija 0.3). *Kritični sifon S* (*zamka Q*) je minimalni sifon (zamka) čije pražnjenje nužno dovodi do zastoja [KEZ09]. Da bi se izbjeglo pražnjenje kritičnih sifona, nužno je odrediti skup poslova čije punjenje dovodi do pražnjenja kritičnih sifona. Uvjet da sustav nije u stanju kružnog blokiranja iznesen je u teoremu 7.1. Prema teoremu kako bi se izbjegla sva kružna blokiranja, ukupan broj poslova koji se izvode unutar kritičnog podsustava mora biti manji ili jednak od $m_0(CW)-1$, gdje je $m_0(CW)$ inicijalni broj oznaka u mjestima kružnog čekanja. Kako bi se izbjegli takvi zastoji prve razine, nužno je uvesti kontrolna mjesta u svakom kritičnom podsustavu te povezati ta kontrolna mjesta sa prijelazima prije i nakon svakog kritičnog podsustava.

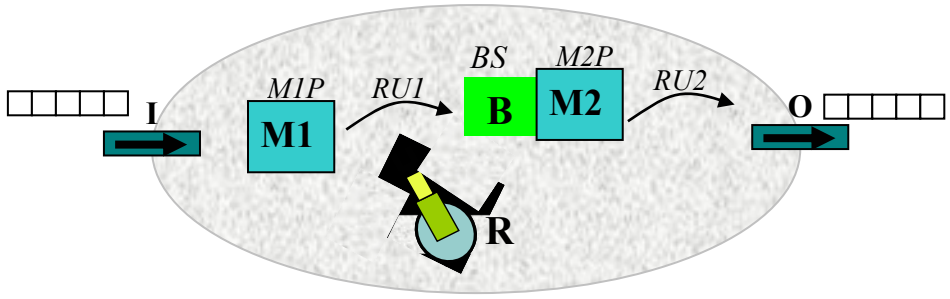
Za MRF sustave koji se nalaze u stanju zastoja druge razine nije istinita tvrdnja da je izbjegavanje CB nužan i dovoljan uvjet za izbjegavanja zastoja u sustavu. Takvi sustavi posjeduju ključne resurse (definicija 0.5, dodatak D3) i nalaze se u stanju koje se naziva cikličko kružno čekanje. Kako bi se izbjegao zastoj druge razine potrebno je pronaći ključni resurs i primijeniti proceduru kojom se osigurava da ključni resurs ne ostane i posljednji raspoloživi resurs. U [KEZ09] autori su opisali metodu računanja kontrolnih mjesta pomoću *P*-invarijante koji kontroliraju konflikte i ograničavaju dostupna stanja kako bi se izbjegli zastoji prve i druge razine.

Primjer 6.1 Određivanje kritičnog sifona u MRF_1 sustavu [KOV02]

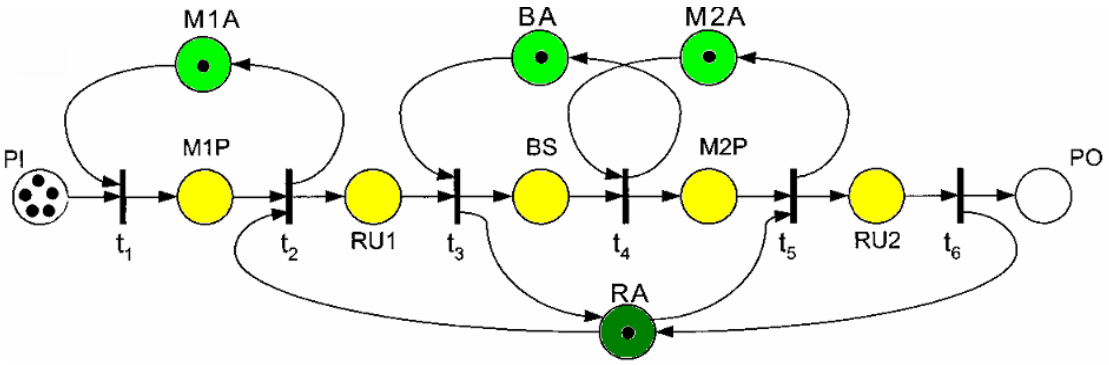
Dio fleksibilnog proizvodnog sustava prikazan je slikom 6.6a. Prikazana je proizvodna linija nekog proizvoda koja zahtijeva niz poslova i skup resursa. Robot R je zajednički resurs

[19] hrv. *završen*
[20] hrv. tekući, trenutni

za poslove pomicanja proizvoda - RU1 i RU2. Odgovarajuća PM prikazana je na slici 6.6b, a značenje mjesta dana je u tablici 6.1.



(a)



(b)

Slika 6.6 Primjer fleksibilni proizvodni sustav opisan Petrijevom mrežom

Tablica 6.1 Značenje mjesta u PM grafu sa slike 6.6b

PI	Ulazni spremnik
M1A	Stroj M1 je dostupan
M1P	Stroj M1 obavlja posao
RU1	Robot pomiče proizvod sa stroja M1 do spremnika B
BS	Spremnik B je pun
BA	Spremnik B je dostupan
RA	Robot R je dostupan
M2A	Stroj M2 je dostupan
M2P	Stroj M2 obavlja posao
RU2	Robot R pomiče proizvod sa stroja M2 do izlaza
PO	Izlazni spremnik

Petrijeva mreža sa slike 6.6(b) pripada MRF_1 klasi i u mreži se mogu pronaći strukturna svojstva navedena u prethodnom tekstu. Kružna čekanja mogu se odrediti izravno iz PM grafa. Kako svako kružno čekanje uključuje barem jedan zajednički resurs, kružna čekanja se mogu odrediti tako da se prate lukovi iz Petrijeve mreže koji povezuju mjesta resursa počevši od zajedničkog resursa. Zajednički resurs RA ima dva izlazna luka, jedan je povezan s prijelazom t_2, a drugi s prijelazom t_5. Ako se prati proizvodna linija od RA prema

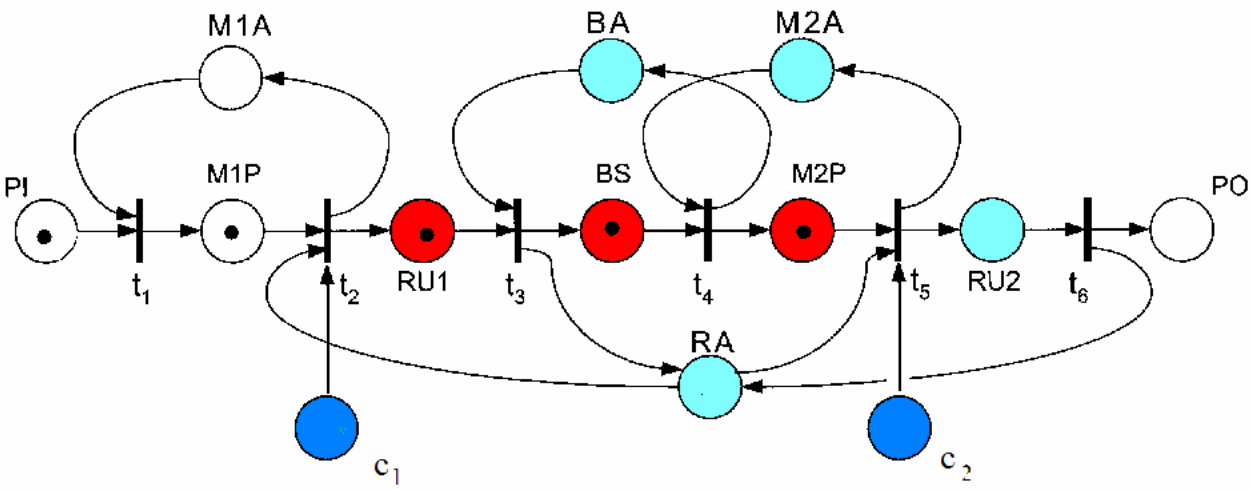
t_2, može se nastaviti lukom $t_2 \rightarrow$M1A, i nadalje po luku M1A $\rightarrow t_1$. Kako ne postoje lukovi koji povezuju mjesta resursa sa prijelazom t_1, linija nije zatvorena. Očito je da nije riječ o kružnom čekanju. S druge strane, ako se prati luk RA$\rightarrow t_5$ staza se nastavlja preko M2A do BA. Preko prijelaza t_3 staza se ponovo vrati u mjesto RA i tako je zatvoreno kružno čekanje. Dakle, resursi u kružnom čekanju su BA, M2A i RA. Pripadajući sifon je BA, M2A, RA i RU2. Kritični podsustav sadrži mjesta RU1, BS i M2P [KEZ09, BOG06].

■

Primjer 6.2 Zastoj MRF_1 sustava

Na slici 6.6b može se vidjeti kako izgleda sustav u konfliktu. Mjesto RA sadrži jednu oznaku koja može otići ili u RU1 ili u RU2.

Kao što je već prije rečeno, oznaka ne može istovremeno otići na oba mjesta nego je potrebno upravljačkim mjestima (c_1, c_2 na slici 6.7) na prijelazima t_2 i t_5 odlučiti koji će prijelaz okinuti.



Slika 6.7 Sustav u kružnom blokiranju

Primjerice, ako je omogućen prijelaz t_2 onda oznaka prelazi na mjesto RU1. U tom slučaju sustav se nalazi u stanju prikazanom slikom 6.7. Oznaka na mjestu RU1 ne može krenuti dalje dok se ne pojavi oznaka na mjestu BA, ona se pak ne može pojaviti dok se ne pojavi oznaka u M2A, koja se također ne može pojaviti dok se ne pojavi oznaka u RA. Oznaka u RA trebala bi doći iz RU1 i time je krug zatvoren. Kružno čekanje $CW = \{BA, M2A, RA\}$ je u stanju kružnog blokiranja jer niti jedan resurs nije spreman za nastavak rada. Posljedica kružnog blokiranja je zastoj u PM (niti jedan prijelaz nije moguće okinuti).

Resurs RA sa mjestima RU1 i RU2 čini *P*-invarijantu. Na slici 6.7 može se vidjeti da je operacija RU1 uzela predmet. Ukoliko bi RU1 bio u mogućnosti "vratiti" predmet nazad te resurs RA osloboditi da bi izvršio RU2, tada bi sustav izašao iz blokiranja. Jedna od značajki koji posjeduje i MRF_1 jest to da jednom zauzet resurs, ostaje zauzet dok se zadatak ne izvrši

do kraja, pa stoga takva mogućnost ne postoji. Stoga je jedino rješenje u upravljanju koje je umjesto prijelaza t_2 trebalo omogućiti prijelaz t_5.

■

6.5. Model implementacije genetskog algoritma

U ovom podpoglavlju opisat će se genetski algoritam koji će se koristi za optimalizaciju sustava sa diskretnim događajima za potrebe određivanja rasporeda poslova. Model koji se predlaže za integraciju započinje time da se definira funkcija cilja i ograničenja problema. Iz PM, GA proceduri moraju se proslijediti informacije o mreži, kao što su vrijeme trajanja poslova, relacije koje se odnose na slijed poslova, raspoloživost resursa, potrebe poslova za resursima. Ove informacije su potrebne jer je svaki projekt poseban. GA parametre, kao što su broj eksperimenata, veličina populacije i vjerojatnost genetskih operatora odredi sam korisnik.

Potrebno je odrediti sintaksu kromosoma, tumačenje i vrednovanje kromosoma kao i potrebni skup genetskih operatora, koji djeluju na kromosomima. Na osnovi podataka iz vremenske MRF_1 PM, određuje se optimalno vrijeme kada će posao započeti. Početna populacija se generira na osnovi podataka iz Petrijeve mreže. Ulazni parametri u genetski algoritam su matrice *F i S*.

6.5.1. Struktura kromosoma i postupak dekodiranja

U genetskom algoritmu koji se rabi u ovom poglavlju kromosom je prikazan slučajnim distribucijom brojeva iz intervala (0,1). Da bi se omogućilo vrednovanje funkcije cilja za svaku jedinku, potrebno je kromosome dekodirati u odgovarajuće rješenje problema, odnosno u raspored poslova. Važna značajka prikaza kromosoma slučajnim realnim brojevima je ta što su potomci, koji su nastali križanjem roditelja, moguća rješenja problema. Naime, ako je moguće vektor slučajnih brojeva protumačiti kao moguće rješenje, onda je moguće primijeniti i operator križanja na takvo rješenje. Kroz dinamiku GA, sustav uči o odnosima između niza slučajno generiranih brojeva i rješenja problema za koje funkcija cilja postiže dobre vrijednosti.

Svaki kromosom je sastavljen od $m+2n+x$ gena, gdje je n broj poslova, m broj projekata, a x broj resursa:

$$kromosom = \left(\underbrace{p_1, p_2, \ldots, p_m}_{prioriteti}, \underbrace{k_1, k_2, \ldots, k_n}_{kašnjenja}, \underbrace{rt_1, rt_2, \ldots, rt_n}_{vrijeme\ raspoloživosti\ projekata}, \underbrace{Rt_1, Rt_2, \ldots, Rt_x}_{vrijeme\ raspoloživosti\ resursa} \right). \quad (6\text{-}15)$$

Dakle, koristi se indirektna reprezentacija kromosoma. Razlog je taj što je direktna reprezentacija kromosoma presložena za predstavljanje rasporeda poslova u više-projektnom sustavu. Naime za direktnu reprezentaciju teško je razviti odgovarajuće operatore križanja i mutacije. Zato su rješenja problema predstavljena indirektno, parametrima, koja se kasnije, pomoću procedure opisane u podpoglavlje 6.6, prevedu u raspored poslova.

6.5.1.1. Dekodiranje gena

Iz relacije (6-15) vidi se da je svaki kromosom podijeljen u četiri dijela. Prvi dio čini prvih *m* gena, koji se koriste za određivanje prioriteta svakog projekta i to prema sljedećoj relaciji

$$prioritet_i = gen_i, \quad \forall i = 1,\ldots m \tag{6-16}$$

Položaj gena u kromosomu predstavlja broj projekta, a vrijednost gena predstavlja prioritet kako bi se napravio izbor među raspoloživim projektima.

Drugi dio čine geni koji se nalaze od ***m+1*** do ***m+n*** pozicije u kromosomu. Ovaj dio kromosoma se koriste za određivanje vremena kašnjenja pojedinog posla u trenutku kada se određuje raspored poslova. U svakoj iteraciji ***g*** vrijeme kašnjenja ***DelayGen*** odredi se sljedećim izrazom

$$DelayGen_g = gen_g \times MaxDur \times 1.5, \quad \forall g = 1,\ldots n \tag{6-17}$$

gdje je ***MaxDur*** maksimalno vrijeme izvođenja između svih poslova[21].

Geni između ***m+n+1*** i ***m+2n*** čine treći dio kromosoma i koriste za određivanje vremena kada je svaki projekt raspoloživ, odnosno kada može započeti s prvim poslom. Vremena se određuju sljedećim izrazom

$$ReadyT_i = ERD_i + gen_i \times (DD_i - ERD_i), \quad \forall i = m+n+1,\ldots,2n \tag{6-18}$$

gdje je ERD[22] najranije vrijeme kad je posao raspoloživ.

Geni između ***m+2n+1*** i ***m+2n+x*** se koriste za određivanje vremena kada je svaki resurs raspoloživ i određuje se sljedećim izrazom

$$ReadyR_i = gen_i, \quad \forall i = m+2n+1,\ldots,m+2n+\mathcal{K} \tag{6-19}$$

6.5.1.2. Evolucijski proces

U ovom podpoglavlju opisat će se proces koji se koristi za generiranje nove populacije jedinki iz postojeće populacije. Kroz cijelu proceduru uzima se konstantan broj jedinki u populaciji. Svaka jedinka početne populacije je niz slučajno generiranih brojeva. Na određenu populaciju primjenjuju se genetski operatori kako bi se dobila sljedeća generacija.

Operatori reprodukcije i križanja uvjetuju koji roditelji će dobiti potomke i kako će se između roditelja izmjenjivati genetski materijal da bi se dobila djeca. Operatori reprodukcije i križanja teže poboljšanju kvalitete populacije i utječu na konvergenciju rješenja ka optimumu. Važno je odabrati takve genetske operatore, koji osiguravaju pouzdanu konvergenciju do konačnog, zadovoljavajućeg rješenja, a istovremeno pružaju dovoljno širok raspon različitih provjera rješenja. Mutacija se opire konvergenciji u smislu da dopušta slučajnu izmjenu genetskog materijala.

---

[21] Faktor 1.5 je uzet kao najbolji parametar nakon testiranja s parametrima iz intervala od 1.0 do 2.0.
[22] MERD pokrata od *Maximum Earliest Relase Date.*

Selekcija: U GA smisao operatora selekcije je "preživljavanje najboljih". Vrijednost fitnes funkcije je glavno mjerilo kvalitete rješenja. U ovom poglavlju primijenit će se elitistička selekcija. Elitizam je odabiranje najboljih jedinki iz trenutne populacije i čuvanje od bilo kakve promjene ili eliminacije tijekom evolucijskog procesa. Te jedinke se izravno prenose u sljedeću novu populaciju. U ovom programu primijenit će se dvostruka selekcija, a vrijednost fitnes funkcija se računa kao normalizirana učinkovitost prema sljedećoj relaciji:

```
Efficiency=(1-pressure)*(FObj_Pop_Max-  Total_FObj)/max([FObj_Pop_Max
            - FObj_Pop_Min, eps]) + pressure;
```

Pri čemu su zadani sljedeći parametri:

- `% Pop_in` - početna populacija jedinki
- `% Indiv1` - prvi skup potomaka generiranih pomoću križanja + mutacija
- `% Indiv2` - drugi skup potomaka generiranih pomoću križanja + mutacija
- `% FObj_Pop_in` -vektor vrijednosti funkcije sposobnosti za svaku jedinku iz Pop_in
- `% FObj_Indiv1` - vektor vrijednosti funkcije sposobnosti za svaku jedinku iz Indiv1
- `% FObj_Indiv2` - vektor vrijednosti funkcije sposobnosti za svaku jedinku iz Indiv2
- `% pressure` - koeficijent pritiska selekcije.

Izlazne vrijednosti su:

- `% Pop_out` - sve selektirane jedinke u populaciji veličine pop_size.
- `% Obj_Pop_out`- sve vrijednosti funkcije cilja za svaku jedinku iz skupa Pop_out
- `% Eficiency` - sve vrijednosti fitnesa 'efficiency' za svaku jedinku iz Pop_out.

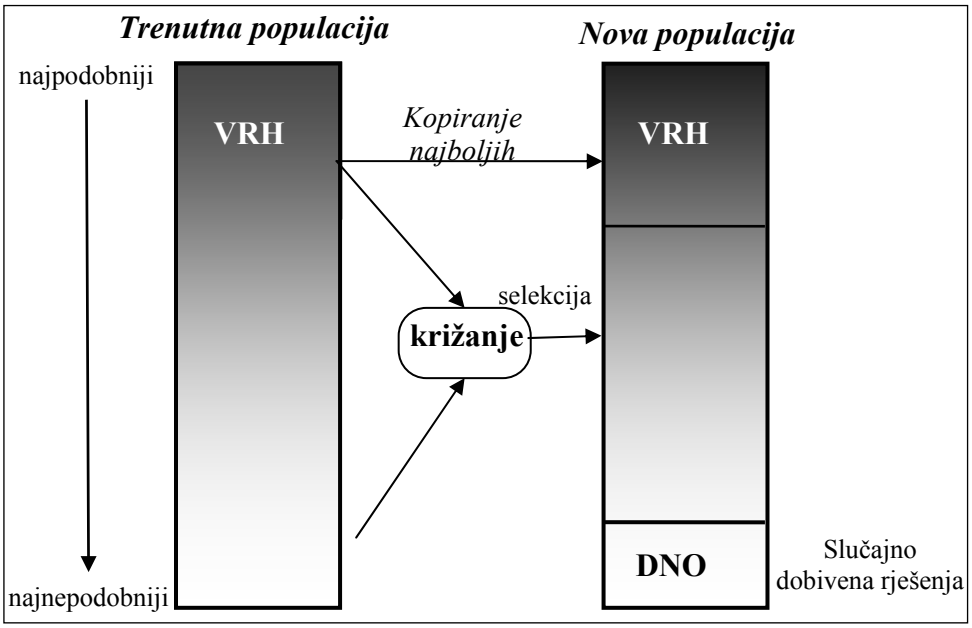
```
   Total_Pop  = [Pop_in; Indiv1; Indiv2];
   Total_FObj = [FObj_Pop_in FObj_Indiv1 FObj_Indiv2]';
    FObj_Pop_Max = max(Total_FObj);
    FObj_Pop_Min = min(Total_FObj);
% Ekstrakcija i Selekcija genotipa
 [Efficiency, Index_sort] = sort(Efficiency,'descend');
  Efficiency = Efficiency(1:XTop);
% Ekstrakcija i Selekcija fenotipa
Total_FObj   = Total_FObj(Index_sort);
FObj_Pop_out = Total_FObj(1:XTop);
        Pop_out=Total_Pop_Novi(1:XTop,:);
```

Slika 6.8 Dio programskog koda za selekciju jedinki

Slika 6.8 prikazuje dio programskog koda datoteke `selection_ga_elitist.m` kojim se vrši sortiranje jedinki prema fitnesu (`Efficiency`) i selektiranje najboljih jedinki. Broj jedinki koji se odabire pri elitističkoj selekciji (varijabla XTop na slici 6.8) je korisnički definiran parametar.

Slika 6.9 prikazuje prijelazni proces između uzastopnih generacija. Primjenom elitističke selekcije, ovisno o vrijednosti fitnesa, jedinke trenutne populacije poredane su od najbolje do najgore. Izdvoje se najbolje jedinke, one s najmanjim fitnesom (najkraće vrijeme izvođenja) čuvaju se u dijelu koji je označen kao VRH populacije na slici 6.9 i izravno se prenose u sljedeću generaciju. U slučaju da dvije ili više jedinke imaju istu vrijednost fitnesa, mogu se odabrati obje ili slučajno neke od njih. Broj jedinki koji se prenosi u VRH nove

populacije jednak je razmjeru *Top* od broja jedinki u populaciji, gdje je *Top* korisnički definirana varijabla.



Slika 6.9 Evolucijski proces među generacijama

Prednost genetskog algoritma s ugrađenim elitizmom, nad tradicionalnom selekcijom je u tome, što GA iz generacije u generaciju, monotono poboljšava rješenje i asimptotski teži ka globalnom optimumu, odnosno rješenju problema. Mogući nedostatak je preuranjena konvergencija, odnosno konvergencija ka lokalnom minimumu. To je moguće izbjeći jakom mutacijom kako bi se postigla različitost u populaciji.

Mutacija: Umjesto izvođenja klasične mutacije gen s genom, s malom vjerojatnosti za svaku generaciju, na dnu nove populacije (dio jedinke označen kao DNO na slici 6.9) uvest će se neke nove jedinke. Broj novih jedinki *Bottom* (linija 132, datoteka 'GA.m') razmjeran je veličini populacije i definira ga korisnik. Te nove jedinke slučajno se generiraju istom distribucijom kao što je generirana početna populacija i na taj se način genetski materijal iz trenutne populacije ne prenosi u sljedeću. Na ovaj način bi se trebala spriječiti preuranjena konvergencija populacije.

Križanje: Preostale jedinke nove populacije se generiraju primjenom parametriziranog uniformog operatora križanja koji je opisan u podpoglavlju 4.2.3.1. Dvije jedinke se slučajno biraju kao roditelji koji će proizvesti potomke. Jedan roditelj se bira s VRHA sadašnje populacije, dok se drugi nasumično odabire iz ostatka trenutne populacije (uključujući i VRH). Za svaki gen, slučajni broj iz intervala [0, 1] se generira. Ako se dobije broj manji od zadane granične vrijednosti[23] (primjerice 0.7), onda se odabire vrijednost gena od prvog roditelja. Inače se odabire od drugog roditelja. Proces križanja se ponavlja onoliko puta koliko je potrebno jedinki da se popuni populacija (varijabla *Nx*, linija 165). Na taj način svaki par

[23] Ova vrijednost naziva se *vjerojatnost križanja* (engl. *crossover probability*, pokrata: CProb).

roditelja proizvede dva potomka. Ponovnom primjenom elitističke selekcije bira se samo *Nx* djece, ona s najboljim fitnesom, kako bi se popunila populacija. Evolucijski proces se nastavlja sve dok se ne zadovolji uvjet za kraj. U ovom programu uvjet je samo na maksimalni broj generacija.

6.6. Procedura generiranja rasporeda

Ova procedura se koristi za generiranje parametriziranog aktivnog rasporeda koji se zasniva na određivanju parcijalnih rasporeda poslova po iteracijama. Za svaku iteraciju određeno je vrijeme t_g - vrijeme kada se pokreće iteracija *g*, odnosno vrijeme kada se raspoređuje određena podgrupa poslova, gdje je *g* redni broj iteracije. Skup *J* obuhvaća sve poslove koji se nisu izvršili do trenutka t_g.

Neka je $A(t_g)$ skup svih poslova koji se još izvode do trenutka t_g, odnosno $A(t_g)=\{j \in J : (F_j - d_j) \le t_g < F_j\}$. Ovaj skup se jednostavno odredi pomoću vektora m_i, jer u iteraciji *i* ako je neki od elemenata $m_i(3) \ldots m_i(12)$ jednak 1 znači da se u trenutku očitavanja stanja mreže $time(i)$ izvode određeni poslovi (*linije* 218 i 248, *Prilog* D).

Kapacitet resursa *k* u trenutku t_g dan je sljedećim izrazom:

$$RD(k,t_g) = RD(k) - \sum_j r_{j,k}, \quad k = 1,\ldots,\mathcal{K},$$

pri čemu *j* označava sve poslove koji se izvode u trenutku t_g (*linija* 198, *Prilog* D), $r_{j,k}$ je broj jedinica resursa r_k koje je zauzeo posao *j* kada se započeo izvoditi, a $RD(k)$ je raspoloživ kapacitet resursa r_k na početku, odnosno u trenutku 0.

Skup S_g uključuje sve poslove koji će se rasporediti u iteraciji *g*, a *FSg* uključuje vrijeme završetka svih poslova iz skupa S_g. Neka je *Delayg* vrijeme kašnjenja pridružen iteraciji *g*, i neka skup E_g uključuje sve poslove koji, po slijedu poslova, mogu započeti u intervalu $[t_g, t_g + Delayg]$, odnosno vrijedi

$$E_g = \{j \in J \setminus S_{g-1} : F_i \le t_g + Delayg(i \in P_j)\}.$$

Vrijednosti varijable *Delayg* se odrede kroz proceduru genetskog algoritma. Broj parametriziranih rasporeda koji se odrede ovim algoritmom dan je sljedećim izrazom:

$$\begin{aligned} &Br\_kromo+[DNO*Br\_kromo+(Br\_kromo-VRH*Br\_kromo-DNO*Br\_kromo)*2]*\ Br\_generacija= \\ &Br\_kromo + [\ Br\_kromo\ *(\ DNO + 2 - 2*VRH - 2*DNO)]*\ Br\_generacija= \\ &Br\_kromo + [\ Br\_kromo\ *(\ 2 - 2*VRH - DNO)]*\ Br\_generacija \end{aligned}$$

pri čemu je *Br_generacija* broj generacija, *Br_kromo* broj kromosoma u populaciji, a DNO i VRH je postotak populacije prethodne populacije koji se odnosi na DNO i VRH sljedeće populacije.

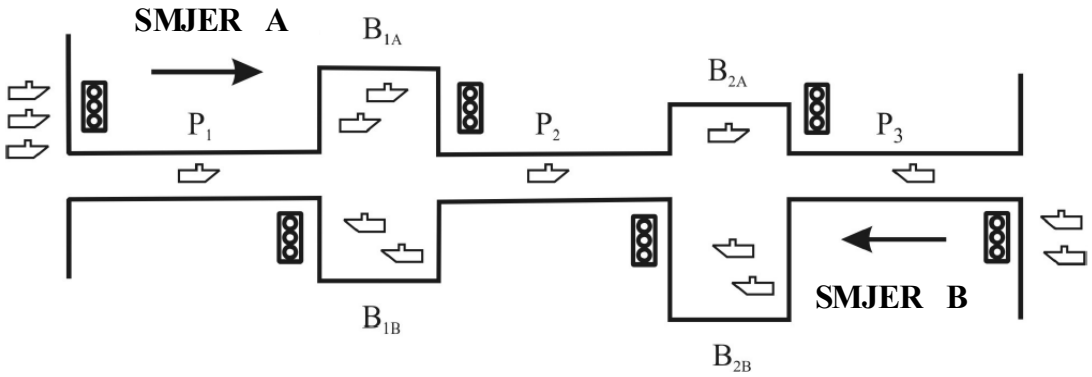
Opis procedure po kojoj se generiraju rasporedi prikazan je slikom 6.10.

```
% Koriste se matrice iz PM modela:
    mf - svako je mjesto PM podijeljeno na dva dijela; mf-zavrsni dio mi-
    inicijalni dio;    r - earliest_start_time za svaki posao
    PNtimes - vektor vremena; mjestima poslova dodijeljena su vremena
    Fr- matrica potrebnih resursa; Sv- matrica pokrenutih poslova;
    k – dekodirani kromosom; DL-dio kromosoma koji se odnosi na kašnjenja
% Korisnički definirani parametri
    TotalTime –ukupno vrijeme simulacije
    TimePeriod – period očitanja stanja mreže
    Num_iter=TotalTime/TimePerid – ukupan broj iteracija za vrem. simulaciju
% Kao rezultat obrade prosljeđuju se sljedeće matrice
    Ft - vektor koji pamti vremena startanja-okidanja poslova u optimalnom
    rasporedu; S0 – optimalan raspored
Algoritam
1.  Inicijalizacija:
     g=1;  t(1)=0; A=[]; S0=[0;0];
     E=[];     % Skup svih poslova koji su planirani za g-tu iteraciju
     FMC=[];  % računanje EFT
     F=min(min(DL)); tg=F; RD(k,1)=RD(k);   (za svaki resurs r_k, k=1,...,K)
2.  Vremenska simulacija: za svaku iteraciju od 1 do Num_iter čini
3.   Vanjska petlja: sve dok vrijedi (J≠∅) čini
       Odrediti skup E - skup poslova koji mogu započeti u iteraciji g
4.   Unutarnja petlja: sve dok vrijedi (E≠∅) čini
     a. Odrediti skup Sg – skup poslova iz E koji zadovoljavaju uvjete na
        slijed poslova i kapacitet resursa. Ako postoji dva ista posla, a
        pripadaju različitim projektima odabrati samo onaj koji pripada
        projektu s najvećim prioritetom.
     b. Za poslove iz skupa Sg odrediti x–vektor okidanja prijelaza u PM.
     c. Ispitati postojanje konflikta ili zastoja u mreži takvim izborom
        poslova. Dati novi vektor okidanja x.
     d. Odrediti matricu stanja mreže za svaku jedinicu vremena dok traje
        iteracija g.
5.    Računanje najranijeg vremena završetka poslova skupa Sg
        FMC(g)=min_{j*∈Sg}(Fx(j*))  &  r_{j*,k} ≤ RD_k(τ);  k=1,...,K, τ∈[t_g, t_g+d_{j*}]
     a.
     b. Obnoviti: S0=S0∪Sg,        Ft=Ft∪FMC(g)
     c. Inkrementiranje broja iteracije: g=g+1;
     d. Obnoviti skup A(t), kapacitete resursa RD(k,t) za
        ∀t∈[Ft(g), FMC(g)].
6.     Kraj unutarnje petlje
7.   t(g)=min(t∈Ft | t>t(g))     %određivanje vremena kada se aktivira
       iteracija g
8.    Kraj vanjske petlje
9.    Kraj ponavljanja
10.   KRAJ procedure
```

Slika 6.10 Pseudo kod procedure kojom se generira raspored poslova

6.7. Optimiranje prometa brodova

Matrični model MRF_1 Petrijeve mreže integriran sa GA, opisan u podpoglavlju 6.5, primijeniti će se na sustav prometa brodova u morskim kanalima, pokrata: SPMK (slika 6.11).



Slika 6.11 Sustav prometa morskim kanalima

Motivacija:

- Razviti odgovarajući MRF_1 model Petrijeve mreže i primjenom opisanog matričnog modela (podpoglavlje 6.4) razviti nadzornik za sprječavanje konflikata i zastoja
- Napraviti program koji generira raspored prolaza brodova sustavom morskih kanala
- Uzeti u razmatranje raspoloživost resursa u vidu kapaciteta kanala i bazena
- Odrediti kada će započeti obavljati svaki posao u sustavu
- Učinkovita uporaba kapaciteta resursa, tako da ukupno vrijeme prolaza svih brodova kroz kanale bude što kraće.

SPMK je sustav s diskretnim događajima, pri čemu su događaji stanja resursa i stanja obavljanja poslova. Kao što je razvidno iz slike 6.11, sustav se sastoji od tri kanala P_1 , P_2 , P_3 i četiri bazena B_{1A} , B_{2A} , B_{1B} i B_{2B} . Plovila na lijevom kraju kanala čekaju za prolaz kanalima prema desnoj strani sustava (na slici je taj smjer označen kao smjer A), a plovila na desnom kraju sustava čekaju za prolaz prema lijevoj strani (smjer B). Plovila se mogu kretati kanalima jedino pomoću vlastite propulzije, ili pomoću brodova tegljača koji su smješteni uz rubove kanala. Staza za plovila u smjeru A je $P_1 \rightarrow B_{1A} \rightarrow P_2 \rightarrow B_{2A} \rightarrow P_3$, dok je za smjer B staza $P_3 \rightarrow B_{2B} \rightarrow P_2 \rightarrow B_{1B} \rightarrow P_1$. Plovila iz oba smjera dijele kanale. Bazeni su izgrađeni za samo jedan smjer kretanja plovila. U bazenima brodovi čekaju dok se ne oslobodi kanal. Ako je određeni resurs zauzet, onda brod mora čekati određeno vrijeme na kraju kanala.

Prolaz pojedinog broda kanalom gledat će se kao pojedini projekt koji se sastoji od jednakog broja poslova koji su navedeni u tablici 6.2 i poslovi se moraju obaviti određenim redoslijedom. Kako brodovi koji prolaze kanalima i u smjeru A i u smjeru B moraju obaviti iste poslove, onda tablica 6.2 sadrži poslove samo za jedan smjer. Svaki posao se izvršava uz uporabu određenog resursa i traje određeni broj vremenskih jedinica[24]. Kanali i bazeni predstavljaju resurse sustava. Kod dinamičkog modela, brodovi su različitog prioriteta i

[24] Kao mjerna jedinica vremena ovog modela koristit će se 1 sat.

postoje kašnjenja u izvršavanju pojedinih poslova. Također, nakon obavljenog posla resursi zahtijevaju određeno vrijeme setiranja.

Tablica 6.2 Opis slijeda poslova u sustavu sa slike 6.11 koji pripadaju *i*-tom projektu

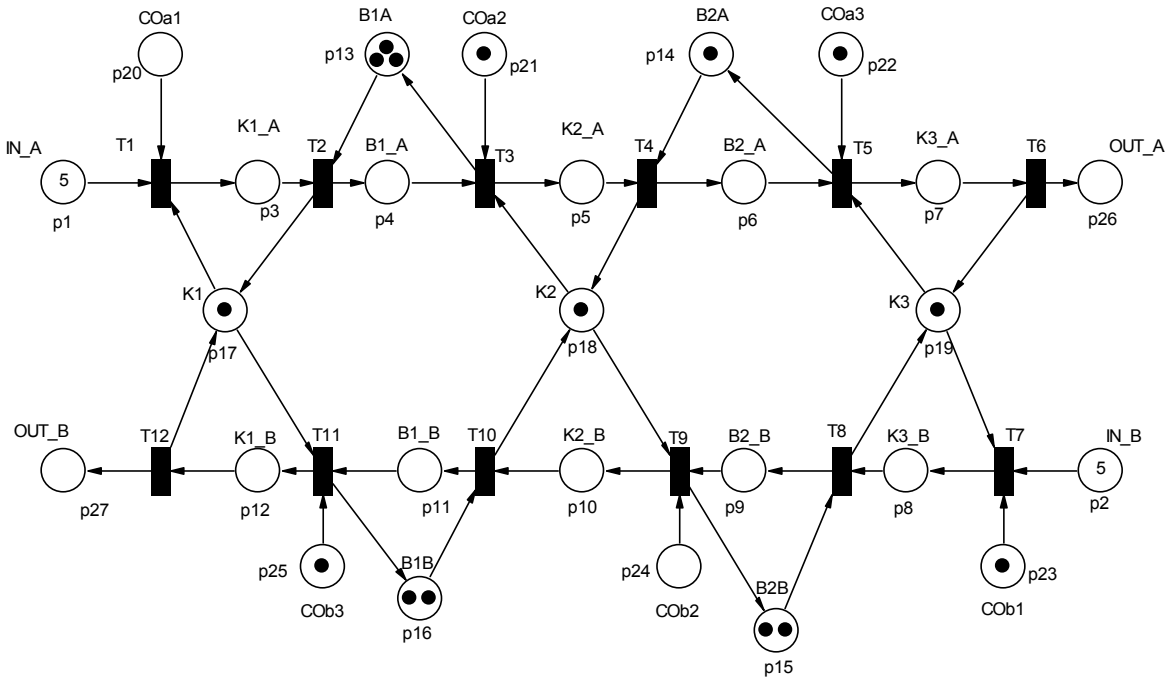
PROJEKT *i*				
Posao 1	**Posao 2**	**Posao 3**	**Posao 4**	**Posao 5**
Prolaz prvim kanalom	*Čekanje u bazenu*	*Prolaz drugim kanalom*	*Čekanje u bazenu*	*Prolaz trećim kanalom*
trajanje t_1	trajanje t_2	trajanje t_3	trajanje t_4	trajanje t_5

6.7.1. Modeliranje sustava s MRF1 klasom Petrijeve mreže

Ovaj sustav pripada MRF klasi zbog sljedećeg:

- Sustav posjeduje svoj ulaz i svoj izlaz
- Ne postoje samostalne petlje, svaki pojedini posao koristi samo jedan resurs te nema poslova sastavljanja
- U sustavu postoje i višeradni resursi (kanali) koji mogu izvršavati više od jednog posla (prolaz kanalom u smjeru A i prolaz kanalom u smjeru B).

Slika 6.12 prikazuje pripadajuća PM koja opisuje ovaj sustav. Mjesta predstavljaju resurse i poslove, a okidanjem prijelaza posao se počinje izvoditi. Mjesta $p_3 - p_{19}$ predstavljaju poslove i resurse sustava. Kontrolna mjesta $COa_1 - COa_3$, $COb_1 - COb_3$ pripadaju kontroleru i osiguravaju da se poslovi u mreži izvode bez konflikta i bez zastoja.



Slika 6.12 MRF_1 PM model pomorskog prometa kanalima

Mjesta p_1, p_2 predstavljaju skup ulaznih mjesta $\{\text{IN\_A, IN\_B}\}$, a broj 5 u mjestu p_1 (p_2) označava da se na ulazu u kanal u smjeru A (smjeru B) nalazi 5 brodova koji čekaju na prolaz kanalima. Skup izlaznih mjesta je $\{\text{OUT\_A, OUT\_B}\}$. Skup poslova u sustavu je

J={K1_A,B1A_A, K2_A, B2A_A, K3_A, K3_B, B2B_B, K2_B, B1B_B, K1_B},

a skup resursa je R={ B1A, B2A, B2B, B1B, K1, K2, K3}.

Kretanja brodova kanalima ograničeno je kapacitetima resursa. U ovom sustavu kapacitet[25] *i*-tog resursa RD(*i*) određen je brojem brodova koji mogu čekati u bazenu, odnosno koji mogu proći kanalom. Na početku simulacije kapaciteti resursa su zadani kako slijedi

RD(1)=3 RD(2)=1 RD(3)=2 RD(4)=2 RD(5)=1 RD(6)=1 RD(7)=1

kapaciteti bazena B1A, B2A, B2B, B1B *kapaciteti kanala* K1, K2, K3

U programskom jeziku Matlab razvijen je program koji simulira opisani sustav sa diskretnim događajima (vidjeti prilog A).

```
% 1. Definiranje matrica iz Petrijeve mreze kanala
% Fv- matrica slijeda poslova (koji posao okida koji prijelaz)
% Fr- matrica potrebnih resursa (koji resurs treba za okidanje prijelaza)
% Sv- matrica pokrenutih poslova (koji prijelazi kada se okidaju aktiviraju  koje poslove
% Sr- matrica raspoloživih resursa (koji prijelaz opušta koji resurs)
Fv=[0 0 0 0 0 0 0 0 0 0;1 0 0 0 0 0 0 0 0 0;0 1 0 0 0 0 0 0 0 0;0 0 1 0 0 0 0 0 0 0;0 0 0 1 0 0 0 0 0 0;0 0 0 0
1 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 1 0 0 0 0;0 0 0 0 0 0 1 0 0 0;0 0 0 0 0 0 0 1 0 0;0 0 0 0 0 0 0 0 1 0;0
0 0 0 0 0 0 0 0 1];
Fr=[0 0 0 0 1 0 0;1 0 0 0 0 0 0;0 0 0 0 0 1 0;0 1 0 0 0 0 0;0 0 0 0 0 0 1;0 0 0 0 0 0 0;0 0 0 0 0 0 1;0 0 1 0 0
0 0;0 0 0 0 0 1 0;0 0 0 1 0 0 0;0 0 0 0 1 0 0;0 0 0 0 0 0 0];
Fud=[1 0 0 0 0 0;0 0 0 0 0 0;0 1 0 0 0 0;0 0 0 0 0 0;0 0 1 0 0 0;0 0 0 0 0 0;0 0 0 1 0 0;0 0 0 0 0 0;0 0 0 0 1
0;0 0 0 0 0 0;0 0 0 0 0 1;0 0 0 0 0 0];
Sv=[1 0 0 0 0 0 0 0 0 0 0 0;0 1 0 0 0 0 0 0 0 0 0 0;0 0 1 0 0 0 0 0 0 0 0 0;0 0 0 1 0 0 0 0 0 0 0 0;0 0 0 0 1 0
0 0 0 0 0 0;0 0 0 0 0 0 1 0 0 0 0 0;0 0 0 0 0 0 0 1 0 0 0 0;0 0 0 0 0 0 0 0 1 0 0 0;0 0 0 0 0 0 0 0 0 1 0 0;0 0 0
0 0 0 0 0 0 0 1 0];
Sr=[0 0 1 0 0 0 0 0 0 0 0 0;0 0 0 0 1 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 1 0 0 0;0 0 0 0 0 0 0 0 0 1 0 0;0 1 0 0 0 0
0 0 0 0 0 1;0 0 0 1 0 0 0 0 0 1 0 0;0 0 0 0 0 1 0 1 0 0 0 0];
Sud=[0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0
0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0];
Fu=[1 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 1 0 0 0 0 0]';
Fy=[0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0]';
Su=[0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0];
Sy=[0 0 0 0 0 1 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 1];
F=[Fu Fv Fr Fud Fy];              % matrica zavrsenih poslova
S=[Su' Sv' Sr' Sud' Sy']';        % matrica poslova koji startaju
```

Slika 6.13 Dio Matlab programskog koda u kojem se definiraju matrice PM

[25] Kapaciteti su prikazani brojem oznak u mjestime p13 – p19 u MRF_1 mreži na slici 6.12.

U skladu s opisanim matričnim modelom MRF_1 PM, za ovaj primjer potrebne su matrice prikazane na slici 6.13. Važno je naglasiti da se razvijeni algoritam može primijeniti na bilo koji DES sustav, samo treba promijeniti vrijednosti u matricama koje opisuju taj sustava (slika 6.13).

6.7.1.1. Vremenska simulacija

Svaki resurs zahtijeva određeno vrijeme (različito od nule) potrebno za izvođenje poslova. Vremena obavljanja poslova koja su dodijeljena mjestima p3–p12 PM mreže (slika 6.12) zadana su vektorom *PNtimes* :

$$\text{PNtimes} = [0\ \ 0\ \underbrace{0.54\ \ 1\ \ 1.08\ \ 1\ \ 0.81}_{\text{poslovi u smjeru A}}\ \underbrace{0.81\ \ 1\ \ 1.08\ \ 1\ \ 0.54}_{\text{poslovi u smjeru B}}\ 0\ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]'$$

Vremena u mjestima 3,5,7 ovise o brzini kretanja brodova kanalima u smjeru A, a vremena u mjestima 8,10,12 u smjeru B. Pretpostavka je da se brodovi moraju kretati brzinom od v = 1čvor = 1milja/h= 1.852km/h te da su duljine kanala redom l = 1km, 2km, 1.5km.

Vrijeme potrebno za prolaz kanalom uz odgovarajuću brzinu izračuna se po formuli $v = \frac{l}{t}$ i vrijedi

$$PNtimes(3) = PNtimes(12) = l / v = 0.54 \quad PNtimes(5) = PNtimes(10) = 1.08 \quad PNtimes(7) = PNtimes(8) = 0.81.$$

Obzirom da se mjestima trebaju pridružiti vremena, planirano je da svako mjesto p3–p12 u mreži sadrži dva dijela. Jedan dio se naziva *ulazni* dio, oznaka m_i, gdje se oznake pozicioniraju kada se okine ulazni prijelaz u to mjesto. Nakon što prođe vrijeme pridruženo mjestu, oznaka se pomiče iz ulaznog dijela m_i u *izlazni* dio m_f. Sada je moguće okinuti prijelaz koji izlazi iz tog mjesta.

Umjesto jednadžbe (3-22) koja opisuje stanje PM nakon okidanja prijelaza k, sada vrijede sljedeće jednadžbe:

$$\begin{aligned} m_f(k+1) &= m_f(k) - F^T \cdot x(k) \\ m_i(k+1) &= m_i(k) + S \cdot x(k) \end{aligned} \qquad (6\text{-}20)$$

Jednadžbe (6-20) su implementirane u linijama 105 i 106 (Prilog A). S ovakvom implementacijom vremenske PM, moguće je odrediti postotak iskoristivosti resursa, proteklo vrijeme, skup A- poslovi koji se još izvode u određenoj iteraciji određivanja rasporeda itd.

6.7.1.2. Detekcija postojanja konflikata i zastoja u sustavu

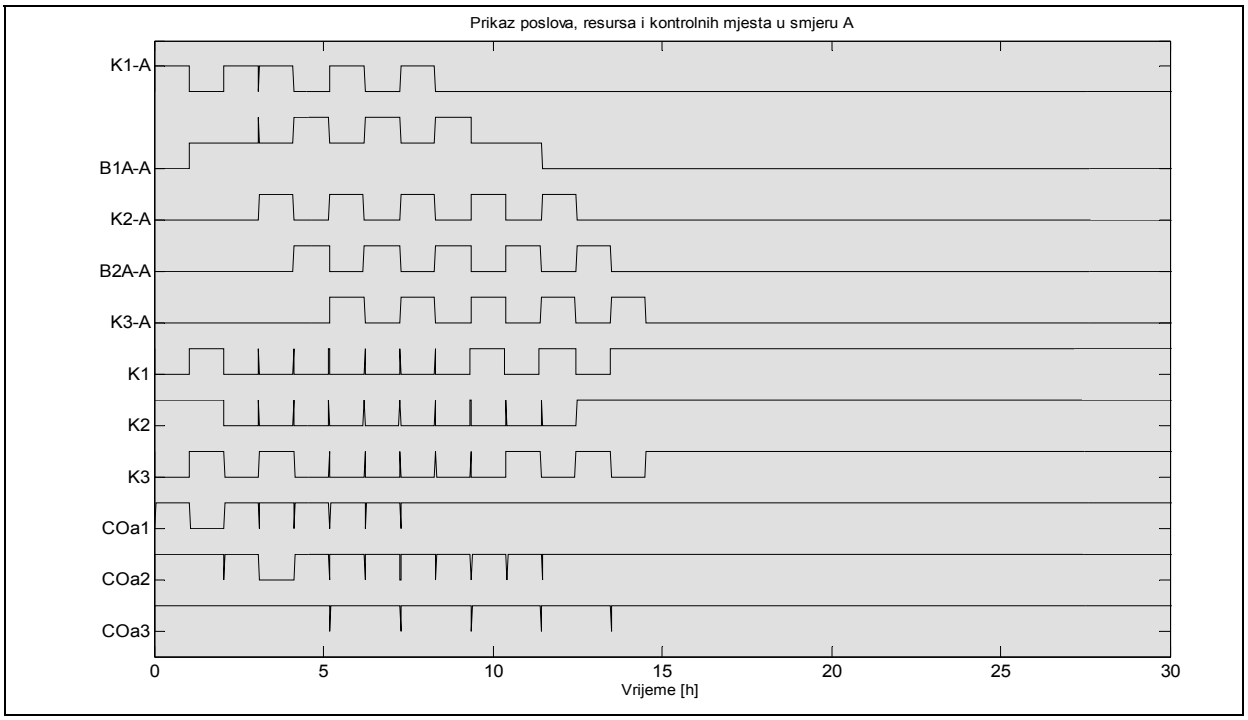
Konflikti nastaju kada brodovi iz oba smjera trebaju iste resurse $\{p_{17} - p_{19}\}$. U programskom kodu iz *Priloga A*, *x* je vektor omogućenih prijelaza (*linija* 44) i izračuna se vektor *conflict (linija* 46), koji se dobije tako da se u onim mjestima vektora *mf* koji prethode omogućenim prijelazima x oduzme broj oznaka za 1. Ako vektor *conflict* ne sadrži niti jednu negativnu vrijednost u stanju m_{k+1}, onda postoji problem dodjele zajedničkog resursa (*linije* 50,58, 66). U tom slučaju, mora se pozvati procedura (*linije* 50-72) koja će

odrediti koji posao će dobiti traženi resurs i određuje se novi vektor x_k. Odabir posla je određen proširenjem vektora stanja tako da se uključe kontrolna mjesta. Procedura dodjeljivanja zajedničkog resursa može sadržavati određeno pravilo dodjeljivanja. Iz linija 52, 60, 68 je razvidno da se u ovom programu rabi procedura dodjeljivanja resursa po slučajnom izboru. Funkcija *rand*(1) generira slučajan broj 0-1, koji se pomnoži sa 2 tako da se dobije slučajan broj od 0.000 do 1.9999 pa se taj broj zaokruži na cijeli.

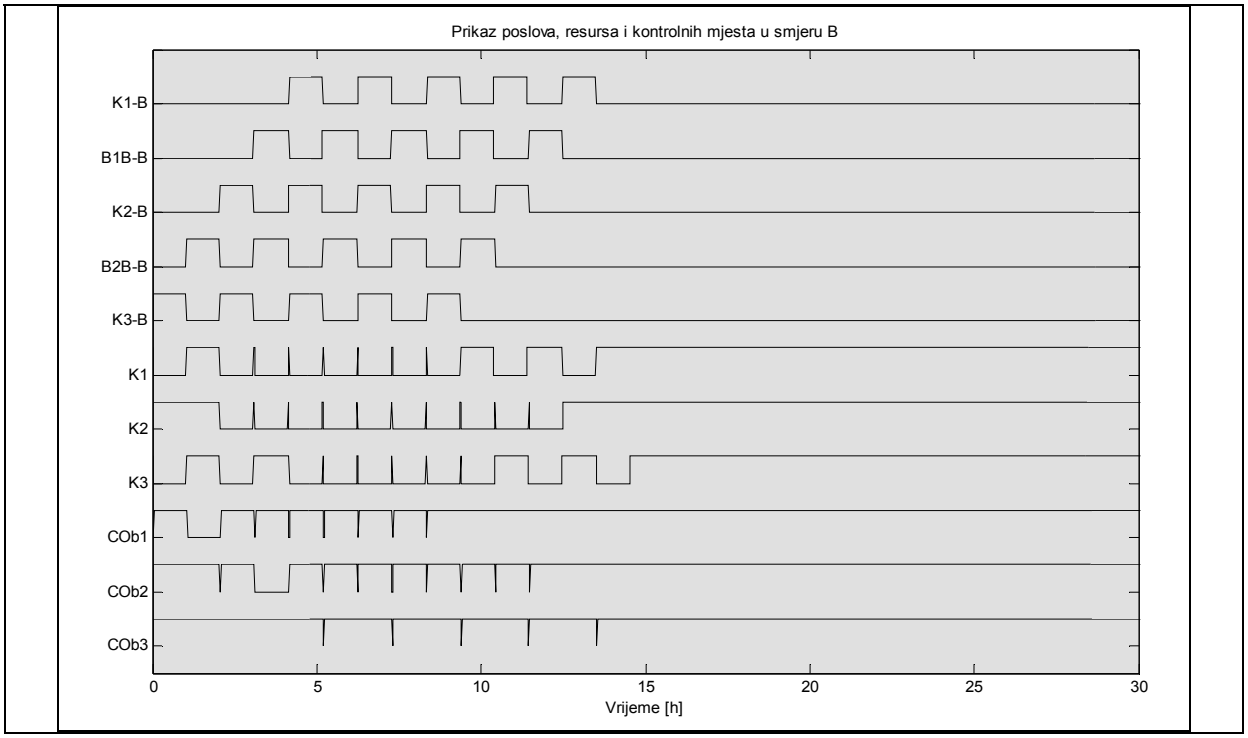
U ovom primjeru prijelazi t_1, t_{11} i t_3, t_9 i t_5, t_7 mogu biti u konfliktu (moguće je oba prijelaza, iz svakog para, okinuti u istom trenutku). Primijenjena procedura kontrole konflikta omogućava prolaz samo jednim smjerom, A ili B, s jednakom vjerojatnošću. Drugi je problem kako omogućiti da se poslovi u sustavu izvršavaju bez zastoja. Procedura izbjegavanja zastoja dana je u datoteci *'deadlock_avoidance.m' (Prilog F).* Kontrolna mjesta ograničavaju broj plovila u kritičnim podsustavima kako bi se izbjegao zastoj prve razine. Ali još uvijek može postojati zastoj druge razine ako je sustav iregularan i ako sadrži ključne resurse [BOG06]. Kako bi se izbjeglo zaglavljenje druge razine, moraju se uzeti u obzir da ključni resurs ne može biti, ujedno, i posljednji raspoloživi resurs u mreži.

6.7.1.3. Rezultati simulacije

Rezultati opisane tehnike i datoteke *'kanal.m'* iz priloga A prikazani su na slikama 6.14 i 6.15. Rezultati prikazuju evoluciju promatranog sustava sa diskretnim događajima kroz vrijeme. Osam gornjih dijagrama na slici 6.14 prikazuju broj plovila u mjestima koja prikazuju poslove sustava $p_3 - p_{12}$, kao i broj brodova koji prolaze kanalima. Primjerice u prvom vremenskom intervalu u prvom dijagramu je vrijednost 1, što znači da se izvodi posao K1_A. Istovremeno je vrijednost u šestom dijagramu za K1 jednaka 0, što znači da je u prvom vremenskom intervalu zauzet kanal K1. Donja tri dijagrama (COa1-COa3 dijagram (a), COb1-COb3 dijagram (b)) prikazuju kontrolna mjesta $p_{20} - p_{25}$. Kontrolna mjesta su "semafori" koji sprječavaju okidanje prijelaza (logička vrijednost "1" znači zeleno svjetlo, dok logička vrijednost "0" znači crveno). Slika 6.15 prikazuje stanja u ulaznim i izlaznim mjestima. Može se uočiti da postoji proces kretanja plovila kroz sustav i da zadnji brod kanalski sustav napušta nakon 15 sata (točnije nakon 15.5800 vrem. jed.). Primijenjena procedura kontrole osigurava kretanje brodova bez konflikta i zastoja u sustavu.

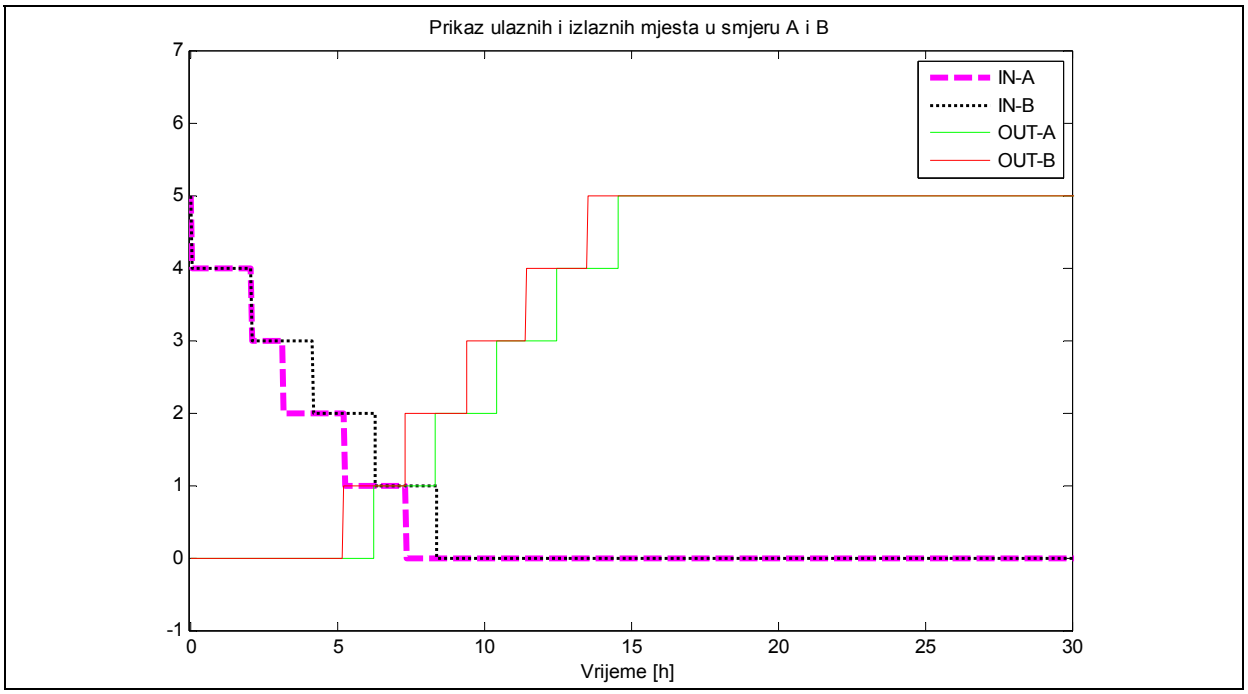


(a)



(b)

Slika 6.14 Broj plovila u mjestima koji predstavljaju poslove i broj oznaka u kontrolnim mjestima; (a) Smjer A, (b) Smjer B



Slika 6.15 Broj plovila u ulaznim/izlaznim mjestima

6.7.1.4. Rezultati optimalizacije

Kako bi se provjerila učinkovitost PMGA algoritma, testiran je na SPMK sustavu s ciljem da se pronađe projektni raspored takav da je ukupno vrijeme prolaza plovila kanalima najkraće, a da su maksimalno iskorišteni resursi, uz minimalno vrijeme čekanja na resurse. Pri tome se moraju zadovoljiti ograničenja slijeda poslova i dostupnosti resursa (izrazi (6-2) - (6-7)). Treba za svaki posao *i* odrediti vrijeme početka $s_i, (i = 1,\ldots, N)$.

Funkcija cilja minimalizira ukupno vrijeme potrebno da se izvrše svi poslovi u sustavu (makespam), odnosno

$$min \quad c_{max} = min(F_1, F_2, \ldots, F_N). \tag{6-21}$$

U ovom podpoglavlju će se prvo razmatrati statički model promatranog sustava. U statičkom modelu svi poslovi su bez kašnjenja, resursi su odmah raspoloživi i raspored je određen vremenom F_j - vrijeme kada je završen posao j.

U statičkom slučaju sva kašnjenja u sustavu su jednaka nuli što se mora uzeti u obzir pri dekodiranju kromosoma. Na slici 6.16 prikazan je primjer jednog kromosoma u GA algoritmu.

Kako je opisano u točki 6.5.1.1, svaki kromosom se mora dekodirati i primjer dekodiranog kromosoma prikazan je slikom 6.17. Prvih deset gena se rabe kao prioriteti projekata, geni od 11 do 60 se rabe za određivanje vremena kašnjenja poslova, pomoću gena od 61 do 70 odrede se vremena kada su projekti raspoloživi, a genima od 71 do 77 kada su resursi raspoloživi.

```
Geni od 1 do 10
0.6726   0.1338   0.8431   0.9407   0.5439   0.0332   0.3589   0.4961   0.1505   0.1206
Geni od 11 do 60
0.9638   0.0109   0.0927   0.0333   0.4664   0.8747   0.6170   0.9171   0.2391   0.1648   0.6385
0.3419   0.7216   0.3878   0.7951   0.7964   0.5631   0.5211   0.0354   0.0934   0.4672   0.1123
0.8579   0.6342   0.9928   0.8738   0.1946   0.1218   0.1563   0.2062   0.8023   0.9262   0.2724
0.1112   0.2720   0.0250   0.1879   0.7173   0.3688   0.5900   0.5096   0.9906   0.5408   0.3477
0.0333   0.2212   0.9451   0.8126   0.4145   0.0113
Geni od 61 do 70
0.6547   0.3268   0.7991   0.1910   0.3714   0.6213   0.2014   0.1556   0.0470   0.7653
Geni od 71 do 77
  0   0   0   0   0   0   0
```

Slika 6.16 Primjer kromosoma

```
Geni od 1 do 10
 0.6726   0.1338   0.8431   0.9407   0.5439   0.0332   0.3589   0.4961   0.1505   0.1206
Geni od 11 do 60
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Geni od 61 do 70
 11.7843   5.5550   9.5895   2.8655   5.9416   9.9415   2.4170   1.8668   0.8461   10.7142
Geni od 71 do 77
 0   0   0   0   0   0   0
```

Slika 6.17 Dekodirani kromosom sa slike 6.16

Optimalizacija je provedena na računalu sa 3.20GHz Pentium procesorom, 2GB RAM-a i MS Windows XP operacijskim sustavom, a programi su razvijeni u programskom sustavu Matlab verzija R2008a. Tablica 6.3 prikazuje kreirane datoteke[26] koje se neophodne za testiranje programa. Dovoljno je pokrenuti datoteku *'kanal_start.m'* koja sadrži matrični opis PM i poziva datoteku *'GA.m'* unutar koje se poziva generator rasporeda i odvija se evolucijski proces. Prethodno opisan algoritam integracije PMGA testiran je sa parametrima koji su prikazani u tablici 6.4. Rezultati optimalizacije prikazani su na slici 6.18.

Tablica 6.3. Nazivi datoteka koje su uključene u program

% PRIMJER_ KANALI	
% crossover_ga	- Uniformno križanje uzima dva roditelja Par1,Par2
% deadlock_avoidance	- Procedura izbjegavanja zastoja u sustavu
% dekod_krom	- Dekodiranje kromosoma
% earliest_finish_time	- Određivanje najranijeg mogućeg završetka izvođenja posla
% earliest_start_time	- Određivanje najranijeg mogućeg početka izvođenja posla
% GA.M	- Genetski algoritam
% ga_populacija	- Funkcija kreira početnu populaciju
% generator_rasporeda	- Generiranje rasporeda poslova
% graf_tokena	- Grafički prikaz rezultata
% kanal	- Simulacija kanala MRF1 Petrijevom mrezom

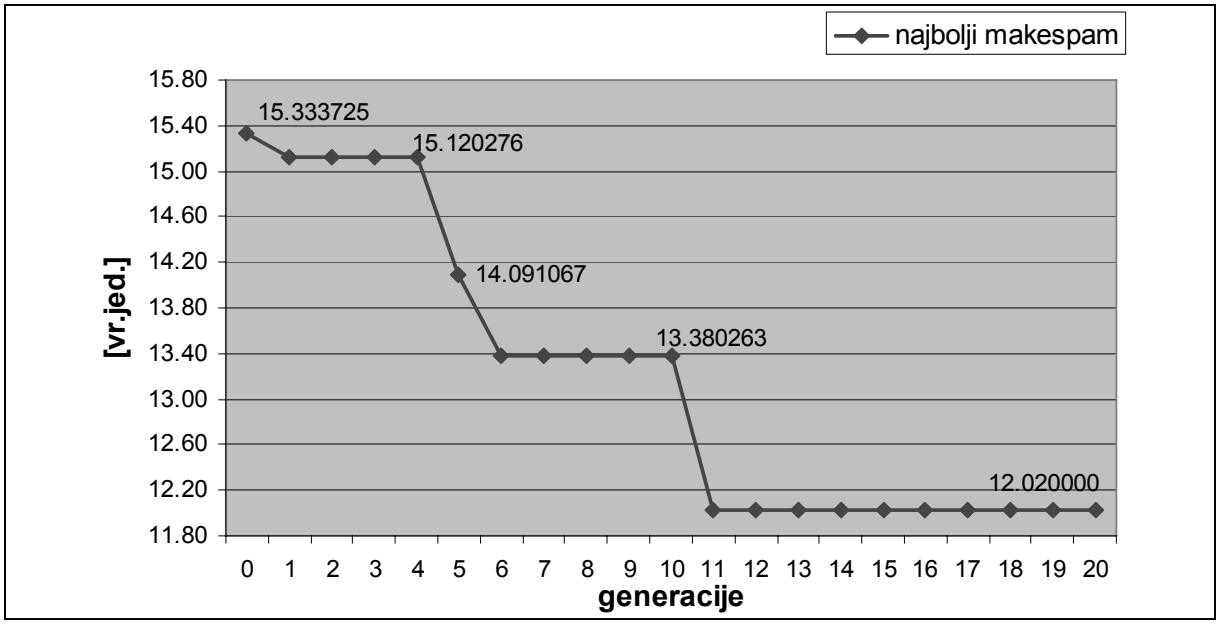
[26] Programski kodovi nekih od navedenih datoteka su prikazani na kraju u prilogu.

Tablica 6.3 nastavak

% kanal_start	- Početni file PMGA algoritma
% konflikt2	- Funkcija za rješavanje konfliktnih situacija
% kromosomi	- Funkcija kojom se generiraju kromosomi
% multCO	-
% multoa	-
% not	- Funkcija koja članove <1 pretvara u jedinicu, a ostale u nulu
% precedence_feasible	- Funkcija pronalazi skup poslova koje je moguće započeti u određenoj iteraciji
% resursi	- n - broj resursa
% selection_ga_elitist	- Elitistička selekcija
% vektor_okidanja	- Odreduje se mogući niz okidanja u PM
% vremenska_simulacija	-

Tablica 6.4 Parametri algoritma

Parametar	Vrijednost
Broj poslova	50
Veličina populacije	0.2*Broj_poslova
Uvjeti zaustavljanja	Nakon 10 (20) generacija
Vjerojatnost mutacije jedinke	{0.5,0.6, 0.70, 0.8}
Način selekcije	elitistička
Pritisak selekcije	0.05
Broj jedinki u VRH-u populacije	{10%,15%,20%} od veličine populacije
Broj jedinki u DNU-u populacije	{20%,30%} od veličine populacije
Funkcija cilja	Najkraće vrijeme izvođenja svih poslova (najkraći makespam)



Slika 6.18 Optimalizacijski proces za ukupno vrijeme izvršavanja svih poslova u sustavu

Iz grafa na slici 6.18 je razvidno da u početnoj populaciji najkraće ukupno vrijeme izvršavanja svih poslova (makespam) u sustavu iznosi 15.333725 vr. jed. i taj najbolji raspored poslova, generiran prema proceduri koja je opisana u podpoglavlju 6.6, prikazan je na slici 6.19.

			Brod	Posao	Vrijeme poc...
1	iteracija:	1	4	K1_A	0.4237
2	iteracija:	2	4	B1A_A	0.9637
3	iteracija:	3	5	K1_A	1.9637
4	iteracija:	3	4	K2_A	1.9637
5	iteracija:	4	5	B1A_A	2.5037
6	iteracija:	5	5	K2_A	3.5037
7	iteracija:	5	4	B2A_A	3.5037
8	iteracija:	6	3	K3_B	4.5037
9	iteracija:	7	5	B2A_A	5.3137
10	iteracija:	7	3	B2B_B	5.3137
11	iteracija:	8	4	K3_A	6.3137
12	iteracija:	8	3	K2_B	6.3137
13	iteracija:	9	2	K1_A	7.1237
14	iteracija:	10	2	B1A_A	7.1237
15	iteracija:	10	5	K3_A	7.1237
16	iteracija:	10	3	B1B_B	7.1237
17	iteracija:	11	5	K3_B	7.9337
18	iteracija:	12	3	K1_A	8.7437
19	iteracija:	12	2	K2_A	8.7437
20	iteracija:	12	5	B2B_B	8.7437
21	iteracija:	13	3	K1_B	8.7437
22	iteracija:	14	3	B1A_A	9.2837
23	iteracija:	14	1	K3_B	9.2837
24	iteracija:	14	5	K2_B	9.2837
25	iteracija:	15	1	K1_A	9.2837

			Brod	Posao	Vrijeme poc...
26	iteracija:	15	3	K2_A	9.2837
27	iteracija:	15	3	B2A_A	9.2837
28	iteracija:	15	1	B2B_B	9.2837
29	iteracija:	16	1	B1A_A	9.8237
30	iteracija:	16	3	K3_A	9.8237
31	iteracija:	16	1	K2_B	9.8237
32	iteracija:	16	5	B1B_B	9.8237
33	iteracija:	17	1	K2_A	10.6337
34	iteracija:	17	2	B2A_A	10.6337
35	iteracija:	17	5	K1_B	10.6337
36	iteracija:	18	2	K3_A	10.6337
37	iteracija:	19	2	K3_B	10.6337
38	iteracija:	20	1	B2A_A	11.4437
39	iteracija:	20	2	B2B_B	11.4437
40	iteracija:	21	1	K3_A	11.4437
41	iteracija:	21	2	K2_B	11.4437
42	iteracija:	22	2	B1B_B	12.2537
43	iteracija:	23	2	K1_B	13.2537
44	iteracija:	24	4	K3_B	13.2537
45	iteracija:	24	1	B1B_B	13.2537
46	iteracija:	25	4	B2B_B	13.2537
47	iteracija:	25	1	K1_B	13.2537
48	iteracija:	26	4	K2_B	13.7937
49	iteracija:	27	4	B1B_B	13.7937
50	iteracija:	28	4	K1_B	14.7937
51			Min Make...		15.3337

Slika 6.19 Prikaz parcijalnih rasporeda najbolje jedinke iz početne populacije

Tablica sa slike 6.19 prikazuje parcijalne rasporede poslova po iteracijama za najbolju jedinku početne (inicijalne) populacije. Prva dva stupca tablice prikazuju redne brojeve iteracija, treći stupac za koji brod (projekt) se pokreće posao, u četvrtom stupcu je naziv posla prema PM sa slike 6.12, a u zadnjem stupcu su prikazana vremena kada ti poslovi započinju (kada se okidaju prijelazi koji prethode mjestima naznačenog posla). Tako, u iteraciji $g = 1$, u trenutku 0.4237 četvrti brod u smjeru A započinje prolaz kanalom K1 (posao K1_A). U drugoj iteraciji $g = 2$, u trenutku 0.9637 isti brod ulazi u bazen B1 (posao B1A_A) i tako se redom nastavljaju poslovi kako je prikazano u tablici. Kako je iz tablice razvidno posljednji posao započinje u 28. iteraciji, u trenutku 14.7937. Tada četvrti brod iz smjera B započinje posljednji posao (K1_B), odnosno ulazi u kanal K1. Kako je K1_B posljedni posao koji se izvodi i mjestu p_{12} (slika 6.12) je pridruženo vrijeme od 0.54 vrem. jed, to znači da će brod isploviti iz kanala u trenutku 14.7937 + 0.54 = 15.3337, što predstavlja i konačno vrijeme izvođenja svih poslova u sustav (ukupni makespam).

Prema koncepciji GA, početna populacija ide u evolucijski proces. Evolucija se odvija kroz 20 generacija i na kraju se dobije najbolji raspored poslova koji je prikazan u tablici 6.5. Razvidno je da taj najbolji raspored poslova završava nakon 12.020000 vrem. jed. U usporedbi sa neoptimaliziranim rezultatom od 15.5800 vr. jed. (slika 6.15), projekt je ubrzan za 3.3 vr. jed. ili za 21%, što predstavlja znatno poboljšanje u optimalizaciji vremena.

Tablica 6.5 Parcijalni rasporedi po iteracijama za najbolje rješenje

	brod	**posao**	**vrijeme početka**
iteracija: 1			
	5	K1_A	0.000000
	2	K3_B	0.000000
iteracija: 2			
	5	B1A_A	0.540000
iteracija: 3			
	2	K1_A	1.540000
	5	K3_B	1.540000
	2	B2B_B	1.540000
iteracija: 4			
	2	B1A_A	2.080000
	5	K2_A	2.080000
	5	B2B_B	2.080000
iteracija: 5			
	3	K1_A	3.080000
iteracija: 6			
	3	B1A_A	3.620000
	2	K2_A	3.620000
	5	B2A_A	3.620000
iteracija: 7			
	1	K1_A	4.620000
	5	K3_A	4.620000
iteracija: 8			
	4	B1A_A	5.160000
iteracija: 9			
	2	B2A_A	5.160000
	5	K2_B	5.160000
iteracija: 10			
	4	K1_A	6.160000
iteracija: 11			
	1	B1A_A	6.700000
	3	K2_A	6.700000
	2	K3_A	6.700000
	5	B1B_B	6.700000
iteracija: 12			
iteracija: 13			
	1	K2_A	7.510000
	3	B2A_A	7.510000
	5	K1_B	7.510000
iteracija: 14			
	4	K2_A	8.050000
	3	K3_A	8.050000
iteracija: 15			
	1	B2A_A	8.050000
	3	K3_B	8.050000
	2	K2_B	8.050000
iteracija: 16			
	4	B2A_A	8.050000
	1	K3_A	8.050000
	3	B2B_B	8.050000
	2	B1B_B	8.050000
iteracija: 17			
	1	K3_B	8.860000

Tablica 6.5 Nastavak

iteracija: 18			
	4	K3_A	8.860000
	3	K2_B	8.860000
	2	K1_B	8.860000
iteracija: 19			
	4	K3_B	9.400000
	1	B2B_B	9.400000
iteracija: 20			
	4	B2B_B	9.400000
	1	K2_B	9.400000
	3	B1B_B	9.400000
iteracija: 21			
	4	K2_B	10.400000
	3	K1_B	10.400000
iteracija: 22			
	4	B1B_B	10.940000
iteracija: 23			
	1	B1B_B	10.940000
	4	K1_B	10.940000
iteracija: 24			
	1	K1_B	11.480000

Najbolji_makespam=12.020000 vr. jed.

6.7.1.5. Dinamički model

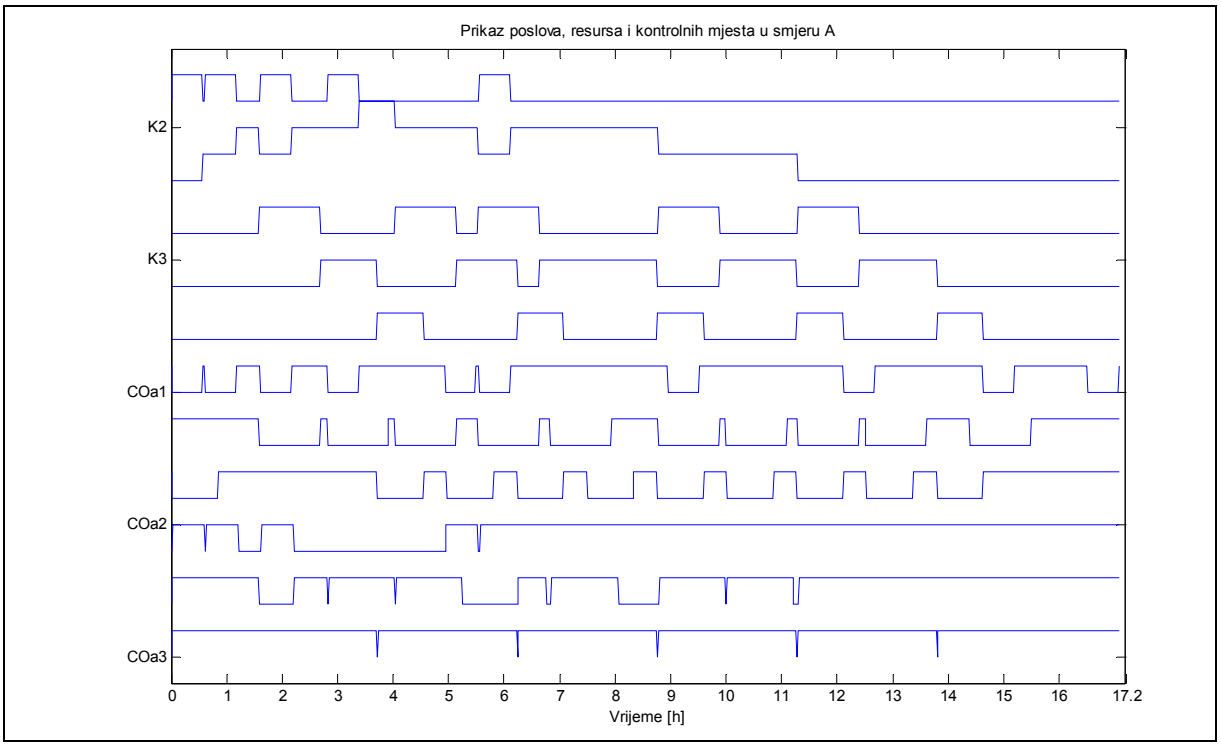
U ovom podpoglavlju će se statički model promatranog SPMK sustava proširiti na dinamički model. Uvest će se vremena 'setiranja' resursa. Svaki resurs zahtijeva određeno vrijeme (različito od nule) potrebno za ponovni rad nakon što je obavio posao. Vremena setiranja resursa dodijeljena su mjestima p13–p19 PM mreže (slika 6.12) i zadana su vektorom *PNtimes* (opisan u potpoglavlju 6.7.1.1.) :

```
PNtimes = [0 0 0.54 1 1.08 1 0.81 0.81 1 1.08 1 0.54
           0.90 0.30 0.60 0.60 0.02 0.02 0.02 0 0 0 0 0 0 0 0]'
```

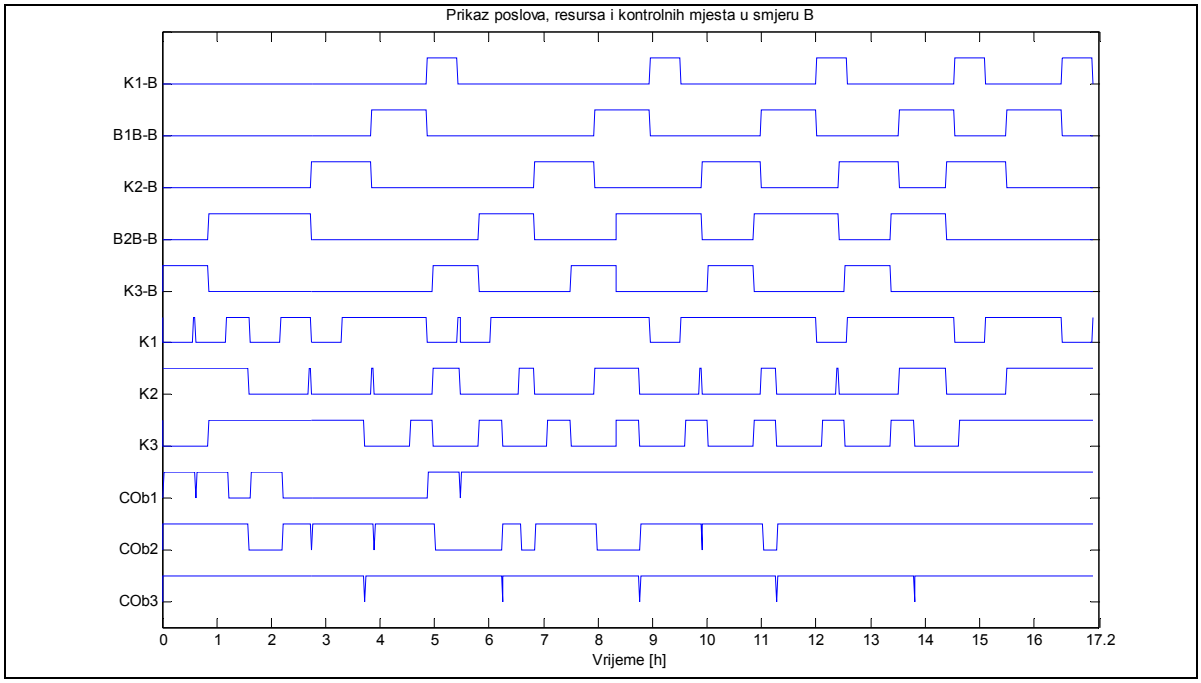
Vremena setiranja resursa

```
st1=0.3*PNtimes(4,1)*mf(13,1);
st2=0.3*PNtimes(6,1)*mf(14,1);
st3=0.3*PNtimes(9,1)*mf(15,1);
st4=0.3*PNtimes(11,1)*mf(16,1);
setup_time= [st1 st2 st3 st4];  % setup time za bazene
[sx sy]=size(setup_time);
 for i=1:sy
     PNtimes(12+i,1)=setup_time(1,i);
 end
```

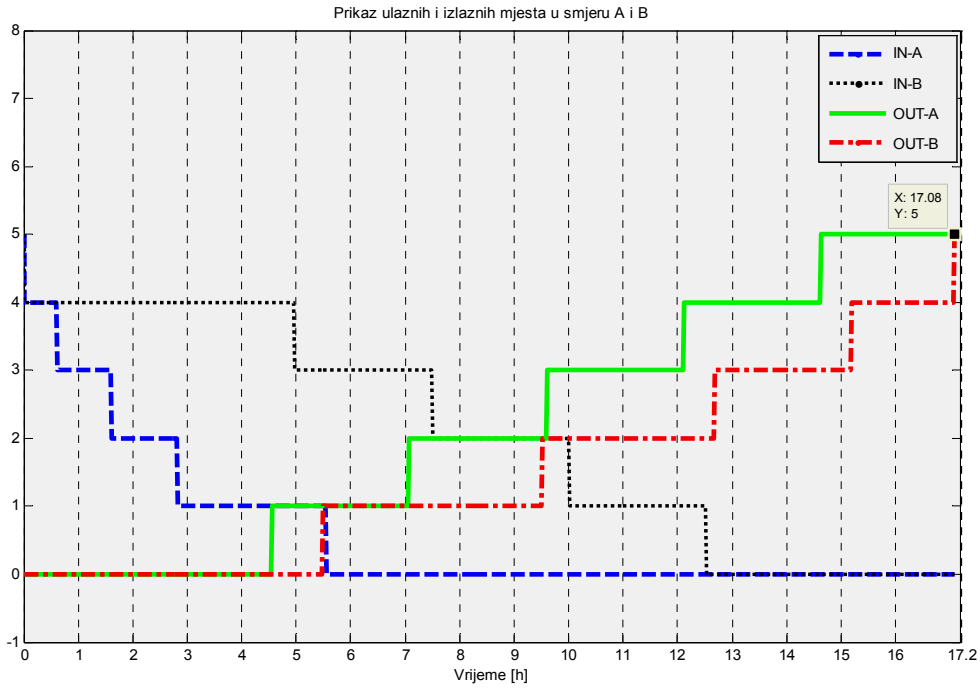
Slika 6.20 prikazuje rezultate simulacije nakon što su uvedena vremena setiranja za resurse. Primijenjena je procedura kontrole konflikta i zastoja, bez GA optimalizacije. Sa slike 6.20(c) može se uočiti da posljednji brod napušta kanal nakon 17.08 vrem.jed.


Prikaz poslova, resursa i kontrolnih mjesta u smjeru A
K2
K3
COa1
COa2
COa3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17.2
Vrijeme [h]

(a)


Prikaz poslova, resursa i kontrolnih mjesta u smjeru B
K1-B
B1B-B
K2-B
B2B-B
K3-B
K1
K2
K3
COb1
COb2
COb3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17.2
Vrijeme [h]

(b)

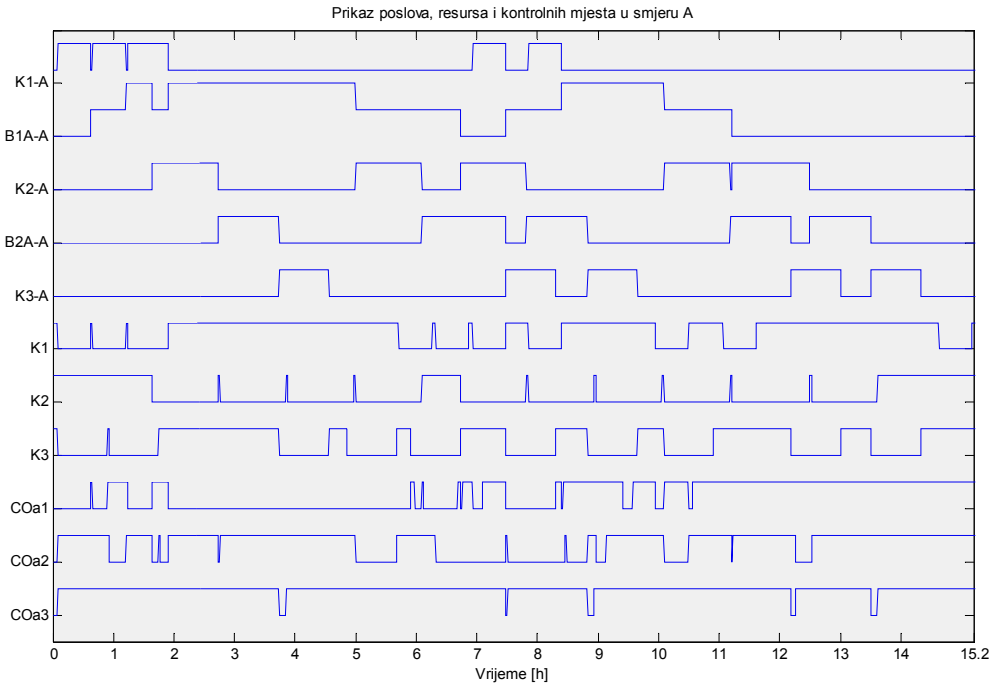


(c)

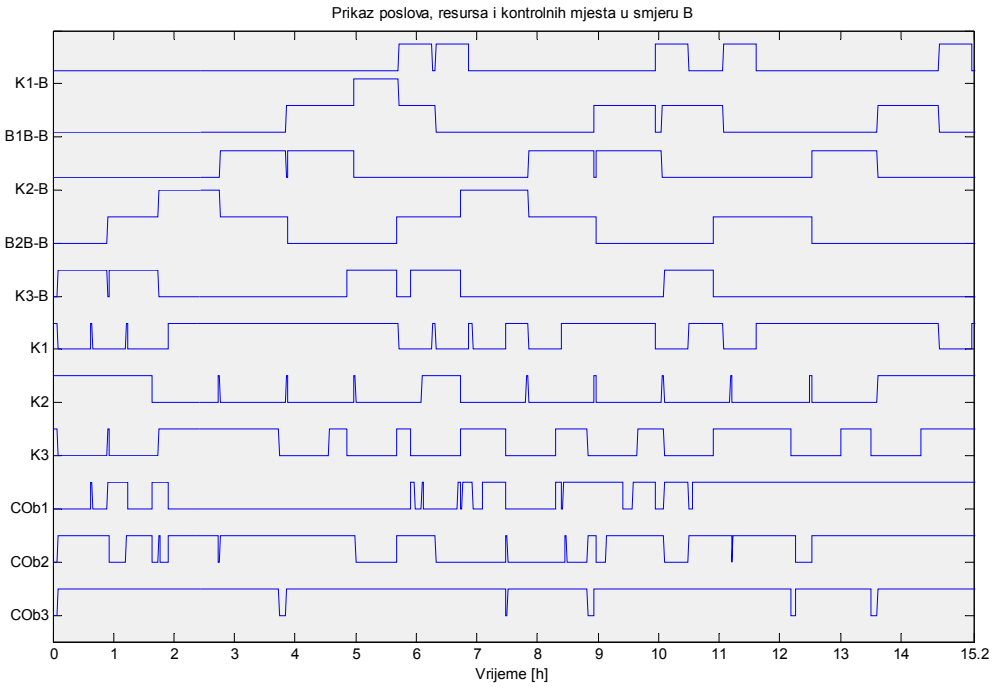
Slika 6.20 Rezultati PM modela dinamičkog sustava sa slike 6.12

Na isti SPMK sustav primijenjen je algoritam integracije PM i GA, opisan u poglavlju 6.5. To je dinamički sustav u kojemu uz vremena setiranja postoje određena kašnjenja poslova. Kako je opisano u poglavlju 6.5.1.1. svaki kromosom se sastoji od $m+2n+\mathcal{K}$ gena, gdje je n broj poslova, m broj projekata, a $\mathcal{K}$ broj resursa: prvih m gena se rabe kao prioriteti projekata, geni od $m+1$ do $m+2n$ rabe za određivanje vremena kašnjenja poslova, dok se geni od m+2n+1 do $m+2n+\mathcal{K}$ se rabe za određivanje vremena kada su projekti raspoloživi. Za razliku od kromosoma u statičkom modelu (slika 6.16), kod dinamičkog modela svi geni su različiti od nule.

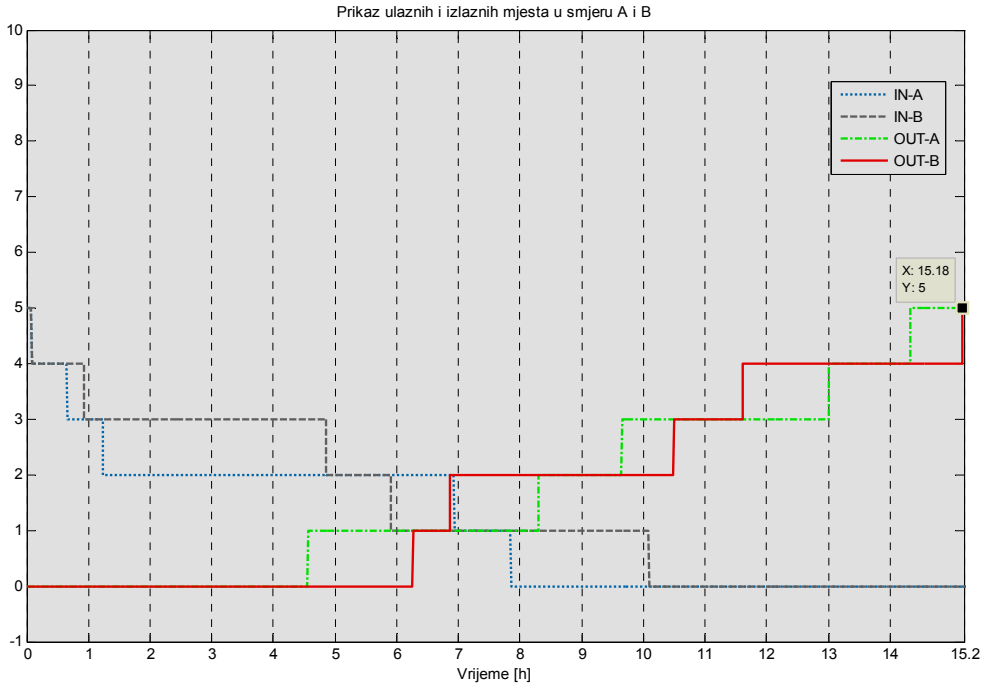
Nakon 10 testiranja, pri čemu je maksimalni broje generacija u GA postavljen na 20, a svaka populacija ima 10 kromosoma, dobiven je optimalni raspored bez konflikata. Slike 6.21 (a) i (b) prikazuju broj plovila u mjestima koja predstavljaju poslove i broj oznaka u kontrolnim mjestima. Može se primijetiti da u sustavu nema konflikata ni zastoja. Sa slike 6.21(c) može se uočiti da kod optimaliziranog rasporeda posljednje plovilo napušta sustav nakon 15.18 vrem.jed.


Prikaz poslova, resursa i kontrolnih mjesta u smjeru A
K1-A
B1A-A
K2-A
B2A-A
K3-A
K1
K2
K3
COa1
COa2
COa3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15.2
Vrijeme [h]

(a)


Prikaz poslova, resursa i kontrolnih mjesta u smjeru B
K1-B
B1B-B
K2-B
B2B-B
K3-B
K1
K2
K3
COb1
COb2
COb3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15.2
Vrijeme [h]

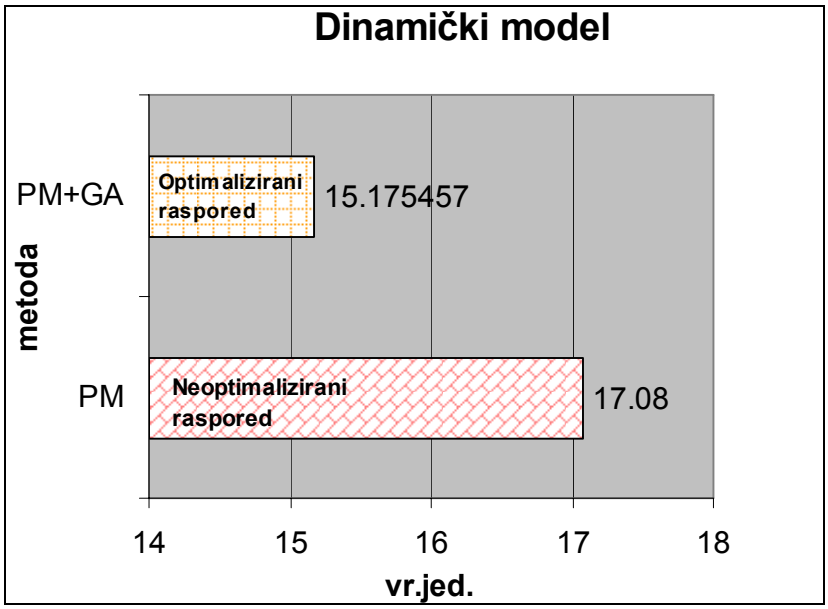
(b)



(c)

Slika 6.21 Optimalni raspored dinamičkog sustava sa slike 6.12: (a) Smjer A- broj plovila u mjestima poslova i oznaka u kontrolnim mjestima; (b) Smjer B- broj plovila u mjestima poslova i oznaka u kontrolnim mjestima; (c) Broj plovila u ulazno-izlaznim mjestima

Slika 6.22 prikazuje usporedbu ukupnog vremena (makespam) koje je potrebno da i posljednje plovilo napusti sustav kanala za neoptimalizirani (slika 6.20) i optimalizirani raspored poslova (6.21c), uz zadana vremena setiranja resursa i uz moguća kašnjenja poslova. Razvidno je da optimalizirani raspored skraćuje vrijeme za 1.9 vremenskih jedinica ili približno 11%. Primjerice, ako je vremenska jedinica 1 sat to je ušteda oko 2 sata. Ovo nije zanemariva ušteda jer i vlasnici brodova, putnici i lučki agenti žele da je vrijeme prolaska broda kroz kanale što kraće. Ako se poveća broj plovila, primjerice po 10 sa svake strane sustava i ako se ne primjene optimalizacijske metode, trebat će oko 40 sati za prolazak brodova kanalima [KEZ09]. Ovim algoritmom optimalizacije ušteda bi bila oko 4.35 sati. Ako se gleda duže vremensko razdoblje, primjerice poslovna godina onda su to velike uštede u vremenu, u potrošnji goriva (ako plovila čekaju s uključenim motorima) te je učinkovita uporabljivost resursa.

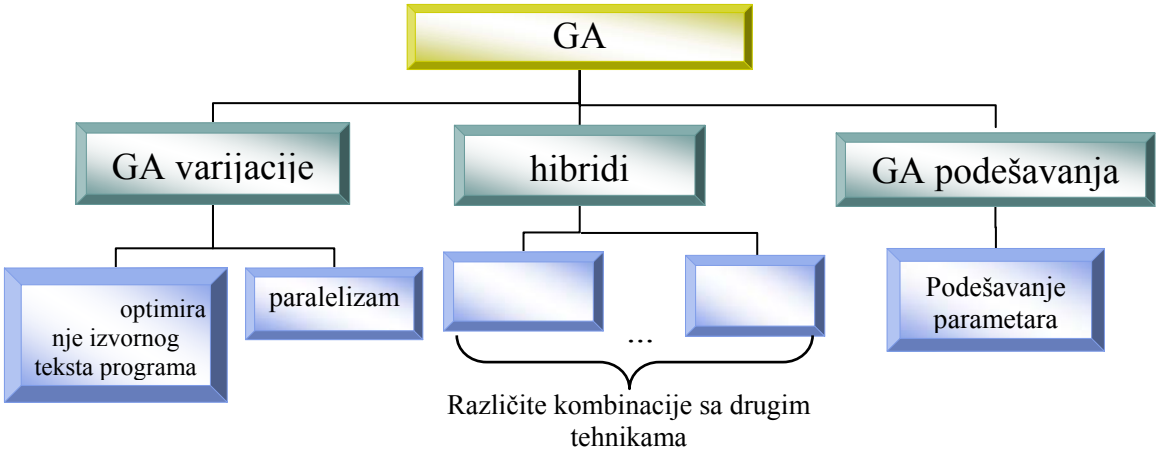


Slika 6.22 Usporedba vremena izvođenja svih poslova u sustavu za optimalizirani raspored i neoptimalizirani

6.7.2. Poboljšanja genetskog algoritma

Poboljšanja GA se izvode u cilju poboljšanja učinkovitosti GA, povećanje kvalitete rješenja ili zbog uštede proračunskog napora, odnosno, skratiti trajanje izvođenja.

Trajanje izvođenja se može skratiti također na tri načina: povećanjem brzine konvergencije čime se smanjuje broj iteracija, smanjenjem trajanja izvođenja jedne iteracije i paralelnim izvođenjem cijelog genetskog algoritma ili samo pojedinih genetskih operatora [GOL01]. Genetski algoritam se može poboljšati na tri načina: GA varijacijama, hibridima i poboljšani GA (slika 6.23).



Slika 6.23 Načini poboljšanja GA

Implementacijom jednostavnog GA dolazi se do rješenja koje je lokalni minimum. To ne mora značiti da se pronašlo i globalno rješenje problema. Tada bi korisnik trebao izvesti veći broj eksperimentiranja istog GA s različitim parametrima, kao što su veličina populacije, vjerojatnost križanja, vjerojatnost mutiranja. Na taj način bi se postigla bolja učinkovitost GA, a ujedno osigurala konvergencija ka globalnom optimumu problema.

U ovom podpoglavlju prezentirat će se rezultati do kojih se došlo testiranjem predloženog algoritma s različitim modifikacijama predloženog GA koji je opisan u poglavlju 6.5, a u cilju poboljšanja izvedbe GA. Zbog toga napravit će se usporedba predložene selekcije s nekim drugim načinima.

Glavni nedostatak GA najčešće je veliki računalni napor koji je potreban kako bi se riješio složeniji problem, a to znači i dugo vrijeme izvođenja. Znanstvenici su proučavali brojne tehnike kako bi riješili ovaj problem, primjerice uvođenjem paralelnog GA (Shonkwiller, 1993 navedeno u [HID05]). Parametri kao što su veličina populacije, broj ili učestalost razmjene jedinki izravno utječe na konvergenciju i kvalitetu rješenja [HID05].

U cilju poboljšanja učinkovitosti predloženog algoritma, statička veličina populacije zamijenit će se dinamičkom. Naime, predloženi GA proširit će se generičkom tehnikom koja se naziva *varijacija populacije* (pokrata: VP), u cilju smanjenja proračunskog napora (engl. *computational effort*) i ako je moguće poboljšati rješenje. Naime, veličina populacije će rasti sve dok je nužno uvoditi novi genetski materijal kako bi se optimalizirala funkcija cilja. Zato se predlaže izmijenjeni GA koji je, za ovaj problem, nazvan *FibonacciGA algoritam* (u prilogu vidjeti pseudokod). Osnovna ideja predloženog algoritma temelji se na problemu razmnožavanja zečeva koje je promatrao sam Fibonacci. Ovaj algoritam počinje Fibonaccijevim brojem 3, odnosno u populaciji postoje 3 para jedinki (ukupno 6 jedinki). U procesu reprodukcije sudjeluju dva par (4 jedinke), a jedan par mora čekati da "sazrije". Nakon reprodukcije dobiju se još dva nova para jedinki tako da u sljedećoj generaciji postoji ukupno 5 parova jedinki, odnosno 10 jedinki. U svakoj generaciji postoje dvije skupine jedinki. Prva skupina je nazvana varijablom `PopNovi`, a okuplja sve parove jedinki koje sudjeluju u reprodukciji. Druga grupa, nazvana `Djeca` okuplja jedinke koje su nastale u postupku reprodukcije i koja ne mogu sudjelovati u sljedećoj reprodukciji već čekaju sljedeću generaciju kako bi sazrjeli. Na ovaj način veličina populacije stalno raste i u svakoj generaciji broj parova jedinki jednak je nekom Fibonaccijevom broju. Varijabla `jedinke` određuje broj jedinki koje mogu sudjelovati u reprodukciji.

Osnovni je problem predloženog algoritma što veličina populacije stalno raste i broj parova jedinki naglo se povećava (3,5,8,13,21,34,55,89,144...) što povlači veliki računalni napor. Cijeli algoritam jako dugo traje, a da pri tome nema značajnih poboljšanja funkcije cilja. Zato se u ovom poglavlju predlaže novi model promjene veličine populacije, koji će kasnije biti opisan i testiran.

6.7.3. Selekcija

U ovom poglavlju usporedit će predložena elitistička selekcija (točka 6.5.1.2) i Fibonaccijeva selekcije.

Fibonaccijevu selekciju prvi put je uveo Bernik [BER01]. Za izbor jedinki koji ulaze u novu generaciju rabe se Fibonaccijevi brojevi. Fibonaccijev niz čine brojevi 0,1,1,2,3,5,8,... (više o Fibonaccijevim brojevima pogledati dodatak D2). Iz trenutne populacije izdvajaju se

samo one jedinke čije mjesto u populaciji odgovara Fibonaccij-evom broju. Na ovaj način kontrolira se raznolikost populacije i GA brže konvergira ka rješenju.

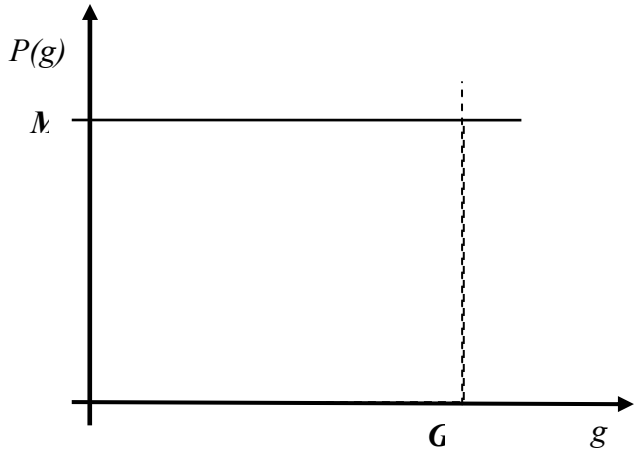
Fibonaccijeva selekcija u kombinaciji s elitizmom primijenit će se za odabir onih jedinki koje će činiti vrh nove populacije; jedinke trenutne populacije prvo, sortiraju prema elitizmu (vrijednostima fitnes funkcije). Na tako sortiranu populaciju primjeni se Fibonaccijeva selekcija. Na ovaj način osigurava se raznolikost populacije, a istovremeno preživljavaju one jedinke koje čine elitu populacije. Programski kod ovog operatora sadržan je u datoteci 'selection_ga_Fib.m" (Prilog I).

6.7.4. Veličina populacije

Brojni su znanstvenici istraživali i utvrdili da učinkovitost genetskih algoritama u traženju optimalnog rješenja uveliko ovisi o veličini populacije. Pregled tih istraživanja mogu se pronaći u [KOZ92], [KOU09], (Eiben 2004, Costa 1999, navedeni u [HID05]). Naime, velika populacija povećava genetsku raznolikost i veća je vjerojatnost da će algoritam pronaći globalni optimum. S druge strane, velika populacija zahtijeva više memorije za pohranjivanje i duže vrijeme računanja.

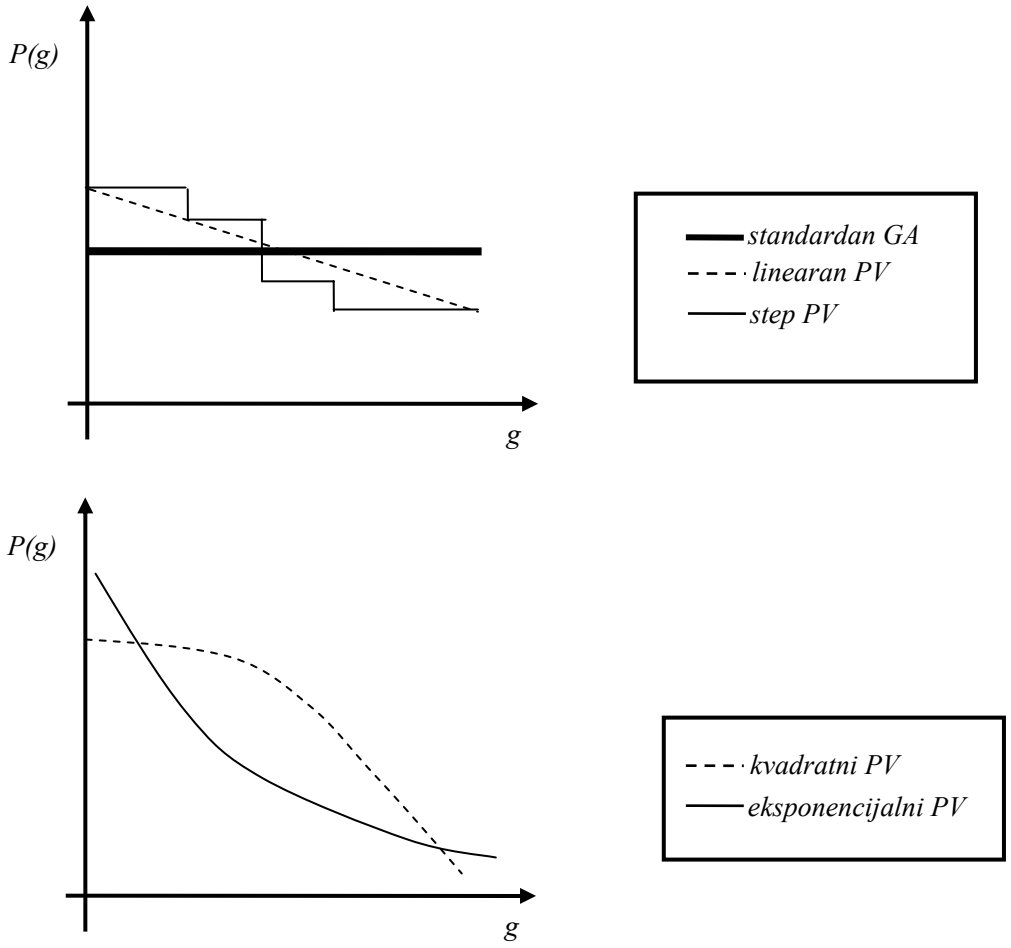
Mnogi istraživači su pokušali odrediti optimalan broj jedinki u populaciji. U jednom od istraživanja [CANT03] analiziraju se prednosti uporabe jednog izvođenja GA s velikom populacijom u odnosu na više izdvojenih izvođenja istog algoritma s manjim uzorcima populacija. U radu je zaključeno da je za teže probleme bolje primijeniti jedno izvođenje (s velikom populacijom uzimajući u obzir raspoložive resurse i ograničenja) nego višestruka izvođenja s manjim populacijama. Međutim, manja pozornost, a samim tim i manji broj istraživanja je posvećen podešavanju veličine populacije. Gledajući na to tehnički, veličina populacije je najfleksibilniji parametar u prirodnim sustavima. Mnogo lakše je podesiti veličinu populacije nego, primjerice, vjerojatnost mutacije. Ipak kod evolucijskog računanja, veličina populacije je, tradicionalno kruti parametar [KOF08].

Kod standardnog GA veličina populacije je konstantna za vrijeme izvođenja GA i ukupan broj jedinki u bilo kojoj generaciji g grafički se može prikazati vodoravnom linijom (slika 6.24). Broj generacija g varira od 0 (početna generacija koja sadrži slučajno generiranu populaciju) do Max$G$ (maksimalan broj generacija koji bi se trebao izvesti). $P(g)$ predstavlja veličinu populacije za svaku generaciju g, a odsječak na osi y je veličina populacije M.



Slika 6.24 Veličina populacije u standardnom GA

Način promjene veličine populacije može biti različit. Broj jedinki u populaciji može se smanjivati ili povećavati ili se slučajno generirati. Načini smanjivanja (povećanja) veličine populacije mogu biti različiti, primjerice linearan, eksponencijalan, kvadratan, hiperbolički ili slučajan (slika 6.25).



Slika 6.25 Načini smanjivanja veličine populacije

U (Tomassini, 2004, navedeno u [KOU09]) odluka treba li povećati ili smanjiti veličinu populacije temeljila se na odnosu između vrijednosti najboljeg fitnesa u populaciji iz trenutne generacije g (f_g) i najboljeg fitnesa iz prethodne generacije (f_{g-1}). Ta vrijednost je spremljena u varijabli nazvana *pivot*. U navedenom radu predstavljene su dvije verzije varijable pivot: (1) $pivot = \frac{\Delta_{g-1}}{\Delta_g}$; (2) $pivot = \Delta_{g-1} - \Delta_g$, gdje je $\Delta_g = f_{g-1} - f_g$. U izrazu (1) pivot funkcija se naziva DIV (pokrata od Division) pivot funkcija, jer je osnovna funkcija dijeljenje, a u izrazu (2) se naziva SUB (pokrata od Substraction) jer je oduzimanje funkcija po kojoj se pivot izračunava.

Pivot funkcija se rabi kao mehanizam odluke treba li nove jedinke dodati u populaciju ili je potrebno neke ukloniti iz populacije.

U navedenom radu veličina populacije se mijenja iz generacije u generaciju prema sljedećem izrazu:

$$\Delta P(g) = \begin{cases} +0.2\% \cdot P(g) \cdot \Delta_g, & \Delta_g < pivot \\ -1\% \cdot P(g) \cdot \Delta_g, & \Delta_g \geq pivot \end{cases}. \tag{6-22}$$

Broj generacija varira od 0 do maksimalnog broja dopustivih generacija, a $P(g)$ je veličina populacije za bilo koju generaciju g.

Dakle, broj jedinki koji se dodaje u populaciju ili uklanja iz nje ovisi isključivo o trenutnoj veličini populacije. Na taj način, kada je trenutna populacija jako velika onda su velike i silovite promjene u veličini te populacije. Posljedica toga je, ako algoritam upadne u fazu stagnacije (nema promjene u veličini fitnes funkcije), treba dovoditi novi genetski materijal u populaciju i veličina populacije će stalno rasti.

Veličina populacije će se dinamički mijenjati ovisno o vrijednosti funkcije cilja, a promjena broja jedinki (kromosoma) u populaciji temeljiti će se na Fibonaccijevim brojevima i zlatnom rezu (vidjeti prilog o Fibonaccijevim brojevima).

Učinkovitost ove ideje će se testirati na predloženom SPMK sustavu i pokazat će se da ima potencijala da dođe do dobrog rješenja uz nižu proračunsku cijenu u odnosu na predloženi algoritm iz poglavlja 6.5. Programski kod ovog algoritma je sačuvan u datoteci *'DinFibonacciGA.m'*.

6.7.5. Računalni napor

Obično kad se određuje učinkovitost GA onda se uspoređuje mjera kvalitete (primjerice, veličina fitnesa) i broj utrošenih generacija. Međutim, ovakva usporedba je valjana samo onda kada je utrošeni računalni napor za svaku genaraciju isti. Međutim kada se veličina populacije mijenja iz generacije u generaciju ili iz eksperimenta u eksperiment onda je ovakva usporedba nevaljana. Stoga će se u ovom poglavlju učinkovitost predloženog GA odrediti usporedbom kvalitete funkcije cilja, veličine populacije i količinom računalnog napora koji je utrošen. Manji računalni napor znači učinkovitiji algoritam. Računalni napor se definira kao ukupan broj jedinki koji se mora vrednovati za zadani broj generacija. Kako generacije posjeduju različiti broj jedinki u populaciji, mora se računati parcijalni računalni napor po generaciji (pokrata: $PN(g)$), kao ukupan broj jedinki koji je ocijenjen u generaciji g. Onda se računalni napor (pokrata: *RN*) može izračunati prema sljedećem izrazu

$$RN = \sum_{g=0}^{G} PN(g), \tag{6-23}$$

Izraz (6-23) ekvivalentan je ukupnom broju evaluacija, pri čemu je G ukupan broj izvedenih generacija.

Neka je *PRN* oznaka za prosječan računalni napor (engl. *Average Computation Effort*). Ako dva algoritma dođu do iste vrijednosti funkcije cilja onda se *PRN* rabi kao mjera superiornosti algoritma. To znači: manji *PRN* ekvivalentan je bržoj konvergenciji k rješenju, odnosno superiornijem algoritmu. Iako ova mjera ovisi o problemu koji se rješava korisno je

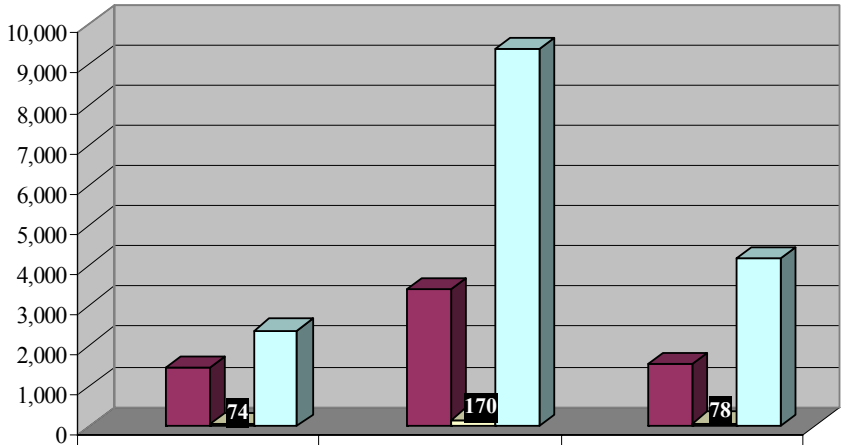
napraviti usporedbu za različite postavke istog problema. Kako bilo, ako se reducira *PRN* to implicira i kraće računalno vrijeme.

6.7.6. Rezultati testiranja poboljšanog GA algoritma

6.7.6.1. Utjecaj selekcije

Prvi eksperiment je izveden u cilju određivanja najbolje metode selekcije. Algoritam koji integrira GA i PM iz poglavlja 6.5 testiran je s tri različite metode selekcije: *ga55* označava algoritam sa elitističkom selekcijom, *ga_samopril55* sa samoprilagođivanjem, *ga_fib_sel55* sa Fibonaccijevom selekcijom. Veličina populacije u ovom eksperimentu je 55 jedinki. Deset optimalizacija je testirano sa svakim algoritmom i najbolji rezultati su prikazani na slici 6.26. Može se uočiti da je razlika između ukupnog makespama zanemariva, dok se to ne može reći i za utrošeni računalni napor.

Elitistička selekcija ima najmanji računalni napor (RN=1475, odnosno prosječni računalni napor po generaciji iznosi PRN=75). Fibonaccijeva selekcija ima znatno manji računalni napor u odnosu na metodu samoprilagođavanja (1555 vs. 3399, odnosno prosječni računalni napor 78 vs. 170). Stoga u nastavku istraživanja će se primjenjivati elitistička selekcija, jer je pokazala najbolje značajke.



	elitistička	samoprilagođavanje	Fibonaccijeva
makespam [vr. jed.]	14.9454070	14.897280	14.895516
RN	1475	3399	1555
PRN	74	170	78
vrijeme izvođenja [s]	2,397.016	9,389.65625	4,171.48438

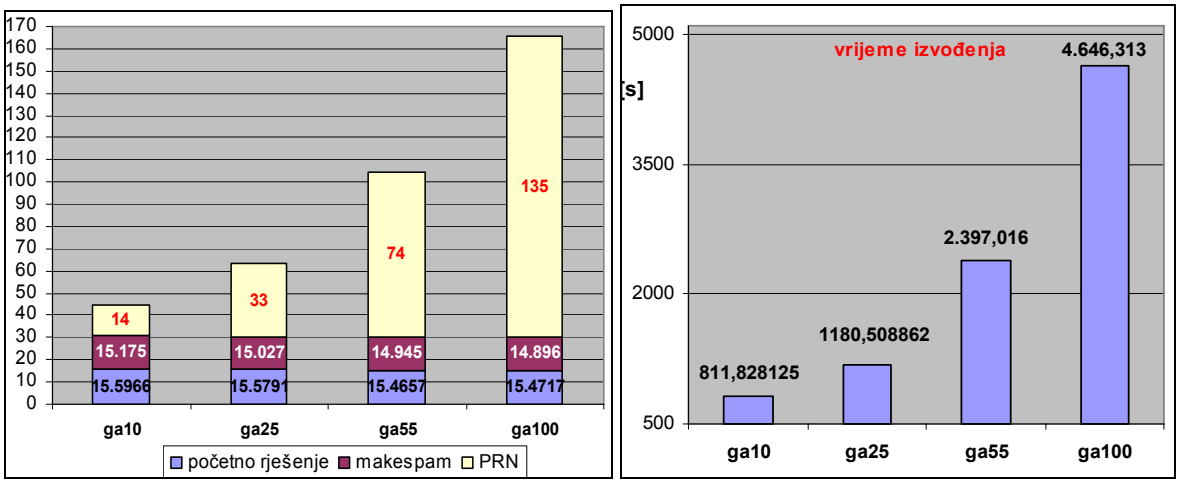
Slika 6.26 Ukupni makespama i računalni napor u odnosu na način selekcije

6.7.6.2. Utjecaj veličine populacije

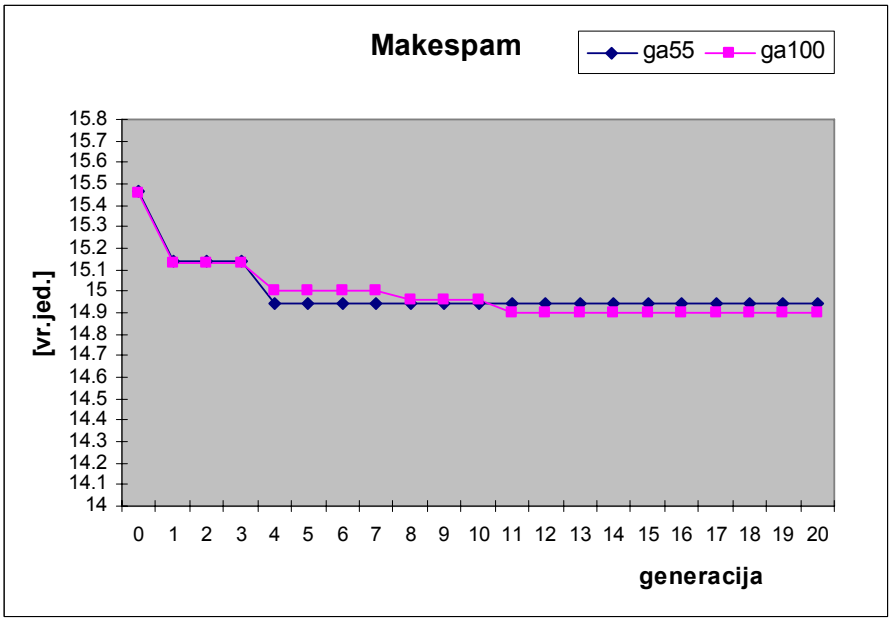
Statička veličina populacije

U cilju poboljšanja makespama (ukupno vrijeme da i posljednji brod napusti kanale) izvedena su tri eksperimenta sa standardnim GA s različitim veličinama populacije, koja je bila nepromijenjena od početne do zadnje generacije. Testiranje je provedeno sa 10, 25, 55, 100 jedinki u populaciji. Slika 6.27 prikazuje rezultate testiranja, pri čemu *ga10* označava GA s veličinom populacije od 10 jedinki, *ga25* s 25 jedinki, *ga55* s 55 jedinki i *ga100* sa 100

jedinki. Može se uočiti da povećanje broja jedinki vodi ka boljoj funkciji cilja, ali se znatno povećava računalni napor i vrijeme izvođenja (engl. *run time*). Slika 6.28 prikazuje konvergenciju algoritama ga55 i ga110 ka rješenju problema. Može se primijetiti da oba algoritma relativno rano dostižu lokalni minimum. Naime, iako su populacije s velikim brojem jedinki i veliki je utrošeni računalni napor, nakon 10-te (11-te) generacije nema poboljšanja funkcije cilja. Kako bi se reducirao računalni napor, a samim tim i vrijeme izvođenja, trebalo bi smanjiti veličinu populacije kada algoritam dosegne lokalni optimum.



Slika 6.27 (a) Odnos početnog rješenja i makespama u odnosu na računalni napor, gdje je PRN prosječni računalni napor; (b) Odnos vremena izvođenja i veličine populacije



Slika 6.28 Konvergencija algoritama ga55 i *ga100* k lokalnom optimumu

Novi model promjene veličine populacije

Novi mehanizam promjene veličine populacije koji se ovdje predlaže temelji se na vrijednostima funkcije cilja, odnosno na određivanju rasporeda poslova koji najkraće traje.

Kako se funkcija cilja poboljšava tako će se u algoritmu povećavati veličina populacije, kratkotrajni nedostatak u poboljšanju funkcije cilja činit će populaciju manjom, dok stagnacija funkcije cilja kroz određeni broj generacija će uzrokovati ponovni rast veličine populacije.

Pseudokod predloženog mehanizma promjene veličine populacije prikazan je slikom 6.29 (sadržan u datoteci *'DinFibonacciGA.m'*).

```
generacija=generacija+1;
delta=(Best(generacija-3).FTime-Best(generacija-1).FTime);
RO=0;
novi_delta=(delta>RO);
          if (not(novi_delta))
              if (not(stari_delta))
               % ni novi ni stari broj jedinki nije postigao cilj
               % veličina populacije se mijenja prema
               % Fibonaccijevim brojevima
                  GA_param.pop_size_stari=GA_param.pop_size;
                   GA_param.pop_size=Fib(niz)*2+size(Djeca,1);
                   jedinke=Fib(niz)*2;
                       if GA_param.pop_size<GA_param.pop_size_stari
                         GA_param.pop_size_stari=GA_param.pop_size;end
                else
                  % novi broj jedinki nije postigao cilj
                  % stari broj jedinki je postigao cilj
                  % veličina populacije se postavlja na početni broj
                GA_param.pop_size_stari=N_poc;
                      GA_param.pop_size=N_poc;
                      niz=2;flag=1;jedinke=niz*2;
                      [Pop_init,Makespam,raspored] = selection_ga_elitist
                        (PopNovi,[],[],Makespam,[],[],...
                          raspored,[],[],[],GA_param,Fib(niz)*2);
                     Makespam=Makespam';
                [Djeca,Djeca_makespam,Djeca_raspored]=
                 selection_ga_elitist(Djeca,[],[],Djeca_makespam',[],[], ...
                  Djeca_raspored,[],[],[],GA_param,(Fib(niz+1)-Fib(niz))*2);
              end
          else
              if (not(stari_delta))
               % novi broj jedinki je postigao cilj
               % stari broj jedinki nije postigao cilj
                     % veličina populacije se mijenja prema parametru GR1
                GA_param.pop_size_stari=GA_param.pop_size;
                GR=round(1.6180*(maxRN-TRN)*(inicijalniF-Best(generacija-
1).FTime)/inicijalniF);
                     GA_param.pop_size=GA_param.pop_size_stari+GR;
                     jedinke=GA_param.pop_size;
                     GA_param.pop_size=GA_param.pop_size+size(Djeca,1);
              else
               % i novi i stari broj jedinki je postigao cilj
               % veličina populacije se ne mijenja
                     GA_param.pop_size=GA_param.pop_size_stari;
                     niz=niz-2;jedinke=Fib(niz)*2;
              end
          end  % kraj if funkcije
   stari_delta=novi_delta;
```

Slika 6.29 Pseudo kod za određuvanje dinamičke veličine populacije

Ovaj kod koristi sljedeće varijable: `broj generacija=20; delta1, novi_delta, stari_delta, RO, N_poc, GR1, maxTRN, TRN, inicijalniF, Best(generacija-1).FTime)`. U nastavku će ove varijable biti opisane, a vrijednosti neke od navedenih varijabli određene su iz niza prethodnih eksperimentiranja koja su provedena u cilju ovog rada.

Varijabla `delta` označava vrijednost fitnes funkcije kojom se aproksimira poboljšanje makespama. U cilju ispitivanja stagnacije populacije uspoređuje se najkraće vrijeme izvođenja (makespam) najboljeg rasporeda trenutne populacije i najkraće vrijeme izvođenja populacije koji je postignut dvije generacije ranije.

Varijabla `stari_delta` (odnosno `novi_delta`) je Boolean-ova varijabla čija je vrijednost 1 (istina, true) ako je najbolji makespam prethodne generacije (odnosno trenutne generacije) zadovoljio cilj, inače je vrijednost 0 (laž, false).

Kako što se iz pseudo koda može primijetiti, mehanizam promjene veličine populacije temelji se na četiri načina promjene:

1. Ako vrijedi da je `stari_delta=0` i `novi_delta=0`, onda nema poboljšanja makespama u posljednje tri generacije i tada treba povećati broj jedinki u populaciji. Povećanje prati zakon razmnožavanja zečeva i Fibonaccije niz brojeva. Broj novih jedinki koji se dodaje trenutnoj populaciji jednak je broju djece koja su stvorena u prethodnoj generaciji. Varijabla `jedinke` označava broj kromosoma koji mogu sudjelovati u reprodukciji i stvaranju potomaka.
2. Ako vrijedi `stari_delta=0` i `novi_delta=1`, to znači da je u populaciji pronađen raspored s kraćim vremenom trajanja od ovog iz prethodne populacije. Dakle, postignuto je poboljšanje funkcije cilja. U tom slučaju mijenja se veličina populacije tako da se na trenutni broj jedinki dodaje ili oduzima njih `GR`. Vrijednost varijable `GR` računa se prema sljedećem izrazu:

$$GR = round\left[\varphi \cdot \left(max\,RN - TRN\right) \cdot \frac{\left(inicija\,ln\,iF - Best\left(generacija-1\right).FTime\right)}{inicija\,ln\,iF}\right]$$

gdje je *round* funkcija kojom se realni broj zaokruži na cijeli.

Naime, varijabla `GR` je funkcija koja ovisi o proračunskom naporu trenutne populacije (parametar `TRN`), o vrijednosti najboljeg makespama te populacije (`Best(generacija-1).FTime`) i o vrijednosti makespama početne populacije (`inicijalniF`). Izraz $\left(\texttt{inicijalniF} - \texttt{Best}\left(\texttt{generacija}-1\right).\texttt{FTime}\right)$ predstavlja poboljšanje koje je postignuto kada je postignut najbolji makespam u trenutnoj populaciji. φ je varijabla koja određuje koliko jak mora bit porast ili pad veličine populacije. U ovom algoritmu za vrijednost varijable φ uzeta je vrijednost Zlatnog reza, 1.6180, jer se željela očuvati harmonija u generacijama. Varijabla `maxRN` je najveći dopustivi računalni napor po generaciji koji je inicijalno postavljen na `80`. Tako kod generacija s velikim brojem jedinki (preko 80), računalni napor koji je potreban za evaluaciju djece i jedinki koji čine dno populacije prelazi 80, a u sljedećem koraku parametar `GR` imat će negativnu vrijednost. To znači da će se u sljedećoj generaciji smanjiti broj jedinki koje mogu sudjelovati u reprodukciji. Na ovaj način ograničava se da se broj jedinki koje sudjeluju u razmnožavanju uveća najviše

za 80 novih. Taj broj je sasvim dovoljan, jer je kroz eksperimentiranje pokazano da se broj jedinki može stalno povećavati i biti jako velik[27] a da nema promjene u vrijednosti makespama.

3. Ako vrijedi `stari_delta=1` i `novi_delta=0`, znači kratkoročno nema poboljšanja funkcije cilja i veličinu populacije valja smanjiti. Ovdje je odabrano vraćanje na početak, odnosno varijabla `N_poc` predstavlja početnu veličinu jedinki u populaciji. Odabran je početni broj jedinki jer se kroz niz eksperimenata pokazao kao dobar izbor između uštede proračunskog napora i očuvanja dobrog genetskog materijala.
4. Ako ne vrijedi ni 1, ni 2, ni 3 znači da poboljšanje funkcije cilja ide u željenom smjeru. U tom slučaju nema promjene u veličini populacije.

Ocjena rezultata

U ovom podpoglavlju prezentirat će se rezultati koji su postignuti primjenom teoretskih koncepata iz prethodnom podpoglavlju, na promatrani SPMK sustav kanala. Glavni cilj je bio reducirati vrijeme izvođenja algoritma i smanjiti računalni napor, te ako je moguće smanjiti vrijeme izvođenja svih poslova u sustavu.

Tablica 6.6 sadrži rezultate izvođenja za tri različita algoritma koji su međusobno uspoređeni. Kod svih algoritama koristi se elitistička selekcija opisana u prethodnom poglavlju, uvjet zaustavljanja je max broj generacija (20), VRH populacije čini 20% jedinki trenutne populacije, a DNO 30% jedinki. Ove vrijednosti su se pokazale kao najbolje kroz niz prethodnih eksperimentiranja s opisanim algoritmima.

GAFibDiv je predloženi *FibonacciGA* algoritam kod kojeg se veličina populacije mijenja prema DIV modelu (slika 6.30), a DinFibGA je oznaka za *DinFibonacciGA* algoritam koji je ranije opisan.

```
delta1=(Best(generacija-2).FTime-Best(generacija-1).FTime);
delta2=(Best(generacija-3).FTime-Best(generacija-2).FTime);
if delta1==0
    RO=1;
else
    RO=delta2/delta1;end
novi_delta=(delta1>RO)
```

Slika 6.30 Pseudo kod za DIV pivot funkciju

Algoritam DinGA je predloženi PMGA algoritam iz poglavlja 6.5 s varijabilnom veličinom populacije. Veličina populacije se mijenja prema Fibonaccijevom nizu. Kako se vidi iz tablice algoritam nije došao do kraja jer je nakon 15 generacije postigao veličinu populacije od 610 jedinki, a da pri tome nema promjene u veličini funkcije cilja. Tu je algoritam prekinut (nakon 1 sata izvođenja) jer u sljedećoj generaciji je imao 987 jedinki i došlo je do zagušenja u računalu. Zato je taj algoritam modificiran i rezultat je DinGA2 algoritam. Ključna varijabla u DinGA2 algoritmu je broj *n*, broj ponavljanja GA koji bi se

[27] Kroz eksperimentiranje ta brojka je narasla do 700, a nije bilo poboljšanja u funkciji cilja.

trebao izvesti kako bi algoritam došao do globalnog optimuma. Broj n varira ovisno o značajkama funkcije cilja. Umjesto da GA radi s velikim brojem jedinki u populaciji i s velikim brojem generacija, algoritam DinGA je proširen tako da radi s malim brojem generacija (10), ali se algoritam ponavlja dok se ne približi globalnom optimumu. Algoritam će se ponavljati sve dok je zadovoljen sljedeći uvjet

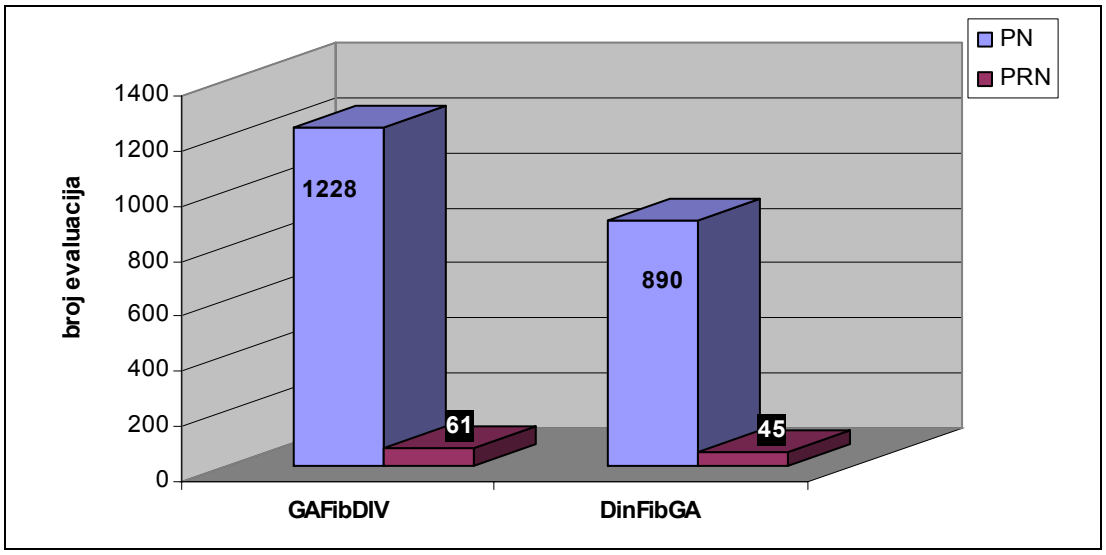
$$uvjet = \left(\left| \frac{fitnes(Fit-2) - fitnes(Fit)}{fitnes(Fit)} \right| \leq 0.02 \right), \tag{6-25}$$

gdje je Fit broj izvođenja GA algoritma ($Fit = 1,\ldots,n$), a $fitnes(Fit)$ je najbolja vrijednost funkcije cilja u Fit-om ponavljanju GA algoritma. Kada varijabla *uvjet* poprimi vrijednost 0 ('false') algoritam se zaustavlja i postigao je globalni optimum.

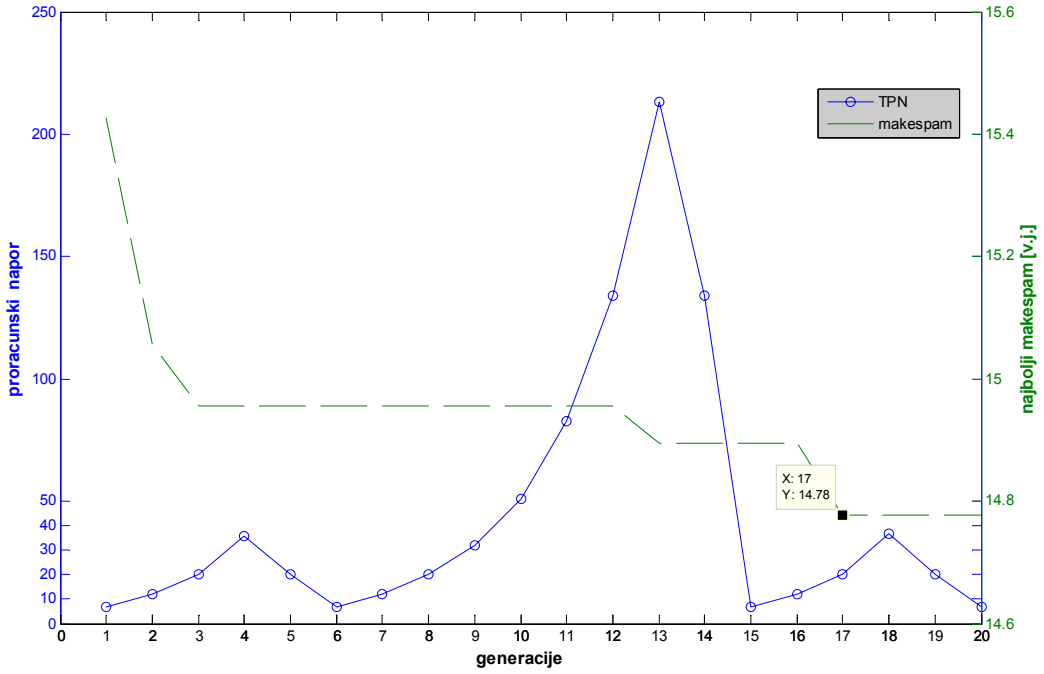
Tablica 6.6 Pregled rezultata za algoritme s dinamičkom veličinom populacije

Algoritam		**DinGA**	**DinGA2**					**GAFibDIV**	**DinFibGA**
Populacija: *dinamička*	Broj jedinki od min do max	5-610	5-144					6-466	6-174
generacije	deklarirani broj generacija	20	10					20	20
	izvedeni broj generacija	16	10	10	10	10	10	20	20
	Najbolje rješenje početne populacije	15.470030	16.1262	17.799	16.7004	15.471	16.7004	15.641519	15.627057
	Najbolji postignuti makespam	14.957993	15.2762	14.996	15.4505	14.9579	15.4505	14.898543	14.776750
	Ukupni računalni napor RN	-	335	465	385	66	82	1228	890
	Prosječni računalni napor PRN	-	28	27	30	8	10	61	45
	Vrijeme izvođenja [s]	Nije došlo do kraja	8,833.562500					4,967.547	1,416.375

Kao što se iz tablice 6.6 i slike 6.31 može vidjeti predloženi algoritam *DinFibonacciGA* postiže najbolje rezultate. Naime, ovaj algoritam je došao do najkraćeg vremena izvođenja svih poslova u promatranom SPMK sustavu od 14.77675 vr. jed. To je najkraće vrijeme koje je u nizu eksperimentiranja postignuto za ovaj dinamički model. Pri tome je i najkraće vrijeme izvođenja i velike su uštede i u računalnom naporu. Naime, kada algoritam dođe do faze stagnacije, pokazano je da velika populacija ne mora poboljšati funkciju cilja, zato ovdje u slučaju stagnacije naglo pada broj jedinki u populaciji (slika 6.32) i sačuvan je dobar genetski materijal.



Slika 6.31 Usporedba računalnog napora obzirom na način promjene veličine populacije



Slika 6.32 Usporedba po generacijama računalnog napora i vrijednosti najbolje funkcije cilja u populaciji

6.8. Sažetak poglavlja

U ovom je poglavlju ostvarena integracija PM i GA namijenjena optimiranju projektnih rasporeda s ograničenim resursima. Cilj je bila provjera mogu li se integracijom PM i GA pronaći rješenja koja su optimalna, a pri tome su izbjegnuti konflikti i zastoji, uz razumno vrijeme izvođenja algoritma.

Sustav je modeliran kao sustav sa diskretnim događajima jer se relativno jednostavno mogu prikazati stvarni sustavi. Odabrana je Petrijeva mreža kao osnova tog modela zbog sljedećih razloga:

- PM posjeduje sposobnost predstavljanja različitih stanja u sustavu na jezgrovit način, mogućnost prikaza slijeda poslova kao i njihovu međusobnu povezanost i konkurentnost;
- Posjeduje grafičku reprezentaciju, koja se lako može nadograditi i promijeniti;
- Njeno ponašanje se može jednostavno matematički izraziti što omogućava formalnu analizu sustava;
- Matrični model PM omogućava jednostavno praćenje dinamičkih svojstava sustava kao što su konflikti i zastoji;
- Dosadašnja istraživanja u primjeni PM na sustave kod kojih je važna sigurnost, kao što je sustav upravljanja zračnim prometom, nuklearni sustavi itd., pokazala su obećavajuće rezultate.

Povezivanje metode GA sa PM pokazalo se učinkovitim, što potvrđuju rezultati testiranja na primjeru kanala. Kada se algoritam primjenjivao samo na statičke modele, odnosno na one kada nema kašnjenja poslova i svi brodovi bili su raspoloživi u trenutku 0, pronalazili su se optimalni rasporedi već nakon prve iteracije i takvo rješenje je u prosjeku poboljšalo neoptimalizirano rješenja za 20-tak postotaka. Kod dinamičkog modela poboljšanja su od 11% do 13.5%, što su značajne uštede u ljudskom vremenu.

Ono po čemu se ovaj algoritam razlikuje od ostalih algoritama koji su nastali integracijom PM i GA, i što je glavni doprinos ovog rada je:

1) Umjesto obične PM predložena je primjena MRF_1 klasa flowline PM-a, pri čemu mjesta predstavljaju i resurse i poslove;
2) Predlaže se drugačiji način dodjeljivanja vremena i poslovima i resursima, svako mjesto je podijeljeno na dva dijela ulazni dio i izlazni dio i pridružena su im odvojena vremena i na taj način je povećana snaga modela, na štetu nešto većeg zauzetog memorijskog prostora i vrlo malo sporije izvedbe programa;
3) Analiziraju se strukturna svojstva mreže kao što je P-invarijantnost, kružna čekanja, kritični sifoni i kritični podsustavi;
4) Kako bi se izbjegli konflikti, zastoji prve i druge razine, predlaže se uvođenje kontrolnih mjesta u mrežu (nadglednik). Kontrolna mjesta omogućuju okidanje točno određenog prijelaza kako bi se izbjegao konflikt i ograničavaju broj oznaka (brodova) u kritičnom podsustavu kako bi se izbjeglo zaglavljenje prve razine. Moguće je da u sustavu postoji zaglavljenje druge razine ako je sustav iregularan i sadrži takozvane ključne resurse. Kako bi se izbjegla ovakva vrsta zastoja, potrebno je osigurati da ključni resurs nije zadnji raspoloživi resurs u mreži. Predložena procedura je primijenjena i vrednovana na sustavu morskih kanala.
5) Raspored poslova generiran je pomoću procedure parametriziranog aktivnog rasporeda kojim se nastojalo smanjiti kašnjenje, a time i prostor pretraživanja rezultata.

6) U metodologiji GA je pored već poznatog principa evolucije jedinke unutar populacije, koji se zasniva na križanju i mutaciji razvijen drukčiji tip elitističke selekcije, koja se paralelno primjenjuje za odabir elite i za odabir djece nastala u reprodukciji. Rezultati pokazuju da primjenom elitističke selekcije rezultati brzo konvergiraju prema rješenju, a time se skraćuje i vrijeme optimalizacije i računalni napor.

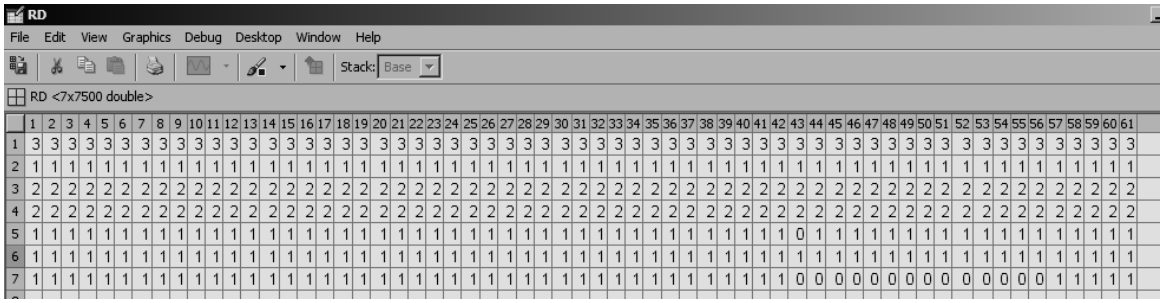
7) U ovom poglavlju ispitana je korisnost veličine populacije. Ideja nije nova kod evolucijskog računanja, ali prema našim saznanjima, nitko do sada se nije bavio tim problemom pri integraciji PM i GA. Testirani su različite varijacije veličina populacije s odgovarajućom shemom povećavanja/smanjivanja broja jedinki u populaciji. Osnovna ideja je bila pokazati ima li promjena veličine populacije ikakvog značenja na algoritam u smislu računalnog napora i ukupnog vremena izvođenja. Nastojao se smanjiti računalni napor dinamičkom prilagodbom veličine populacije rješenju i trenutnom broju jedinki u populaciji. Razvijen je GA kod kojeg se reprodukcija odvija prema Fibonaccijevom nizu i predložen je mehanizam dinamičke promjene veličine populacije. Rezultati testiranja pokazali su ispravnost ovog pristupa.

Razvijeni algoritmi vrijedi za sve projekte s ograničenim sredstvima neovisno o izgledu projektnog plana, složenosti, broju poslova i resursa. Također, algoritam bi se mogao proširiti na nekoliko različitih načina.

- Preuranjenost/kašnjenje u modelu rasporeda ima veliko značenje zbog važnosti i primjerenosti u praksi. Naime, problemi rasporeda kod kojih se vrednuje i preuranjenost i kašnjenje su u skladu s načelom proizvodnje "Baš na vrijeme" i "Upravljanje distribucijskim lancem" (engl. *Just in Time / Supply Chain Management*).
- Kao funkcija cilja može se uzeti i linearna preuranjenost i kvadratna funkcija cijene kašnjenja:

$$C_a \cdot \sum_i T_i^3 + C_b \cdot \sum_i E_i^2 + C_c \cdot \sum_i FD_i^2 + C_{\mathrm{d}} \cdot \sum w \qquad (6\text{-}26)$$

8) Moguće je u svakom trenutku on-line kontrolirati raspoloživost resursa preko editora varijabli. Na slici 6.32 prikazan je editor varijable RD-kapaciteti resursa, gdje su redci resursi, a stupci iteracije. U slučaju otkazivanja nekog resursa u i-toj iteraciji, odnosno u trenutku vrijeme (i-1) treba u tom stupcu smanjiti vrijednost tog resursa za 1.



Slika 6.33 Editor varijable RD-kapaciteti resursa

7. MODEL RASPOREDA AUTOMATSKI UPRAVLJANIH VOZILA U KONTEJNERSKIM LUKAMA

Luke predstavljaju golemu ekonomsku snagu, imaju važnu ulogu u svjetskoj privredi i međunarodnoj robnoj razmjeni. Osnovni zadatak i svrha postojanja luka jest povezivanje kopnenog i pomorskog prometa, a na to se nadovezuju mnogobrojne djelatnosti koje se obavljaju u lukama. To su mjesta za prihvat, ukrcaj ili iskrcaj tereta i putnika na brodove i na kopnena prijevozna sredstva. Postoje dva tipa teretnih brodova. Brodovi koji spadaju u prvi tip prenose velike količine tereta kao što je nafta, ugljen, žito, itd. U drugi tip spadaju brodovi koji prenose teret u željeznim kontejnerima koji su standardne veličine. Ekonomičnost takvog pomorskog prijevoza uvidio je i Malcom McLean sredinom 20. stoljeća koji je prvim kontejnerskim brodom Ideal X-om, otvorio novu stranicu povijesti. Od tada, prijevoz kontejnerskim brodovima postaje uobičajeni način prijevoza različitih vrsta proizvoda.

Glavna funkcija kontejnerskih terminala je isporučivanje kontejnera do primatelja i prihvaćanje pristiglih kontejnera, ukrcavanje kontejnera na plovila, iskrcavanje s njih te privremeno skladištenje kontejnera zbog razlike u vremenu dolaska kontejnera i vremena kada se mora proslijediti klijentu ili zbog učinkovitijeg korištenja opreme na terminalu.

Još od 1960. godine, zajedno s porastom kontejnerizacije (znači da broj proizvoda koji se prevoze u kontejnerima konstantno raste) i razvojem svjetske trgovine, grade se novi kontejnerski terminali, a oni postojeći se proširuju. Steenken et al. (2004) su obavili klasifikaciju različitih kontejnerskih terminala na svijetu kao i pregled literature [STE04]. Njihova studija je pokazala da je na Hong Kong terminalu, transport kontejnera porastao s 9 milijuna TEU[28] (pokrata od *Twenty-foot equivalent unit*) iz 1993 na 19 milijuna TEU u 2002. Istovremeno, u Singapore, kao drugoj svjetskoj luci, transport je porastao s 9 milijuna na 18 milijuna TEU. Dakle, kontejnerski terminali se stalno suočavaju s problemom porasta broja kontejnera.

Kako bi se zadovoljili zahtjevi potrošača, nužno je da se kontejneri jako brzo iskrcavaju s broda i ukrcavaju na njega. Zbog toga je važno razviti visoko sofisticirane, automatizirane kontejnerske transportne sustave, koji će omogućiti učinkovito premještanje kontejnera unutar terminala.

Logistika kontejnerskih terminala je značajno područje istraživanja, a pojam automatski upravljana vozila (engl. *Automated Guided Vehicle*, pokrata: AGV) postaje ključna riječ u publikacijama. Automatski upravljana vozila danas postaju dostupan način transporta kontejnera unutar kontejnerskih terminala. Posebice, velika pozornost je usmjerena ka određivanju rasporeda poslova za AGV. Predloženi su brojni modeli kontejnerskog terminala, kao i različite tehnike kako riješiti problem rasporeda AGV-ova.

U ovom poglavlju će se kao primjer sustava s diskretnim događajima razmatrati kontejnerski terminal luke Koper, Slovenija. Luka Koper odgovara na modernizaciju i širenje izgradnjom 150 metara obale na svom kontejnerskom terminalu te nabavkom novih dizalica. U modernizaciju luke Kopar ove će se godine uložiti 51 milijun eura. Međutim, projekt modernizacije još uvijek ne predviđa uvođenje AGV vozila. Stoga, u ovom poglavlju će se predstaviti prijedlog kojim se pokušava poboljšati proces modernizacije kontejnerskog terminala luke Koper. To znači da bi plan modernizacije predvidio i uvođenje AGV vozila,

[28] Mjerna jedinica za kapacitet ukrcaja broda.

dok bi obalne i skladišne dizalice bile ručno upravljane. Fokus istraživanja je usmjeren na određivanje rasporeda poslova za AGV-a i njegov utjecaj na ukupno vrijeme potrebno za obavljanja iskrcavanja kontejnera s broda.

U tom cilju, algoritam opisan u šestom poglavlju primijenit će se na problem rasporeda poslova za AGV vozila na kontejnerskom terminalu. Problem je kako dodijeliti AGV-ove kontejnerima tako da se minimalizira ukupni makespam i poveća protok terminala. Pri tome pod makespamom se podrazumijeva vrijeme koje brod provede na vezu u luci. Također, na automatiziranom kontejnerskom terminalu veliki poseban problem su konflikti i zaglavljena, koji mogu blokirati AGV-ove. Stoga, će se u ovom poglavlju u algoritam određivanja rasporeda poslova za AGV-ove integrirati i procedura kojom će se uspješno predvidjeti i riješiti konflikti i zaglavljenja u sustavu.

Struktura ovog poglavlja je kako slijedi. U podpoglavlju 7.1. opisat će se kontejnerski terminal, a u 7.2 automatizirani kontejnerski terminal s opisom njegovih osnovnih podsustava. Podpoglavlje 7.3 je opis osnovnih operacija na kontejnerskom terminalu, a 7.4 opis kontejnerskog terminala luke Koper na koji će se primijeniti predloženi algoritam. U podpoglavlju 7.5 prikazat će se MRF1 PM model problema promatranog sustava kao i procedura za rješavanje konflikta i zaglavljenja. U podpogljavlju 7.6 primjenit će se PMGA algoritam na promatrani sustav. U podpoglavljima 7.7 i 7.8 prezentirat će se rezultati testiranja PMGA algoritma na promatrani sustav, dok će se u podpoglavlju 7.9 testirati više-ciljni PMGA algoritam.

7.1. Sustav kontejnerskog terminala

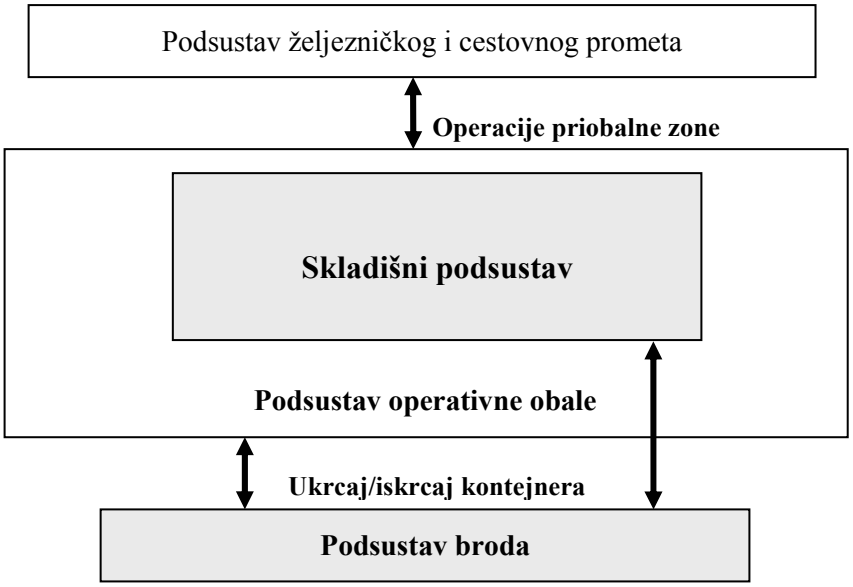
Lučki kontejnerski terminal dio je lučkog sustava, koji predstavlja posebno izgrađen i opremljen objekt namijenjen prekrcaju kontejnera izravnim ili posrednim rukovanjem između morskih brodova i kopnenih prijevoznih sredstava. Kontejnerski terminal povezuje najmanje dva prometna sustava. To su mjesta opremljena specijaliziranim prekrcajnim sredstvima i uređajima gdje se prazni ili puni kontejneri odlažu, ukrcavaju na brodove, vlakove, kamione ili se skladište.

S ekonomskog gledišta terminala, nastoji se smanjiti kašnjenje u isporuci robe brodovima, vlakovima i kamionima a to znači skraćivanje vremena pomorskog, cestovnog ili željezničkog transporta. Očito je da kontejnerski terminali imaju važnu ulogu u razmjeni unutar cestovne, željezničke i morske mreže i zbog toga su obično opremljeni modernom opremom, dodatnim transportnim sustavima i modernim informacijskim i komunikacijskim tehnologijama. Zbog toga svi poslovi koji se odnose ne promet kontejnera moraju biti optimalizirani. Dakle, učinkovitost određenog terminala ovisi o njegovoj unutarnjoj organizaciji prema strategijama planiranja i upravljanja.

Glavna djelatnost lučkog kontejnerskog terminala je prijenos i pohrana kontejnera. Danas, diljem svijeta postoje brojni terminali koji se razlikuju po izgledu, opremi, stupnju automatizacije, organizacijskom procesu, itd. Lučki kontejnerski terminal složen je sustav koji se sastoji od sljedećih podsustava [ZEN06]:

- podsustav *brod* predstavlja element na koji je usmjerena aktivnost, a obuhvaća brodove, koji se razlikuju prema vrsti i veličini,
- podsustav *operativna obala* uključuje pristran, obalne kontejnerske dizalice i krcalište (operativnu površinu namijenjenu poslovima s kontejnerima),
- podsustav *skladište* je otvoreni prostor uređen za smještaj i čuvanje različitih vrsta kontejnera do njihovog ukrcaja na brod ili utovara na kopneno vozilo,

- podsustav *prometnica* za unutarnji prijevoz čine željeznički kolosijeci,
- podsustav *rukovanja* kontejnerima obuhvaća operacije s kontejnerima na sidrištu, pristanu i skladištu,
- podsustav *organizacije* je element terminala sa zadatkom planiranja, koordinacije, nadzora i kontrole prekrcajnog procesa, administrativnog praćenja kontejnera, te fakturiranja usluga lučkoga kontejnerskog terminala.



Slika 7.1 Sustav tipičnog lučkog kontejnerskog terminala

Prostor operativne obale obuhvaća tri zone:

- prostor za prekrcaj širine 30-50 m na kojem su kontejnerske obalne dizalice i kolosijeci, koji zauzima 10% ukupne površine pristana,
- prostor za skladištenje, na koji se odnosi oko 55% ukupne površine pristana, a namijenjen je za skladištenja kontejnera (oko 0,1 ha za 1000 t kontejnera) i
- prostor za primanje i otpremu kopnenih transportnih sredstava koji je udaljen od obale i sastoji se od pristupnih cesta i manipulativnog prostora, a obuhvaća 23% ukupne površine pristana.

Većina terminala se služi opremom kojom se ručno upravlja (prikolice, viljuškari, portalni prijenosnik, dizalice, vučna vozila). Međutim, neki terminali, kao što su terminali u luci Rotterdam, su automatizirani. Na takvim terminalima transport se obavlja pomoću automatski upravljanog vozila. Čak se i proces skladištenja može automatski obavljati pomoću automatskih skladišnih dizalica (engl. *automated stacking cranes*, pokrata: ASCs).

7.2. Automatizirani kontejnerski terminali

Prvi u nizu automatiziranih kontejnerskih terminala (pokrata: AKT) pušten je u rad 1993, na ECT poluotoku u luci Rotterdam, Nizozemska. Ovaj terminal koristi automatske skladišne dizalice za odlaganje i otpremanje kontejnera na skladištu. Čak se i transport kontejnera od obalnih dizalica do skladišnih i obratno odvija pomoću automatski upravljenih vozila (AGV). Jedino se obalnim dizalicama ručno upravlja.

7.2.1. Kontejnerska skladišta

Skladišni dio prekriva najveći dio terminala i služi za privremeno odlaganje ulaznih ili izlaznih kontejnera. Ulazni kontejneri su stigli u luku brodom te se oni transportiraju u kontinentalni dio, dok su izlazni kontejneri stigli kamionima ili željeznicom za daljnji transport brodom. Veliki dio skladišnog dijela terminala podijeljen je na skladišne linije. [ZHA02]. U svakoj liniji, kontejneri su naslagani jedan do drugoga i jedan na drugi u obliku pravokutnika koji se zove blok [ZHA02]. Na krajevima linija nalaze se transportne točke gdje neko sredstvo za transport prihvaća kontejner i prenese ga ili do obale ili do kamiona, odnosno željeznice. Odlaganje i preuzimanje kontejnera sa skladišta obavlja dizalica (slika 7.2), koja je kod AKT potpuno automatizirana.

Slika 7.2 Skladišna dizalica

Unutar automatiziranog kontejnerskog terminala, kretanje kontejnera obuhvaća podizanje potrebnog kontejnera sa skladišta, prebacivanje i pozicioniranje. Ovo se sekvencijalno izvodi. Kod ulaznih kontejnera dizalica podiže kontejner s vozila, a kod izlaznih kontejnera dizalica ga odlaže na prazno transportno sredstvo koje bi moralo čekati na dizalicu.

7.2.2. Automatski upravljana vozila (AGV)

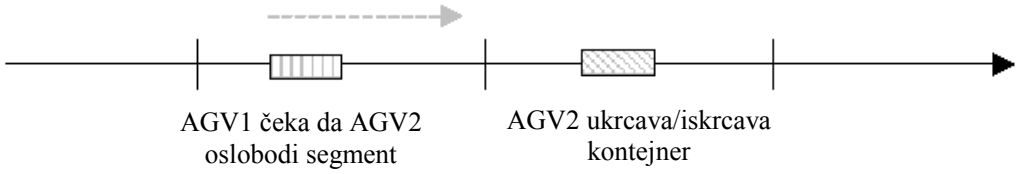
Na terminalima se koriste različita vozila za transport kontejnera unutar obalnog podsustava. Sva vozila se mogu svrstati u dvije grupe. U prvu grupu spadaju "pasivna" vozila, odnosno, ona vozila koja nisu sposobna samostalno zahvaćati kontejnere. Ukrcavanje i iskrcavanje kontejnera na (s) tih vozila obavlja se uz pomoć dizalica, obalnih ili skladišnih dizalica. Traktori s prikolicama, povezane prikolice i AGV-i, (slika 7.3a) pripadaju toj skupini.

(a) (b)

Slika 7.3 "Pasivna" transportna sredstva na terminalu: (a) automatski upravljano vozilo; (b) prikolica (izvor: www.pomorskodobro.com)

AGV-i su automatski upravljana vozila koja se pokreću pomoću elektromotora i akumulatora. AGV ima mnoštvo senzora koji centralnom mikroračunalu šalju podatke o poziciji vozila. Kontrolni sustav preuzima te podatke i odlučuje treba li se vozilo pokrenuti ili ne, prati prijenos kontejnera, usmjerava AGV, itd. Pomoću takvog sustava kontrole, u svakom trenutku poznata je točna pozicija AGV-a, brzina AGV-a i uvjeti prometa u određenoj zoni.

AGV vozila su obično duga oko 17m, široka 2.5m i mogu prenijeti kontejner od 40 ili 45ft ili dva od 20ft. AGV-ovi imaju unaprijed određenu putanju po kojoj voze. Obično ta putanja ima oblik kružne petlje. Kako bi se izbjegli sudari među vozilima putanja po kojoj vozila voze podijeljena je na manje segmente i promet se kontrolira tim segmentima. Primjerice, kada je sljedeći segment kojim treba voziti AGV blokiran od drugog vozila, prvi AGV mora čekati dok se ne oslobodi segment. To je prikazano slikom 7.4.



Slika 7.4 Primjer segmenta kontroliranih AGV-ova

Stoga bi u informacijski sustav AGV-ova trebalo implementirati i učinkovite algoritme dodjeljivanja AGV-ova, određivanja njihovih ruta kao i strategije upravljanja prometom. Kako primjena AGV sustava zahtijeva ogromna ulaganja, danas se takva vozila primjenjuju samo na terminalima ECT/Rotterdam i HHLA/Hamburg.

Transportna vozila druge grupe su sposobna samostalno podizati kontejnere. viljuškari, autodizalica i portalni prijenosnik pripadaju toj skupini (slika 7.5).

(a) (b) (c)

Slika 7.5 (a) Viljuškar; (b) Autodizalica; (c) Portalni prijenosnik

7.2.3. Obalne dizalice

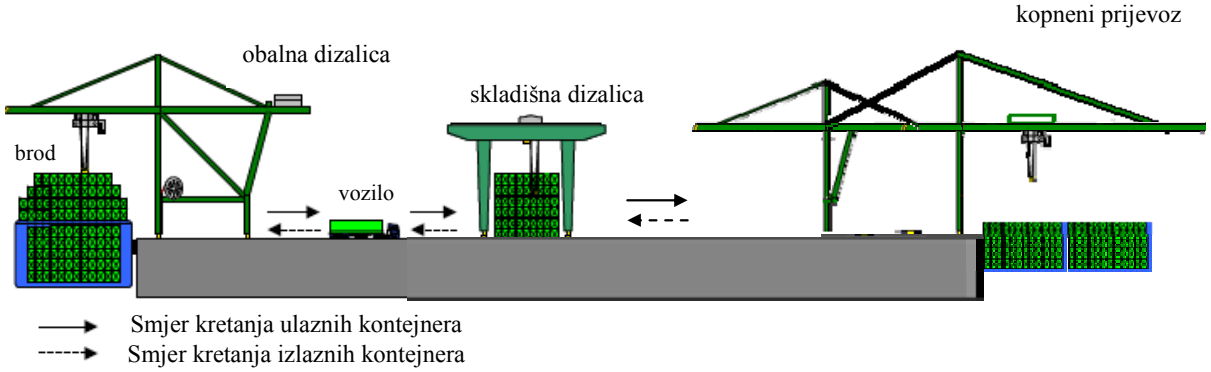
Obalne ili kontejnerske dizalice (engl. *quay* (*gantry*) *cranes*, pokrata: QC) (slika 7.6) su dizalice kojima se isključivo ručno upravlja. Dizalice ukrcavaju i iskrcavaju kontejnere sa brodova, približnom brzinom od 40 do 50 kontejnera po satu. Njihova učinkovitost je izravno povezana sa radnim učinkom AGV vozila i kontejnerskih dizalica, koji moraju osigurati da određeni kontejneri stignu u pravo vrijeme na prvi pristran. QC dizalice imaju spreader, poseban uređaj koji zakači kontejner. Spreaderi se pomiču pomoću kabela. QC dizalice rade s

kontejnerima prema unaprijed definiranoj listi, koja mora biti u skladu s planom skladištenja i rasporedom dizalica.

Slika 7.6 Lučke obalne dizalice (izvor:www.destinacije.com)

7.3. Upravljanje kontejnerima na kontejnerskom terminalu

Proces iskrcavanja ili ukrcavanja kontejnera prikazan je na slici 7.7. Nakon što brod pristigne u luku, dodijeli mu se pristan ili vez opremljen QC dizalicama. Pristan za kontejnerske brodove duži je od uobičajenih pristana jer su i kontejnerski brodovi veći. Duljina pristana ovisi o veličini brodova koji se očekuju, a prema iskustvenim podacima ne bi smjela biti manja od 260 m [VIS04].



Slika 7.7 Uobičajeni smjerovi kretanja kontejnera na terminalu

Dizalice podižu kontejnere sa palube broda ili s brodskog skladišta i premještaju ih na vozila koja prevoze kontejnere između broda i skladišta na terminalu. Skladišta se sastoje od određenog broja staza gdje se čuvaju kontejneri neko određeno vrijeme. Kontejneri koji su pristigli željezničkim ili cestovnim putem se prihvaćaju na ulazu terminala. S internom opremom kontejneri se preuzmu i prenesu do određenom mjesta na skladištu. Kad je vozilo stiglo do određenog mjesta na skladištu, skladišna dizalica (engl. *stack crane*, pokrata: SC) ili viljuškar podiže kontejner s vozila i stavlja ga na skladište. Nakon određenog vremena sa skladišta izlazni kontejneri se vozilima transportiraju do mjesta gdje se prebace na drugo prometno sredstvo, primjerice, vlak, kamion, traktor ili na drugi brod za daljnji transport.

Prije samog početka iskrcavanja/ukrcavanja kontejnera, mora se napraviti radni raspored za QC dizalice. Temeljem tog radnog rasporeda, napravi se lista kojom je određen

slijed operacija iskrcavanja ili ukrcavanja za svaki kontejner posebno. Stvarne operacije s kontejnerima na brodu se odvijaju potpuno istim redoslijedom kako je navedeno u listama.

Ako na terminalu nema privremeno odlagalište kontejnera gdje QC i SC odlažu kontejnere nakon što ga podignu s broda ili skladišta, onda AGV mora čekati ispod dizalica sve dok dizalica ne završi posao transfera kontejnera. Također, moguća kašnjenja u izvođenju poslova iskrcavanja (ukrcavanja) kontejnera rezultat su čekanja QC-ova na AGV-ove. Kako bi se poboljšala produktivnost i smanjila kašnjenja u sustavu, nužno je minimalizirati čekanja QC-ova i vrijeme vožnje za AGV-ove. Na kontejnerskom terminalu, kada se AGV-ima dodjeljuju poslovi moraju se uzeti u obzir i budući zadaci isporuke kako bi se minimalizirala kašnjenja QC-ova. Razlog tome je što su QC-ovi daleko najskuplji resursi u usporedbi i s AGV-ovima i skladišnim dizalicama. Također, QC su resursi koji uzrokuju zaglavljenja u sustavu. Stoga je cilj raspoređivanja AGV-ova taj da budu podrška obalnim dizalicama kako bi one bile u potpunosti iskorištene. Zbog toga minimalizacija kašnjenja u radu resursa mora biti većeg prioriteta nego minimalizirati vrijeme vožnje AGV-ova.

Složenost operacija na terminalu proizlazi iz međusobne povezanosti operacija i zbog različitih stohastičkih procesa [PES98] (primjerice, učestalost dolaska brodova, vrijeme kvara opreme, vrijeme potrebno za popravak kvara na opremi, broj kontejnera koji se ukrcava i iskrcava). Moguće je simulacijskim modelima potpuno opisati sustav i ti modeli su dugotrajni. S druge strane, analitički model kontejnerskog terminala se sastoji od postavljanja matematičkih modela i jednadžbi koji opisuju određenu fazu u funkcioniranju danog sustava. Glavni problem analitičkih modela kontejnerskog terminala povezan je sa činjenicom da se kod tih modela gube detalji i fleksibilnost i oni tako pojednostavnjuju stvarnu situaciju. Zato je simulacijsko modeliranje bolje od analitičkog u predstavljanju složenog okruženja kontejnerskog terminala. Također, analitički ili simulacijski modeli mogu se rabiti za uravnoteživanje opterećenja, za definiranje radnih pravila i određivanje rasporeda osoblja za kraći ili duži vremenski period.

U ovom radu razmatra se kako PMGA algoritam primijeniti u okruženju kontejnerskih terminala luke Koper. Kako bi se održala konkurentska prednost i povećala učinkovitost kontejnerskog terminala nužno je uvesti AGV-ove za unutarnji transport kontejnera i formulirati dobru strategiju rasporeda poslova za ta vozila.

Cilj rasporeda AGV-ova je poboljšati ukupnu produktivnost sustava i smanjiti kašnjenja u nizu poslova tipa pokupi-spusti kontejner, a pri tome se moraju uzeti u obzir mogući zastoji, prioriteti… Krajnji ciljevi se odnose na optimalizaciju vremena obrade, kašnjenja i/ili na minimalizaciju broja AGV koji su uključeni i održavaju protok u sustavu uz minimalizaciju ukupnog vremena vožnje svih vozila. Predlagani su mnogi algoritmi za određivanje rasporeda AGV-ova, ali su se oni odnosili na raspored AGV u proizvodnom okruženju gdje je mrežna struktura jednostavna i zahtijeva se mali broj AGV. U lučkim terminalima mrežna struktura je složenija što povlači veću skupinu, od 80 ili više AGV-ova.

7.4. Studija slučaja

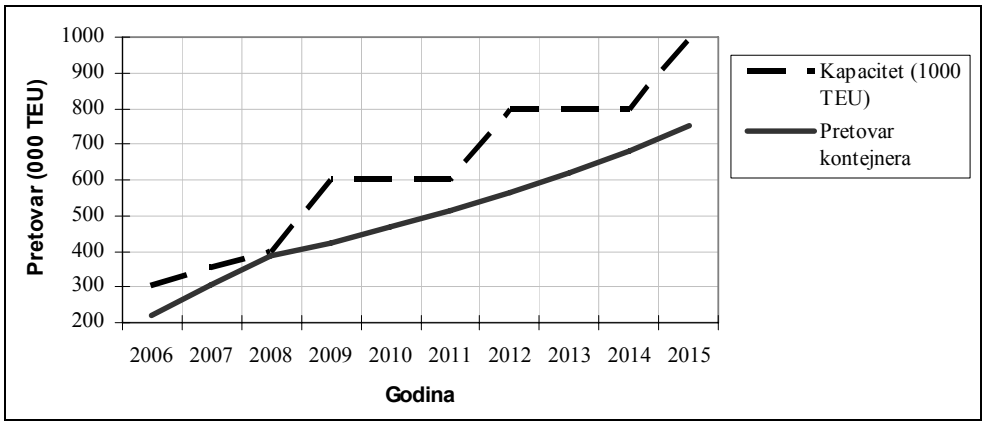
U ovom poglavlju, razmatrat će se kontejnerski terminal luke Koper (Slovenija). Luka Koper pripada lukama koja se koristi za prijevoz kontejnera za potrebe srednje i istočne Europe. U odnosu na sjeverne europske luke, morska ruta iz Kopra u zemlje Mediterana i zemlje izvan Sueza je kraća za više od 2000 nautičkih milja. U ovom trenutku, to je najveća prednost slovenske luke. Luka Koper je višenamjenska luka, opremljena i osposobljena za pretovar i skladištenje bilo koje vrste robe.

Pristupanjem Slovenije EU-a, 1. svibnju 2005, uočen je porast prometa (tablica 7.1,) na kontejnerskom terminalu luke Kopar. Postigli su 15,4 milijuna tona pretovara robe u 2007 godini i premašili su iznos pretovara u odnosu na 2006 za 10% [2]. Pretovar kontejnera iznosio je rekordnih 305.648 TEU. Stoga je uprava luke donijela stratešku odluku o proširenju terminala (npr. proširenje obalnog i skladišnog područja, uvođenje nove opreme, povećanje kapaciteta TEU i razvoj infrastrukture) kako bi udovoljila zahtjevima europskog tržišta.

Tablica 7.1 Pretovar kontejnera i kapacitet kontejnerskog terminala od 2006 do 2015 godine

Godina	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
Kapacitet (1000 TEU)	300	350	400	600	600	600	800	800	800	1000
Promet - 10% porast	218.97	305.65	385.12	423.63	465.99	512.59	563.58	620.24	682.26	750.49

Izvor: U. Horvat, E. Twrdy: "The impact of introducing sea motorways on increase of container transshipment in the Port of Koper", 11th International Conference on Traffic Science, Portorož, Slovenija, 2008 [GUD10]



Slika 7.8 Promjene pretovara kontejnera u odnosu na kapacitet kontejnerskog terminala od 2006 do 2015 [GUD10]

Luka Koper ima deset terminala: kontejnerski i ro-ro terminal (500m operativne obale), automobilski terminal (500m operativne obale i 4 ro-ro rampe za prihvat automobila), terminal za voće (427m operativne obale), terminal za opće terete (833m operativne obale), terminal za drvo (250m operativne obale), terminal za stoku (86m operativne obale), terminal za glinu (200m operativne obale), terminal rude i minerala (500m operativne obale), terminal za žitarice i krmno bilje (200m operativne obale), terminal za tekuće terete (200m operativne obale).

Kontejnerski terminal luke Koper (slika 7.9) ima 596m operativne obale i 18 ha skladišnog prostora. Od obale skladište je udaljeno je 56m, širina skladišnog prostora je 27m, a dužina 248m. Njegov trenutni godišnji kapacitet je ograničen na 182.250 TEU (engl. *Twenty-foot Equivalent Unit*), jednokratno skladištenje je ograničeno na 11,500 TEU i tri veza za kontejnerske brodove s dubinom mora od 9 do 12m, i 3 željezničke linije koje vode do terminala.

Slika 7.9 Kontejnerski terminal luke Koper [http://www.luka-kp.si/slo/fotogalerija-popup/110?tip=1]

Terminal je trenutno opremljen sa 4 kontejnerske dizalice uz obalu (slika 7.10), 4 prijenosne dizalice u skladištu i jednom prijenosnom dizalicom za ukrcaj ili iskrcaj vagona. Također, opremljen je sa 4 manipulatora i 3 viličara za prazne kontejnere i remorkere, 40 kamiona i 30 prikolica. Kontejneri se prevoze između terminala i zaleđa cestom ili željeznicom, koja je od pristrana udaljena 360m. Oba toka kontejnera, ulazni i odlazi, istovremeno se odvijaju. (Izvor: Interni materijal Luke Koper).

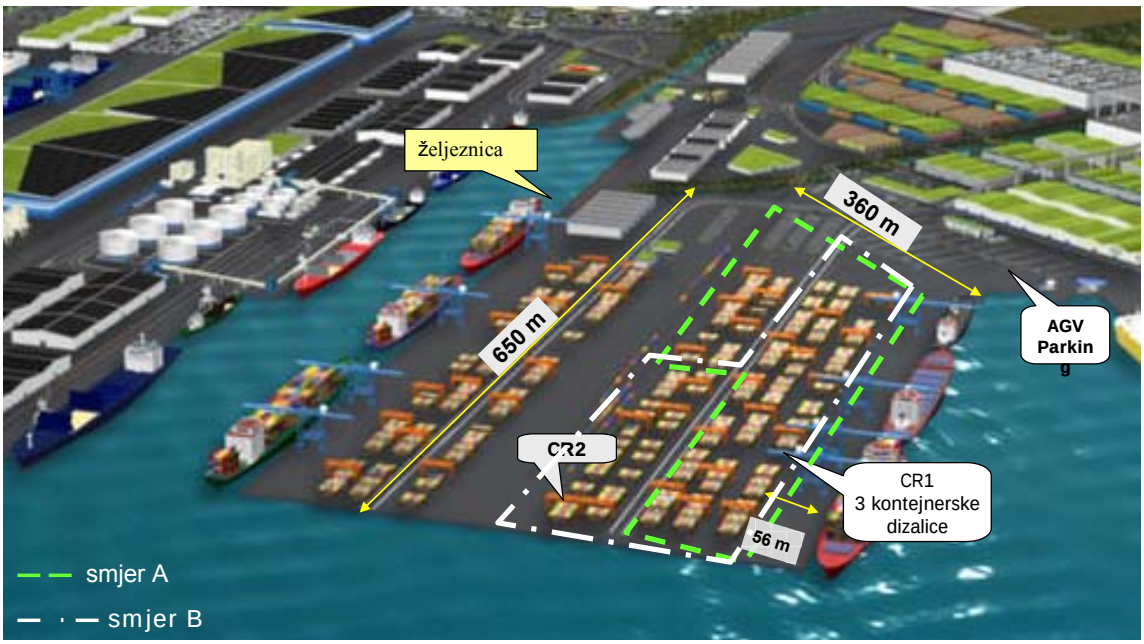
Slika 7.10 Pristran kontejnerskog terminala luke Koper

Kako bi se dodatno unaprijedile usluge terminala, trebali bi se automatizirati poslovi na terminalu. U tom smislu, u ovom poglavlju, kroz jedan primjer, iznesen je prijedlog kako uvesti AGV sustav za transport kontejnera unutar kontejnerskog terminala. Razvoj automatiziranog terminala, zahtijeva i dobre strategije raspoređivanja i nadzora AGV-ova, kako bi se mogao implementirati transport kontejnera pomoću AGV-ova. Stoga se u ovom poglavlju predlaže primjena PMGA algoritma (opisan u 6. poglavlju) kako bi se AGV-ovi rasporedili određenim kontejnerskim poslovima. Algoritam će se pokazati jako učinkovitim jer razvija optimalan raspored i sprječava moguće konflikte i zaglavljenja u sustavu transporta kontejnera. Ovakav algoritam bi mogao doprinijeti boljoj produktivnosti kontejnerskog terminala i donošenju boljih strateških odluka.

7.5. Simulacijski model rasporeda automatski upravljanih vozila

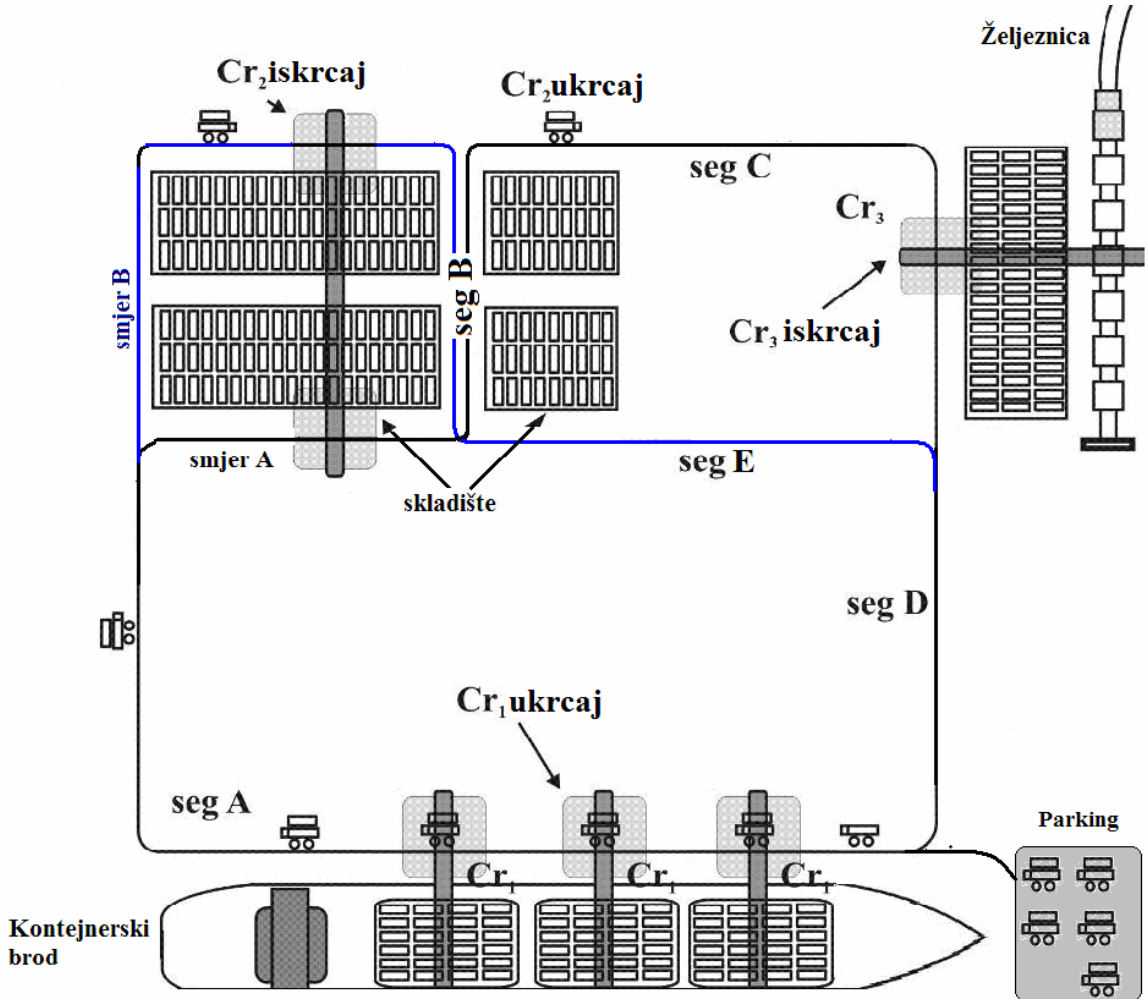
Kako bi se razvio veoma učinkovit automatiziran kontejnerski terminal, uprava je izrazila potrebu za uvođenjem AGV-ova kao i za potrebnom strategijom dinamičkog raspoređivanja AGV-ova unutar terminalskog prostora. U tom cilju, u ovom poglavlju opisat će se prijedlog modela kako uvesti AGV-ove za prijevoz kontejnera na kontejnerskom terminalu luke Koper (slika 7.11). Matrični model MRF_1 Petrijeve mreže integriran sa GA, opisan u podpoglavlju 6.5, primijeniti će se na određivanje rasporeda poslova pri iskrcaju kontejnera s broda i transport do skladišta na terminalu luke Koper. Svaki posao se odnosi na ukrcavanje kontejnera na AGV, kretanje AGV do odredišta kontejnera i iskrcavanje kontejnera s AGV. U ovoj studiji, sljedeće pretpostavke će se uvažavati:

- Svaki AGV može opsluživati više od jedne obalne dizalice;
- Sva vozila su ista i po veličini, po funkcionalnosti i mogu prevoziti samo jedan kontejner u određenom trenutku;
- Točna lokacija i smjer kretanja AGV-a odredit će se u algoritmu i ovisi o shemi rasporeda;
- Kretanje AGV modelirat će se s konstantnom brzinom, bez ikakvog ubrzanja ili usporenja;
- Brzina AGV i vrijeme koje je potrebno da AGV dođe iz jedne točke u drugu je dinamičko i odredit će se iz sheme rasporeda;
- Vrijeme koje je potrebno dizalici da ukrca ili iskrca kontejner na(sa) AGV-a je zadano. Ovo vrijeme se mora uzeti u obzir i za sve vrste dizalica jer ono utječe i na vrijeme čekanja AGV-ova;
- Uzet će se u obzir i prosječno vrijeme koje protekne od trenutka kada QC položi kontejner na AGV do trenutka kada podigne sljedeći kontejner i zadat će se kao vrijeme setiranja resursa.



Slika 7.11 Računalna simulacija kontejnerskog terminala (izvor: Luka Koper)

Slika 7.12 prikazuje prijedlog automatizacije razmatranog kontejnerskog terminala tako da se uvedu AGV vozila za transport kontejnera. Ova vozila bi, po unaprijed definiranoj stazi, prevozila kontejnere. Kako bi se povećala učinkovitost AGV-ova, odnosno smanjilo vrijeme čekanja, predviđeno je da uz glavnu stazu kojom se kreću AGV-i, postoji i alternativna. To je moguće uvesti jer se radi o relativno jednostavnom terminalu na kojem bi bilo uključeno 20-tak AGV-ova. Naime, postojale bi dvije kružne staze. Prva, staza A, bi bila od obalnih dizalica, do dizalice na skladištu, željeznice i natrag do obalnog dijela. Druga staza B, bi bila od obalnih dizalica do skladišta i natrag do veza. Dakle kako bi se smanjilo čekanje staza B bi bila alternativna ruta. Staze se sastoje od nekoliko segmenata. Segmentom A vozila prevoze kontejnere od obalnih dizalica do određenog mjesta u skladištu gdje se kontejneri odlažu. Nakon toga, segmentom B, AGV se premjesti na poziciju gdje će ukrcati kontejner sa skladišta i segmentom C ga transportira do željeznice. Nakon iskrcavanja kontejnera na željeznici prazno vozilo segmentom D ide do mjesta gdje čeka sljedeći posao koji treba obaviti. Ako AGV vozilo nije ukrcalo kontejner koji treba prebaciti do željeznice, onda nakon iskrcavanja kontejnera na skladištu, prazno vozilo ide segmentom E do parkinga.



Slika 7.12 Sustav transporta kontejnera

Kada se brod veže za obalu terminala moguće su dvije vrste poslova koje je potrebno obaviti. To su ili iskrcaj kontejnera s broda ili ukrcaj na brod. Zbog složenosti sustava u ovom poglavlju razmatrat će se samo proces istovara kontejnera s broda.

Sustav sadrži 3 obalne dizalice (Cr_1) koje obavljaju iskrcaj kontejnera s broda i postavljaju ga na AGV, 1 skladišne dizalice (Cr_2) koja iskrcava kontejner s AGVa i stavlja ga

na određenu poziciju na skladištu i 1 mobilne dizalice uz željeznicu koja iskrcava kontejner sa AGVa i ukrcava ga na vlak.

Kada obalna dizalica podigne kontejner s broda mora ga prebaciti na AGV koji je dodijeljen dizalici za taj transport. Kako ne bi bilo kašnjenja, dodijeljeno AGV vozilo bi trebalo čekati na dizalicu na vezu. Čim se AGV dodijeli dizalici, odmah mu se odredi ruta, smjer A ili smjer B. Ako je vozilo slobodno onda ono čeka na mjestu označenom kao parking. Ako nema slobodnih vozila, onda se dizalici dodjeljuje novo vozilo koje je u spremištu.

Staze kojima se kreću vozila gledat će se kao jedan projekt koji se treba obaviti. I projekti su različitog prioriteta. Svaki projekt se sastoji od određenog broja poslova, koji se trebaju obaviti u određenom redoslijedu. Svaki posao se obavlja na određenoj vrsti resursa i traje određeno vrijeme. Segmenti staze i dizalice predstavljaju resurse sustava.

Integraciju MRF_1 Petrijeve mreže i GA primijenit će se na određivanje rasporeda poslova AGV-a. Kao primjer uzet će se kontejnerski brod SAFMARINE KOMATI (IMO: 9342176) koji je bio 2.veljače 2010. na vezu u luci Koper (slika 7.13). Brod je duljine 300 m i kapaciteta 6160 TEU. Predviđen je i obavljen ukrcaj i iskrcaj 1000 kontejnera.

Slika 7.13 Kontejnerski brod usidren u luci Koper (izvor: luka Koper)

7.5.1. Modeliranje sustava s MRF1 klasom Petrijeve mreže

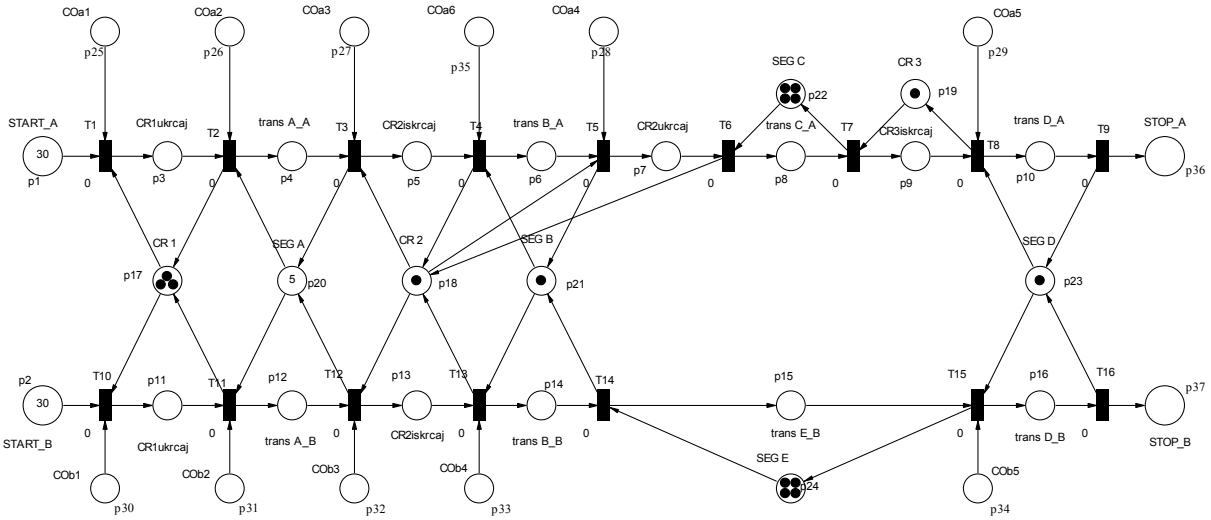
Slika 7.14 prikazuje model sustava izgrađen Petrijevom mrežom. Mjesta $p_3 - p_{24}$ pripadaju MRF_1 Petrijevoj mreži, a predstavljaju resurse i poslove. U tablici 7.2 je prikazano značenje tih mjesta.

Dakle, skup poslova u sustavu je:

$$J=\{CR1ukrcaj\_A\ transA\_A\ CR2iskrcaj\_A\ transB\_A\ CR2ukrcaj\_A\ transC\_A\ CR3iskrcaj\_A\ transD\_A\ CR1ukrcaj\_B\ transA\_B\ CR2iskrcaj\_B\ transB\_B\ transE\_B\ transD\_B\}, \quad (7\text{-}1)$$

a skup resursa je:

$$R=\{\ CR1\ CR2\ CR3\ SEG\_A\ SEG\_B\ SEG\_C\ SEG\_D\ SEG\_E\ \}. \quad (7\text{-}2)$$



Slika 7.14 MRF_1 PM model kontejnerskog terminala

Tablica 7.2 Značenje mjesta MRF_1 na Petrijevoj mreži sa slike 7.14

ZNAČENJE MJESTA / POSLOVI		Pridruženo vrijem (min)
	Smjer A	
p_3	Kontejnerska dizalica CR_1 ukrcava kontejner, koji je podignut sa broda, na AGV	1
p_4	Segmentom A u smjeru A, AGV vozi kontejner do skladišta	1.5
p_5	Skladišna dizalica CR_2 podiže kontejner sa AGV i odlaže ga na skladište	2
P_6	Prazan AGV vozi segmentom B do mjesta u skladištu gdje će ukrcati kontejner za vlak	0
p_7	CR_2 podiže kontejner i ukrcava ga na AGV	2
p_8	Puni AGV vozi segmentom C do željeznice	0
p_9	Dizalica CR_3 podiže kontejner s AGV-a	2
p_{10}	Prazan AGV vozi segmentom D do parkinga2	0
	Smjer B	
p_{11}	Kontejnerska dizalica CR_1 ukrcava kontejner, koji je podignut sa broda, na AGV koji ide smjerom B	1
p_{12}	Segmentom A AGV vozi kontejner do skladišta	1.5
p_{13}	Skladišna dizalica CR_2 podiže kontejner sa AGV i odlaže ga na skladište	2
p_{14}	Prazan AGV vozi segmentom B	0
p_{15}	Prazan AGV vozi segmentom E	0
p_{16}	Prazan AGV vozi segmentom D do parkinga2	0
ZNAČENJE MJESTA / RESURSI		Vrijeme setiranja
p_{17}	CR_1 - Obalna dizalica za kontejnere je raspoloživa	1
p_{18}	CR_2 – Skladišna dizalica je raspoloživa	1
p_{19}	CR_3 – Dizalica za željeznicu	1
p_{20}	Segment A je slobodan	0
p_{21}	Segment B je slobodan	0
p_{22}	Segment C je slobodan	0
p_{23}	Segment D je slobodan	0
p_{24}	Segment E je slobodan	0

Resursi su ograničenog kapaciteta RD(i), $(i = 1,\ldots,8)$. Kapaciteti resursa 1-3 su određeni brojem dizalica koje mogu opsluživati skup AGV-ova, a kapaciteti resursa 4-8 odnose se na broj AGV-ova koji mogu voziti određenim segmentom. Na početku simulacije kapaciteti resursa su zadani kako slijedi

RD(1)=3 RD(2)=1 RD(3)=2 RD(4)=5 RD(5)=1 RD(6)=4 RD(7)=1 RD(7)=4

kapaciteti dizalica CR1, CR2, CR3 *kapaciteti segmenata* SEG_A, SEG_B, SEG_C, SEG_D SEG_E

Mjesta p_1, p_2 predstavljaju skup ulaznih mjesta $\{$START_A, START_B$\}$, a skup izlaznih mjesta je $\{$STOP_A, STOP_B$\}$. Ulazna mjesta označavaju polazišne lokacije gdje obalne dizalice podižu kontejnere i prebacuju ih na vozila. Izlazna mjesta su odredišta AGV vozila. Broj 30 u mjestu p_1 označava da će 30 AGV-ova obaviti transport kontejnera s broda u smjeru A, a 30 u smjeru B. Kontrolna mjesta $COa_1 - COa_6$, $COb_1 - COb_5$ pripadaju kontroleru i osiguravaju da se poslovi u mreži izvode bez konflikta i bez zaglavljenja.

U skladu s opisanim matričnim modelom MRF_1 PM, za ovaj primjer potrebne su matrice prikazane su na slici 7.15.

```
Fv=[0 0 0 0 0 0 0 0 0 0 0 0 0 0;1 0 0 0 0 0 0 0 0 0 0 0 0 0; 0 1 0 0 0 0 0
0 0 0 0 0 0 0; 0 0 1 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 1 0 0 0 0 0 0 0 0 0 0; 0
0 0 0 1 0 0 0 0 0 0 0 0 0; 0 0 0 0 0 1 0 0 0 0 0 0 0 0; 0 0 0 0 0 0 1 0 0 0
0 0 0 0; 0 0 0 0 0 0 0 0 1 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 0 0 0 0; 0 0 0 0 0
0 0 0 1 0 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0; 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0; 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0; 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0; 0 0 0 0 0 0 0
0 0 0 0 0 0 1];
Fr=[1,0,0,0,0,0,0,0; 0,0,0,1,0,0,0,0; 0,1,0,0,0,0,0,0; 0,0,0,0,1,0,0,0;
    0,1,0,0,0,0,0,0; 0,0,0,0,0,0,1,0; 0,0,1,0,0,0,0,0; 0,0,0,0,0,0,0,1,0;
    0,0,0,0,0,0,0,0; 1,0,0,0,0,0,0,0; 0,0,0,1,0,0,0,0; 0,1,0,0,0,0,0,0;
    0,0,0,0,1,0,0,0; 0,0,0,0,0,0,0,1; 0,0,0,0,0,0,1,0; 0,0,0,0,0,0,0,0];
Fud=[1,0,0,0,0,0,0,0;0,1,0,0,0,0,0,0;0,0,1,0,0,0,0,0;0,0,0,0,0,0,0,0;
     0,0,0,1,0,0,0,0;0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0;
     0,0,0,0,0,0,0,0;0,0,0,0,1,0,0,0;0,0,0,0,0,0,1,0;0,0,0,0,0,0,0,1,0;
     0,0,0,0,0,0,0,1;0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0,0]
Sv=[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0;0,0,1,0
,0,0,0,0,0,0,0,0,0,0,0,0;0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0;0,0,0,0,1,0,0,0,0,
0,0,0,0,0,0,0,0;0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0
,0;0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0;0,0,0,0,
0,0,0,0,0,0,0,1,0,0,0,0,0;0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0;0,0,0,0,0,0,0,0,0,0
,0,0,1,0,0,0;0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0;0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,
0;];
Sr=[0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0;0,0,0,1,0,1,0,0,0,0,0,0,1,0,0,0;0,0,0,0
,0,0,0,1,0,0,0,0,0,0,0,0;0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0;0,0,0,0,1,0,0,0,0,
0,0,0,0,1,0,0;0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0,1,0,0,0,0,0,0
,1;0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0;];
Sud=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;
     0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0;0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
Fu=[10 0 0 0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]';
Fy=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]';
Su=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];
Sy=[0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1];
```

Slika 7.15 Matrice Petrijeve mreže sa slike 7.14

7.5.2. Vremenska simulacija

Svaki resurs zahtijeva određeno vrijeme (različito od nule) potrebno za izvođenje poslova. Vremena obavljanja poslova koja su dodijeljena mjestima p_3–p_{16} PM mreže (slika 7.14) zadana su vektorom *PNtimes* :

```
PNtimes = [0 0 1 1.5 2 0 2 0 2 0 1 1.5 2 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0]';
```

Mjestima p_6, p_8, p_{10}, p_{14}, p_{15}, p_{16} početna vremena postavljena su na nulu. Naime, kako vremena u tim mjestima odgovaraju vremenima koja su potrebna da vozilo prođe određeni segment na terminalu i kako to vrijeme ovisi o udaljenostima na terminalu (matrica D) te vozi li AGV prazan ili pun jer su mu brzine različite, onda se za ta mjesta vrijeme računa kako slijedi:

```
D=  [0   445   495   545   866   893     0     0     0
     0     0    50   100   421     0     0     0     0
     0     0     0    50   371     0     0     0     0
     0     0     0     0   321     0     0     0     0
     0     0     0     0     0    27     0     0     0
     0     0     0     0     0     0   680   258     0
     0     0     0     0     0     0     0   260   360
     0     0     0     0     0     0     0     0   100
     0    70     0     0     0     0     0     0     0];

ve=330;   %brzina AGV kad je prazan  (m/min)
vf=180;   %brzina AGV kad je pun(m/min)

PNtimes(6,1)=(D(5,6)+10)/ve;PNtimes(8,1)=D(6,7)/vf;PNtimes(10,1)=D(7,9)/ve;
PNtimes(14,1)=D(5,6)/ve;PNtimes(15,1)=D(6,8)/ve;PNtimes(16,1)=D(8,9)/ve;
```

Mjesta p_4 i p_{12} predstavljaju transport AGV vozila segmentom A, odnosno AGV vozilo je preuzelo kontejner od obalne dizalice i transportira ga do skladišta. Udaljenost koju AGV mora prijeći ovisi o mjestu gdje se nalazi dizalica, koja je ukrcala kontejner na vozilo, pa se i vremena za mjesta p_4 i p_{12} dinamički određuju kako slijedi:

```
PNtimes(4,1)=D(d2+1,5)/vf; PNtimes(12,1)=D(d2+1,5)/vf;
```

gdje je d_2 broj dizalice koja je dodijeljena.

7.5.3. Konflikti i zaglavljenja u sustavu

Konflikt se dešava kada se u jednom trenutku ostvare uvjeti za pokretanje dva ili više poslova koji zahtijevaju isti resurs. Strategija određivanja rute AGV vozila trebala bi uključivati rješenja za slučaj konflikta za istim resursom, kao i sprječavanje zaglavljenja. Na obalnom dijelu terminala, zaglavljenje se može definirati kao stanje kada je automatski sustav u nemogućnosti nastaviti s izvođenjem poslova.
U AGV sustavu dizalice i segmenti kojima prometuju AGV-ovi su zajednički resursi i dijele ih sva vozila na terminalu. Također, u određenom trenutku vozilo može voziti samo jednim segmentom.

Konflikti se pojavljuju kada AGV-ovi koji voze u različitim rutama pokušavaju zauzeti isti resurs (dizalicu ili segment) $\{p_{17} - p_{24}\}$. U ovom primjeru mogući konflikti su između prijelaza t_1, t_{10} ; t_2, t_{11} ; t_4, t_{13} i t_8, t_{15} (ne mogu biti istovremeno omogućena oba prijelaza). Međutim, moguć je konflikt između tri prijelaza t_3, t_5, t_{12}, kada tri vozila traže istu dizalicu (resurs p_{18}). Treba omogućiti samo jedan prijelaz s jednakom vjerojatnošću. Na slici

7.16 prikazan je programski kod procedure koja ispituje postojanje konflikta u sustavu i rješava ih.

```
% provjera da li ima negativnog broja tokena u p17
if (conflict(17) < 0)       % Rutina po slucajnom izboru omogucuje okidanje
     jednog od prijelaza t1 ili t10, ukoliko su ovi prijelazi u konfliktu
          if (fix(2*rand(1))==0)
              mf(COa1)=1; mf(COb1)=0;
          else
              mf(COa1)=0; mf(COb1)=1;    end
 end
% provjera da li ima negativnog broja oznaka u p18
 if (conflict(18) < 0);  % Ova rutina po slucajnom izboru omogucuje okidanje
  jednog od prijelaza t3,t5 ili t12, ukoliko su ovi prijelazi u konfliktu
    if (x(3)==1 && x(5)==1 && x(12)==0)
          sb=1;
     else
          if (x(3)==1 && x(5)==0 && x(12)==0)
               sb=0;
          else
           if (x(3)==0 && x(5)==0 && x(12)==1)
               sb=2;
            else
             if (x(3)==0 && x(5)==1 && x(12)==1)
                sb=3;
             else
                sb=4;  end
            end
          end
    end
     switch (sb)
          case 0
               mf(COa3)=0; mf(COa4)=1;mf(COb3)=0;
          case 1
               mf(COa3)=0; mf(COa4)=1;mf(COb3)=0;
          case 2
               mf(COa3)=0; mf(COa4)=0;mf(COb3)=1;
          case 3
               mf(COa3)=0; mf(COa4)=1;mf(COb3)=0;
          otherwise
               if (fix(2*rand(1))==0)
                   mf(COa3)=1; mf(COb3)=0;mf(COa4)=0;
               else
                   mf(COa3)=0; mf(COb3)=1;mf(COa4)=0;
               end
       end
  end
  % provjera da li ima negativnog broja tokena u p20
   if (conflict(20) < 0)
        if (fix(2*rand(1))==0)
                  mf(COa2)=1; mf(COb2)=0;
        else
                mf(COa2)=0; mf(COb2)=1;   end
  end
  % provjera da li ima negativnog broja tokena u (21)
           if (conflict(21) < 0)
              if (fix(2*rand(1))==0)
                  mf(COa6)=1; mf(COb4)=0;
              else
                  mf(COa6)=0; mf(COb4)=1;
              end
```

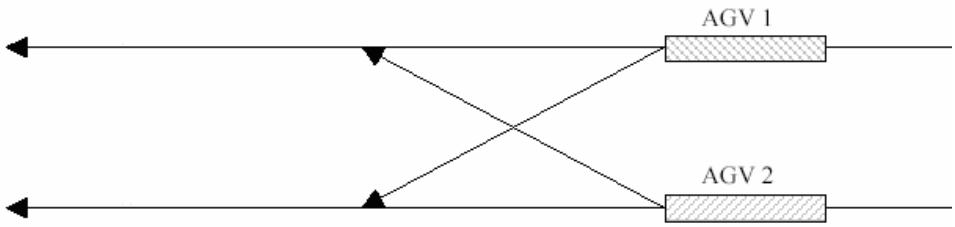
```
57.              end
58.    % provjera da li ima negativnog broja tokena u p23
59.       if (conflict(23) < 0)
60.          if (fix(2*rand(1))==0)
61.                mf(COa5)=1; mf(COb5)=0;
62.           else
63.               mf(COa5)=0; mf(COb5)=1;   end
64.    end
```

Slika 7.16 Programski kod procedure koja rješava konflikte u sustavu

Drugi problem je kako omogućiti prometovanje AGV-ova bez zaglavljenja u sustavu. Obzirom na izgled terminala i rute kojima voze AGV-ovi, moguća su različita zaglavljenja u sustavu koja ovise o načinu upravljanja AGV-ovima. Na kontejnerskom terminalu najčešće se pojavljuju dvije vrste zaglavljenja koja su opisana u nastavku.

Zaglavljenje zbog križanja staza

Prvi tip zaglavljenja koji se može pojaviti je slučaj kada se za dva vozila njihove staze križaju. Primjer je prikazan slikom 7.17, kada AGV1 želi prijeći u stazu kojom vozi AGV2 i AGV2 želi prijeći u stazu kojom vozi AGV1.

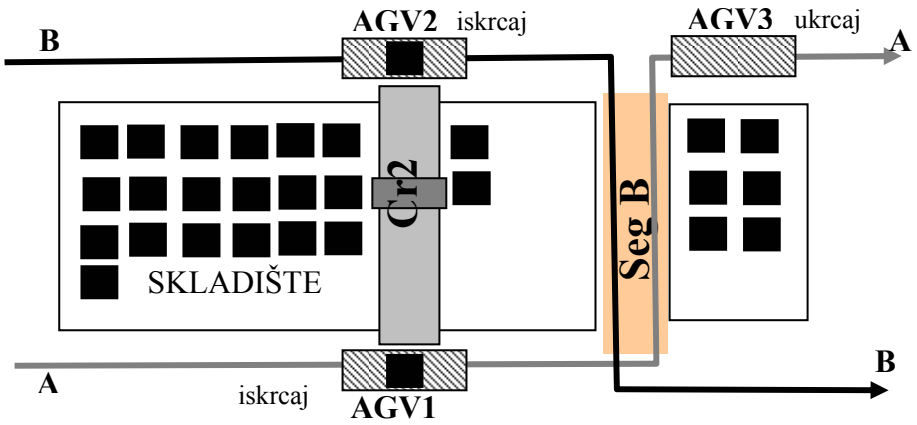


Slika 7.17 Prikaz zaglavljenja u slučaju križanja staza kojima voze dva vozila

Tada se može dogoditi da AGV1 ne može zaokrenuti u drugu liniju ako se AGV2 nalazi preblizu križanja, odnosno, AGV1 se može sudariti s AGV2 u momentu skretanja. Kako bi se izbjegao sudar, na terminalu luke Rotterdam, upravljanje prometom AGV-ova riješeno je pomoću navigacijskog sustava u vozilima. Međutim, može se dogoditi situacija da će navigacijski sustav na AGV1 dati nalog vozilu da se zaustavi. Isto se može dogoditi kod AGV2. U tom slučaju svako vozilo čeka da se pomakne ono drugo, što rezultira da se niti jedno vozilo ne miče. U modelu sa slike 7.14 sudar se može dogoditi ako dva vozila, koji se kreću različitim smjerom, istovremeno skreću u segment B (slika 7.18). U PM modelu koji se predlaže u ovom radu problem sudara je riješen tako da se kapacitet segmenta B ograniči na 1. Dakle, dva vozila ne mogu istovremeno zauzeti segment B.

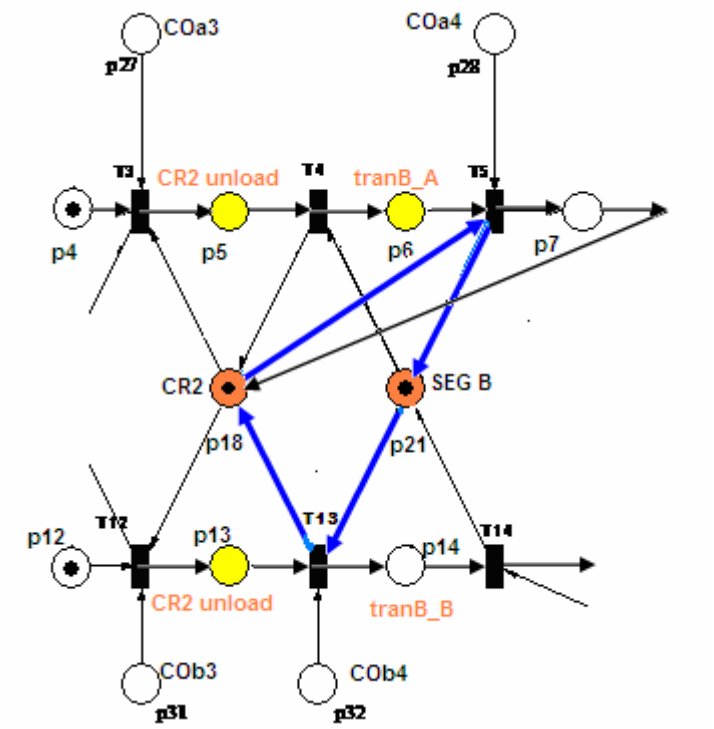
Kružno zaglavljenje

Druga vrsta zaglavljenja koje se može pojaviti u AGV sustavu sa slike 7.14 je kružno čekanje prikazano slikom 7.18. U šestom poglavlju je opisano i na primjeru pokazano da kružno čekanje, a time i pripadajući kritični podsustavi mogu stvarati probleme koji utječu na stabilnost sustava. Kružno čekanje se javlja kada vozila čekaju jedan na drugog da oslobode zajedničke resurse. Dakle, par $C = \{CR2, SEG\_B\}$={dizalica Cr2, segment B} je kružno čekanje.



Slika 7.18 AGV vozila u kružnom čekanju

Kružno čekanje se izravno može odrediti iz grafa PM tog sustava (slika 7.19). Vidljivo je da su mjesta p18 i p21 zajednički resursi i svako mjesto p18 ima jednu oznaku. Oznaka iz mjesta p18 može otići u p5, p7, p13. Ako je omogućen prijelaz t5 onda oznaka prelazi u mjesto p7 i u mjesto p21. Oznaka iz p21 ne može krenuti prema mjestu p14 sve dok se ne pojavi oznaka u mjestu p13, a kada se pojavi okidanjem prijelaza t13 oznaka bi trebala doći u p18 i time je krug zatvoren. Mjesta čekaju jedan na drugog i sustav se nalazi u kružnom zaglavljenju. Dakle, resursi u kružnom čekanju su CR2 i SEG_B.



Slika 7.19 Kružno čekanje u PM modelu sa slike 7.14

Iz kružnog čekanja, sustav se može naći u stanju kružnog zaglavljenja. Naime, pri upravljanju AGV sustavom veoma je važno predvidjeti postojanje kružnog čekanja, jer ono uzrokuje čekanje AGV-ova na dizalice. To je nedopustivo jer se javlja zagušenje u sustavu, odnosno kašnjenja u obavljanju poslova. Kako bi se izbjegla takva situacija, na MRF_1 sa slike 7.14 primijenit će se matrična metoda izbjegavanja zaglavljenja. Ista metoda je primijenjena i u točki 6.7.1.2, a izvorno je razvijena za FMS [LEE07].

Potrebno je odrediti poslove koji pripadaju kružnom čekanju i kritični podsustav.

Poslovi koji pripadaju kružnom čekanju čine skup

$$J(CW)=\{p5,p6,p7,p13,p14\}=\{CR2iskrcaj\_A, transB\_A, CR2ukrcaj, CR2iskrcaj\_B, transB\_B\}.$$

Kritični podsustav sačinjavaju svi poslovi koji trebaju ili resurs CR2 (slijedi nakon SEG_B) ili resurs SEG_B (slijedi nakon CR2):

$$J_0(CW)=\{p5,p6,p13\}=\{CR2iskrcaj\_A, transB\_A, CR2iskrcaj\_B\}.$$

Iz kritičnog podsustava odredi se kružno zaglavljenje. Kružno zaglavljenje u MRF1 nastaje ako je broj oznaka u kritičnom podsustavu jednak ukupnom broju resursa u kružnom čekanju $\left[m(J_0(CW))=m_0(CW)\right]$, gdje je $m_0(CW)$ inicijalni broj oznaka u kružnom čekanju. Dakle, ovim uvjetom se sprječava situacija u sustavu da su i dizalica i segment istovremeno zauzeti.

Slika 7.20 prikazuje programski kod procedure za izbjegavanje kružnog zaglavljenja u sustavu. Kako bi se izbjeglo kružno zaglavljenje (ili zagušenje) u MRF_1 PM sa slike 7.14, ukupan broj poslova koji se izvršava u kritičnom podsustavu $(J_0(CW))$ trebao bi se ograničiti da bude ili manji ili jednak od $m_0(CW)-1$ (prema teoremu 1 iz dodatka). Ovo pravilo će se primijeniti tako da se promatra stanje u kojem će PM naći okidanjem omogućenih prijelaza (*linije* 2,3, slika 7.20). Nakon što je detektirano postojanje zaglavljenja (*linija* 4), procedura mora zaustaviti ulazak oznaka u kritični podsustav, a omogućiti izlaz oznaka iz njega. To je riješeno dodavanjem i oduzimanjem oznaka kontrolnih mjesta COa3, COa4, COb3, COb4 (*linije* 6-9), što izvršava nadzornik.

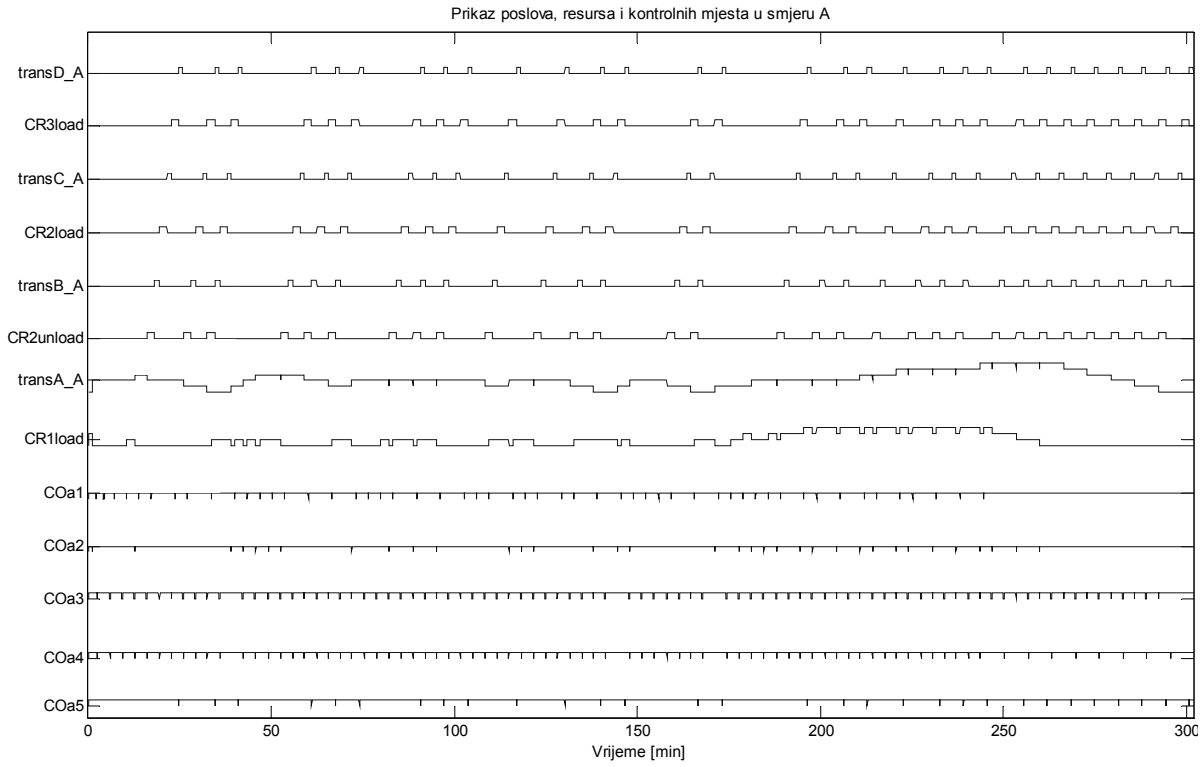
```
1. % ispitivanje kriticnog podsustava
2.  x=multoa(not(F),not(not(mf))); % odrediti koji su prijelazi omoguceni
3.  mtest=(mf+mi)+(M'*x); % odrediti stanje mtest nakon okidanja prijelaza x
4.      if (mtest(5)+ mtest(6)+ mtest(13) > (m0(18)+ m0(21)-1))
5.   % omoguciti izlaz iz kriticnog podsustava i zabraniti ulaz u njega
6.                    mf(COa3)=0;
7.                    mf(COb3)=0;
8.                    mf(COa4)=1;
9.                    mf(COb4)=1;          end
```

Slika 7.20 Procedura za predviđanje i izbjegavanje kružnog zaglavljenja

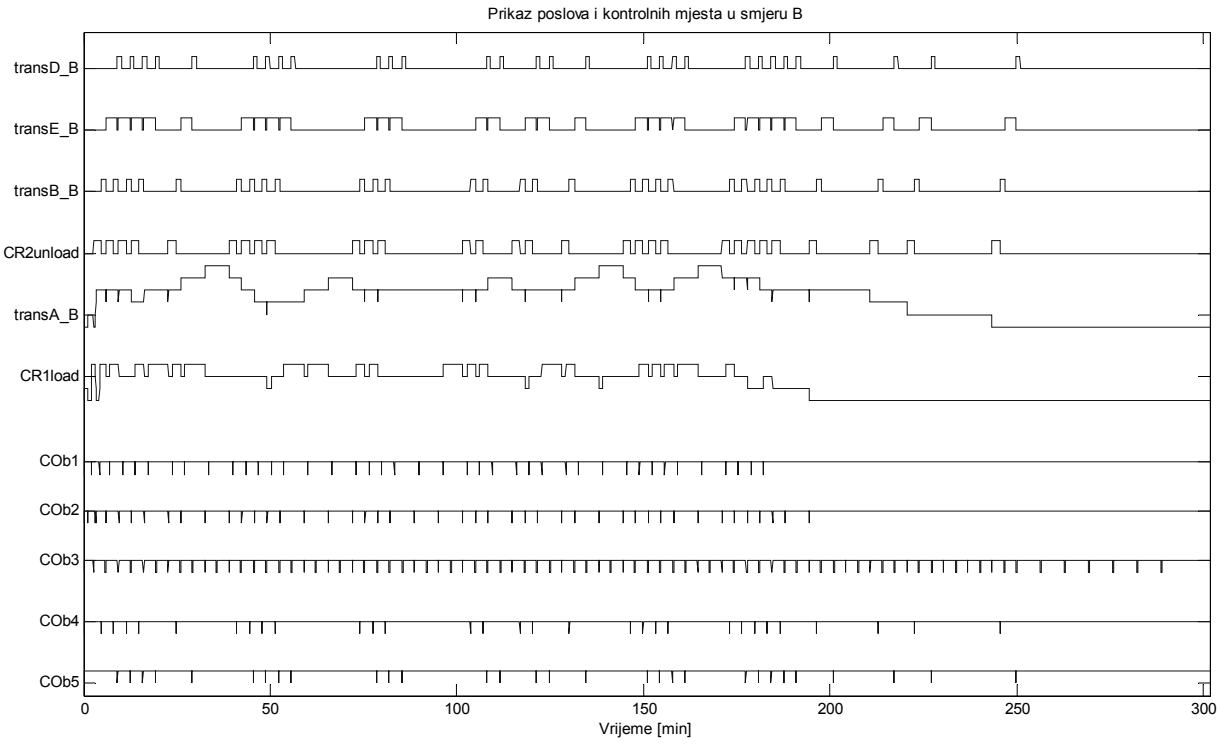
7.5.4. Rezultati simulacije

Programski kod koji simulira PM sa slike 7.14 naveden je u prilogu u datoteci '*AGV.m*', a rezultati simulacije su prikazani na slikama 7.21 i 7.22. Rezultati prikazuju evoluciju promatranog sustava sa diskretnim događajima kroz vrijeme. Sa slike 7.22 je vidljivo da su obavljeni svi poslovi u sustavu jer je na početku u početnom mjestu START_A bilo 30 kontejnera i svi su stigli do izlaznog mjesta STOP_A, a isto vrijedi i za smjer B. Vidljivo je da posljednji kontejner nakon 301.6 minuta dolazi na odredište, odnosno, svi poslovi su tada završeni. Primijenjena procedura kontrole osigurava odvijanje procesa bez konflikta i zaglavljenja u sustavu, što se vidi sa slike 7.21. Osam gornjih dijagrama na slici 7.21(a) prikazuju broj AGV-ova u mjestima p_3-p_{10} koja prikazuju poslove sustava u smjeru A, dok na slici 7.21(b) prikazuju broj AGV-ova u mjestima $p_{11}-p_{16}$ koja prikazuju poslove sustava u smjeru B. Donjih pet dijagrama (COa1-Coa,5 dijagram (a), COb1-Cob5, dijagram

(b)) prikazuju kontrolna mjesta $p_{25} - p_{34}$. Kontrolna mjesta su "semafori" pomoću kojih je rješava problem dodijele zajedničkog resursa.

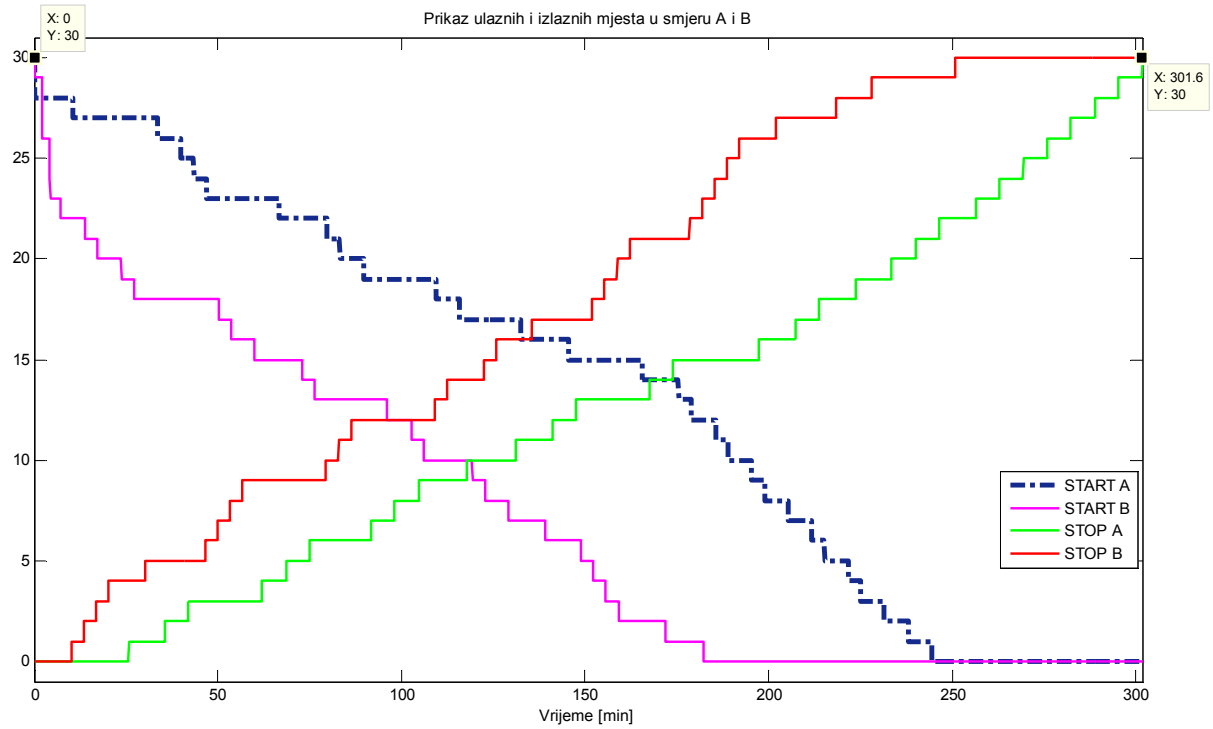


(a)



(b)

Slika 7.21 Rezultati simulacije: Broj kontejnera u mjestima koji predstavljaju poslove i broj oznaka u kontrolnim mjestima; (a) Smjer A; (b) Smjer B



Slika 7.22 Rezultati simulacije: vremenski tok broja kontejnera od ulaznog mjesta do izlaznog

7.6. Određivanje rasporeda automatski upravljanih vozila

Fokus ovog poglavlja je primijeniti PMGA algoritam kako bi se odredio raspored AGV-ova. Cilj određivanja rasporeda AGV vozila je dodijeliti poslove skupu vozila i odrediti vrijeme početka izvođenja svakog posla za svako vozilo. Svaki posao se odnosi na transport kontejnera od lokacije, gdje je ukrcan na vozilo, do lokacije gdje će se iskrcati s vozila. Svakom AGV vozilu može se dodijeliti samo jedan posao u jednom trenutku. Nakon što je AGV završio s jednim poslom trebao bi započeti sa sljedećim. skratilo vrijeme koje kontejnerski brod provede u luci uz maksimalizaciju produktivnosti AGV-ova i obalnih dizalica. Naime, kako bi se održala konkurentska prednost i povećala učinkovitost kontejnerskog terminala nužno je odrediti odgovarajući broj AGV i odrediti dobre strategije rasporeda za ta vozila

Određivanje rasporeda i upravljanje AGV vozilima na kontejnerskom terminalu je prilično kompliciranо, prvenstveno, zbog zaglavljenja i konflikata. Jedan od razloga tome je veliki broj vozila koji je uključen, kao i njihova veličina. Primjerice, na ukrcaju kontejnera na brod može biti uključeno i do 30 vozila, koja su duga 17m i široka 2.5m. Također, određivanje rasporeda AGV vozila izravno je povezano s rasporedom obalnih i skladišnih dizalica i njihovih poslova, što je otežavajući faktor.

U ovom radu, cilj rasporeda automatski upravljanih vozila je rasporediti skup AGV vozila kako bi se poboljšala ukupna produktivnost sustava i smanjilo kašnjenje u nizu poslova tipa ukrcaj-iskrcaj, a da se pri tome uzmu u obzir zastoji, konflikti, prioritet i dr. Krajnji ciljevi ovog rada se odnose na optimalizaciju vremena obrade i na minimalizaciju broja AGV koji su uključeni u transport te održavaju nesmetani protok kontejnera u sustavu.

S tom namjerom, u ovom primjeru određivanja rasporeda vozila razmatrat će se proces transporta kontejnera s područja pristrana do skladišta pomoću AGV-a.

7.6.1. Pregled dosadašnjih istraživanja

Posljednjih nekoliko desetljeća mnogo se istraživala tehnologija AGV sustava i zabilježen je veliki napredak i to uglavnom u okruženju FMS sustava. Kao jedna od mogućih tehnologija, određivanje rasporeda AGV privukla je značajnu pozornost znanstvenika (Egbelu i Tanchoco 1984, [Vis06a]). Yim i Linn [YIM93] modelirali su sustav pomoću PM i koristili su simulaciju kako bi istražili utjecaj različitih rasporeda AGV vozila na značajke FMS sustava. Krajnji cilj je bio skratiti ukupno vrijeme izvođenja svih poslova.

Mali je broj objavljenih radova koji se bave AGV sustavima u okruženju kontejnerskog terminala.

Glavna razlika između kontejnerskog terminala i proizvodnog AGV sustava je ta što kod kontejnerskog terminala, kontejneri imaju određene prioritete i ograničenja na skladišne lokacije i što kontejneri konkuriraju za određeni skup resursa, primjerice dizalice. Posljednjih nekoliko godina mnogi optimalizacijski problemi koji su povezani s kontejnerskim terminalom su opsežno proučavani. Vis i Koster De (2003) (citirano u [VIS06a]) iznijeli su cjelovit pregled literature.

Nekoliko je radova koji su proučavali određene optimalizacijske probleme kod kontejnerskih terminala s automatskom opremom. Općeniti modeli za određivanje rasporeda opreme, kao što su AGV vozila ili automatske dizalice, ili neautomatski resursi kao što su viljuškari, predloženi su od strane Bösel (2002), Chen (2006), Kim (2004), Zaffalon (2003). Bish i drugi (2001) razvili su heuristički algoritam za analizu novog rasporeda vozila i problema dodjeljivanja lokacija. Qiu i drugi [QIU02] predstavili su algoritam za određivanje rasporeda i rute AGV vozila pri dvosmjernom kretanju.

Općeniti model određivanje rasporeda AGV-a predložen je u [CHE97] i [KIM04]. Chen i ost. (1997) razmatrali su problem dodjeljivanja AGV-a na automatiziranom kontejnerskom terminalu. Kako bi pojednostavnili analizu, oni su pretpostavili da su skladišne dizalice uvijek raspoložive za ukrcaj/iskrcaj kontejnera na/s AGV-a. Za slučaj kada je samo jedna obalna dizalica, koja ukrcava ili iskrcava kontejnere, autori su razvili pohlepni (engl. greedy) algoritam za problem dodjeljivanje kontejnera AGV vozilima, za kojeg su pokazali da je optimalan. Algoritam radi na načelu da je prvih k AGV-ova dodijeljeno prvim k dostupnim kontejnerima. Nakon toga sljedeći kontejner je dodijeljen prvom raspoloživom AGV-u. Za slučaj kada je uključeno više obalnih dizalica, pokazalo se da greedy algoritam dodjeljuje raspoloživi AGV prvoj raspoloživoj QC dizalici i kroz primjere je pokazano da algoritam nužno pronaći optimalno rješenje, u ovom slučaju. Drugi pristup određivanju rasporeda AGV-ova predložen je od strane Kim i Bae (2004). Za svaki kontejner koji se treba transportirati, uveli su vremena nastupanja događaja koji bi se trebao dogoditi. Vremena se odnose na događaje podizanja/spuštanja kontejnera i transport kontejnera.

Vis i ost. [VIS06b] predlažu uporabu privremenih odlagališta u transportu kontejnera od pristrana do skladišta, tako da su proces transporta raščlanili na dva podprocesa: iskrcaj i transport. Razvili su model s cjelobrojnim programiranjem i odredili minimalan broj vozila potreban da se transport obavi u zadanom vremenskom okviru. Analitički rezultati su vrednovani simulacijom: numerički eksperimenti pokazuju da model pruža dobru procjenu potrebnog broja vozila.

Cheng i ost. (2005) su proučavali problem raspodjele AGV-a uzimajući u obzir utjecaj zastoja u sustavu. Mrežni model je predstavljen i koristi se kako bi se odredio potreban broj vozila; cilj je minimalizirati vrijeme čekanja AGV-ova na pristranu. Rezultati simulacije pokazuju da predloženi model povećava propusnost terminala.

Lee i ost. (2007) su predstavili model kojim rješavaju problem rasporeda u dvo-transtainerskom[29] sustavu: jedna QC opslužuje dva transtainera koji prihvaćaju kontejnere s dva različita područja terminala. Cilj je minimalizirati ukupno vrijeme ukrcavanja kontejnera na brod. Problem je riješen simuliranim kaljenjem.

7.6.2. Matematička formulacija

Za formulaciju problema potrebni su sljedeći indeksi:

i, j	= indeksi kontejnera
k,l	= indeks za dizalicu
	Oznake podataka
NcA	= broj kontejnera koje treba iskrcati s broda
J	= skup poslova koja treba obaviti
V	= skup AGV-ova
K	= skup dizalica
R	=skup svih resursa, $R = K \cup \left\{ \bigcup_{Z=A}^{E} segmZ \right\}$
$\boldsymbol{r}_{j,m}$	= 1 ako posao j treba resurs m kako bi se posao obavio, = 0, inače
$s_{i_j,j}$	= vrijeme setiranja resursa prije izvođenja posla j
$\boldsymbol{A}(t)$	= skup poslova koji se izvode u trenutku t
y_i^k	= vrijeme početka postavljanja (podizanja) i-tog kontejnera na(s) AGV od strane k-te dizalice
e_j^r	=najraniji mogući početak posla za resurs r ($s_i^0 = 0$, za $\forall i \in V$)
F_j	= vrijeme završetka posla $j \in J$
p_j	=duljina trajanja izvođenja posla $j \in J$
M	= dovoljno velik pozitivan broj
	Varijable odluke
x_{ij}^k	=1 ako je j-tom AGV-u, dodijeljen i-ti kontejner k-te dizalice

Neka je $J = \{1,2,\ldots,n\}$ skup poslova koje treba izvršiti skup $\mathcal{R} = \{r_1, r_2, \ldots, r_K\}$ resursa. Svaki posao se izvršava pomoću jednog resursa. Svaki resurs može obavljati samo jedan posao u određenom trenutku.

Svaki posao $j \in J$ se izvodi određeno vrijeme $p_j > 0$. Prije nego posao započne zahtijeva se određeno vrijeme setiranja. Neka $i_j \in J$ označava posao kojeg obavlja resurs $r \in \mathcal{R}$ neposredno prije izvođenja posla j. Dakle, vrijeme setiranja resursa $r \in \mathcal{R}$, $s_{i_j,j} \geq 0$ se mora uzeti u obzir prije no što se počne izvršavati posao j. Jednom započet posao se mora izvršiti do kraja, bez prekidanja.

Neka je α cijena trajanja izvođenja poslova u jedinici vremena i β je kazna (engl. *penalty cost*) po jedinici vremena za čekanje AGV-ova na dizalice. Pretpostavka je da vrijedi $\alpha << \beta$. Matematički se problem može opisati kako slijedi:

---

[29] Transtainer je vrsta skladišne dizalice velikog raspona.

$$Min \left(\alpha \sum_{j \in J} F_j + \beta \sum_{l \in V} \sum_{k \in K} \left(y_j^k - e_{i_l}^l \right) \right) \tag{7-3}$$

Tako da vrijedi

$$\sum_{i=1}^{NcA} x_{ij} = 1, \qquad \forall j \in V \tag{7-4}$$

$$\sum_{j \in V} x_{ij} = 1, \qquad \forall i = 1,2,\ldots,NcA \tag{7-5}$$

$$F_{i_j} + s_{i_j,j} + p_j \le F_j, \qquad \forall\, j \in J \tag{7-6}$$

$$e_{i_j} + s_{i_j,j} + p_j \le F_j, \qquad \forall\, j \in J \tag{7-7}$$

$$\sum_{j \in A(t)} r_{j,k} \le RD(k), \qquad r_k \in \mathcal{R},\ t \ge 0, \tag{7-8}$$

$$x_{i,j} = \begin{cases} 1 \\ 0, \end{cases} \qquad \forall i \in 1,2,\ldots,NcA,\ \forall j \in V \tag{7-9}$$

$$F_j \ge 0,\ RD(k) \ge 0, \quad j = 1,2,\ldots,n,\ r_k \in \mathcal{R},\ k = 1,\ldots,\mathcal{K} \tag{7-10}$$

Problem je pronaći vektor rješenja $x_{i,j}$, $\forall i \in 1,\ldots,NcA$, $\forall j \in V$ i raspored poslova $j \in J$, takav da resurs r_j i vrijeme završetka f_j za svaki posao $j \in J$ zadovoljavaju kriterije od (7-4) do (7-10). U toku iterativnog procesa, cilj je minimalizirati ukupno vrijeme izvođenja svih poslova $j \in J$, što je prvi dio funkcije cilja (7-3), i minimalizirati ukupno čekanje AGV-ova na dizalice. Kako je $\alpha << \beta$, zbroj kašnjenja poslova koje obavljaju dizalice će se prvo minimalizirati. Ograničenja (7-4) i (7-5) osiguravaju da je mogući raspored preslikavanje 1 na 1; za svaki kontejner koji se iskrca s broda dodijeli mu se jedan AGV za transport do skladišta. Ograničenja (7-6) i (7-7) definiraju ograničenje slijeda poslova za bilo koji par poslova (i, j), pri čemu posao *i* neposredno prethodi poslu *j*. Izraz (7-6) ograničava da se posao j ne može započeti izvoditi nakon što ne prođe vrijeme "setiranja" od trenutka kada je prethodni posao *i* napustio taj resurs, a uvjetom (7-7) se ograničava da se posao ne može početi izvoditi prije no što je resurs raspoloživ za izvođenje posla *j* nakon što je obavio posao *i*. Obzirom na kapacitet resursa, ograničenje (7-8) se odnosi na dodjeljivanje resursa *k* poslu *j* koji se počeo izvoditi u trenutku *t*.

Dakle problem spada u više-kriterijske i više-ciljne probleme rasporeda poslova.

7.6.3. Aplikacija

Kako je sustav presložen i vrijeme izvođenja algoritma je dugo, neće se uzeti da je s broda SAFMARINE KOMATI iskrcano 1000 kontejnera. Pretpostavka je da se s broda treba iskrcati *NcA* = 30 kontejnera i da se na skladištu nalazi *NcB* = 30 kontejnera koji čekaju na transport do željeznice. Statička strategija pretpostavlja da je uvijek isti broj AGV-ova dodijeljen obalnim dizalicama za vrijeme iskrcavanja kontejnera s broda. Međutim takva strategija, zbog malog manevarskog prostora dizalica, može dovesti do zastoja u transportu. Za vrijeme iskrcavanja svakih t_g sekundi kontejner je raspoloživ za iskrcaj. Ovo vrijeme je uzeto kao *setup* vrijeme za obalnu dizalicu. Također, pretpostavka je da je prosječno vrijeme obalne dizalice za podizanje kontejnera s broda jednako 66 sekundi. Nakon tog vremena kontejner je raspoloživ za ukrcaj na AGV i određena je ruta kojom će AGV voziti (smjer A ili

B). Broj AGV-ova potreban da se minimalizira vrijeme potrebno za iskrcavanje kontejnera s broda nije konstantan već je ograničen varijablom *N,* u skladu s cijenom rada i veličinom terminala. Pretpostavka je da su sva vozila potpuno istih mogućnosti i kapaciteta. Kontejneri dodijeljeni dizalicama, kao i ruta vozila određuju niz poslova $j_{g;i}$, gdje je $i \in \{1, \ldots, n_g\}$, n_g je broj poslova koje *dodijeljeni* AGV mora obaviti pri transportu *g*-tog kontejnera. Posao *j* označava transport kontejnera s jedne lokacije na drugu.

Kod ove aplikacije pojedina ruta (smjer A ili B) kojim AGV može voziti gledat će se kao pojedini projekt koji se sastoji od određenog broja poslova koji su navedeni u (7-1) i poslovi se moraju obaviti određenim redoslijedom. Svaki posao se izvršava uz uporabu određenog resursa i traje određeni broj vremenskih jedinica (minuta). Segmenti i dizalice predstavljaju resurse sustava. Razmatrat će se dinamički model, tj. projekti su različitog prioriteta i postoje kašnjenja u izvršavanju pojedinih poslova. Također dizalice zahtijevaju određeno vrijeme za 'setup'.

Svaki kromosom je sastavljen od $m+n+m$ gena, gdje je $n=14$ broj poslova, $m=2$ broj projekata:

$$kromosom = \left(\underbrace{p_1, p_2}_{\text{prioriteti}}, \underbrace{k_1, k_2, \ldots, k_n}_{\text{kašnjenja}}, \underbrace{rt_1, rt_2}_{\text{vrijeme raspoloživosti projekata}} \right). \qquad (7\text{-}11)$$

Dakle, i u ovoj aplikaciji koristi se indirektna reprezentacija kromosoma. Pomoću procedure opisane u podpoglavlje 6.6, dekodirani kromosomi će se prevesti u odgovarajući raspored poslova.

Generiranim rasporedom odredit će i potreban broj AGV-a, kao i niz poslova koje AGV-ovi trebaju izvršiti. U ovoj aplikaciji za određivanje koji AGV-e će se pridružiti kojoj QC dizalici koristi će se pravilo: slobodno AGV vozilo koje je najmanje udaljeno od dizalice bit će joj dodijeljeno. Slika 7.23 opisuje način izračuna funkcije cilja za transport jednog kontejnera.

```
1.  NcA              (broj kontejnera koje treba iskrcati s broda)
2.  Tcrane           (vrijeme kada je dizalica podigla kontejner s broda)
3.  crane            (niz obalnih dizalica koje trebaju  AGV u iteraciji )
4.  i ← 1             (redoslijed u nizu kontejnera)
5.  N ← 1             (broj AGV-ova)
6.  Tagv ← 0; Dagv ← 0; Wagv ← 0 (ukupno vrijeme vožnje, vrijeme kašnjenja i čekanja AGVova
7.      m(:,it)                    (vektor oznaka PM u iteraciji it)
8.  if  m(17,it)>0 & (m(1,it)+m(2,it))<m(17,it)   /* ako ima slobodnih dizalica treba im dodijeliti
                                                      kontejner */
9.  while i <= NcA do
10.  m(1,it) ← m(1,it) +1  /*uvodi se nova oznaka u ulazno mjesto PM (1 ili 2 ovisno o AGV ruti) */
11.     [Duration, Delay, Wait]= gettime(i);
12.          Tagv ← Tagv + Duration;
13.          Dagv ← Dagv + Delay;
14.          Wagv ←Wagv + Wait;
15.     if (Tagv >T) or (crane(i) < crane(i-1)) then
16.              N = N+ 1
17.              Tagv ← 0; Dagv ← 0; Wagv ← 0;
18.     else
19.          i ← i + 1;    endif
20.     end
```

Slika 7.23 Pseudo kod za računanje funkcije cilja

Funkcijom *gettime(i) (linija* 11*)* trebalo bi se izračunati vrijeme koje je AGV-u potrebno da obavi transport *i*-tog kontejnera. To vrijeme bi trebalo uključivati vrijeme da AGV dohvati početnu poziciju (od parkinga do dodijeljene obalne dizalice), vrijeme transporta kontejnera i vremena čekanja koje se može dogoditi ako se AGV vrati prije nego dobije poziv za sljedeći posao. Za svaki AGV određena je ruta. Ako se radi o smjeru B, AGV treba kontejner prebaciti do skladišta, a zatim se vratiti na privremeno parkirališta gdje čeka na sljedeći poziv dizalice. U drugom slučaju, AGV nakon što transportira kontejner do skladišta treba otići do lokacije gdje će preuzeti drugi kontejner koji će prebaciti do dizalice na željeznici i nakon toga se vraća na privremeno parkiralište. Važno je napomenuti da se broj AGV-a uvijek povećava (*linija* 16) ako u trenutku kada dizalica treba prebaciti kontejner s broda nema slobodnih AGV-ova (kada vrijeme obavljanja poslova za AGV prelazi vrijeme *Tcrane* ili ako je slobodno AGV vozilo dodijeljeno za transport kontejnera (*i*-1) i čeka kod dizalice Q_{i-1} koja prethodi dizalici Q*i*.

U procesu reprodukcije koristit će se genetski operatori opisani u 6. poglavlju. Prvo će se primjenom elitističke strategije kreirati vrh populacije, odnosno 30% najboljih jedinki će se izravno proslijediti u sljedeću generaciju i one prolaze kroz bazen za reprodukciju. Osnovni nedostatak elitističke strategije je što algoritam može prebrzo konvergirati ka lokalnom minimumu. Ovaj problem će se riješiti operatorom mutacije vrlo velike vjerojatnosti, odnosno kreirat će se dno populacije (40% jedinki cijele populacije činit će dno) na način da se uvedu potpuno nove jedinke u populaciju.

Također, u ovom poglavlju predlaže se novi kombinirani operator selekcije. Sastoji se od predloženog elitističkog operatora selekcije (poglavlje 6.5.2) i Fibonaccijeve selekcije.

Kombinirana elitističko-Fibonaccijeva selekcija će se primijeniti za preživljavanje onih jedinki koje će činiti vrh nove populacije i radi na način da se jedinke trenutne populacije, prvo, sortiraju prema elitizmu (vrijednostima fitnes funkcije). Na tako sortiranu populaciju primjeni se Fibonaccijeva selekcija. Na ovaj način trebala bi se osigurati raznolikost populacije, a istovremeno bi preživjele jedinke koje čine elitu populacije.

7.7. Rezultati eksperimentiranja

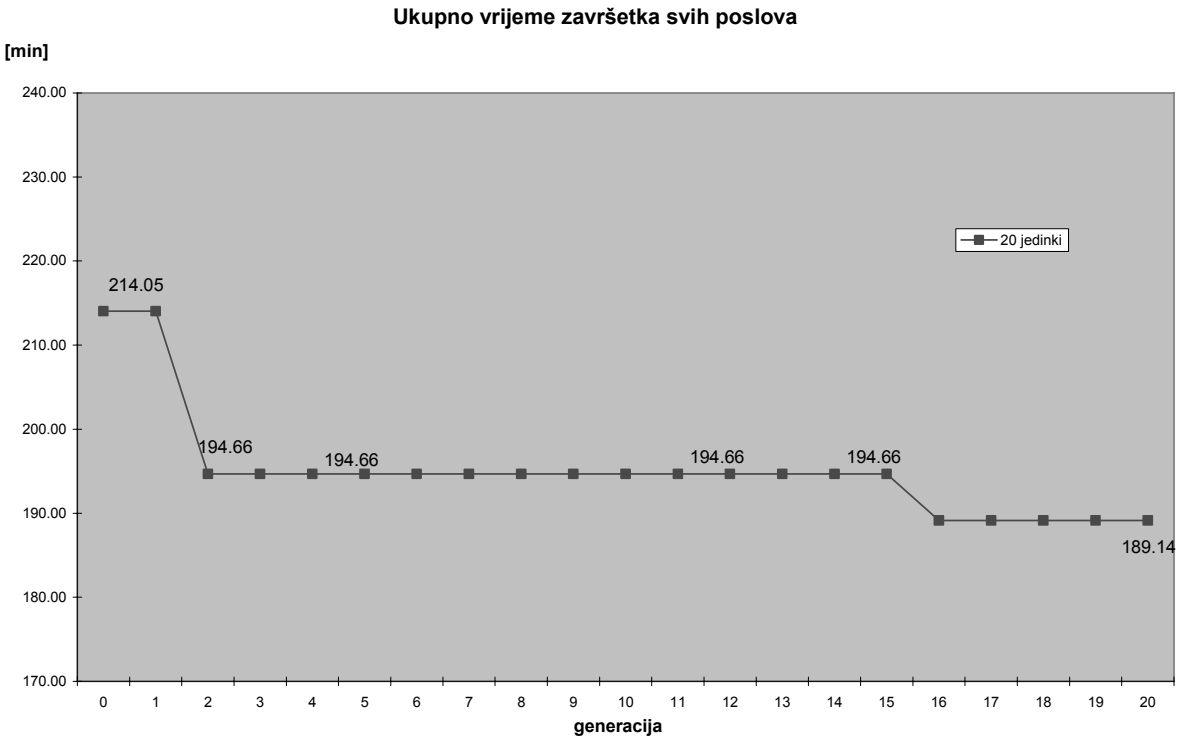
U ovom poglavlju izneseni su rezultati eksperimentiranja primjene PMGA algoritma na opisani problem. Posebna pažnja posvećena je trajanju izvođenja raznih verzija paralelnih genetskih algoritama.

Pri testiranju uvažene su sljedeće pretpostavke. Svi kontejneri su istog kapaciteta, 40 stopa (engl. *feet*). Brzina punog AGV_a (s kontejnerom) je 3 m/s, a brzina praznog AGVa je 5.5 m/s. Za potrebe genetskog algoritma korišteni su sljedeći parametri:

Veličina Populacije = 20 jedinki
Vjerojatnost križanja = 0.500
Kriterij za kraj = 20 generacija.

Za početak, problem optimalizacije funkcije cilja (7-3), razdvojit će se u dva podproblema optimalizacije. Kod prvog podproblema, pokušat će se pronaći takav raspored svih poslova tako da se minimalizira ukupno vrijeme završetka svih poslova (minimalizira se prvi dio funkcije cilja u izrazu (7-3)). Kod drugog podproblema će se minimalizirati drugi dio funkcije cilja iz izraza (7-3), odnosno, minimalizirat će se prosječno vrijeme čekanja AGV-ova na dizalice. Iako je glavni cilj kako poboljšati krajnje rješenje, razmatrat će se i vrijeme izvođenja GA. Algoritam je testiran po 5 puta za svaki slučaj kako bi se provjerila konzistentnost rješenja. Niže su predočeni najznačajniji rezultati.

Slika 7.24 pokazuje najmanju vrijednost ukupnog izvođenja svih poslova za svaku generaciju. Razvidno je da je u početnoj populaciji najbolje vrijeme bilo od 214.04 min, dok se evolucijskim procesom kroz 20 generacija to vrijeme uspjelo smanjiti na 189.1414 minute. Lokalni optimum se postiže nakon 2269.27 sekundi.



Slika 7.24 Vrijeme završetka svih poslova u odnosu na generacije

Tablica 7.3 prikazuje postupak generiranje tog rasporeda po fazama, odnosno, prema proceduri iz podpoglavlja 6.6, u svakoj fazi je određeno koji su poslovi raspoloživi i vrijeme kada će započeti. Vremenski slijed izvođenja poslova za 30 kontejnera koje treba iskrcati s broda i transportirati do skladišta, posebno za svaki smjer transporta, prikazan je slikama 7.25(a) i 7.25(b). Slika 7.26 prikazuje postojanje konflikta u 164.2 minuti izvođenja. Naime, nakon što je jedno vozilo prošlo segmentom A u smjeru A, a drugo vozilo segmentom B u smjeru A, za nastavak obavljanja poslova oba vozila trebaju dizalicu CR_2. U tom momentu kontrolno mjesto COa3 se isključilo, a uključilo se COa4. Na taj način resurs je dodijeljen drugom vozilu i ono nastavlja s izvođenjem posla CR2ukrcaj_A (dizalica na vozilo ukrcava kontejner koji treba doći do željeznice).

Tablica 7.3 Redoslijed izvođenja poslova za optimalni raspored

kontejner	posao	Start time
faza: 1		
2	CR1ukrcaj_A	1.700000
3	CR1ukrcaj_B	1.700000
faza: 2		
1	CR1ukrcaj_B	2.700000
faza: 3		
2	transA_A	3.750000
3	transA_B	3.750000
faza: 4		
3	CR2iskrcaj_B	5.300000
faza: 5		
1	transA_B	5.533333

```
faza: 6
    4           CR1ukrcaj_A         7.083333
faza: 7
    4           transA_A            9.083333
    3           transB_B            9.083333
faza: 8
    5           CR1ukrcaj_B         11.422222
    1           CR2iskrcaj_B        11.422222
    3           transE_B            11.422222
faza: 9
    5           transA_B            11.472222
    3           transD_B            11.472222
faza: 10
    6           CR1ukrcaj_A         13.422222
faza: 11
    7           CR1ukrcaj_B         14.472222
    1           transB_B            14.472222
faza: 12
    6           transA_A            15.472222
    1           transE_B            15.472222
faza: 13
    7           transA_B            17.533333
    5           CR2iskrcaj_B        17.533333
    1           transD_B            17.533333
faza: 14
faza: 15
    8           CR1ukrcaj_B         19.083333
    5           transB_B            19.083333
faza: 16
    5           transE_B            20.083333
faza: 17
    8           transA_B            20.865152
    7           CR2iskrcaj_B        20.865152
faza: 18
    5           transD_B            20.915152
faza: 19
    9           CR1ukrcaj_B         23.415152
    7           transB_B            23.415152
faza: 20
    7           transE_B            24.415152
faza: 21
    6           CR2iskrcaj_A        25.196970
    10          CR1ukrcaj_B         25.196970
    9           transA_B            25.196970
    7           transD_B            25.196970
faza: 22
faza: 23
    10          transA_B            27.196970
faza: 24
    6           transB_A            28.696970
faza: 25
    11          CR1ukrcaj_B         26.746970
faza: 26
    6           CR2ukrcaj_A         29.746970
faza: 27
    6           transC_A            31.796970
    12          CR1ukrcaj_B         31.796970
faza: 28
    9           CR2iskrcaj_B        35.574747
```

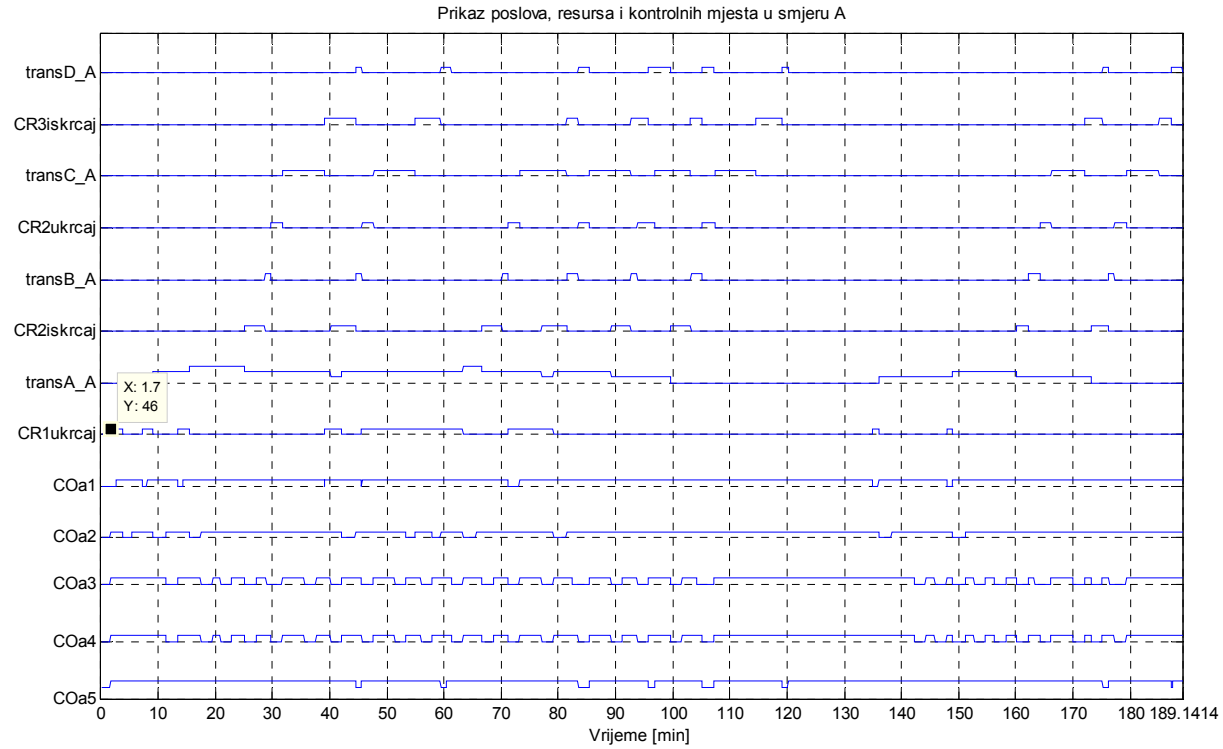
```
faza: 29
    13          CR1ukrcaj_A         35.624747
    6           CR3iskrcaj_A        35.624747
faza: 30
    11          transA_B            37.574747
faza: 31
    9           transE_B            37.624747
faza: 32
    9           transB_B            39.074747
faza: 33
    2           CR2iskrcaj_A        40.124747
    9           transD_B            40.124747
faza: 34
    13          transA_A            42.124747
faza: 35
faza: 36
    2           transB_A            44.463636
    6           transD_A            44.463636
faza: 37
    14          CR1ukrcaj_A         44.513636
faza: 38
    2           CR2ukrcaj_A         45.554545
faza: 39
    2           transC_A            47.604545
faza: 40
    11          CR2iskrcaj_B        51.382323
faza: 41
    2           CR3iskrcaj_A        51.432323
    15          CR1ukrcaj_B         51.432323
faza: 42
    12          transA_B            53.382323
faza: 43
    11          transB_B            54.882323
faza: 44
    11          transE_B            54.932323
faza: 45
    10          CR2iskrcaj_B        55.932323
    11          transD_B            55.932323
faza: 46
    15          transA_B            57.932323
faza: 47
    10          transB_B            57.982323
faza: 48
    2           transD_A            59.432323
faza: 49
    10          transE_B            60.523232
faza: 50
faza: 51
    16          CR1ukrcaj_B         61.305051
    8           CR2iskrcaj_B        61.305051
    10          transD_B            61.305051
faza: 52
    8           transB_B            61.658081
faza: 53
faza: 54
    14          transA_A            63.305051
faza: 55
    8           transE_B            65.643939
faza: 56
```

```
    4           CR2iskrcaj_A        66.693939
    17          CR1ukrcaj_B         66.693939
    8           transD_B            66.693939
faza: 57
    16          transA_B            68.693939
faza: 58
    4           transB_A            70.193939
faza: 59
    18          CR1ukrcaj_A         71.243939
    4           CR2ukrcaj_A         71.243939
faza: 60
    4           transC_A            73.293939
faza: 61
    14          CR2iskrcaj_A        77.071717
faza: 62
    4           CR3iskrcaj_A        77.121717
    19          CR1ukrcaj_B         77.121717
faza: 63
    18          transA_A            79.071717
faza: 64
    14          transB_A            79.121717
faza: 65
    14          CR2ukrcaj_A         83.460606
    4           transD_A            83.460606
faza: 66
    14          transC_A            85.460606
faza: 67
    18          CR2iskrcaj_A        89.238384
faza: 68
    18          transB_A            89.288384
    14          CR3iskrcaj_A        89.288384
    20          CR1ukrcaj_B         89.288384
faza: 69
    17          transA_B            91.238384
faza: 70
    18          CR2ukrcaj_A         93.788384
faza: 71
    14          transD_A            95.788384
faza: 72
    18          transC_A            95.838384
faza: 73
    13          CR2iskrcaj_A        99.616162
faza: 74
    18          CR3iskrcaj_A        99.666162
    21          CR1load_B           99.666162
faza: 75
    19          transA_B            101.616162
faza: 76
    13          transB_A            103.116162
faza: 77
    13          CR2ukrcaj_A         105.166162
    18          transD_A            105.166162
faza: 78
faza: 79
    13          transC_A            107.307071
faza: 80
    16          CR2iskrcaj_B        111.084848
faza: 81
    13          CR3iskrcaj_A        111.134848
```

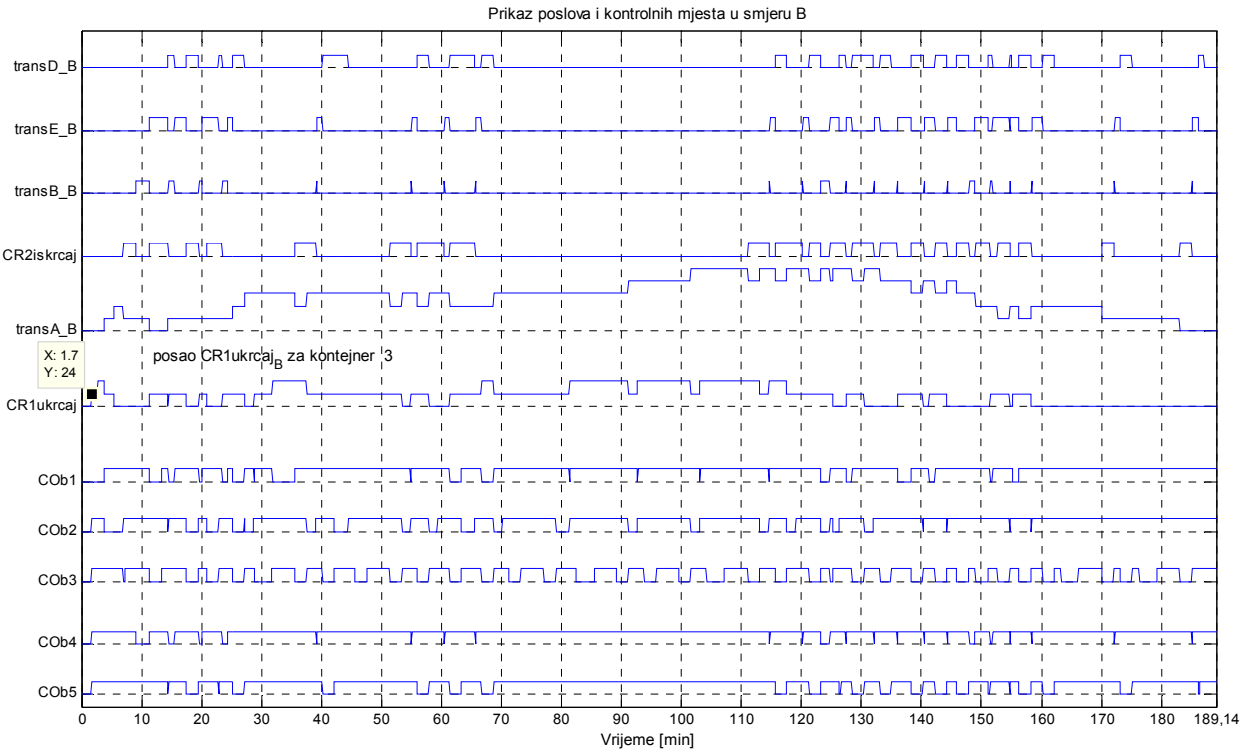
```
    22          CR1ukrcaj_B         111.134848
faza: 82
    20          transA_B            113.084848
faza: 83
    16          transB_B            113.134848
faza: 84
    16          transE_B            114.584848
faza: 85
    17          CR2iskrcaj_B        115.634848
    16          transD_B            115.634848
faza: 86
    22          transA_B            117.634848
faza: 87
    13          transD_A            119.134848
faza: 88
    17          transE_B            119.184848
faza: 89
    17          transB_B            120.225758
faza: 90
    22          CR2iskrcaj_B        121.275758
    17          transD_B            121.275758
faza: 91
    23          CR1ukrcaj_B         123.275758
    21          transA_B            123.275758
    22          transB_B            123.275758
faza: 92
    20          CR2iskrcaj_B        123.325758
    22          transE_B            123.325758
faza: 93
    23          transA_B            124.825758
faza: 94
    22          transD_B            126.325758
faza: 95
    20          transB_B            127.375758
faza: 96
    24          CR1ukrcaj_B         127.457576
    20          transE_B            127.457576
faza: 97
    15          CR2iskrcaj_B        128.457576
    20          transD_B            128.457576
faza: 98
    24          transA_B            130.457576
faza: 99
faza: 100
    15          transB_B            130.507576
faza: 101
    15          transE_B            131.957576
faza: 102
    24          CR2iskrcaj_B        133.007576
    15          transD_B            133.007576
faza: 103
    25          CR1ukrcaj_A         135.007576
faza: 104
    24          transB_B            135.057576
faza: 105
    25          transA_A            136.007576
    26          CR1ukrcaj_B         136.007576
    24          transE_B            136.007576
faza: 106
```

```
    26          transA_B            136.839394
faza: 107
    19          CR2iskrcaj_B        138.346465
    24          transD_B            138.346465
faza: 108
faza: 109
    19          transB_B            140.396465
faza: 110
    19          transE_B            140.478283
faza: 111
    27          CR1ukrcaj_B         141.260101
faza: 112
    12          CR2iskrcaj_B        142.260101
    19          transD_B            142.260101
faza: 113
    27          transA_B            142.310101
faza: 114
    12          transB_B            144.260101
faza: 115
    12          transE_B            143.860101
faza: 116
    26          CR2iskrcaj_B        145.860101
    12          transD_B            145.860101
faza: 117
faza: 118
    28          CR1ukrcaj_A         147.860101
    26          transB_B            147.860101
faza: 119
    26          transE_B            147.910101
faza: 120
    28          transA_A            148.860101
    21          CR2iskrcaj_B        148.860101
faza: 121
    29          CR1ukrcaj_B         150.910101
    21          transB_B            150.910101
faza: 122
    26          transD_B            151.198990
faza: 123
    21          transE_B            151.910101
faza: 124
    27          CR2iskrcaj_B        152.691919
faza: 125
    29          transA_B            152.741919
    21          transD_B            152.741919
faza: 126
    27          transB_B            154.691919
faza: 127
    27          transE_B            154.291919
faza: 128
faza: 129
    30          CR1ukrcaj_B         155.291919
faza: 130
    29          CR2iskrcaj_B        156.291919
    27          transD_B            156.291919
faza: 131
    30          transA_B            156.644949
faza: 132
    29          transB_B            158.194949
faza: 133
```

```
    29          transE_B            158.291919
faza: 134
    25          CR2iskrcaj_A        160.194949
    29          transD_B            160.194949
faza: 135
faza: 136
    25          transB_A            162.194949
faza: 137
    25          CR2ukrcaj_A         164.244949
faza: 138
    25          transC_A            166.294949
faza: 139
    30          CR2iskrcaj_B        170.072727
faza: 140
    25          CR3iskrcaj_A        170.122727
faza: 141
    30          transB_B            172.122727
faza: 142
    30          transE_B            172.122727
faza: 143
    28          CR2iskrcaj_A        173.122727
    30          transD_B            173.122727
faza: 144
faza: 145
    25          transD_A            175.122727
faza: 146
    28          transB_A            175.172727
faza: 147
faza: 148
    28          CR2ukrcaj_A         177.263636
faza: 149
    28          transC_A            179.313636
faza: 150
    23          CR2iskrcaj_B        183.091414
faza: 151
    28          CR3iskrcaj_A        183.141414
    23          transB_B            183.141414
faza: 152
    23          transE_B            185.141414
faza: 153
    23          transD_B            186.141414
faza: 154
faza: 155
    28          transD_A            186.444444
```

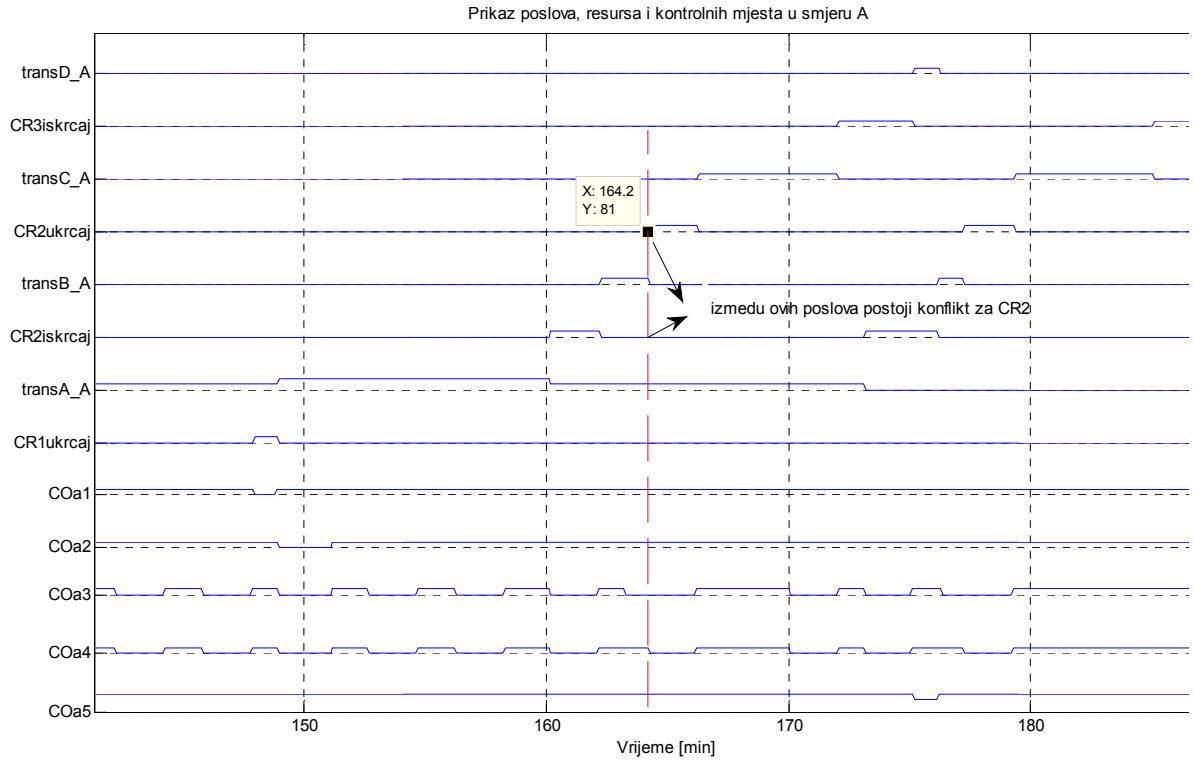


(a)



(b)

Slika 7.25 Vremenski slijed izvođenja poslova: (a) smjer A; (b) smjer B



Slika 7.26 Konfliktna situacija u smjeru A

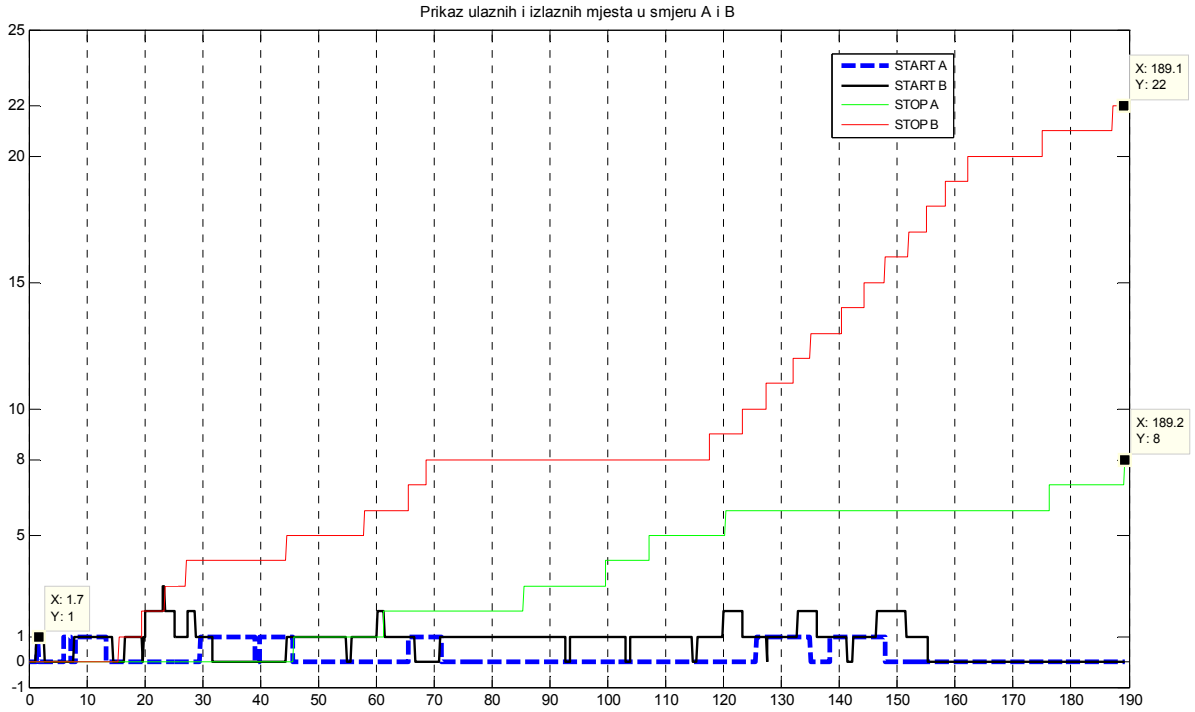
Slika 7.27 prikazuje raspodjelu AGV-ova po poslovima. Razvidno je da je potrebno 11 AGV kako bi se postigao optimalni raspored. Razvidno je da su ta vozila 22 puta raspoređena za smjer B, a 8 puta za smjer A. Može se uočiti da je ovo pridruživanje preslikavanje 1 na 1, svakom kontejneru je pridružen sam jedan AGV. Također svih 30 kontejnera su dodijeljeni jednom vozilu.

```
broj agv-a            #kontejner-smjer (A/B)
-------------------------------------------------
agv    1:            1-B     13-A     27-B
agv    2:            2-B      7-B     19-B   28-A
agv    3:            3-A     12-B     21-B   29-B
agv    4:            4-A     20-B     30-B
agv    5:            5-B     15-B     24-B
agv    6:            6-A     16-B     23-B
agv    7:            8-B     25-A
agv    8:            9-B     22-B
agv    9:           10-B     17-B     26-B
agv   10:           11-B     18-A
agv   11:           14-A
```

Slika 7.27 Raspodjela poslova po skupu AGV-ova

Vremenski slijed pridruživanja kontejnera AGV-ovima razvidan je iz slike 7.28. Kako je dizalica CR_1 podigla kontejner s broda, pozove se AGV koji dođe do mjesta ispod dizalice, što je prikazano inkrementiranjem broja oznaka ili u ulaznom mjestu START_A ili

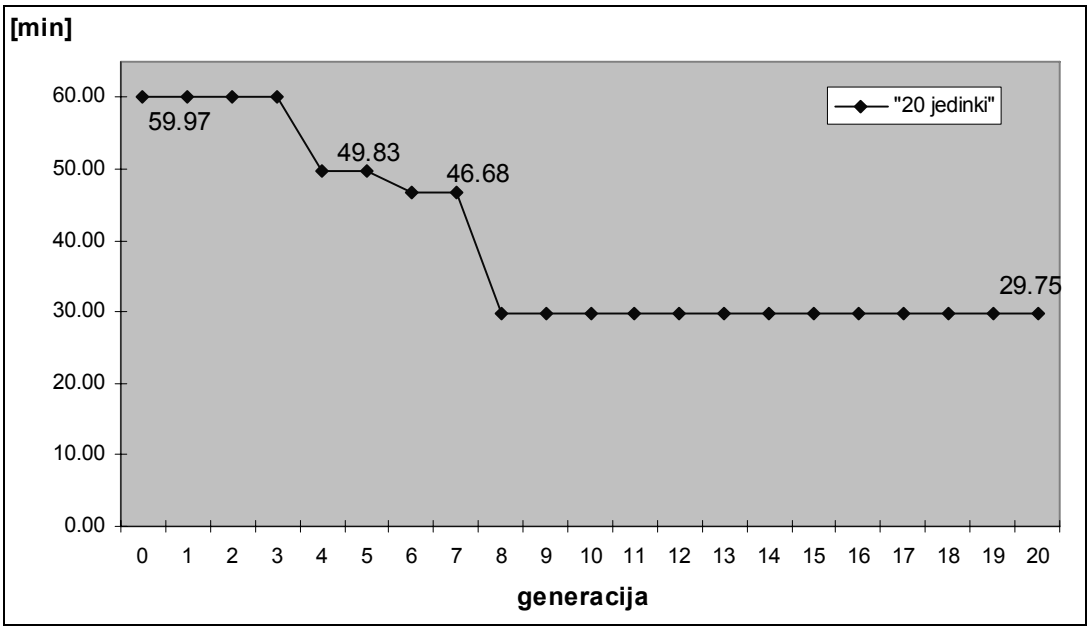
START_B, ovisno o ruti AGV-a. Kada AGV isporuči kontejner na odredište onda se inkrementira broj oznaka u izlaznim mjestima. Dakle AGV-ovi su 22 puta stigli do izlaznog mjesta STOP_B i 8 puta do STOP_A, znači svih 30 kontejnera s broda je stiglo do skladišta na obali.



Slika 7.28 Promjene broja oznaka u ulazno/izlaznim mjestima PM kroz vrijeme

Drugi podproblem se svodi na pronalaženje rasporeda poslova tako da je vrijeme čekanja AGV na dizalice što kraće. Dakle treba minimalizirati drugi dio funkcije cilja iz izraza (7-3). Kao funkcija cilja uzet će se prosječno vrijeme čekanja AGV na dizalice, odnosno treba zbrojiti sva čekanja AGV-ova i podijeliti s ukupnim brojem uključenih vozila. Algoritam je testiran s potpuno istim parametrima kao i kod rješavanja prvog podproblema. Datoteka *'agv_wait.m'* u prilogu L, sadrži programski kod koji računa ukupno vrijeme čekanja svih uključenih vozila na dizalice CR_2 i CR_3. To vrijeme se lako izračuna iz matrice m(1:36, *it*), koja pamti broj oznaka u PM u iteraciji *it*. Dovoljno je za četvrti, šesti, osmi i dvanaesti redak matrice *m* pratiti kada je broj oznaka povećan za 1 (AGV je stiglo do dizalice) i kada su im pridruženi prijelazi okinuti (tada im je dizalica stavila ili podigla kontejner). Vrijeme čekanja vozila na obalne dizalice CR_1 računa se u datoteci *'pozovi_novi.m'*, prilog K, u linijama 17 i 18.

Rezultati su prikazani na slikama 7.29 i 7.30. Vidljivo je da najbolji raspored postiže prosječno vrijeme čekanja na dizalice od 29.75 minuta i taj raspored uključuje 11 AGV vozila. Inače za taj raspored ukupno vrijeme izvođenja svih poslova iznosi 220.5449 minuta. Vrijeme za koje algoritam postiže optimum je 1921.328125 sek.



Slika 7.29 Ukupno vrijeme čekanja AGV-a na dizalice

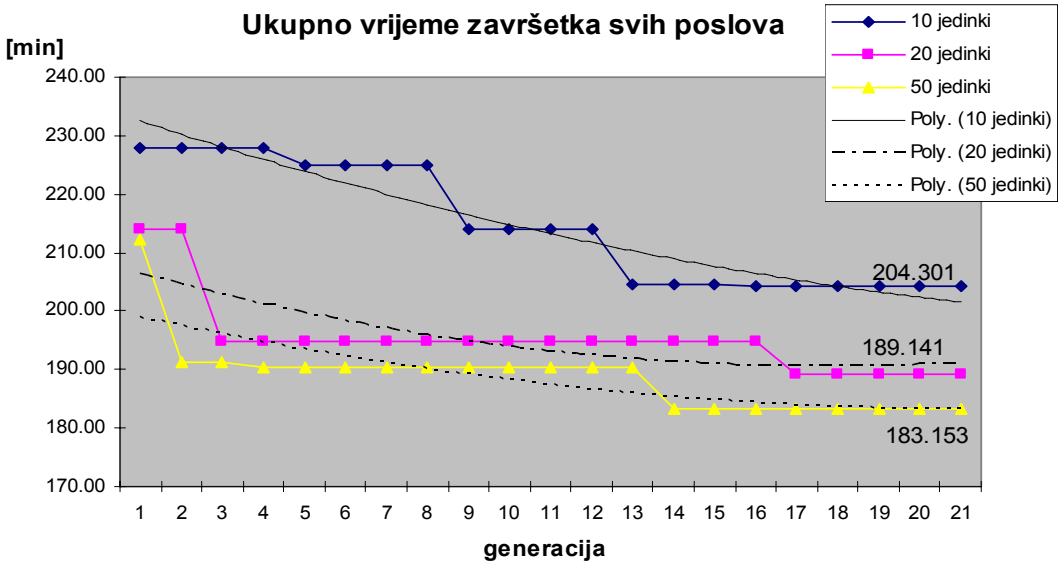
```
broj agv-a          broj kontejnera - RUTA (A/B)
-------------------------------------------------------------------------
agv    1:           1-B          8-B         16-A         22-B         28-B
agv    2:           2-B          9-B         23-B
agv    3:           3-A         18-B
agv    4:           4-B         12-B         25-B
agv    5:           5-B         13-A         20-B
agv    6:           6-B         14-B         19-B         27-B
agv    7:           7-B         15-A         24-B
agv    8:          10-B         30-A
agv    9:          11-B
agv   10:          17-B         26-A
agv   11:          21-A         29-A
```

Slika 7.30 Raspored i rute vozila

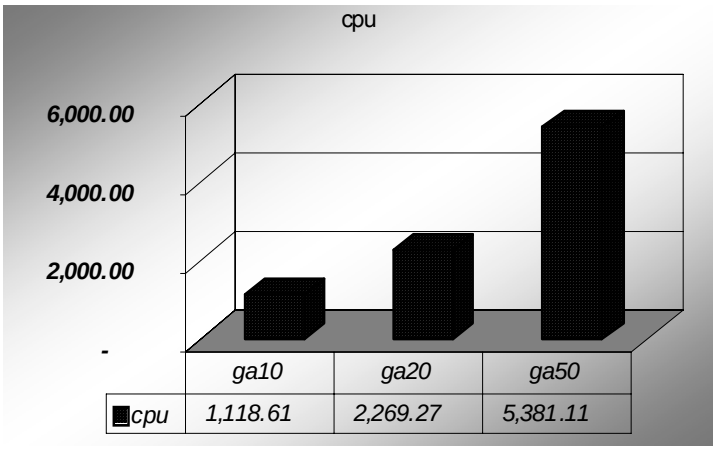
7.8. Utjecaj veličine populacije

Kao i u 6. poglavlju ispitat će se utjecaj veličine populacije na rezultat funkcije cilja te na duljinu izvođenja algoritma kao i na računalni napor.

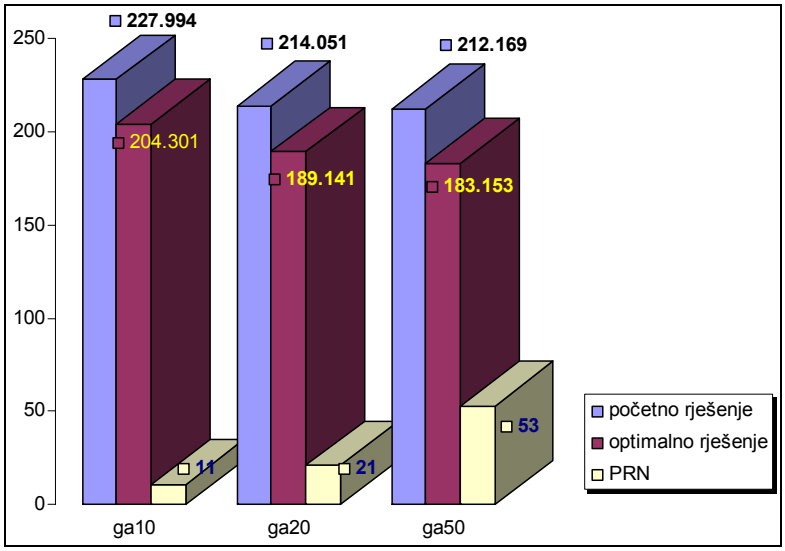
Analiza će se napraviti za oba podproblema iz predhodnog podpoglavlja. Usporedit će se rezultati postignuti s 10, 20 i 50 jedinki u populacijama. Već broj jedinki se neće uzeti u obzir jer se pokazalo da takav algoritam jako dugo traje uz neznatno poboljšanje funkcije cilja. Za optimalizaciju ukupnog vremena izvođenja svih poslova rezultati su prikazani slikama od 7.31 do 7.33. Razvidno je da algoritam s 50 jedinki u populaciji postiže najbolje rezultate, ali treba i najduže vrijeme izvođenja i najveći računalni napor. U usporedbi s algoritmom od 20 jedinki (slika 7.31) rješenje je malo poboljšano (oko 3%), ali vrijeme izvođenja algoritma je gotovo dvostruko duže (slika 7.32).



Slika 7.31 Usporedba najboljih rezultata po generacijama



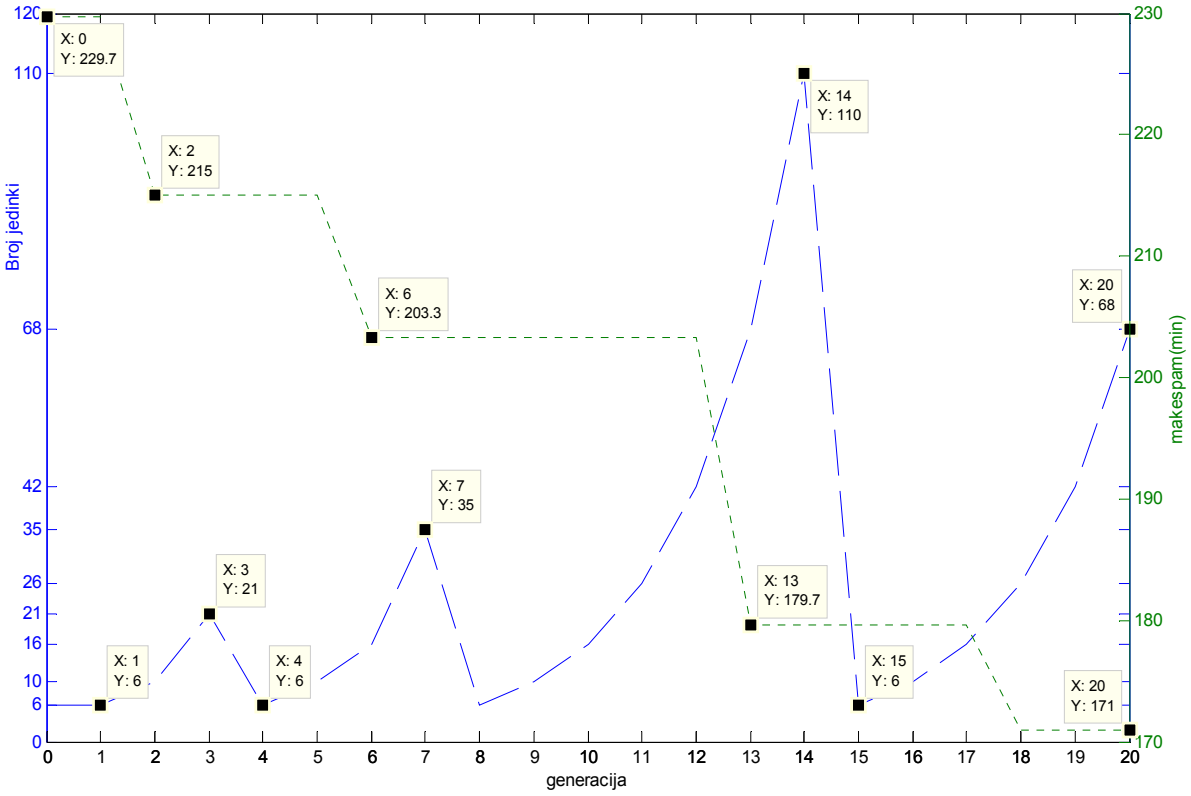
Slika 7.32 Vrijeme izvođenja u odnosu na broj jedinki u populaciji



Slika 7.33 Usporedba početnog i optimalnog rješenja te prosječnog računalnog napora obzirom na broj jedinki u populaciji

Kako bi se poboljšalo vrijeme izvođenja svih poslova, uz razumno vrijeme izvođenja, statička veličina populacije zamijenit će se dinamičkom. Pri promjeni veličine populacije koristit će se isti mehanizam opisan u *podpoglavlju 6.7.7, a* koji se temelji na Fibonaccijevim brojevima (datoteku *'DinFibonacciGA.m')*.

Slika 7.34 pokazuje da ovaj algoritam postiže najbolji rezultat od 170.971717 minuta. Broj jedinki u populacijama je od 6 do najviše 110 jedinki. U odnosu na isti algoritam sa statičkom veličinom populacije od 50 jedinki (*ga50)*, ovaj algoritam je skratio vrijeme izvođenja poslova za 19 minuta ili oko 10%, ali je istovremeno smanjio i računalni napor (469 vs 1050 što je oko 55% manje). Vrijeme izvođenja je 2434.344 sek što je u odnosu na *ga50* gotovo dvostruko brže.



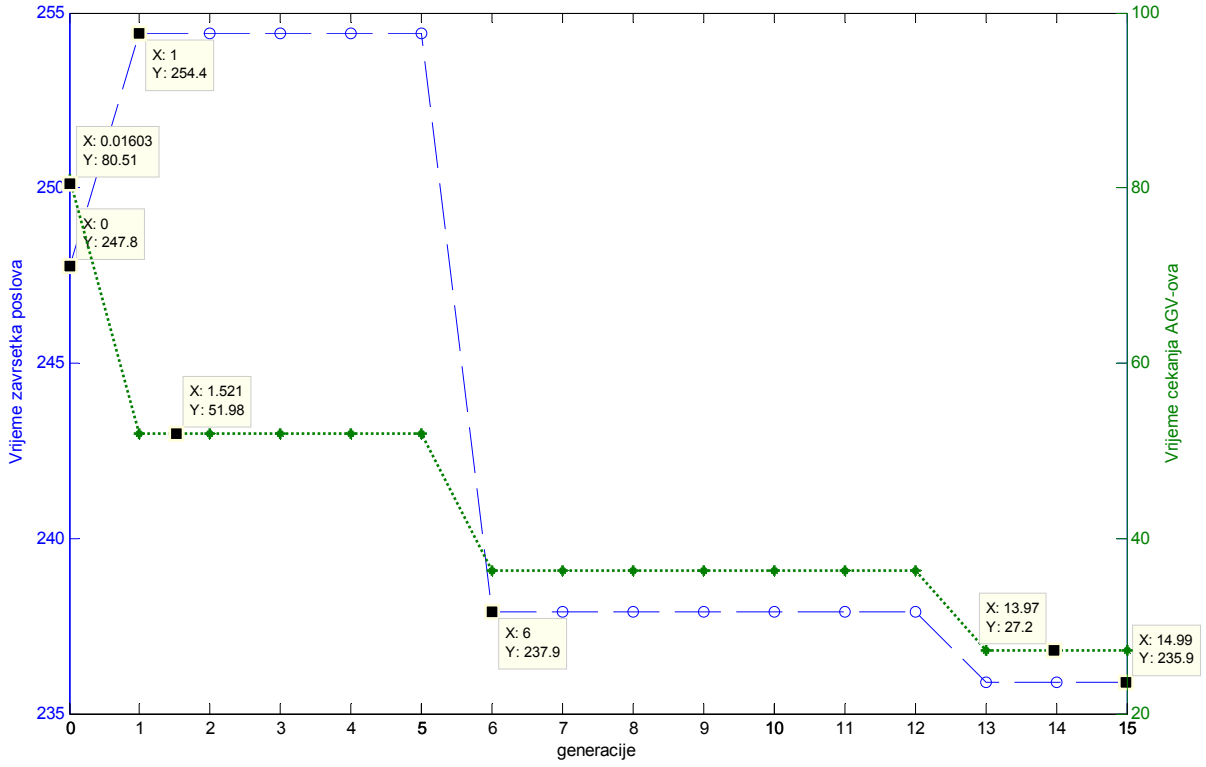
Slika 7.34 Odnos najboljeg rezultata i broja jedinki po generacijama

```
broj agv-a            broj kontejnera - smjer
-------------------------------------------------------
agv    1:          1-A     8-B         18-A        26-A
agv    2:          2-B     16-B        23-B
agv    3:          3-B     13-B        30-B
agv    4:          4-B     17-B        25-B
agv    5:          5-B     22-B
agv    6:          6-B     14-B        27-B
agv    7:          7-A     15-B
agv    8:          9-B     19-B        29-B
agv    9:          10-A    21-B
agv   10:          11-B    20-A        28-B
agv   11:          12-B    24-B
```

Slika 7.35 Raspored i rute vozila

7.9. Više-ciljni genetski algoritam

U ovom podpoglavlju primijenit će se već opisani algoritam na problem opisan u točki 7.6.2. Treba odrediti raspored poslova AGV-ova tako da se prvo minimaliziraju kašnjenja dizalica, a zatim se minimalizira ukupno vrijeme izvođenja svih poslova. Zbog toga u izrazu (7-3) mora vrijediti $\alpha << \beta$. α i β su težine i postavljene su na vrijednosti $\alpha = 0.1,\ \beta = 0.9$ što je razuman izbor u praksi [VIS06b]. Kako se pokazalo da se najbolji rezultati, obzirom i na računalni napor i na konačno rješenje, postižu primjenom dinamičke veličine populacije, ovaj problem će se testirati tako da se veličina populacije mijenja u skladu s Fibonaccijevim brojevima. Iz slike 7.36 razvidno je da se kroz generacije smanjuje i vrijeme izvođenja svih poslova i vrijeme čekanja AGV-ova, ali se prvo minimalizira vrijeme jer to vrijeme brže konvergira ka optimumu.

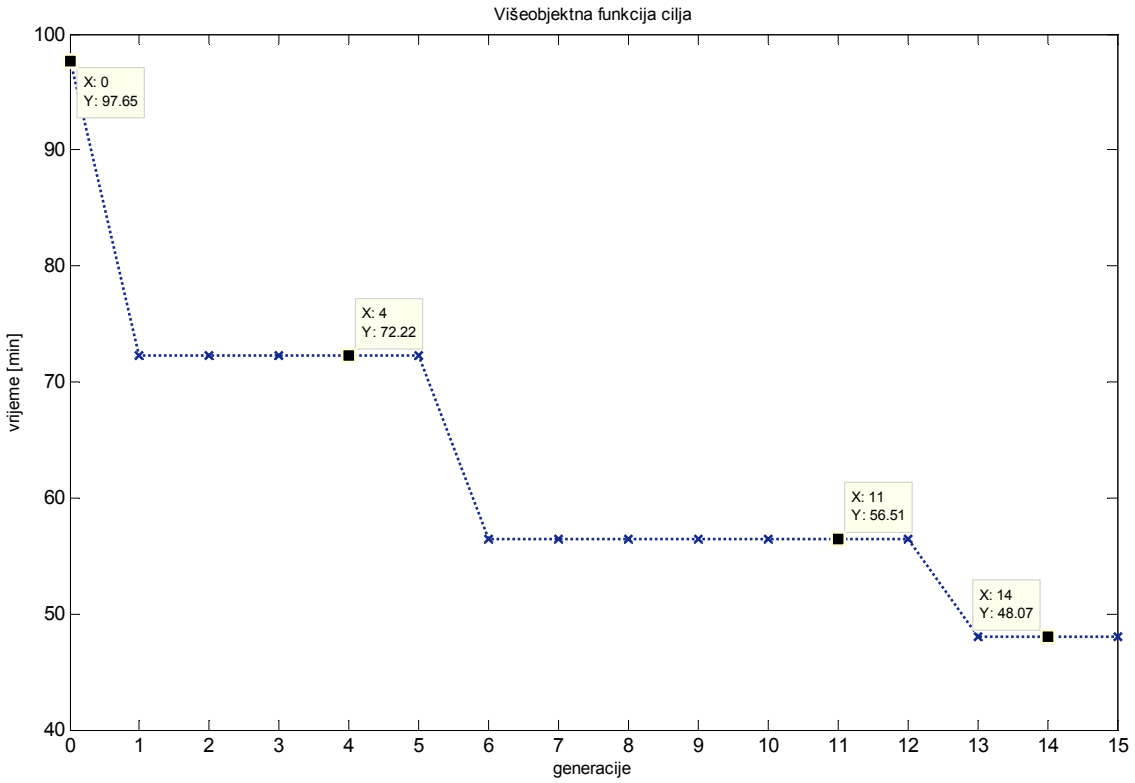


Slika 7.36 Odnos vremena završetka svih poslova i vremena čekanja AGV-ova na dizalice

Slika 7.37 prikazuje rezultat istovremene minimalizacije i funkcije završetka svih poslova i funkcije čekanja AGV-ova na dizalice po generacijama. Vidljivo je da se najbolji rezultat postiže za 48.07 minuta, odnosno vrijeme završetka svih poslova je 235.9 minuta, a ukupno vrijeme čekanja svih AGV-ova na dizalice je 27.2 minute. Dakako, da ovo vrijeme izvođenja svih poslova nije najkraće, ali sada korisnik mora odlučiti što mu je važnije; da maksimalno iskoristi resurse i tako smanji cijenu troškova resursa ili smanjiti vrijeme obavljanja poslova i tako omogućiti da brod što prije napusti luku.

Naime, iz slike 7.38 vidljivo je da je potrebno 11 AGV-ova za izvođenje svih poslova, koji su 13 puta vozili smjerom A, kako bi se postigao ovaj optimum. Dakle od 30 kontejnera koji su se trebali sa skladišta prebaciti do željeznice ostalo ih je još 17. Kada se optimaliziralo

samo vrijeme obavljanja svih poslova, iz slike 7.27 je razvidno da je 11 AGV-ova raspoređeno i da su 8 puta išli smjerom A. Dakle na skladištu je ostalo 22 kontejnera koje još treba prebaciti do željeznice. Korisnik mora odabrati povoljniji raspored, ovisno o tome što mu je važnije.



Slika 7.37 Evolucijski razvoj funkcije cilja

broj agv-a	broj kontejnera - smjer			
agv 1:	1-B	8-A		
agv 2:	2-B	18-B	25-B	30-A
agv 3:	3-A	12-A	24-A	
agv 4:	4-A	14-B	22-A	29-A
agv 5:	5-B	13-A	28-B	
agv 6:	6-A	17-A		
agv 7:	7-B	19-B	27-A	
agv 8:	9-B	20-B		
agv 9:	10-B	21-B		
agv 10:	11-B	16-B	26-A	
agv 11:	15-B	23-B		

Slika 7.38 Raspored poslova AGV-ova

7.10. Sažetak poglavlja

U ovom poglavlju složeni kombinatorni problem je riješen integracijom MRF1 PM i GA. Problem rasporeda poslova AGV-ova uključuje iskrcavanje kontejnera s broda, transport do mjesta u skladištu i transport kontejnera iz skladišta do željeznice. Glavni cilj je minimalizirati vrijeme koje brod provede privezan u luci, kao i vrijeme čekanja AGV-ova na dizalice, jer se na taj način povećava protočnost cijelog terminala. Sporedni ciljevi su minimalizirati broj AGV koje treba uključiti u proces i maksimalizirati broj kontejnera koje treba prebaciti iz skladišta do željeznice.

Odabrana je MRF1 klasa Peterijevih mreža jer ta vrsta mreža pruža matričnu metodu za predviđanje i izbjegavanje konflikata i zastoja u sustavu, te se tako učinkovito upravlja realnim sustavom. Naime, jedna od metoda analize stanja sustava u kojima može doći do zastoja jeste testiranje ponašanja sustava simulacijom. Ovdje prikazan simulacijski model matricama. Ono što je korisno i različito od nekih drugih pristupa je to što je matricama dodana vremenska komponenta. Na ovaj način se u svakom trenutku može pratiti stanje izvođenja poslova i stanje resursa.

Kako je problem složen zbog velikog broja kombinacija rasporeda poslova, bez imalo gubitka općenitosti, algoritam je primijenjen i testiran na jednom segmentu realnog sustava. Rezultati pokazuju da sustav pouzdano i ekonomično izvršava slijed zadataka što je važno svugdje, bez obzira o kojoj vrsti sustava je riječ. Kako bi se naglasila važnost balansiranja između optimalnog rješenja i vremena izvođenja algoritma, problem je testiran sa statičkom i dinamičkom veličinom populacije. Rezultati pokazuju da se povećanjem veličine populacije može dobiti optimalnije rješenje, ali se daleko više povećava vrijeme izvođenja algoritma i veliki je izazov primijeniti takav algoritam u realan sustav. Zbog toga, mišljenja smo da je primjenom dinamičke veličine populacije moguće poboljšati rezultate, a da se pri tome drastično ne poveća vrijeme izvođenja algoritma. Mehanizam promjene veličine populacije koji se temelji na Fibonaccijevim brojevima, koji je implementiran u poglavlju i ovdje je uveden pokazao se jako uspješnim. Odnos između veličine populacije i najboljeg rješenja u populaciji i ovdje je demonstriran.

8. ZAKLJUČAK

Sustavi koji, s jedne strane, pouzdano i sigurno izvršavaju slijed poslova, a s druge strane su ekonomični sa stajališta cijene i troškova, važni su u bilo kojoj aplikacijskoj domeni. Za takove sustave se podrazumijeva da će izvršiti zadani slijed poslova i da će izbjeći stanja koja bi ga dovela do potpunog zastoja ili neke druge neželjene situacije koja može dovesti do materijalnih ili čak ljudskih gubitaka. Ekonomičnost sustava podrazumijeva racionalnost obzirom na vrijeme izvođenja i iskoristivosti resursa, odnosno što manje troškove prilikom izvođenja poslova.

PMGA model, koji integrira Petrijevu mrežu i genetski algoritam, predstavljen u ovoj disertaciji je doprinos pronalaženju optimalnog načina upravljanja ovakvih sustava. Predloženi model je unaprjeđenje postojećih modela integracije, jer primjena matričnog modela na jednostavniji i učitljiviji način opisuje sustav, a matematička algebra omogućuje jednostavni proračun upravljanja konfliktima i zastojima. Također, iz predloženog modela pomoću istih matrica može se naći optimalni raspored izvođenja poslova i raspored dodjeljivanja resursa korištenjem genetskog algoritma. Također, kako bi se uspostavila ravnoteža između pronađenog optimalnog rješenja i vremena izvođenja algoritma, u ovom prijedlogu GA radi s varijabilnom veličinom populacije. Predložen je i mehanizam koji određuje veličinu populacije koristeći Fibonaccijeve brojeve.

U ovoj disertaciji opisana je metodologija višekriterijskog problema rasporeda poslova pomoću metode genetskih algoritama (GA). Iznesen je pregled metoda evolucijskog računanja, kao i detaljan opis modeliranja i simulacije sustava s diskretnim događajima pomoću Petrijevih mreža (PM). Temeljem toga je pomoću metodologije PM napravljen simulator realnog sustava, pomoću kojeg su u GA razvijeni različiti rasporedi poslova. Matrična metoda MRF_1 PM omogućava vizualno predstavljanje optimalnog rasporeda poslova simuliranog sustava.

Također, ono po čemu se ova integracija razlikuje od prethodnih je ta što integrirani GA zajedno s procedurom za generiranje rasporeda razvija parametrizirani aktivni raspored. Pojam parametriziranog rasporeda prvi put je uvedeno u [GON08], ali se u ovom radu prvi put razvija iz modela PM. Pojam parametriziranog aktivnog rasporeda se temelji na tome da se utječe na vrijeme kašnjenje poslova. Ako je maksimalno dopustivo vrijeme kašnjenja preveliko onda je i prevelik prostor rješenja koje GA mora pretražiti. Dakle, kako bi se reducirao prostor pretraživanja rješenja i kako bi se kontroliralo vrijeme kašnjenja poslova, primijenjen je koncept parametriziranog aktivnog rasporeda.

U metodologiji GA je uz poznate principe evolucije jedinki, razvijen novi način određivanja broja jedinki u populaciji primjenom Fibonaccijevih brojeva. Rezultati testiranja pokazuju da se poboljšanje makespama može postići povećanjem broja jedinki u populaciji (occ. 100 jedinki). Međutim to povlači i duže vrijeme izvođenja, odnosno poboljšanje funkcije cilja je oko 0,2% dok se vrijeme izvođenja algoritma može povećati i preko 300%. Krajnja odluka je na korisniku, ali kako se zahtijeva svakodnevna optimalizacija, korisnik će se vjerojatno odlučiti na kraće vrijeme izvođenja algoritma i nešto duže vrijeme izvođenja poslova u sustavu. Međutim, što je problem rasporeda poslova složeniji to je duže vrijeme izvođenja, te veći računalni napor. Kako bi se uspostavila ravnoteža između vremena izvođenja, odnosno računalnog napora i makespama, u ovom radu je testiran algoritam s

dinamičkom veličinom populacije. Broj jedinki u populaciji ovisi o vrijednosti funkcije cilja te generacije populacije i predložen je novi mehanizam koji računa potreban broj jedinki pomoću Fibonaccijevih brojeva. Rezultati testiranja su se pokazali dobrim i obećavajućim.

8.1. Ocjena rezultata i znanstveni doprinos

U ovoj disertaciji uspješno je povezana metoda višekriterijskog odlučivanja, genetski algoritam za određivanje rasporeda poslova u realnom sustavu s diskretnim događajima i vizualna simulacija pomoću matrične metode MRF_1 PM. Kod ove klase PM mjesta predstavljaju i poslove i resurse. Opće PM nisu dovoljne za potpunu strukturnu analizu sustava jer ne uzimaju u razmatranje temporalno ponašanje sustava koje je veoma važno za izvođenje poslova u sustavu. Zato je u ovom radu sustav modeliran P-vremenskom MRF_1 PM, gdje je vrijeme pridruženo mjestima.

Korištenjem i/ili matrične algebre, analiziraju se strukturne značajke Petrijeve mreže (kružna čekanja, P-invarijante, kritični sifoni i podsustavi, ključni resursi) i na nekoliko načina moguće je izbjeći stanja zastoja:

1) Zastoji prve i druge razine se mogu izbjeći održavanjem broja oznaka u kritičnim podsustavima i osiguravanjem da ključni resurs ne ostane zadnji slobodni resurs u sustavu. To se postiglo dodavanjem kontrolnih mjesta koja blokiraju okidanje pojedinih prijelaza i ograničavaju broj oznaka u kritičnom podsustavu dodavanjem ili uklanjanjem oznaka u kontrolnim mjestima.
2) Uporaba pravila za određivanje prioriteta kako bi se izbjegao konflikt između dva ili više poslova koji trebaju zajednički resurs.
3) Brisanjem nezadovoljavajućeg testiranog rasporeda i biranjem novog kromosoma među kandidatima kao mogući novi raspored.

Dakle, hipoteza H1, *da je moguće razviti učinkovit GA za određivanje rasporeda poslova uz simulaciju PM je potvrđena i kroz studije slučaja je pokazano da predložena integracija znatno pridonosi poboljšanju u upravljanju rasporedom poslova u realnom sustavu.*

Zbog složenosti sustava i velikog broja poslova koji se dodjeljuju resursima, u ovom radu nije primijenjena direktna reprezentacija kromosoma u GA. Zato je svaki kromosom kreiran kao konačan niz slučajnih realnih brojeva i ovakav pristup spada u indirektnu reprezentaciju. Za ovaj model koji integrira PM i GA, indirektna reprezentacija ima određene prednosti nad direktnom, kao što su jednostavnost strukture kromosoma, jednostavnost GA operatora, kraće vrijeme izvođenja i manji računalni napor. Kada bi se koristila direktna reprezentacija onda bi broj gena u kromosomu bio barem onoliki koliki je broj poslova u sustavu. Primjerice za sustav kanala iz točke 6.7 broj gena bi iznosio 50, a kod AGV sustava broj gena bi odgovarao broju kontejnera koje treba transportirati (može bit i nekoliko stotina).

Nakon što su definirani, posebno kreiranom procedurom i matricom koja sadrži vremena pridružena mjestima u PM, geni se dekodiraju u prioritete i vremena kašnjenja poslova kao i vremena raspoloživosti resursa. Ovako dekodirani kromosom se prosljeđuje u

proceduru kojom se generira vrijeme okidanja pojedinih prijelaza u PM, odnosno generira se raspored izvođenja poslova u sustavu. Također, broj gena u kromosomu se odredi iz matrica PM koje opisuju resurse i poslove.

Dakle i pretpostavka H2, *da je moguće iz Petrijeve mreže generirati kromosome za potrebe genetskog algoritma je potvrđena.*

U ovom radu se kao primjer sustava s diskretnim događajima razmatrao kontejnerski terminal, te raspored AGV vozila. Kako bi se održala konkurentska prednost i povećala učinkovitost kontejnerskog terminala nužno je odrediti odgovarajući broj AGV-a i odrediti dobre strategije rasporeda za ta vozila. Kao studij slučaja uzet je kontejnerski terminal luke Koper. To je terminal koji nije potpuno automatiziran, odnosno nema AGV-a, ali ima stratešku važnost za transport Jadranskim morem. U tom kontekstu razvijen je algoritam predloženog modela integracije za taj terminal kako bi se poboljšala ukupna produktivnost i smanjilo kašnjenje u nizu poslova tipa ukrcaj-iskrcaj, uzevši u obzir zastoje, konflikte, prioritete i dr. Cilj je odrediti optimalan broj potrebnih AGV-ova i odrediti njihov raspored. Krajnji ciljevi se odnose na optimalizaciju vremena obrade ili na minimalizaciju broja AGV-a koji su uključeni u sustav te održavaju nesmetani protok kontejnera u sustavu, a istovremeno minimaliziraju ukupno vrijeme izvođenja svih poslova.

Dok su ranija istraživanja problemu rasporeda AGV vozila u kontejnerskim terminalima pristupala kao statičkom problemu, tj. broj AGV-a potrebnih za opsluživanje plovila u lukama je konstantan, u ovom radu tom problemu se pristupilo kao složenom dinamičkom problemu. Naime, potreban broj AGV-ova se dinamički odredio kroz procedure generiranja rasporeda poslova i taj broj je ovisio o broju kontejnera koji se trebaju transportirati. Drugi kriterij je bio što bolja iskoristivost resursa, odnosno smanjiti vrijeme čekanja AGV-ova i dizalica kao jako skupih resursa.

Ranija istraživanja u problem rasporeda AGV-a nisu integrirala i problem zastoja u sustavu. U ovim radom predloženom algoritmu optimalnog rasporeda AGV-a uzeta je u obzir i važnost predviđanja i izbjegavanja zastoja u sustavu.

Prema tome i hipoteza H3, *da je moguće sustav AGV-a modelirati vremenskim Petrijevim mrežama, te razviti genetski algoritam za rješavanje više-ciljnog problema optimalizacije rasporeda kojim će se pridružiti operacije AGV vozilima, odrediti slijed operacija za svako vozilo, minimalizirati broj AGV vozila, optimalizirati ukupno vrijeme izvođenja svih poslova i optimalizirati ukupni troškovi je potvrđena.*

Temeljem navedenog, cilj uporabe metode genetskih algoritama i simulacije pomoću Petrijeve mreže za rješavanje višekriterijskog problema rasporeda u sustavima s diskretnim događajima u potpunosti je ostvaren.

DODATAK

D1. Osiguravanje nenegativne vrijednosti funkcije dobrote

U velikom broju optimizacijskih problema vrijednost $f(v)$ za pojedino rješenje *v* je oblika:

$$\forall v \quad (f(v) \geq 0),$$

a problem je $f(v)$: odrediti takav $f(v^*) = max_v f(v)$.

Za ovakav tip problema maksimalizacije s nenegativnom funkcijom cilja može se izravno koristiti funkciju cilja kao funkciju dobrote, odnosno

$$d(v) = f(v) \tag{D-1}$$

U problemima maksimalizacije gdje $f(v)$ može poprimiti i negativnu vrijednost ne može se koristiti gornja jednadžba jer bi se u tom slučaju dobile negativne vrijednosti vjerojatnosti selekcije.

U tom slučaju koristi se sljedeća transformacija:

$$d(v) = \begin{cases} f(v) + C_{min}, & \text{ako je } f(v) + C_{min} > 0 \\ 0, & \text{inače} \end{cases} \tag{D-2}$$

gdje je C_{min} apsolutna vrijednost jedne od slijedećih vrijednosti:

- minimalna vrijednost koju $f(v)$ može imati (ako je poznata),
- minimalna vrijednost $f(v)$ u trenutnoj ili zadnjih *n* generacija,
- funkcija varijance $f(v)$ u trenutne populacije, primjerice:

$$\mu(f(v)) - 2\sqrt{\sigma^2(f(v))}$$

gdje je: μ - srednja vrijednost, a σ^2 - varijanca.

U problemima minimalizacije (npr. cijene, troškova ili greške) problem se može matematički postaviti kao:

$$E(v^*) = min_v E(v),$$

U slučaju da je $\forall v \quad (E(v) \leq 0)$, može se za funkciju dobrote uzeti:

$$d(v) = -E(v) \tag{D-3}$$

U slučaju da $-E(v)$ može biti i negativan koristi se transformacija:

$$d(v) = \begin{cases} C_{max} - f(v), & \text{ako je } C_{min} > f(v) \\ 0, & \text{inače} \end{cases} \tag{D-4}$$

gdje je C_{max} jedna od slijedećih vrijednosti:

- maksimalna vrijednost koju $f(v)$ može imati (ako je poznata),
- maksimalna vrijednost $f(v)$ u trenutnoj ili zadnjih n generacija
- funkcija varijance $f(v)$ u trenutne populacije, primjerice:

$$\mu(f(v)) + 2\sqrt{\sigma^2(f(v))}.$$

Nakon što se na neki od navedenih načina dobije vrijednost dobrote ista se može koristiti u postupku selekcije. Vjerojatnost odabira *i*-te jedinke dana je relacijom:

$$p_{sel}^{i} = \frac{d_i}{\sum d} \qquad \text{(D-5)}$$

gdje su:

d_i - vrijednost dobrote *i*-te jedinke,

$\sum d$ - suma dobrota svih jedinki populacije.

D2. Fibonaccijev niz i broj Phi

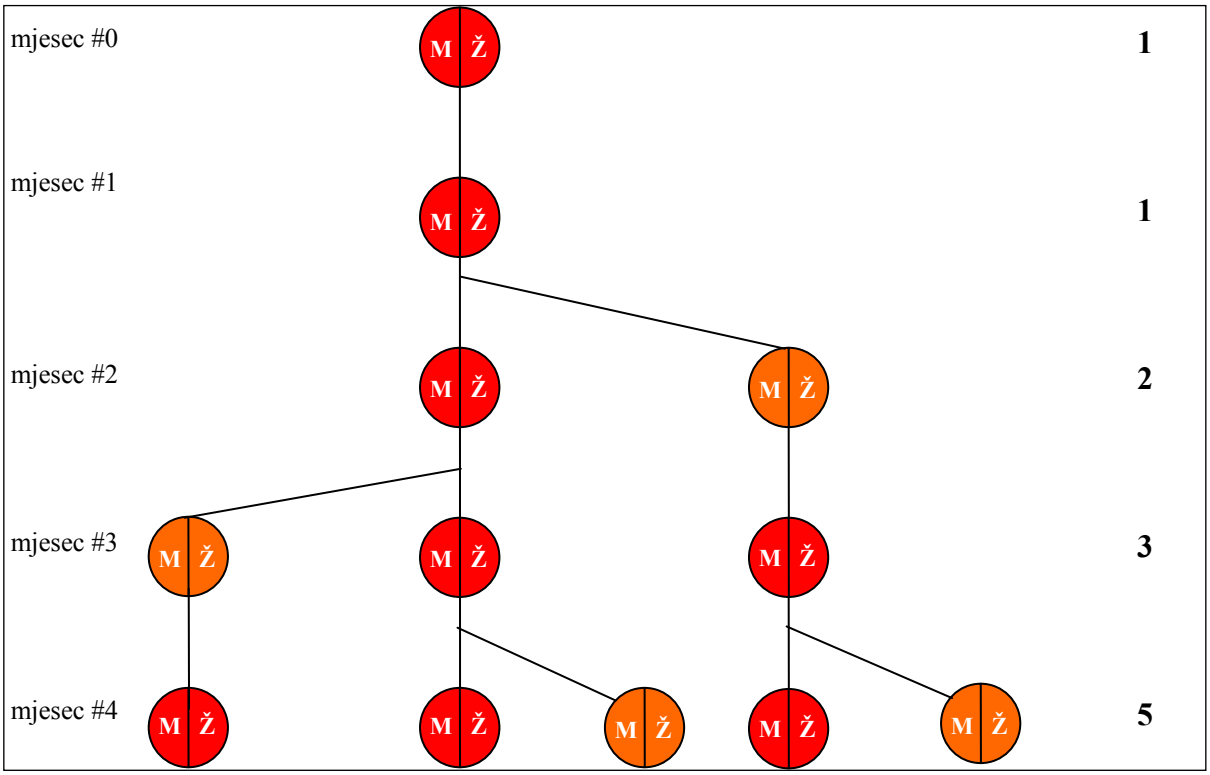
"Najveći matematičar srednjeg vijeka", Leonardo Pisano, poznat kao Fibonacci, otkrio je neobično svojstvo matematičkog niza koje danas nosi njegovo ime. Njegovim najvećim doprinosom matematici smatra se djelo *Liber Abaci* ("Knjiga računanja", 1202.g.), koje je utjecalo na uporabu arapskog (decimalnog) brojevnog sustava u Europi.

Jedan od primjera koji je Leonardo naveo u Liber Abaci kako bi predstavio primjenu arapskih brojeva doveo je do niza kojeg danas poznajemo kao Fibonaccijev niz. Iako izvorni tek varira od jednog izvora do drugog, britannica.com navodi sljedeću verziju: "*Jedan je čovjek stavio par zečeva u izolirani prostor, te oni nakon mjesec dana dostižu spolnu zrelost i novi par se rađa. Koliko parova zečeva će biti u tom prostoru nakon jedne godine, uz pretpostavku da niti jedan zec nije uginuo*?"

U početku postoji 1 par zečeva (početni uvjeti, mjesec # 0). Nakon mjesec dana zečevi će dostići zrelost, no još se neće razmnožiti. Tako na početku prvog mjeseca u prostoru je još samo jedan par. U toku prvog mjeseca će se razmnožiti i na početku drugog mjeseca bit će dva para zečeva. Za vrijeme drugog mjeseca početni par će se ponovo razmnožiti, dok drugi par sazrijeva i ne mogu dati djecu, tako da će na početku trećeg mjeseca biti tri para zečeva. Od ta tri para u toku trećeg mjeseca dva će se razmnožiti i kao rezultat bit će pet parova na početku četvrtog mjeseca. To najbolje prikazuje slika 0.1.

Pa koliko je onda novorođenih parova svaki mjesec?

Kako treba 2 mjeseca da svaki novi par daje mlade, svaki stari par koji je živ do prije 2 mjeseca, daje nov par. Prema redoslijedu riječi, broj novorođenih parova svakog mjeseca jednak je broju parova već živih do prije 2 mjeseca. Zatim trebamo pronaći broj parova koji su bili živi prije novorođenih. Trebalo bi biti očito da je to broj parova živih prije mjesec dana. Drugim riječima, da se nađe ukupan broj parova zečeva, jednostavno se zbroji broj parova koji su bili živi u prethodna dva mjeseca.



Slika 0.1 Broj parova zečeva po mjesecima

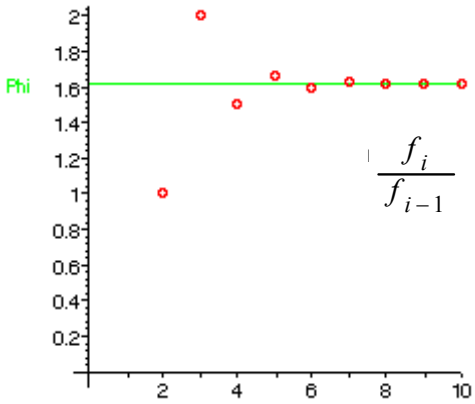
Taj zamišljen primjer prikazuje Fibonaccijev niz kojeg možemo definirati kao niz brojeva koji započinju brojevima 1, 1 i dalje se niz nastavlja u beskonačnost, pri čemu se svaki sljedeći broj računa kao zbroj prethodna dva broja u niz.

Fibonaccijev niz se definira kako slijedi:

$$fb_i = \begin{cases} 0 & \text{ako je } i = 0; \\ 1 & \text{ako je } i = 1; \\ fb_{i-1} + fb_{i-2} & \text{ako je } i > 1. \end{cases}$$

Tako dobiven niz, sastavljen je od brojeva $1,1,2,3,5,8,13,21,34,55,89,\ldots f_i$.

Bitna značajka Fibonaccijevog niza, za koju nije sigurno da ju je bio svjestan i sam Fibonacci, je ta da se omjer bilo kojeg člana Fibonaccijeva niza i prvog nižeg člana približava broju φ=1.61803398, dok je omjer sa prvim većim članom niza približno 0.618. Točnost se povećava kako niz napreduje (slika 0.2). Broj φ naziva se *Zlatna rez* (engl. *Golden Ratio*).



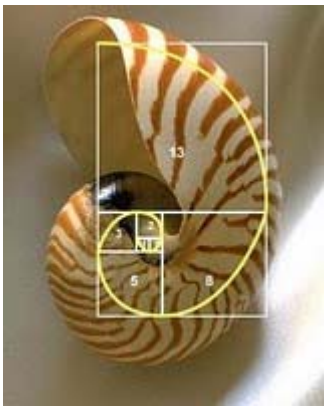
Slika 0.2 Kretanje Fibanoccijevih brojeva [2]

Drugi važan omjer je između bilo kojeg člana niza i člana koji je za dva mjesta veći u nizu, a koji teži broju 0.382. Sami Fibonaccijevi brojevi ovdje neće biti važni, već odnos između članova niza.

Važnost Zlatnog reza pokazala se u 19. stoljeću, kada su znanstvenici otkrili povezanost zlatnog reza s prirodom. Tako je otkriveno da se odnosi mjera kod biljaka, životinja i ljudi zapanjujućom preciznošću približavaju broju φ.

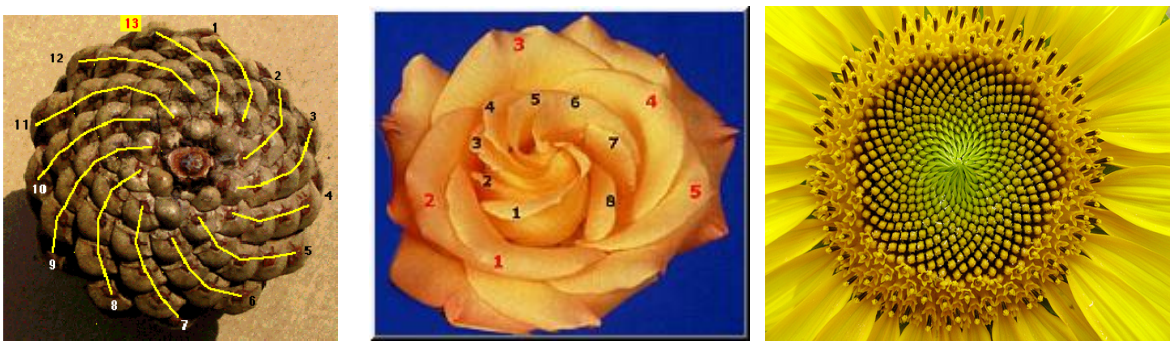
Slijedi nekoliko primjera broja φ i njegove povezanosti sa Fibonaccijem i prirodom:

- Zlatni rez nalazimo u građi egipatskih piramida. Zbroj površina stranica piramide u omjeru prema površini baze piramide daje φ.
- U pčelinjoj zajednici, košnici, uvijek je manji broj mužjaka pčela nego ženki pčela. Kada bi podijelili broj ženki sa brojem mužjaka pčela, uvijek bi dobili broj φ.
- Puževa kućica u svojoj konstrukciji ima spirale (slika 0.3 i kao da je izrastao po zakonu zlatnog reza. Pogleda li se pravokutnik i slijedeći dinamičku spiralu umanjuju li se pravokutnici do najmanjega uočava se da je puževa kućica izrasla iz Fibonaccievog niza: 2, 3, 5, 8, 13. Takva struktura se naziva *phyllotaxis spirala*. Kada bi izračunali odnos svakog spiralnog promjera prema slijedećem dobili bi broj φ. Kod takve spirale, dužine gornjeg i donjeg dijela spirale uvijek će zapravo biti dva susjedna broja Fibonaccijevog niza. Isto je i s lijevim i desnim dijelom.



Slika 0.3 Puževa spirala [3]

- Fibonaccijevi brojevi nalaze se kao spinovi (okreti) na češeru. Na slici 0.4(a) su spinovi popraćeni žutim krivuljama.
- Kada se biljka grana, obično se grana po Fibonaccijevim brojevima. Broj latica na nekom cvijetu je, obično, 3, 5 ili 13, no ima ih i sa 55 i 89 – neke porodice tratinčica. Malo koja biljka ima 4 ili 6 latica (slika 0.4(b)).
- Sjemenke na cvijetu suncokreta rastu u obliku spirala. Može se vidjeti 55, 89, a možda još i više spirala koje prelaze jedne preko drugih. Međusobni odnosi promjera rotacije je broj φ.

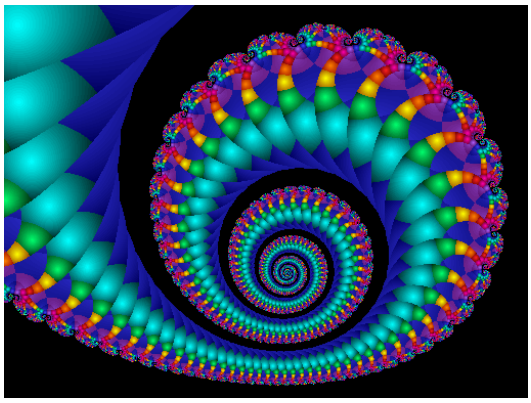


(a) (b) (c)

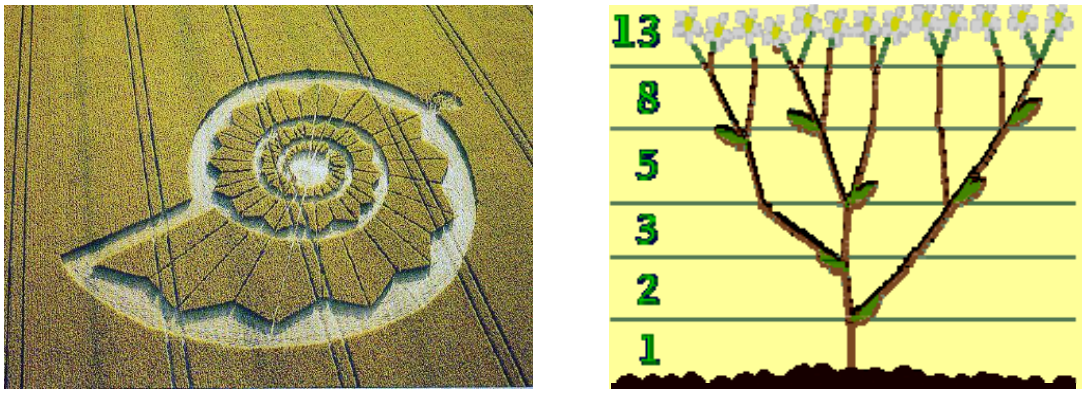
Slika 0.4 Primjeri pojave Zlatne spirale u prirodi: (a) spinovi na češeru; (b) latice cvijeta ruže; (c) sjemenke suncokreta [30]

U prirodni postoje još mnogi drugi primjeri pojave Zlatnog reza, primjerice, uzorci rasta bakterija, mikroskopsko kreiranje kvazi kristala, životinjski rogovi, rep komete se spiralno udaljuje od sunca. Čak se pokazalo da i ljudsko tijelo sadrži elemente Zlatnog reza. Prosječna duljina tijela od pupka do vrha glave je 0.618 duljine tijela od stopala do pupka, dok je duljina tijela od stopala do pupka prosječno jednaka 0.618 duljine cijelog tijela osobe.

Ako je φ sila koja potiče rast u svemiru*, moguće je da je to poticaj za porast učinkovitosti GA. Ako se čovjekov napredak temelji na produkciji i reprodukciji, onda nije nemoguće, čak je i razumno očekivati, da je takav napredak spiralnog oblika i da se takav oblik može primijeniti i na proces reprodukcije u GA.*



[30] Preuzeto sa: http://bs.wikipedia.org/wiki/Fibonaccijev_broj.



Slika 0.5 Neki primjeri Fibonaccijevih brojeva u prirodi [3]

D3. Uvjeti zastoja sustava

Kako bi se izvela pravila za izbjegavanje zaglavljenja za MRF_1 PM, potrebno je objasniti nekoliko pojmova kao što su kružna čekanja, kružno zaglavljenje, kritični resursi i kritični podsustavi, citirani su u [LEE07]. Kako bi se objasnila procedura izbjegavanja zaglavljenja te definicije su niže navedene.

Za bilo koja dva resursa $r_i, r_j \in R$ kaže se da r_i *čeka* na r_j (oznaka $r_i \to r_j$) ako otpuštanju resursa r_i neposredno prethodi dostupnost resursa r_j.

Definicija 0.1 Kružno čekanje (Lewis, 1998)[31]

Kružno čekanje (engl. *Circular Wait*, pokrata: *CW*) je skup resursa $\{CW \subset R(resursa) : |CW| > 1\}$, tako da za bilo koji par resursa $\{r_i, r_j\} \subset CW$, postoji dva moguća slijeda resursa; jedan sadrži slijed od r_i do r_j, a drugi od r_j do r_i.

Definicija 0.2 Kritični podsustav (Lewis, 1998)

Kritični podsustav kružnog čekanja $[J_0(CW)]$ u MRF_1 je skup svih mjesta koji predstavljaju poslove koji trebaju resurse u *CW*, osim onih mjesta (poslova) kod kojih mjesta koja slijede trebaju one resurse koji nisu uključeni u *CW*. Dakle,

$$J_0(CW) = \{p \in J(CW) : p\bullet \in \bullet r_{io}\}, \text{ gdje je } \bullet r_{io} = \{x \in T : \bullet x \cap CW \neq \check{R}\},$$

gdje je $J(CW)$ je skup poslova kružnog čekanja, odnosno, skup mjesta koja predstavljaju one poslove koji zahtijevaju resurse u CW.

[31] Lewis, F.L., Gurel, A., Bogdan, S., Doganalp, A., Pastravanu, O.: "*Analysis od deadlock and circular waits using matrix model for flexible manufacturing systems*", Automatica, vol.34,No. 9, pp. 1083-1100,1998. (citirano u [LEE07])

Definicija 0.3 Kružno blokiranje (Lewis, 1998)

Kružno blokiranje (engl. *Circular Blocking*, pokrata: CB) je stanje PM gdje za $CW = \{r_1, r_2, \ldots, r_q\}$ vrijedi:

1. $m(CW) = 0$ (nema oznaka u *CW*), gdje je $m(CW)$ broj oznaka u kružnom čekanju;
2. Za svaki $r_i \in CW$ i svaki $p \in J(r_i)$ takav $m(p) \neq 0$ vrijedi $p \in J_0(CW)$.

To znači da svi poslovi u kritičnom podsustavu trebaju sve resurse koji su u kružnom čekanju.

Definicija 0.4 Kritični sifon (Lewis, 1998)

Kritični sifon S je minimalni sifon koji ne sadrži skup mjesta koja čuvaju ukupan broj inicijalnih oznaka u svakom dostupnom stanju. U MRF_1 kritični sifon se dobije kao

$$S = CW \cup J(CW). \tag{D-6}$$

Teorem 0.1 Uvjet da sustav nije u stanju kružnog blokiranja (Lewis, 1998)

Za zadanu MRF1 PM ne postoji kružno blokiranje ako i samo ako za svako kružno čekanje CW i za svako dostupno stanje vrijedi $\left[m(J_0(CW)) = m_0(CW)\right]$, gdje je $m_0(CW)$ inicijalni broj oznaka u kružnom čekanju.

Definicija 0.5 Cikličko kružno čekanje, kritični resurs [BOG06]

Ukoliko je kružno čekanje sastavljeno od više kružnih čekanja tada se takvo kružno čekanje zove *ciklično kružno čekanje* (engl. *cyclic circular wait*), a zajednički resurs zove se *kritični resurs.*

LITERATURA

[ABD98] Abdallah, I.B., ElMaraghy, H.A.: "*Deadlock prevention and avoidance in FMS – a Petri net based approach*", International Journal of Advanced Manufacturing Technology, Vol. 14, No. 10, pp. 704-715, 1998.

[BAC97] Bäck, T., Fogel, D.B., Michalewicz Z. (eds.): "*Handbook of Evolutionary Computation*", Oxford University Press, New York, pp. 988, 1997.

[BER01] Bernik, I.: *"Večkriterijsko razporejanje z genetskimi algoritmi in vizualno simulacijo s Petri mrežami"*, Doktorska disertacija, Univerza u Mariboru Fakulteta za Organizacijske Vede, Kranj, 2001.

[BIE90] Biegel, J.E., Davern, J.J.: "*Genetic algorithm and job shop scheduling*", Computers and Industrial Engineering 19, pp. 81–91, 1990.

[BIE99] Bierwirth C., Mattfeld, D.C.: "*Production Scheduling and Rescheduling with Genetic Algorithms*", Evolutionary Computation 7(1), pp. 1-17, 1999.

[BOG06] Bogdan, S., Lewis, F.L., Kovačić, Z., Mireles, J. Jr.: "*Manufacturing Systems Control Design: A Matrix-based Approach*", Springer-Verlag, London, 2006.

[BÖS02] Böse1, J., Reiners, T., Steenken, D., Voß, S.: "*Vehicle dispatching at seaport container terminals using evolutionary algorithms"*, Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, R.H. Sprage (Ed), IEEE, Piscataway, pp. 1-10, 2002.

[BRU94] Brucker, P., Jurisch, B., Sievers, B.: "*A branch & bound algorithm for the job shop problem",* Discrete Applied Mathematics, 49, pp. 107-127, 1994.

[BRU95] Brucker, P.: "*Scheduling Algorithms*", Springer, Berlin, 1995.

[BYK03] Bykov, Y., "*Time-Predefined and Trajectory-Based Search: Single and Multiobjective Approaches to Exam Timetabling*", PhD Dissertation, University of Nottingham, 2003.

[CAM93] Camurri, A., Franchi, P., Gandolfo, F., Zaccaria, R.: "*Petri net based process scheduling*: *A model of the control system of flexible manufacturing systems*", Journal of Intelligent and Robotic Systems, 8, pp. 99–123, 1993.

[CAS99] Cassandras, C.G., Lafortune, S.: "*Introduction to Discrete Event Systems"*, Kluwer Academic Publishers, Boston, 1999.

[CAV95] Cavory, G., Dupas, R., Goncalves, G.: "*A genetic approach to solving the problem of cyclic job shop scheduling with linear constraints*", Eur J Oper Res 161(1), pp. 73–85, 2005.

[CAV98] Cavalieri, S.: "*Petri Nets and Genetic Algorithms to Increase Productivity in FMS*", Second International Conference on Knowledge-Based Intelligent Electronic Systems, pp. 21-23, Adelaide, Australia, 1998.

[CHE01] Chen, J.H., Fu, L.C., Lin, M.H., Huang, A.C.: *"Petri-Net and GA-Based Approach to Modeling, Scheduling, and Performance Evaluation for Wafer Fabrication"*, IEEE Transactions on Robotics and Automation, Vol. 17, No. 5, pp. 619-636, 2001.

[CHE93] Chen, C., Yu, D., Zhang, B.: "*Scheduling parallel processing by Petri nets*", Automatic-Control World Congress, 2 (176), pp. 739–742, 1993.

[CHE97] Chen, Y., Leong, T-Y., Ng, J.W.C., Demir, E.K., Nelson, B.L., Simchi-Levi, D.: "*Dispatching automated guided vehicles in a mega container terminal*", The National University of Singapore/Dept. of IE & MS, Northwestern University, 1997.

[COE04] Coelho, R. F.: "*Multicriteria Optimization with Expert Rules for Mechanical Design*", PhD Dissertation, Faculté des Sciences Appliquée, Université Libre de Bruxelles, 2004.

[CRO95] Croce, F.D., Tadei, R., Volta, G., "*A genetic algorithm for the job shop problem*", Computers and Operations Research 22 (1), pp. 15–24, 1995.

[CVE00] Cvetković, D.: "*Evolutionary Multi–Objective Decision Support Systems for Conceptual Design*", PhD Thesis, School of Computing, Faculty of Technology, University of Plymouth, Plymouth, 2000.

[DAG90] Daganzo, C.F.: "*The productivity of multipurpose seaport terminals*", Transport Science 24(3), pp. 205-216, 1990.

[DAV85] Davis, L.: "*Job Shop Scheduling with Genetic Algorithms*", Presented at International Conference on Genetic Algorithms and Their Applications, 1985.

[FAN94] Fang, H. L.: "*Genetic Algorithms in Timetabling and Scheduling"*, Doktorska disertacija, Department of Artificial Intelligence University of Edinburgh, 1994.

[FON93] Fonseca, C.M., Fleming, P.J.: "*Genetic Algorithms for Multiobjective Optimisation: Formulation, Discussion and Generalization*". In Proceedings of the 15th International Conference on Genetic Algorithms, San Mateo, California, pp. 416-423, 1993.

[FRA06] Frankola, T.: "*Rješavanje problema raspoređivanja aktivnosti projekata evolucijskim algoritmima*", diplomski rad br. 1626, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, listopad 2006.

[FU97] Fu, M. C., Hill, S. D.: "*Optimization of discrete event systems via simultaneous perturbation stochastic approximation*", IIE Transactions 29, pp. 233-243, 1997.

[GAN04] Gang, X., Wu, Z.: "*Deadlock-free scheduling strategy for automated production cel*", IEEE Trans Syst Man Cybern Part C 34(1), 113–122, 2004.

[GEN97] Gen, M., Cheng, R.: "*Genetic Algorithms & Engineering Design*", A Wiley-Interscience Publication, New York, 1997.

[GHO04] Ghosh, A., Dehuri, S.: "*Evolutionary Algorithms for Multi-Criterion Optimization: A Survey*", International Journal of Computing & Information Sciences Vol. 2, No. 1, 2004.

[GIR03] Girault, C., Valk, R.: *Petri nets for Systems Engineering – a Guide to Modeling, Verification, and Application*, Springer, Berlin Heidelberg New York, 2003.

[GOL01] Golub, M.: *"Poboljšavanje djelotvornosti paralelnih genetskih algoritama*", doktorska disertacija, Zagreb, 2001.

[GOL02] Golub, M: "*Genetski algoritmi* – Drugi dio", Fakultet elektrotehnike i računarstva, Zavod za elektroniku, mikroelektroniku, računalne i inteligentne sustave, 2002. Dostupno na Internet adresi: http://www.zemris.fer.hr/~golub/ga/ga.html

[GOL89] Goldberg, D.E., "*Genetic Algorithms in Search, Optimization and Machine Learning*", Addison-Wesley, 1989.

[GOL94] Goldberg, D.E.: "*Genetic and Evolutionary Algorithms Come of Age*", Commun. ACM 37, pp. 113–119, 1994. Dostupno na Internet adresi: www.acm.org/dl.

[GOL96] Golub, M.: *"Vrednovanje uporabe genetskih algoritama za aproksimaciju vremenskih nizova"*, magistarski rad, Zagreb, 1996. Dostupno na Internet adresi: http://www.zemris.fer.hr/~golub/magrad.html

[GON05] Goncavles, J.F., Mendes, J.J.M., Resende, M.G.C.: "*A hybrid genetic algorithm for the job shop scheduling problem*", European Journal of Operational Research 167, pp. 77-95, 2005.

[GRO02] Groumpos, P. P., Merkuryev, Y.: "*A Methodology of Discrete-Event Simulation of Manufacturing Systems: An Overview*", Studies In Informatics and Control, Vol. 11, No. 11, pp. 53-60, 2002. Dostupno na Internet adresi: http://www.ici.ro/new_ici/prezentare/revista/sic2002_1/art05.htm

[GUD08] Gudelj, A., Vidačić, S.: "*Scheduling model of an automated guided vehicle system in seaports*", Modern Traffic, Vol. 28, Special Issue, pp. 10-15. Institutes for Mechanical Engineering University of Mostar, Bosnia and Herzegovina, 2008.

[GUD10] Gudelj, A.; Krčum, M.; Twrdy, E.: "*Models and Methods for Operations in Port Container Terminal*", PROMET - Scientific Journal on Traffic and Transportation Research. Vol.22, No. 1, pp. 43-52, 2010.

[GUN05] Günther, H.O., Kim, K.H.: "*Container Terminals and Automated Transport Systems*", Springer Berlin Heidelberg, 2005.

[HID05] Hidalgo, I., Fernández, F.: "*Balancing the Computation Effort in Genetic Algorithms*", The 2005 IEEE Congress, Vol. 2, pp. 1645-1652, IEEE Press, 2005.

[HOR94] Horn, J., Nafpliotis, N., Goldberg, D.E.: "*A niched pareto genetic algorithm for mu-ti-objective optimization*", Proceedings of the 1st IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, volume 1, pp. 82-87, New Jersey, 1994. IEEE Service Center.

[HUA92] Huang, H. P., Chang, P. C.: "*Specification, modeling and control of a flexible manufacturing cell*", International Journal of Production Research, 30 (11), pp. 2515–2543, 1992.

[JEN97] Jensen, K.: "*Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*", Monographs in Theoretical Computer Science, vol 1-3, Springer-Verlag, Berlin 2nd corrected printing, 1997.

[JEN98] Jensen, K.: "*An Introduction to the Pratical Use of Coloured Petri Nets*", In Reisig, W., Rozenberg, G. (eds.); Lectures on Petri Nets II: Applications, Lecture Notes in Computer Science vol. 1492, Springer-Verlag, pp. 237-292, 1998. Dostupno na Internet adresi: http://www.daimi.au.dk/~kjensen/papers_books/rec_papers_books.html

[KEZ04] Kezić, D.: "*Sprječavanje potpunog zastoja u sustavima s diskretnim događajima primjenom Petrijevih mreža*", Doktorska disertacija, Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, 2004.

[KEZ09] Kezić, D., Matić, P., Račić, N.: "*P-invariant Based Petri net Traffic Controller*", 17th Mediterranean Conference on Control & Automation, Greece, pp. 1096-1101, 2009.

[KIM04] Kim, K.H., Bae, J.W.: "*A look-ahead dispatching method for automated guided vehicles in automated port container terminals*", Transportation Science, Vol. 38, No. 2, pp. 224-234, 2004.

[KOF08] Kofjač, D., Kljajić, M.: "*Application of Genetic Algorithms and Visual Simulation in a Real-Case Production Optimization*", WSEAS Transactions on System and Control, Volume 3, No 12, pp. 992-1001, 2008.

[KOV02] Kovačić, Z., Bogdan, S., Krajči, V.: " *Osnove robotike*", Graphis, Zagreb, 2002.

[KOU09] Kouchakpour, P., Zaknich, A., Braunl, T.: "*Dynamic population variation in genetic programming*", Information Sciences, Vol. 179, No. 8, pp. 1078-1091, 2009.

[KOZ92] Koza J.R.: "*Genetic Programming, On the Programming of Computers by Means of Natural Selection*", MIT Press, Cambridge, MA, 1992.

[KRI98] Kristensen, L.M., Christensen, S., Jensen, K.: "*The Practitioner's Guide to Coloured Petri Nets",* International Journal on Software Tools for Technology Transfer, No.2 Springer Verlag, pp. 98-132, 1998. Dostupno na Internet adresi: http://www.daimi.au.dk/~kjensen/papers_books/rec_papers_books.html (pregledano 11.7.2007).

[LAC01] Lacksonen, T.: "*Empirical comparison of search algorithms for discrete event simulation*", Computers & Industrial Engineering 40, pp. 133-148, 2001.

[LEE07] Lee, S., Tilbury, D.M.: "*Deadlock-Free Resource Allocation Control for a Reconfigurable Manufacturing System With serial and Parallel Configuration*", IEEE Transaction on System, Man and Cybernetics-Part C: Applications and Rewiews, Vol. 37, No. 6, pp. 1373-1381, 2007.

[LEE94] Lee, Y., Dicesare, F.: "*Scheduling flexible manufacturing systems using Petri nets and heuristic search*", IEEE Transactions on Robotics and Automation, 10 (2), pp. 998–1003, 1994.

[LEW02] Lewe, J.H., "*A Spotlight Search Method for Multi-Criteria Optimization Problems*", 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Atlanta, Georgia, 2002.

[LLO95] Lloyd, S., Yu, H., Konstas, N.: "*FMS scheduling using Petri net modeling and branch and bound search*", Proceedins of the IEEE International Symposium on Assembly and Task Planning, Pittsburgh, USA, pp. 141–146, 1995.

[MED03] Medaglia, A.L., Fang, S.C.: "*A genetic-based framework for solving (multi-criteria) weighted matching problems"*, European Journal of Operational Research 149, pp. 77-101, 2003.

[MOO96] Moore, K.E., Gupta, S.M.: "*Petri Net Models and Automated Manufacturing Systems: A Survey*", International Journal of Production Research, 34, No 11, pp. 3001-3035, 1996.

[MUD02] Mudnić, E.: "*Genetički algoritam za optimizaciju lociranja kondenzatorskih baterija u distributivnoj mreži*", magistarski rad, Sveučilište u Splitu, Fakultet elektrotehnike, strojarstva i brodogradnje, Split, 2002.

[MUR89] Murata, T.: "*Petri Nets: Properties, Analysis and Applications*", Proc. IEEE, Vol. 77, pp. 541-580, 1989.

[MUR97] Murata, T.: "*Genetic algorithms for multi-objective optimization*", PhD. Thesis, Osaka Prefecture University, Osaka, Japan, February 1997.

[ONA91] Onaga, K., Silva, M., Watanebe, T.: "*On periodic schedules for deterministically timed Petri net systems*", Proceedings of the Fourth International Workshop on Petri Nets and Performance Models, Melbourne, Australia, 210–215, 1991.

[OSK99] Oskarsson, M.: *"A Decision Support System for Scheduling a Shop Floor Work Center"*, Licentiate thesis, Department of Mathematics, Chalmers University Of Technology, Göteborg University, NO 1999-4/ISSN 0347-2809, 1999.

[PES98] Pesenti, R., Ukovich, W., Castelli, L., Longo, G.: "*Quantitative Models and Methods for Locating Logistics Nodes in Maritime Transportation*", Quaderni di Trasporti Europei: Study on Location of Logistic Nodes, No. 8/9, pp.47-58, 1998.

[PET81] Peterson, J., L.: "*Petri Net Theory and Modeling of Systems*", Prentice Hall, Englewood Cliffs, 1981.

[QIU02] Qiu, L., Hsu, W. J., Huang, S. Y., Wang, H.: "*Scheduling and Routing Algorithms for AGVs: a Survey*", International Journal of Production Research, Vol. 40, No. 3, pp. 745-760, 2002.

[RAD97] Radatz, J.: "*The IEEE Standard Dictionary of Electrical and Electronics Terms*", 6th edition, ISBN:1559378336, 1997

[SRI94] Srinivas, N., Deb, K.: "*Multiobjective optimization using nondominated sorting in genetic algorithms*", Evolutionary Computation, 2(3), pp. 221-248, 1994.

[STE04] Steenken, D. et al: "*Container terminal operation and operations research - A classification and literature review*", OR Spectrum, Vol. 26, No. 1, pp.3-49, 2004.

[SWI00] Swisher, J. R., Hyden, P. D., Jacobson S. H., Schruben, L. W.: "*A Survey of Simulation Optimization Techniques and Procedures*", Proceedings of the 2000 Winter Simulation Conference WSC'00, Orlando, Florida, U.S.A, Ed. by Jeffrey A. Joines, Russel R. Barton, Keebom Kang and Paul A. Fishwick, pp. 119-128, 2000.

[TAC97] Tacconi, D.A., Lewis, F.L.: "*A New Matrix Model for Discrete Event Systems: Application to Simulation*", Control Systems Magazine, IEEE, Volume 17, No 5, pp. 62 – 71, 1997.

[TOD97] Todd, D.S., Sen, P.: "*A multiple criteria genetic algorithm for containership loading*", Proceedings of the Seventh International Conference on Genetic Algorithms, East Lansing, MI, Morgan Kaufmann Publishers, 1997.

[TOD98] Todd, D.S., Sen, P.: "*Tackling Complex Job-Shop Problems Using Operations Based Scheduling*", Proceedings of ACDM 98, the International Conference on Adaptive Computing in Design and Manufacture, Plymouth, UK, Springer Publishers, 1998.

[VIS06a] Vis, I.F.A.: "*Survey of research in the design and control of automated guided vehicle systems*", European Journal of Operational Research 170, pp. 677–709, 2006.

[VIS06b] Vis, I.F.A., Bakker, M.: "*Dispatching and layout rules at an automated container terminal*", http://zappa.ubvu.vu.nl/20050008.pdf (pregledano 19.10.2007).

[VUČ07] Vučijak, B.: "*Višekriterijalna optimizacija u upravljanju prostorom*", Prostor, znanstveni časopis za arhitekturu i urbanizam, Vol.15 No.1(33), pp. 109-117, lipanj 2007.

[WAN98] Wang, J.: "*Timed Petri Nets – Theory and Application*", Kluwer Academic Publishers, Boston, 1998.

[WAN07] Wang, J.: "*Petri Nets for Dynamic Event-Driven System Modeling"*, In Handbook of Dynamic System Modeling, Paul A. Fishwick (ed.), CRC Press, Boca Raton, pp. 24.1 – 24.16, 2007.

[YIM93] Yim, D.S., Linn, R.J.: "*Push and pull rules for dispatching automated guided vehicles in a flexible manufacturing system*", Int J Prod Res 31(1),pp.43–57, 1993.

[ZEN06] Zenzerović, Z.: "*Kvantitativne metode u funkciji optimalnog funkcioniranja sustava kontejnerskoga prijevoza morem*", Pomorski zbornik 43 (1), pp. 165-191, 2006.

[ZEN08] Zeng, Q., Wang, H., Xu, D., Han, Y.: "*Conflict detection and resolution for workflows constrained by resources and non-determined durations*", Journal of System and Software, Vol. 81, No. 9, pp. 1491-1504, 2008.

[ZHA02] Zhang, C., Wan, Y., Liu, J., Linn, R.J.: "*Dynamic Crane Deployment in Container Storage Yard*",. Transportation Research B 36. 2002.

[ZUR94] Zurawsk, R., Zhou, M. C.: " *Petri Nets and Industrial Applications: A Tutorial*", IEEE Transactions on Industrial Electronics, Vol. 41, No. 6, pp. 567-583, 1994.

Dodatni izvori

[1] Luka Koper, naslovna stranica, www.luka-kp.si (pregledano 4.06.2008.)

[2] http://web.zpr.fer.hr/ergonomija/2005/elassadi/index.html

[3] http://zlatni-rez.blogspot.com/

[4] http://bs.wikipedia.org/wiki/Fibonaccijev_broj

Bazična dokumentacijska kartica na hrvatskom jeziku

DDFOI (Sveučilište Zagreb) UDK 007.822(043.3)
Doktorska disertacija 621.316.1(043.3)

Optimalizacija sustava s diskretnim događajima primjenom Petrijevih mreža i genetskih algoritama

A. Gudelj
Fakultet organizacije i informatike
Varaždin, Hrvatska

Rad obrađuje pretpostavke za izgradnju i primjenu općenitog modela, koji integrira Petrijeve mreže i genetske algoritme s ciljem kontinuiranog nadzora poslova u sustavu s diskretnim događajima (DES) i usmjeravanja sustava u željenom smjeru.

U prvoj fazi detaljno je opisana metoda simulacije DES-a pomoću općih Petrijevih mreža (PM). U nastavku predstavljene su osnove određivanja rasporeda poslova, metode evolucijskog računanja, s posebnim naglaskom na genetskih algoritam (GA). U drugoj fazi pristupilo se izradi modela i algoritma uvođenjem matričnog modela MRF1 klase PM i GA, s ciljem određivanja rasporeda poslova u više-projektnom sustavu s višeradnim resursima ograničenog kapaciteta pomoću heurističkih pravila, u kojem su prioriteti, kašnjenja i raspoloživost poslova definirani kroz genetski algoritam.

U trećoj fazi algoritam je verificiran na dva sustava. Prvi sustav je pomorski prometni sustav kanala u kojem može nastupiti stanje potpunog zastoja neodgovarajućim zauzimanjem kanala od strane brodova koji prolaze suprotnim smjerovima. Drugi sustav je kontejnerski terminal. Razmatra se problem rasporeda poslova za automatski upravljana vozila. Cilj je izbjeći konflikte i zastoje među vozilima te minimalizirati vrijeme čekanja na dizalice, uz što kraća zadržavanja broda u luci. Algoritam je vrednovan s različitim veličinama populacije kako bi ispitao utjecaj tog parametra na konvergenciju rezultata ka konačnom rješenju.

Rezultati primjene predloženog algoritma, kao i matrične metode nadzornika za sprječavanje zastoja, ukazuju na njegovu učinkovitost i robusnost.

Voditelj rada: Prof.dr.sc.S.Vidačić, Prof. dr. sc. D. Kezić
Povjerenstvo za ocjenu i obranu:

Obrana: xxxx.2010. **Promocija:**

Rad je pohranjen u Biblioteci Fakulteta organizacije i informatike u Varaždinu.

(273 stranice, 144 slike, 21 tablica, 12 priloga, 87 bibliografskih podataka, original na hrvatskom jeziku).

A.Gudelj

DDFOI-2

UDK 007.822(043.3)
621.316.1(043.3)

1. Optimalizacija sustava s diskretnim događajima primjenom Petrijevih mreža i genetskih algoritama

Integracija
Optimalizacija
Pomorski prometni sustavi
Raspored poslova
Sprječavanje
Sustav s diskretnim događajima

I. Gudelj, A.
II. Fakultet organizacije i informatike, Varaždin, Hrvatska

Bazična dokumentacijska kartica na engleskom jeziku

DDFOI (Univ. Zagreb) UD. 007.822(043.3)
Doctor of Science Thesis 621.316.1(043.3)

The optimization of discrete event systems by Petri nets and genetic algorithm

A. Gudelj
Faculty of Organization and Informatics,
Varaždin, Croatia

This paper deals with assumptions for creation and implementation of the universal model for integration Petri nets and genetic algorithms, whose primary goal is to constantly control all jobs in a discrete event system and to guide the system to the targeted destination.

In the first phase, the method of simulation discrete event system using Petri nets (PN) was described. Hereafter the basis for determining work schedules, methods of evolutionary computation, with special emphasis on genetic algorithm were presented. The aim of the second phases was to construct the integration method of genetic algorithm (GA) and matrix models MRF1 Petri nets into a comprehensive system of assigning jobs. It is necessary to determine the job schedule in a multi-project system with shared resources using heuristic rules which are priorities, delays and availability of jobs defined by genetic algorithm.

The emphasis of the third phase was the verification of proposed algorithm at two systems. The first system is maritime traffic canal system in which to complete deadlock can be reached by taking improper channels by ships passing in opposite directions. The second system is a container terminal. We consider the problem of distribution jobs for unmanned vehicles to transport containers within the terminal. To increase the efficiency of container terminals, the proposed algorithm involves the procedure for forecasting and avoiding the conflicts and deadlocks. Also, the algorithm minimizes the waiting time for resources and minimizes the time that a ship spends in port. The algorithm was evaluated with different population sizes to examine the influence of this parameter on the convergence of results towards the final solution. The test results show the effectiveness and robustness of the proposed integration of GA and PN to determine the job schedule in a discrete event system.

Supervisor: Prof. dr. S. Vidačić, Prof. dr. sc. D. Kezić
Examiners:

Oral examination: xxxxx.2010. **Graduation ceremony:**

The thesis deposited at the Library of the Faculty of Organiziation and Informatiacs, Varaždin

(273 pages, fig 144, 21 table, 12 appendix, 87 references, original in Croatian).

A.Gudelj

DDFOI-2

UD 007.822(043.3)
621.316.1(043.3)

1. The optimization of discret even systems by Petri nets and genetic algorithm

Control
Discrete event system
Optimization
Integration
Job schedule
Maritime traffic system

I. Gudelj, A.
II. Faculty of Organization and Informatics, Varaždin, Croatia

Životopis

OSNOVNI PODACI

Ime i prezime	Anita Gudelj (rođ. Bolanča)
Godina i mjesto rođenja	1970., Split
Državljanstvo:	Republike Hrvatske
Bračno stanje	udana
Matični broj iz Upisnika znanstvenika	278411

OBRAZOVANJE

Datum:	5. studenog 1993.
Mjesto:	Split
Ustanova:	Sveučilište u Splitu Fakultet prirodoslovno-matematičkih znanosti i odgojnih područja u Splitu
Zvanje	Profesor matematike i informatike

Datum	9. svibnja 2000.
Mjesto	Varaždin
Ustanova	FAKULTET ORGANIZACIJE I INFORMATIKE
Zvanje	Magistar informacijskih znanosti

USAVRŠAVANJE

Godina:	2002.
Mjesto :	Zagreb
Ustanova:	Pro-Anima, Pučko otvoreno učilište
Područje:	Programer-Visual Basic-a

Sudjelovanje u znanstvenim projektima:	"Upravljanje sustavima u uvjetima brodskim energetskim kvara i otkaza", broj projekta: 250-2502209-2366, voditelj: dr. sc. Marijo Oršulić s Pomorskog fakulteta u Splitu.

RADNO ISKUSTVO

Datumi (od – do):	travanj 1994. – 1. srpnja 1995.
Ustanova zaposlenja:	1. Gimnazija, Split
Naziv radnog mjesta:	Profesor informatike
Područje rada:	nastava

Datumi (od – do):	12. rujna 1995. – 31. kolovoza 1996.
Ustanova zaposlenja:	Zdravstvena škola u Splitu
Naziv radnog mjesta :	Profesor informatike
Područje rada:	nastava

Datumi (od – do):	1. rujna 1996. – 10.11. 1997.
Ustanova zaposlenja:	Pomorski fakultet u Dubrovniku, Sveučilište u Splitu
Naziv radnog mjesta:	Mlađi asistent za informatiku
Područje rada:	obrazovanje

Datumi (od – do):	10.11. 1997. -
Ustanova zaposlenja:	Pomorski fakultet u Splitu
Naziv radnog mjesta:	Predavač (16.07.2001.), viši predavač (28.03.2006.) za područje društvenih znanosti, polje informacijske znanosti, grana informacijski sustavi i informatologija
Područje rada:	obrazovanje

Curriculum vitae

PERSONAL INFORMATION

Name: Anita Gudelj (rođ. Bolanča)
Year, place of birth: 1970., Split
Nationality: of Republic Croatia
Bračno stanje udana
Identification number from
Records of Scientific Workers: 278411

EDUCATION :

Date: November 5th 1993.
Place of education: Split
Name and type of organization: University of Splitu, Faculty of Natural Sciences, Mathematics and Education, Split
Title or qualification awarded: B.S. degree in mathematics and computer science

Date: May 9th 2000.
Place of education: Varaždin
Name and type of organization: University of Zagreb, Faculty of Organization and Informatics, Varaždin
Title or qualification awarded: M.Sc degree in Information science

TRAINING

Year: 2002.
Place of training : Zagreb
Name of organization: Pro-Anima, Pučko otvoreno učilište Invictus
Occupational skills covered: Programmer-Visual Basic-a

Participation in the scientific projects: "Marine Power Plant Control in Faulty and Failure Conditions“, No. 250-2502209-2366, dr.sc. Marijo Oršulić, Faculty Maritime Studies Split

WORK EXPERIENCE

Date (from – to): April 1994. – July 1st 1995.
Name of employer: 1. Gymnasia, Split
Position held: Professor of informatics
Main activities: education

Date (from – to): September 12th 1995. – August 31, 1996.
Name of employer: Medical School Splitu
Position held: Professor of informatics
Main activities: education

Date (from – to): September 1st 1996. – November 10th 1997.
Name of employer: Maritime Faculty in Dubrovnik, University of Split
Position held: Mlađi asistent za informatiku
Main activities: education

Date (from – to): November 10th 1997. -
Name of employer: University of Split, Faculty of Maritime Studies Split
Position held: Lecturer (16.07.2001.), sen. lecturer (28.03.2006.)
Main activities: education

POPIS OBJAVLJENIH RADOVA

A) Znanstveni radovi iz kategorije (a1)

1. Gudej, Anita; Krčum, Maja: *Managing temporal knowledge in port management systems*, PROMET - Traffic & Transportation, Scientific Journal on Traffic and Transportation Research, No. 3, Vol 18, Fakultet prometnih znanosti Zagreb, 2006, pp. 215-221 (pregledni znanstveni rad).
2. A. Gudelj, S. Vidačić: *Scheduling model of an automated guided vehicle system in seaports*, Suvremeni promet, No. 6, Vol. 27, Hrvatsko znanstveno društvo za promet, Zagreb, 2007, pp.446-451 (izvorni znanstveni rad).
3. Kezić, Danko; Vujović, Igor; Gudelj Anita: *Petri net Approach of Collision Prevention Supervisor design in Port transport system,* Scientific Journal on Traffic and Transportation Reasearch, Vol. 19, No. 5; pp. 269-275, 2007 (izvorni znanstveni rad).
4. Gudelj, Anita; Krčum, Maja; Krčum, Predrag: *The role of information technology in maritime education for hazard avoidance*. International Journal of Emergency Management(IJEM). Vol. 5, No. 3/4, pp. 219-234, 2008 (članak, znanstveni).
5. Gudelj, Anita; Krčum, Maja; Twrdy, Elen: *Models and Methods for Operations in Port Container Terminal.* PROMET - Scientific Journal on Traffic and Transportation Research. Vol.22, No. 1, pp. 43-52, 2010 (prethodno priopćenje, znanstveni).
6. Kezić, Danko; Gudelj, Anita: *Design of river system deadlock avoidance supervisor by using Petri net*. PROMET – Traffic & Transportation, Scientific Journal on Traffic and Transportation Research, No. 3, Vol. 22, pp. 215-221, 2010 (prethodno priopćenje, znanstveni).

B) Znanstveni radovi u zbornicima skupova s međunar.rec.

1. Gudelj, Anita; Krčum, Maja; Krčum, Predrag: *Review of methods and information technology for ship collision avoidance.* Proceedings of 16th TIEMS Annual Conference 2009, Istanbul, Turska, pp. 210-220, 2009 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).
2. Krčum, Maja; Gudelj, Anita; Antonić, Radovan: *Power plant system – electrical safety on ship*. 12th International Conference on Transport Science - ICTS 2009, Portorož, 2009. (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).
3. Krčum, Maja; Gudelj, Anita; Đula, Ivan: *Primjena gorivih ćelija na brodovim.* 2nd International Maritime Science Conference- IMSC 2009, (predavanje,međunarodna recenzija, objavljeni rad,znanstveni).
4. Krčum, Maja; Gudelj, Anita; Vicko Batinica: *Ship Power Plant - Safety Design*. Proceedings of 16th TIEMS Annual Conference 2009, Istanbul, Turska, pp. 221-230, 2009 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).
5. Krčum, Maja; Gudelj, Anita; Žižić, Leo: S*hip electrical system– safety design, simulating and training.* POWA 2009 Conference Proceedings, University of Zagreb, Faculty of Transport and Traffic Sciences, 2009. pp. 1-9 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).
6. Gudelj, Anita; Krčum, Maja; Twrdy, Elen: *A Case Study for Improving Modeling and Simulation of Container Terminals*. POWA 2008 Conference Proceedings, University of Zagreb, Faculty of Transport and Traffic Sciences, 2008. pp. 1-9 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).
8. Krčum, Maja; Gudelj, Anita: T*he new curriculum of education, trainng and certification of seamen at the faculty of maritime studies – Split.* International Conference on Transport Science - ICTS 2008, Portorož, 2008. (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).
9. Gudelj, Anita; Krčum, Maja; Krčum, Predrag: *The Role of Information Technology in Maritime Education for Hazard Avoidance*. 15th TIEMS Annual Conference, proceedings. 2008. (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

10. Krčum, Maja; Gudelj, Anita; Krčum, Predrag: *Further encouragement for the best use of simulations.* 14th TIEMS Annual Conference 2007, pp. 167-176, Split, 2007. (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).
11. Krčum, Maja; Gudelj, Anita; Vlahinić, Saša: *Genetic Algorithm for Solving Berth and Quay Cranes Assignment problems*. 2nd International Conference on Ports and Waterways – POWA 2007, pp.165-177, Zagreb 2007. (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).
12. Gudelj, Anita; Krčum, Maja; Čišić, Dragan: *Container terminal planning by Petri-net and genetic algorithms.* 10th International Conference on Traffic Science-ICTS 2006, Portorož, Slovenija 2006 – rad publiciran na CD-u ovog simpozija. (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).
13. Krčum, Maja; Gudelj, Anita; Batinica, Vicko: *contribution to the research in the application of engine simulators* // ICERS 7 - Proceedings of the 7th International Conference On Engine Room Simulators, pp. 18 – 2614, Portorož, Slovenia, 2005. (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).
14. Krčum, Maja; Gudelj, Anita; Jurić, Zdeslav: *Shipboard power supply - optimisation by using genetic algorithm*. Proceedings of the 11th IEEE International Conference on Methods and Models in Automation and Robotics, MMAR 2005, August 29 - September 1, 2005. Miedzyzdroje, Poland, p.p. 201-206 (Volume 1) (pozvano predavanje, znanstveni rad).
15. Krčum, Maja; Gudelj, Anita; Jurić, Zdeslav: *Dynamic simulation of permanent magnet synchronous machine.* The 12th IEEE Mediterranean Electrotechnical Conference – MELECON 2004, may 12-15, 2004., Dubrovnik, Croatia (međunarodna recenzija, nerazvrstan rad).
16. Gudelj, Anita; Krčum, Maja; Jurić, Zdeslav: *Implementation of the temporal databases.* Proceedings of the IASTED International Conference on Databases and Applications, February 17-19, 2004. Innsbruck, Austria, p.p. 169-173.
17. Krčum, Maja; Gudelj, Anita; Jurić, Zdeslav: *Dinamička simulacija sinkronog stroja pod stalnom uzbudom.* The 12th IEEE Mediterranean Electrotechnical Conference –MELECON 2004, pp. 1117-1120, Dubrovnik 2004. (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).
18. Krčum, Maja; Gudelj, Anita; Jurić, Zdeslav: *Genetic Algorithm Application in the Optimization of the Ship's Electrict Power System Load*. 10th IEEE International Conference on Methods and Models in Automation and Robotics, pp. 201-206, Międzyzdroje, Poland, 2004. (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).
19. Krčum, Maja; Gudelj, Anita; Jurić, Zdeslav: Fuel Cells for Marine Application // 46th International Symposium Electronics in Marine, pp. 491-495, Zadar, Croatia, 2004. (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

C) Drugi radovi u zbornicima skupova s recenzijom

1. Krčum, Maja; Gudelj, Anita; Oršulić, Marijo: Prilog istraživanju u primjenu brodostrojarskog simulatora // . (demonstracija,međunarodna recenzija,objavljeni rad,stručni).

PRILOZI

- Prilog A. Datoteka 'kanal.m'
- Prilog B. Datoteka 'kanal_start.m'
- Prilog C. Datoteka 'GA.m'
- Prilog D. Datoteka 'generator_rasporeda_setiranje.m'
- Prilog E. Datoteka 'konflikt2.m'
- Prilog F. Datoteka 'deadlock_avoidance.m'
- Prilog G. Datoteka 'vremenska_simulacija.m'
- Prilog H. Datoteka 'selection_ga_elitist.m'
- Prilog I. Datoteka 'selection_ga_Fib.m'
- Prilog J. Datoteka 'agv_pozovi.m'
- Prilog K. Datoteka 'pozoviNovi.m'
- Prilog L. Datoteka 'agv_wait.m'

Prilog A. Datoteka 'kanal.m'

```
%  MRF1 PM i simulacija kanala
% v=[K1_A B1A_A K2_A B2A_A K3_A K3_B B2B_B K2_B B1B_B K1_B]
% r= [B1A B2A B2B B1B K1 K2 K3]
% CO=[COa1 COa2 COa3 COb1 COb2 COb3]
% definirane matrica Petrijeve mreze
Fv=[0 0 0 0 0 0 0 0 0 0;1 0 0 0 0 0 0 0 0 0;0 1 0 0 0 0 0 0 0 0;0 0 1 0 0 0 0 0 0
   0;0 0 0 1 0 0 0 0 0 0;0 0 0 0 1 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 1 0 0 0
   0;0 0 0 0 0 0 1 0 0 0;0 0 0 0 0 0 0 1 0 0;0 0 0 0 0 0 0 0 1 0;0 0 0 0 0 0 0 0 0
   1];
Fr=[0 0 0 0 1 0 0;1 0 0 0 0 0 0;0 0 0 0 0 1 0;0 1 0 0 0 0 0;0 0 0 0 0 0 1;0 0 0 0
   0 0 0;0 0 0 0 0 0 1;0 0 1 0 0 0 0;0 0 0 0 0 1 0;0 0 0 1 0 0 0;0 0 0 0 1 0 0;0 0 0
   0 0 0 0];
Fud=[1 0 0 0 0 0;0 0 0 0 0 0;0 1 0 0 0 0;0 0 0 0 0 0;0 0 1 0 0 0;0 0 0 0 0 0;0 0
   0 1 0 0;0 0 0 0 0 0;0 0 0 0 1 0;0 0 0 0 0 0;0 0 0 0 0 1;0 0 0 0 0 0];
Sv=[1 0 0 0 0 0 0 0 0 0 0 0;0 1 0 0 0 0 0 0 0 0 0 0;0 0 1 0 0 0 0 0 0 0 0 0;0 0 0
   1 0 0 0 0 0 0 0 0;0 0 0 0 1 0 0 0 0 0 0 0;0 0 0 0 0 0 1 0 0 0 0 0;0 0 0 0 0 0 0 1
   0 0 0 0;0 0 0 0 0 0 0 0 1 0 0 0;0 0 0 0 0 0 0 0 0 1 0 0;0 0 0 0 0 0 0 0 0 0 1 0];
Sr=[0 0 1 0 0 0 0 0 0 0 0 0;0 0 0 0 1 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 1 0 0 0;0 0 0
   0 0 0 0 0 0 0 1 0;0 1 0 0 0 0 0 0 0 0 0 1;0 0 0 1 0 0 0 0 0 1 0 0;0 0 0 0 0 1 0 1
   0 0 0 0];
Sud=[0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0;0 0
   0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0];
Fu=[1 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 1 0 0 0 0 0]';
Fy=[0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0]';
Su=[0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0];
   Sy=[0 0 0 0 0 1 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 1];
   F=[Fu Fv Fr Fud Fy];
   S=[Su' Sv' Sr' Sud' Sy']';
   M= S'- F;                %matrica incidencije
   %Postavljanje inicijalnih uvijeta
   % m(t0)=[PI v r Ud PO];
   % ulazna mjesta...........(1)PI_A, (2)PI_B,
   % poslovi ....(3)K1_A, (4)B1A_A, (5)K2_A, (6)B2A_A,(7)K3_A,(8)K3_B, ↵ (9)B2B_B,
      (10)K2_B,(11)B1B_B, (12)K1_B,
   %resursi.......(13)B1A, (14)B2A, (15)B2B, (16)B1B, (17)K1, (18)K2, (19)K3,
   %kontrolna mjesta...(20)COa1, (21)COa2, (22)COa3, (23)COb1, (24)COb2,(25)COb3
   % izlazna mjesta .......(26)PO_A, (27)PO_B
   % svako je mjesto podijeljeno na dva dijela
   mf=[5 5 0 0 0 0 0 0 0 0 0 0 3 1 2 2 1 1 1 0 0 0 0 0 0 0 0 0]';   % zavrsni dio
   mi=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'; % inicijalni dio
   % Vektor vremena - mjestima poslova su dodjeljena vremena
   % poslovi .........(3)K1_A, (4)B1A_A, (5)K2_A, (6)B2A_A, (7)K3_A, (8)K3_B, ↵
     (9)B2B_B, (10)K2_B,(11)B1B_B, (12)K1_B,
PNtimes = [0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]';
COa1=20; COa2=21; COa3=22; COb1=23; COb2=24; COb3=25;
TotalTime=50;                                          % Ukupno vrijeme
     simulacije
TimePeriod=0.02;                                       % Period ocitavanja
     stanja mreze
Num_iter = TotalTime / TimePeriod;  % Ukupni broj iteracije
T=[PNtimes PNtimes PNtimes PNtimes PNtimes];
U(:,1)=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'; % inicijalni
     vektor korisnost (redci - korisnost, stupci - iteracije)
W(:,1)=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'; %inicijalni ↵
     vektor proteklog vremena
N(:,1)=mf + mi; % inicijalni vektor broja tokena
Nok = NaN * ones(2,1);   % inicijalni vektor broja tokena
j1=1;
```

```
37. vr(1)=0;
38. time(1)=0;  %inicijalno vrijeme
39.  % Pokretanje simulacije
40. for i=2:Num_iter;  % za za svaku iteraciju
41.  % Procedura za rješavanje konflikata
1.       mf(COa1)=1; mf(COa2)=1; mf(COa3)=1; mf(COb1)=1;
2.        mf(COb2)=1; mf(COb3)=1;
3.        x=multoa(not(F), not(not(mf)));     % racunanje omogucenih prijelaza
4.           conflict=(mf-(F'*x));  % izracunava se vector conflict
5.
6.           % Provjra da li je bilo koje mjesto ima negativan broj tokena.
7.           % Jedino resurs koji je u konfliktu ima negativan broj tokena !!!
8.        % Provjera da li mjesto (17)K1 ima negativnog broja tokena
9.      if (conflict(17) < 0)
10.     % okidanje jednog od prijelaza t1, t11, ukoliko su ovi prijelazi u konfliktu
11.          if (fix(2*rand(1))==0)
12.                mf(COa1)=1; mf(COb3)=0;
13.          else
14.              mf(COa1)=0; mf(COb3)=1;  end
15.     end
16.       % Provjera da li mjesto (18)K2 ima negativnog broja tokena
17.     if (conflict(18) < 0)
18.      % okidanje jednog od prijelaza t3, t9, ukoliko su ovi prijelazi u konfliktu
19.        if (fix(2*rand(1))==0)
20.           mf(COa2)=1; mf(COb2)=0;
21.         else
22.           mf(COa2)=0; mf(COb2)=1;   end
23.     end
24.       % Provjera da li mjesto (19)K3 ima negativnog broja tokena
25.     if (conflict(19) < 0)
26.    % okidanje jednog od prijelaza t5, t7, ukoliko su ovi prijelazi u konfliktu
27.          if (fix(2*rand(1))==0)
28.              mf(COa3)=1; mf(COb1)=0;
29.          else
30.              mf(COa3)=0; mf(COb1)=1;  end
31.          end
32.     % Procedura za rjesavanje zastoja (engl. Deadlock Avoidance Subroutine)
33.         % Broj tokena u kriticnom podsustavu 1 mora biti <= broja inicijalnih ↵
             tokena u kruznom cekanju 1 da bi se izbjeglo zaglavljenje
34.         % K1_A + B1A_A + K2_B + B1B_B <=(B1A + K2 + B1B + K1)-1
35.          x=multoa(not(F), not(not(mf)));
36.          mtest=(mf+mi)+(M'*x); % mtest - stanje nakon okidanja prijelaza x
37.                 if (mtest(26,:)==20 && mtest(27,:)==20)
38.                     if(flag==0)
39.                         UKUPNtime=time(i-1)+TimePeriod;    flag=1;   end
40.                 end
41.      if (mtest(3)+ mtest(4)+ mtest(10)+ mtest(11)) >= (mtest(13)+   ↵
             mtest(16)+ mtest(17)+ mtest(18)+1);
42.          % Omoguciti izlaz iz kriticnog podsustava 1, a zabrani ulaz u njega
43.           mf(COa1)=0;  mf(COb2)=0;  mf(COa2)=1;  mf(COb3)=1;
44.     end
45.         % Ispitivanje kriticnog podsustava 2
46.         % Broj tokena u kriticnom podsustavu 2 mora biti <= broja inicijalnih ↵
            tokena u kruznom cekanju 1 da bi se izbjeglo zaglavljenje
47.         % K2_A + B2A_A + K3_B + B2B_B <=(B2A + K3 + B2B + K2)-1
48.          x=multoa(not(F), not(not(mf))); % omoguceni prijelazi
49.          mtest=(mf+mi)+(M'*x);  % stanje mtest nakon okidanja prijelaza x
50.      if (mtest(5)+ mtest(6)+ mtest(8)+ mtest(9)) >= (mtest(14)+ mtest(15)+   ↵
             mtest(18)+ mtest(19)+1);
51.          % Omoguciti izlaz iz kriticnog podsustava 2, a zabrani ulaz u njega
52.             mf(COa2)=0; mf(COb1)=0;  mf(COa3)=1;  mf(COb2)=1;
53.      end
```

```
54.     % Osigurati da kljucni resurs K2 ne ostane zadnji slobodni resurs
55.      % K1_A + B1A_A + K3_B + B2B_B <=(B1A + K1 + B2B + K3)-1
56.          x=multoa(not(F), not(not(mf)));
57.          mtest=(mf+mi)+(M'*x);
58.        if (mtest(3)+ mtest(4)+ mtest(8)+ mtest(9)) >= (mtest(13)+ mtest(15)+
           mtest(17)+ mtest(19)+1);
59.        % Omoguciti izlaz iz skupa mjesta i onemoguciti ulaz u njega
60.           mf(COa1)=0;  mf(COb1)=0;  mf(COa2)=1;  mf(COb2)=1;
61.        end
104.     x=multoa(not(F), not(not(mf)));
105.     mf=mf+(-F'*x);     % broj tokena u izlaznom dijelu mjesta nakon okidanja
106.     mi=mi+(S*x);    %broj tokena u ulaznom dijelu mjesta nakon okidanja
107.  % ------------------------------------------------------------------------
108.     % Proceduea za proracun proteklog vremena
109.  for j=1:length(mf); % prelazi sva mjesta za svaku iteraciju
110.      % Izracunati U - korisnost, W - utroseno vrijeme, N - broj tokena
111.        % rutina za proracun U
112.  if mi(j)>0;
113.     U(j,i)=U(j,i-1)+TimePeriod;  % uveca korisnost prethodne iteracije za
     TimePeriod
114.      else
115.  U(j,i)=U(j,i-1);
116.  Vr_ok(j,:)=U(j,i);
117.  end
118.        % rutina za proracun W
119.  if mf(j)>0
120.     W(j,i)=W(j,i-1)+TimePeriod;
121.  else
122.     W(j,i)=W(j,i-1);
123.  end
124.  N(:,i)=mf+mi;  % broj tokena u mjestu je zbroj tokena oduzetih i tokena koji
     su pridodani nakon odredjenog vremena
125.  % rutina pregledava broj tokena u svakom mjestu vektora mi
126.  if mi(j)>0;
127.   for k= 1:mi(j);
128.    % matrica T(j,k) redovi-mjesta, stupci-redni broj tokena k
129.    % T(j,k) prikazuje tablicu vremena pojedinih tokena za svako mjesto
130.     % ima 14 redova (mjesta) i 5 identicnih stupaca sa
131.  % vremenima pojedinih mjesta (moze biti max 5 tokena u mjestima mi)
132.  %
133.    if (T(j,k)>0)
134.     T(j,k)=T(j,k)-TimePeriod ;
135.    else
136.      for kt=2:mi(j);
137.            T(j,kt-1)=T(j,kt);     end
138.       T(j,mi(j))=PNtimes(j); % vrati vrijeme za mjesto kad je vrijeme isteklo
139.        mf(j)=mf(j)+1;     % token prelazi u izlazni dio mjesta
140.         mi(j)=mi(j)-1;     % broj tokena se smanjuje u ulaznom dijelu mjesta
141.    end
142.   end
143.  end
144.   end           % KRAJ linije 109
145.   time(i) = time(i-1)+TimePeriod;
146.  end   % KRAJ vanjske FOR petlje (linija 40)
147.           % KRAJ
```

Prilog B. Datoteka 'kanal_start.m'

```
% Pocetni file - Simulacija kanala
% Author:     Anita Gudelj
% History:    kolovoz 2009     file created
clc; clear all; flag=0;
global TotalTime;
global TimePeriod;
% 1. Definiranje matrica iz Petrijeve mreze kanala
% Fv- matrica slijeda poslova (koji posao okida koji prijelaz)
% Fr- matrica potrebnih resursa (koji resurs treba za okidanje
prijelaza)
% Sv- matrica pokrenutih poslova
% Sr- matrica raspoloživih resursa (koji prijelaz opušta koji resurs)
Fv=[0 0 0 0 0 0 0 0 0 0;1 0 0 0 0 0 0 0 0 0;0 1 0 0 0 0 0 0 0 0;0 0 1 0 0 0 0
0 0 0;0 0 0 1 0 0 0 0 0 0;0 0 0 0 1 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 1
0 0 0 0;0 0 0 0 0 0 1 0 0 0;0 0 0 0 0 0 0 1 0 0;0 0 0 0 0 0 0 0 1 0;0 0 0 0 0
0 0 0 0 1];
Fr=[0 0 0 0 1 0 0;1 0 0 0 0 0 0;0 0 0 0 0 1 0;0 1 0 0 0 0 0;0 0 0 0 0 0 1;0 0
0 0 0 0 0;0 0 0 0 0 0 1;0 0 1 0 0 0 0;0 0 0 0 0 1 0;0 0 0 1 0 0 0;0 0 0 0 1 0
0;0 0 0 0 0 0 0];
Fud=[1 0 0 0 0 0;0 0 0 0 0 0;0 1 0 0 0 0;0 0 0 0 0 0;0 0 1 0 0 0;0 0 0 0 0 0;0
0 0 1 0 0;0 0 0 0 0 0;0 0 0 0 1 0;0 0 0 0 0 0;0 0 0 0 0 1;0 0 0 0 0 0];
Sv=[1 0 0 0 0 0 0 0 0 0 0 0;0 1 0 0 0 0 0 0 0 0 0 0;0 0 1 0 0 0 0 0 0 0 0 0;0
0 0 1 0 0 0 0 0 0 0 0;0 0 0 0 1 0 0 0 0 0 0 0;0 0 0 0 0 0 1 0 0 0 0 0;0 0 0 0
0 0 0 1 0 0 0 0;0 0 0 0 0 0 0 0 1 0 0 0;0 0 0 0 0 0 0 0 0 1 0 0;0 0 0 0 0 0 0
0 0 0 1 0];
Sr=[0 0 1 0 0 0 0 0 0 0 0 0;0 0 0 0 1 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 1 0 0 0;0
0 0 0 0 0 0 0 0 0 1 0;0 1 0 0 0 0 0 0 0 0 0 1;0 0 0 1 0 0 0 0 0 1 0 0;0 0 0 0
0 1 0 1 0 0 0 0];
Sud=[0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0;0
0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0];
Fu=[1 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 1 0 0 0 0 0]';
Fy=[0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0]';
Su=[0 0 0 0 0 0 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 0];
Sy=[0 0 0 0 0 1 0 0 0 0 0 0;0 0 0 0 0 0 0 0 0 0 0 1];
F=[Fu Fv Fr Fud Fy];
matrice=struct('Fv',Fv,'Fr',Fr,'Fud',Fud,'Fu',Fu,'Fy',Fy,'Sv',Sv,...
   'Sr', Sr,'Sud',Sud,'Su',Su,'Sy',Sy);
  matrice.F=F;
    S=[Su' Sv' Sr' Sud' Sy']';    matrice.S=S;
  M= S'- F;   matrice.M=M;              %matrica incidencije
% svako je mjesto podijeljeno na dva dijela
% mf-zavrsni dio;  mi-inicijalni dio
% tokeni ulaze u mi i nakon odredjenog vremena prelaze u mf
mf=[5 5 0 0 0 0 0 0 0 0 0 0 3 1 2 2 1 1 1 0 0 0 0 0 0 0 0]';
mi=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]';
% ////// Definiranje parametara projekta //////////////////////////
param_projekt=[];
param_projekt.PrNo=10;             % broj projekata
param_projekt.RNo = 7;                  %broj resursa
param_projekt.PNo = 10;                 % broj poslova
%
PNtimes = [0 0 0.54 1 1.08 1 0.81 0.81 1 1.08 1 0.54 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0]';
st1=0.3*PNtimes(4,1)*mf(13,1);
st2=0.3*PNtimes(6,1)*mf(14,1);
st3=0.3*PNtimes(9,1)*mf(15,1);
st4=0.3*PNtimes(11,1)*mf(16,1);
setup_time= [st1 st2 st3 st4];  % setup time za bazene
```

```
41. [sx sy]=size(setup_time);
42. % Linije 43-45 treba ukljuciti u program kada se ukljuce setup vremena za
    resurse
43. %  for i=1:sy
44. %      PNtimes(12+i,1)=setup_time(1,i);
45. %  end
46.     matriceCO.COa1=20;
47.     matriceCO.COa2=21;
48.     matriceCO.COa3=22;
49.     matriceCO.COb1=23;
50.     matriceCO.COb2=24;
51.     matriceCO.COb3=25;
52. U(:,1)=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'; %
    inicijalni vektor korisnost (redovi - korisnost, stupci - iteracije)
53. W(:,1)=[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]'; %
    inicijalni vektor proteklog vremena (redovi - proteklo vrijeme, stupci
    - iteracije)
54. N(:,1)=mf + mi; % inicijalni vektor broja tokena (redovi - mjesta,
    stupci - iteracije)
55. %  ##############   earliest_start_time  (EST)  za svaki posao
56. r=earliest_start_time(mf,PNtimes);
57.      [i1 i2]=size(r);
58.  fprintf('\n earliest start time \n ');
59.        for i=1:i1
60.           for j=1:i2
61.             fprintf('%.3f\t',r(i,j));
62.            end
63.            fprintf('\n');
64.         end

65. %  ************* POSLOVI ******************
66. posao=struct('mjesto',{'K1_A','B1A_A','K2_A','B2A_A','K3_A',...
      'K3_B','B2B_B','K2_B','B1B_B','K1_B'},'rj',[],'ImeResursa',[]);
67.    for i=1 : param_projekt.PNo;
68.      posao(i).rj=r(:,i+2);
69.    end
70. fprintf('\n*************************************************\n');
71. fprintf('\t\t  POSLOVI u Petrijevoj Mrezi \n\t Mjesto\t \t  Resurs
    koji ga izvodi\n broj mjesta-naziv\t\t  broj-naziv');
72. fprintf('\n-------------------------------------------------\n');
73.  for i=1 : param_projekt.PNo;
74.    fprintf('\t%s\t\t\t  %s\n',posao(i).mjesto,posao(i).ImeResursa);
75. end
76. J=[];    % J skup svih poslova koje treba obaviti
77.    for i =1:5
78.      for j=3:12
79.         J=cat(2,J,[i; j]) ;    % redni broj mjesta-posla, projekt
80.      end
81.    end
82. %  RESURSI
83. fprintf('+++++ Kapacitet resursa na pocetku : \n');
84.  for i=1:param_projekt.RNo
85.            RD(i,1)=mf(12+i);
86.             fprintf('RD(%d)=%d\t',i,RD(i,1));
87.  end
88.  fprintf('\n');
89. % Pokretanje simulacije
90. TotalTime=30;
91.  TimePeriod=0.02;             % Period ocitavanja stanja mreze
```

```
92.  % $$$$$$$   Poziva se GA i GENERATOR  RASPOREDa $$$$$$$$$$$$$$$$$$$$$$
93.  [Best Total_Best GIx]=
     GA(param_projekt,posao,J,matrice,matriceCO,r,mf,mi,PNtimes);
94.  N=Total_Best(GIx).Schedule.Ntokena;
95.  % Graficki prikaz rezultata
96.  graf_tokena(Total_Best,GIx,TimePeriod,TotalTime);
97.  KRAJ
```

Prilog C. Datoteka 'GA.m'

```
function [Best Total_Best Gindex]
        =GA(param_projekt,posao,J,matrice,matriceCO,r,mf,mi,PNtimes)
1.   % Genetski algoritam
2.   % Author:     Anita Gudelj
3.   % file created, 10.10
4.   Izlazne datoteke
5.   fid = fopen('exp.txt', 'wt');
6.   fid_ras = fopen('exp_ras.txt', 'wt');
7.   fid_time = fopen('exp_time.txt', 'wt');
8.   % Definiranjee GA parametara
9.   GA_param=[];
10.  NoJob=50;
11.  GA_param.pop_size=25;    % broj jedinki u populaciji
12.  GA_param.crossover_func = 'crossover_ga';
13.  GA_param.mutation_func = 'mutation_ga';
14.  GA_param.selection_func = 'selection_ga_elitist';
15.  GA_param.nb_couples  = 100;
16.  GA_param.selek_pressure = 0.05;
17.  GA_param.Proba_cross = 0.75;
18.  Log=true; Makespam=[];
19.  MAXGEN = 20;    % Maksimalni broj generacija u gen.evoluciji
20.  Fit=1;
21.  % Inicijalizacija populacije
22.  fprintf(fid_ras,'Inicijalizacija populacije \n');
23.  Pop_init = [];
24.   Pop_init = ga_populacija(param_projekt,GA_param);
25.  raspored=struct('niz_poslova',[],'niz_okidanja',[],'param',[]);
26.  % DEKODIRANJE jedinki
27.  mod =input('Za staticki raspored bez kasnjenja pritisni tipku 0, inace
     1: ');
28.  Pop_in=Pop_init;
29.  y=param_projekt.PrNo;
30.  Pop_out=dekod_krom(Pop_in,r,PNtimes,y,mod);
31.  fprintf(fid_ras,'Dekodirana populacija \n');
32.  Best=struct('populacija',[],'jedinka',[],'kromosom',[],'FTime',[],'Sch
     edule',[]);
33.   CE=0; % pocetni racunalni napor (computational effort)
34.   AES=0; % pocetni prosjecni broj evaluacija (average evaluations)
35.  CUTime=cputime; % pocetno racunalno vrijeme
36.  E=[];
37.  generacija=1;
38.  % ////////////////////////////////////////////////
39.  %              Evolucijska petlja
40.  % % Kreiranje stvarnog rasporeda
41.       k1=[];
42.      for i=1:GA_param.pop_size
43.          k1(i,:)=Pop_out(i,:);
44.          k=k1(i,:);
```

```
45.          [Fg  Ft S0 RD N1] = generator_rasporeda_setiranje
               (r,mf,mi,PNtimes,matrice,matriceCO,param_projekt,posao,J,k);
46.          ns=size(RD,2);
47.          CE=CE+1;
48.          NizOk=[];
49.           for ok=1:size(Fg,2)
50.               NizOk(1,ok)=Fg(ok).Start;
51.           end
52.          BrGen=size(Ft,2);
53.          raspored(i).niz_poslova=S0;
54.          raspored(i).niz_okidanja=NizOk ;          % raspored poslova
55.          raspored(i).param=Fg ;
56.          raspored(i).Ntokena=N1;
57.          raspored(i).resursi=RD;
58.          Makespam(i)=Ft(BrGen);
59.       end
60.  % racunjanje funkcije cila-makespam za svaku generaciju
61.  [FitFja,C]=min(Makespam);
62.  F0=mean(Makespam);
63.  Best(generacija).populacija=generacija;
64.  Best(generacija).jedinka=C;
65.  Best(generacija).kromosom=k1(C,:);
66.  Best(generacija).FTime=Makespam(C);
67.  Best(generacija).Schedule=raspored(C);
68.  disp('&&&&&&&&&&&&&&&&&&&&&&&&&&&6&&&&&&&&&&&&&&&&&&&');
69.  fprintf('Najbolja jedinka pocetne populacije je %d s vremenom
               završetka svih poslova  %f\n',Best(generacija).jedinka,...
               Best(generacija).FTime);
70.  fprintf('Procjecno vrijeme izvodjenja je %f\n', F0);
71.  fprintf(fid_ras,'Najbolja jedinka pocetne populacije je %d s vremenom
               završetka svih poslova  %f\n',
               Best(generacija).jedinka,Best(generacija).FTime);
72.   fprintf(fid_ras,'Procjecno vrijeme izvodjenja je %f\n', F0);
73.   % ****************** POCETNA POPULACIJA
74.  % *****************************************************************
75.  if generacija==1
76.    fprintf('\n * * * * * POCETNA POPULACIJA * * * * *\n');
77.    fprintf(fid_ras,'\n * * * * * POCETNA POPULACIJA * * * * *\n');
78.      for i=1:GA_param.pop_size
79.       fprintf(fid_ras,'********** %d. jedinka:   \n',i);
80.           for j1=1:size(k1,2)
81.               fprintf(fid_ras,'%.3f\t',k1(i,j1));end
82.                fprintf(fid_ras,'   \nmakespam=%f\n\n', Makespam(i));
83.            end
84.          S01=Best(1).Schedule.niz_poslova;
85.          BrGen1=size(Best(1).Schedule.param,2);
86.       fprintf(fid_ras,' _____________________________________\n
               brod\t\t posao \t\t \tzauzeti resursi\n
               _____________________________________\n');
87.        for i=1:size(S01,2)
88.          if ((S01(2,i)~=0) && (S01(2,i)~=26) && (S01(2,i)~=27))
89.             fprintf(fid_ras,'\t%d \t\t\t %s \t\t\t %s\n', S01(1,i),
               posao(S01(2,i)-2).mjesto,posao(S01(2,i)-2).ImeResursa );
90.          end
91.         end
92.       [S0x S0y] = size(S01);
93.  f = figure('Name' , 'Prikaz pocetnog rasporeda poslova' ,'Position',
               [100 100 450 600]);
94.  tab = uitable('Parent', f, 'Position', [2 4 400 570]);
```

```
95.  set(tab, 'ColumnName', {'Broj faze','','Brod', 'Posao','Start time'});
96.  red=1;
97.  for i=1:BrGen1
98.         backgroundColor = [.7 .1 .1;.2 .1 .5];
99.         set(tab, 'BackgroundColor', backgroundColor);
100.      for j=1:size(Best(1).Schedule.param(i).brod,2)
101.       if (raspored(C).param(i).aktivni_poslovi(j)~=0)  &&
               (raspored(C).param(i).aktivni_poslovi(j)~=26)  &&
               (raspored(C).param(i).aktivni_poslovi(j)~=27)
102.        complexData(red,:)={'faza:' i Best(1).Schedule.param(i).brod(j)
            posao(Best(1).Schedule.param(i).aktivni_poslovi(j)-2).mjesto...
            Best(1).Schedule.param(i).Start;};
103.        red=red+1;
104.       end
105.      end
106.    end
107.    set(tab, 'Data', complexData);
108.    set(tab, 'ColumnWidth', {20 10 40 'auto'});
109.    foregroundColor = [1 1 1];
110.    set(tab, 'ForegroundColor', foregroundColor);
111.  end          % kraj if funkcije - linija 75
112. complexData(red,:)={'' '' 'Min Makespam' '' Best(generacija).FTime};
113. set(tab,'Data',complexData);
114. fprintf(fid_ras,'%g \t',Makespam);
115. while (generacija <= MAXGEN)
116.                % ELITISTICKA SELEKCIJA
117. FObj_TopPop_out=[];
118. FObj_BotPop_out=[];
119. FObj_Pop2_out=[];
120. Top=0.2;  %  20% populacije se izravno proslijeduje u sljedecu generaciju
121. Bottom=0.3;     % 30% populacije se zamjenjuje s novim-slucajno
                       generiranim kromosomima
122. % ////////////// Kreiranje VRHA   ////////////////////
123. XTop=round(GA_param.pop_size*Top);
124. Top_Pop_out=[];
125. FObj_TopPop_out=[]; PopNovi=[];
126. rasporedNovi=[];
127. [Top_Pop_out,FObj_TopPop_out,raspored_out,Efficiency,MO_Total_FObj_out
      ]= selection_ga_elitist(Pop_init,[],[],Makespam,[],[],...
      raspored,[],[],[],GA_param,XTop);
128. PopNovi=[PopNovi Top_Pop_out];
129. rasporedNovi=raspored_out;
130. BOT_depop=[];Bot_raspored=[];BOT_pop=[];
131.    raspored2=[];
132.    raspored3=[];Makespam1=[];
133. % ///////////////  Kreiranje DNA POPULACIJE   ///////////////////
134. XBot=round(GA_param.pop_size*Bottom);
135.   for i=1:XBot
136.       BOT_depop(i,:) = kromosomi(param_projekt)  ;
137.       BOT_pop=dekod_krom(BOT_depop,r,PNtimes,param_projekt.PrNo,mod);
138.      [BotFg BotFt BotS0 BotRD BotN] = generator_rasporeda_setiranje(r,
           mf,mi,PNtimes,matrice,matriceCO,param_projekt,posao,J,BOT_pop);
139.      Bns=size(BotN,2);
140.      CE=CE+1;
141.      NizBOT=[];
142.       for ok=1:size(BotFg,2)
143.            NizBOT(1,ok)=BotFg(ok).Start;
144.        end
145.      BrGen=size(BotFt,2);
```

```
      Bot_raspored(i).niz_poslova=BotS0;
      Bot_raspored(i).niz_okidanja=NizBOT ; % raspored poslova
      Bot_raspored(i).param=BotFg ;
      Bot_raspored(i).Ntokena=BotN;
      Bot_raspored(i).resursi=BotRD;
      Makespam1(i)=BotFt(BrGen);
  end
  BOT_Pop_out=[];FObj_BotPop_out=[];
  BOT_Pop_out=BOT_depop;
   FObj_BotPop_out=Makespam1;
   Npop=size(raspored,2);
  % ///////////////     Kreiranje OSTATKA       ///////////////////////
  Nx=GA_param.pop_size-XBot-XTop;
  In1_raspored=[];Makespam3=[];FObj1=[];
  In2_raspored=[];Makespam2=[];FObj2=[];
for i =1 :Nx
      k1=round((XTop-1) * rand)+1;
      k2=round((GA_param.pop_size-1)*rand)+1;
         while FObj_TopPop_out(k1,1)==Makespam(k2)
             k2=round((GA_param.pop_size-1)*rand)+1;
         end
     par1=PopNovi(k1,:);    par2=Pop_init(k2,:);
   % ////////////// KRIZANJE  ////////////////////////
      Beta=GA_param.Proba_cross;
       % Beta=0.25*rand+0.6;
      [Ind1,Ind2] = crossover_ga(par1,par2,Beta);
       Indiv1(i,:)=Ind1;   Indiv2(i,:)=Ind2;
   Vek_Ind1(i,:)=
         dekod_krom(Indiv1(i,:),r,PNtimes,param_projekt.PrNo,mod);
   Vek_Ind2(i,:)=
         dekod_krom(Indiv2(i,:),r,PNtimes,param_projekt.PrNo,mod);
  % Racunanje FUNKCIJE CILJA
    [In1Fg  In1Ft In1S0 In1RD In1N] =
   generator_rasporeda_setiranje(r,mf,mi,PNtimes,matrice,matriceCO,param
   _projekt,posao,J,Vek_Ind1(i,:));
    ins1=size(In1N,2);
    CE=CE+1;       NizIn1=[];
         for ok=1:size(In1Fg,2)
                NizIn1(1,ok)=In1Fg(ok).Start;  end
      BrGen=size(In1Ft,2);
      In1_raspored(i).niz_poslova=In1S0;
      In1_raspored(i).niz_okidanja=NizIn1 ; % raspored poslova
      In1_raspored(i).param=In1Fg;    In1_raspored(i).Ntokena=In1N;
      In1_raspored(i).resursi=In1RD;   Makespam3=In1Ft(BrGen);
       FObj1(1,i)=Makespam3;
  [In2Fg  In2Ft In2S0 In2RD In2N] =generator_rasporeda_setiranje (r,mf,
       mi,PNtimes,matrice,matriceCO,param_projekt,posao,J,Vek_Ind2(i,:));
     ins2=size(In2N,2);
     CE=CE+1;     NizIn2=[];
          for ok=1:size(In2Fg,2)
                 NizIn2(1,ok)=In2Fg(ok).Start;        end
        BrGen=size(In2Ft,2);
        In2_raspored(i).niz_poslova=In2S0;
        In2_raspored(i).niz_okidanja=NizIn2 ; % raspored poslova
        In2_raspored(i).param=In2Fg ;
        In2_raspored(i).Ntokena=In2N;
        In2_raspored(i).resursi=In2RD;
        Makespam2=In2Ft(BrGen);     FObj2(i)=Makespam2;
   end                  % kraj for petlje i (linija 161)
```

```
200.    INraspored=[In1_raspored In2_raspored];
201.    [Pop2_out,FObj_Pop2_out,raspored_out2,Efficiency,MO_Total_FObj_out] =
            selection_ga_elitist([],Indiv1,Indiv2,[],FObj1,FObj2,...
            INraspored, [],[],[],GA_param,Nx);
202.     PopNovi=[PopNovi; Pop2_out; BOT_Pop_out];
203.     raspored=[rasporedNovi raspored_out2 Bot_raspored];
204.     Makespam=[FObj_TopPop_out' FObj_Pop2_out' FObj_BotPop_out];
205.    Pop_init=[];   Pop_init=PopNovi;
206.   % racunjanje najbolje FUNKCIJE CILJA-MAKESPAM za svaku jedinki
207. [FitFja,C]=min(Makespam);
208. Best(generacija).populacija=generacija;
209. Best(generacija).jedinka=C;
210. Best(generacija).kromosom=PopNovi(C,:);
211. Best(generacija).FTime=Makespam(C);
212. Best(generacija).Schedule=raspored(C);
213.  fprintf('Najbolja jedinka %d. populacije je %d s
            fitnesom %f\n',generacija,C,FitFja);
214.  fprintf(fid,'Najbolja jedinka %d. populacije je %d s
            fitnesom %f\n',generacija,C,FitFja);
215.  prosjek(generacija)=mean(Makespam);
216.  fprintf('Procjecno vrijeme ove populacije: %f\n', prosjek(generacija));
217.  fprintf(fid,'Procjecno vrijeme ove populacije: %f\n',
            prosjek(generacija));
218.   generacija=generacija+1;
219.  end  % kraj while petlje
220. % Najbolji_____________________________________
221. Bx=size(Best,2);
222.  for i=1:Bx
223.    Rk(i,:)=Best(i).kromosom;
224.   end
225.  for l=1:Bx
226.     R(1,l)=Best(l).FTime;   end
227.   [r1,r2]=size(Rk);
228.     [min1,NoPop]=min(R);    Total_Best(Fit)=Best(NoPop);

229. %   NAJBOLJI RASPORED POSLOVA
230.  S02=Total_Best(Fit).Schedule.niz_poslova;
231.  BrGen2=size(Total_Best(Fit).Schedule.niz_okidanja,2);
232.   fprintf(' _____________________________________\n   brod\t\t posao \t\t
           \t zauzeti resursi\n _____________________________________\n');
233.   fprintf(fid,' _____________________________________\n   brod\t\t posao
           \t\t \t zauzeti resursi\n _____________________________________\n');
234.      for i=1:size(S02,2)
235.         if ((S02(2,i)~=0) && (S02(2,i)~=26) && (S02(2,i)~=27))
236.            fprintf('\t%d \t\t\t %s \t\t\t %s\n', S02(1,i),posao(S02(2,i)-
                       2).mjesto,posao(S02(2,i)-2).ImeResursa );
237.            fprintf(fid,'\t%d \t\t\t %s \t\t\t %s\n', S02(1,i),posao
                       (S02(2,i)-2).mjesto,posao(S02(2,i)-2).ImeResursa );
238.         end
239.       end
240.   fprintf('_____________________________________\n  brod\t\t posao \t\t\t
              Vrijeme pocetka _____________________________________\n');
241.   fprintf(fid,'_____________________________________\n  brod\t\t posao \t\t
              \t Start time _____________________________________\n');
242.  [S0x S0y] = size(S02);
243. f1 = figure('Name' , 'Prikaz najboljeg rasporeda poslova' ,'Position',
              [100 100 450 600]);
244. tab1 = uitable('Parent', f1, 'Position', [2 4 400 570]);
245. set(tab1, 'ColumnName', {'', '', 'Brod', 'Posao', 'Vrijeme pocetka' });
```

```
246. red1=1;
247. %
248. Ac=Total_Best(Fit).Schedule.param.aktivni_poslovi;
249. Ax=size(Total_Best(Fit).Schedule.param,2);
250.      for i4=1:Ax
251.            fprintf('\n generacija: %d\n ',i4);
252.            fprintf(fid,'\n generacija: %d\n ',i4);
253.            size(Total_Best(Fit).Schedule.param(i4).brod,2);
254.          for j=1:size(Total_Best(Fit).Schedule.param(i4).brod,2)
255.            if(Total_Best(Fit).Schedule.param(i4).aktivni_poslovi(j)~=0) &&
                 (Total_Best(Fit).Schedule.param(i4).aktivni_poslovi(j)~=26)&&
                 (Total_Best(Fit).Schedule.param(i4).aktivni_poslovi(j)~=27)
256.           fprintf('\t%d \t\t\t %s \t\t\t %f\n',
               Total_Best(Fit).Schedule.param(i4).brod(j),posao(Total_Best(Fit
               ).Schedule.param(i4).aktivni_poslovi(j)-2).mjesto,
               Total_Best(Fit).Schedule.param(i4).Start );
257.  fprintf(fid,'\t%d \t\t%s
      \t\t %f\n',Total_Best(Fit).Schedule.param(i4).brod(j),posao(Total_Best(F
      it).Schedule.param(i4).aktivni_poslovi(j)-
      2).mjesto,Total_Best(Fit).Schedule.param(i4).Start);
258.         complexData1(red1,:) = {'faza:' i4
              Total_Best(Fit).Schedule.param(i4).brod(j)
              posao( Total_Best(Fit).Schedule.param(i4).aktivni_poslovi(j)-
              2).mjesto Total_Best(Fit).Schedule.param(i4).Start;};
259.        red1=red1+1;
260.       end
261.      end
262.      fprintf('_____________________________________\n');
263.     end
264.      complexData1(red1,:)={'' '' 'Min Makespam' '' Total_Best(Fit).FTime};
265.      set(tab1,'Data',complexData1);
266.      set(tab1, 'ColumnWidth', {'auto' 10 40 'auto'});
267.       Fib(Fit)=Total_Best(Fit).FTime;
268.       fprintf(fid,' Fit=%d   Best=%f\n', Fit, Fib(Fit));
269.       fprintf(' Fit=%d   Best=%f\n', Fit, Fib(Fit));
270.     CUTime2=cputime;
271.     E=CUTime2-CUTime; % ukupno vrijeme izvodenja (run-time)fit-og ponavljanja GA
272.    fprintf(fid,'CUTime2= %f \t Runtime vrijeme ove iteracije
           =%f\n',CUTime2,E);
273. Global, Gindex]=min(Fit);
274. fprintf(fid,'globalni optimum= %f\n',Fib(Gindex));
275. E=E/(Fit-1);% prosjecno vrijeme izvodenja (run-time)po GA
276. fprintf(fid,'CUTime2= %f \t Prosjeèno runtime vrijeme=%f\n',CUTime2,E);
277. AES=round(CE/(generacija-1));
278.    fprintf(fid,'prosjecan broj evaluacija je = %d \t \n',AES);
279.    fprintf(fid,'ukupan racunalni napor = %d \t \n',CE);
280. fclose(fid); close(fid_ras); fclose(fid_time);
281. % KRAJ
```

Prilog D. Datoteka 'generator_rasporeda_setiranje.m'

```
function [F_g  Ft S0 RD N] = generator_rasporeda_setiranje
             (r,mf,mi,PNtimes,RD,matrice,matriceCO,param_projekt,posao,J,k)
1.   global TotalTime;
2.   global TimePeriod;
3.   n=param_projekt.RNo;
4.   [Rd,ResPosao,resurs,posao] = resursi(n,mf,matrice,posao);
5.   prioriteti(1:5,1)=k(1:5);
6.   prioriteti(1:5,2)=k(6:10);
7.   DeleyGen=k(11:60);  % kasnjenja poslova
8.   DL(:,1)=k(61:65); % DL release date za svaki projekt
9.   DL(:,2)=k(66:70);
10.  RD=[];
11.  % 1. Inicijalizacija
12.  g=1;          %   prva iteracija
13.  t=[];
14.  t(1)=0;   %  za svaku iteraciju g, t(g) je vrijeme rasporedjivanja
15.  A=[];     % Skup A(g) sadrži sve poslove koji su aktivni u trenutku
     t(g)
16.  S0=[0;0];  % Skup svih poslova koji su pokrenuti do g-te iteracije
17.  Sg(:,1)=[0 0];
18.  E=[]; % Skup poslova koje su raspoloZivi u trenutku tg+kašnjenje (za
     iteraciju g)
19.  FMC=[]; %raCunanje EFT (ako se samo uzme redoslijed poslova i
     kapacitet resursa)
20.  Ft=[];     Ft=min(min(DL));
21.  MinCol=int32(Ft/TimePeriod)+1;
22.  tg=[];     tg(g)=Ft;  % ZA svaku iteraciju tg je vrijeme
     rasporedivanja
23.  Fg=struct('Start',[],'eft',[],'brod',[],'aktivni_poslovi',[]);
24.  Fg(1).eft=0;
25.  [nx mx] = size(Sg);
26.  T=[PNtimes PNtimes PNtimes PNtimes PNtimes];
27.  PNtimes=PNtimes';
28.  StartTime=[];
29.  FTime=[];
30.  m0=mf,                  %inicijalni broj oznaka
31.  %   2. ________while |Sg|<n+2
32.  for i =1 : 5
33.  for j0=1:2
34.  FinishTime(i,j0)=DL(i,j0);  %#ok<AGROW>
35.  end
36.  for j= 3:7
37.  FinishTime(i,j)=DL(i,1)+r(i,j)+PNtimes(1,j);   %#ok<AGROW>
38.  end
39.  for j1= 8:12
40.  FinishTime(i,j1)=DL(i,2)+r(i,j1)+PNtimes(1,j1) ;  %#ok<AGROW>
41.  end
42.  end
43.  FTime=FinishTime;
44.  vrijeme(1)=0;                              %inicijalno vrijeme
45.  %  TimePeriod=0.02;                      % Period očitavanja stanja
     mreze
46.  Num_iter = TotalTime / TimePeriod;     % Ukupni broj iteracije
47.  N(:,1)=mi+mf;
48.  % ************** RESURSI ******************
49.  n=param_projekt.RNo;
50.  [Rd,ResPosao,resurs,posao] = resursi(n,mf,matrice,posao);
```

```
51.  %('+++++ Kapacitet resursa na pocetku : \n');
52.  Num_iter1 = TotalTime / TimePeriod;
53.  for i=1:Num_iter1
54.  RD(:,i)=Rd;
55.  end
56.  for it=2:Num_iter;  % za za svaku iteraciju
57.  if vrijeme(it-1)>=tg(g)
58.  %
     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     +++
59.  %  5. Izbor posla s najvecim prioritetom
60.  while (not(isempty(J)))                %  Pocetak vanjske WHILE  petlje
61.  J=(setdiff(J',S0','rows'))';
62.  [E Epr]=precedence_feasible(g,Sg,J,tg(g),FTime,prioriteti,DeleyGen,
     ResPosao,RD);
63.  %  3 . update Eg
     ****************************************************
64.  while ~isempty(E)
65.  [E1,E2]=size(E);
66.  E_N=[];
67.  ep=[];
68.  E_N(:,1)=E(:,1);
69.  ep(1)=Epr(1);
70.  if E2>1
71.  kn=1;
72.  for i1=2:E2
73.  tr=0;
74.  for i2=1:i1-1
75.  if E(1,i1)==E(1,i2) && (E(2,i2)-E(2,i1)<=0) && ((E(2,i1)<=5 &&
     E(2,i2)<=5) || (E(2,i1)>5 && E(2,i2)>5))
76.  tr=1;
77.  end
78.  end
79.  if tr==0
80.  kn=kn+1;
81.  E_N(:,kn)=E(:,i1);
82.  ep(1,kn)=Epr(1,i1);
83.  end
84.  end
85.  end
86.  Sg=[];
87.  [ep, Index_sort] = sort(ep,'descend');  % sortiranje poslova po
     prioritetima od najvećeg prema najmanjem
88.  Sg=E_N(:,Index_sort(1));
89.  [E1 E2]=size(E_N);
90.  for j=2:E2
91.  flag=1;
92.  [Sgx Sgy]=size(Sg);
93.  for k1=1:Sgy
94.  if isequal(E_N(2,Index_sort(j)), Sg(2,k1))
95.  flag=0;
96.  end
97.  end
98.  if flag==1
99.  Sg=[Sg E_N(:,Index_sort(j))]; %#ok<AGROW>
100. end
101. end
102. %~~~~~~~ omoguceni prijelazi u PM
103. x1=zeros(12,1);
```

```
tr=[];     % skup prijelaza-transition
SV=matrice.Sv;
SV=SV';
for i=1:size(Sg,2)
tr(i)=find(SV(:,Sg(2,i)-2)); %#ok<AGROW>
x1(tr(i),1)=~x1(tr(i),1);         end
res=[];
FR=matrice.Fr;
FR=FR';
Sx=size(Sg,2);
for i=1:Sx
pr(i,:)= FR(:,tr(i));
res(i)=find(FR(:,tr(i)));
end
y=mf(13:19)';
x2=vektor_okidanja(x1,y,pr);  % prijelaze koji su omoguceni
%<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
% Conflict resolution subrutine
% po jedan token u svako kontrolno mjesto
mf(matriceCO.COa1)=1;
mf(matriceCO.COa2)=1;
mf(matriceCO.COa3)=1;
mf(matriceCO.COb1)=1;
mf(matriceCO.COb2)=1;
mf(matriceCO.COb3)=1;
%
Xok=multoa(not(matrice.F), not(not(mf)));
mf=konflikt2(mf,x2,matrice.F,matriceCO);
Xok=multoa(not(matrice.F), not(not(mf)));
x2=multCO(matrice,mf,x2);
mf=deadlock_avoidance(x2,mf,m0,matrice,matriceCO);
Xok=multoa(not(matrice.F), not(not(mf)));
x2(6,1)=Xok(6,1);
x2(12,1)=Xok(12,1);
% ##################################################
x2=multCO(matrice,mf,x2);
for ix=1:size(Xok,1)
x(ix,1)=x2(ix,1) && Xok(ix,1);
end
if x(6,1)==1
Sg=[Sg [0;26]];
end
if x(12,1)==1
Sg=[Sg [0;27]];
end
if x==0
E=[];
break; end
mf=mf+(-matrice.F'*x);  %broj tokena u izlaznom dijelu mjesta dijelu
    mjesta nakon okidanja
mi=mi+(matrice.S*x);    %broj tokena u ulaznom dijelu mjesta nakon
    okidanja
[mf mi T N1]=vremenska_simulacija(PNtimes,T,TimePeriod,mf,mi,it);
%
A=find(mi)';
StartTime(g)=vrijeme(it-1);
%###########%############################################################
z=find(x);
Sg;
```

```
161. SG=[];
162. for i=1:size(z,1)
163. if z(i,1)==6
164. j1=26;
165. elseif z(i,1)==12
166. j1=27;
167. else
168. j1=find(matrice.Sv(:,z(i)))+2;
169. end
170. j2=find(Sg(2,:)==j1);
171. SG=[SG Sg(:,j2)];
172. end
173. E=SG;
174. [E1,E2]=size(E);
175. res=[];
176. for i=1:E2
177. if ((E(2,i)~=26) && (E(2,i)~=27)   )
178. for j=1:10
179. if ResPosao(2,j)==(E(2,i)-2)
180. res(i)=ResPosao(1,j);
181. end
182. end
183. end
184. end
185. % 6. Racunanje EFT-earliest finish time (uzimajuci u obzir samo
     prioritete)
186. Fx=0;
187. col=[];
188. for i =1:E2
189. Fx(i)=Ft(g)+PNtimes(1,E(2,i));
190. col(i)=round(Fx(i)/TimePeriod)+1;
191. end
192. %  7. Racunanje EFT-earliest finish time (uzimajuci u obzir
193. S0=[S0 E];
194. Ex=size(E,2);
195. for l=1:Ex
196. if ((E(2,l)~=26) && (E(2,l)~=27)   )
197. FTime(E(1,l),E(2,l))=Fx(l);
198. RD(res(l),MinCol:col(l))=RD(res(l),MinCol:col(l))-1;
199. end
200. end
201. Fzx=find(Fx>0);
202. FMC(g)=min(min(Fx(Fzx(1:size(Fzx,2)))));
203. MinCol=(int32(FMC(g)/TimePeriod))+1;
204. [nx mx] = size(S0);
205. Sg=S0;
206. Ft=cat(2,Ft,FMC(g));
207. Fg(g).Start=Ft(g);
208. Fg(g).eft=FMC(g);
209. Fg(g).brod=E(1,:);
210. Fg(g).aktivni_poslovi=E(2,:);
211. res=[];
212. % Ažururanje skupa A-skup aktivnih poslova
213. tg(g);
214. while vrijeme(it-1)<tg(g)+DeleyGen(g)
215. vrijeme(it) = vrijeme(it-1)+TimePeriod;
216. [mf mi T N1]= vremenska_simulacija( PNtimes,T,TimePeriod,mf,mi,it);
217. N(:,it)=N1(:,1);
218. A=find(mi)';  it=it+1;
```

```
219. end
220. mf(13:19,1)=RD(1:7,it-1);
221. g=g+1;          %  9. inkrementiranje broja iteracija
222. tg(g)=tg(g-1);
223. J=(setdiff(J',S0','rows'))';
224. if isempty(J)
225. break;end
226. [E Epr]=precedence_feasible(g,S0,J,tg(g),FTime,prioriteti,
     DeleyGen,ResPosao,RD);
227. end                  % KRAJ WHILE PETLJE
228. if isempty(J)
229. break;        end
230. while isempty(E)
231. M=find(Ft<=tg(g));
232. z1=setdiff(Ft,Ft(M));
233. if isempty(z1)
234. t(g)=tg(g)+1;
235. fl=0;
236. else
237. t(g)=min(z1);
238. fl=1;
239. end
240. tg(g)=t(g);
241. [E Epr]=precedence_feasible(g,S0,J,tg(g),FTime,prioriteti,
     DeleyGen,ResPosao,RD);
242. end
243. while vrijeme(it-1)<tg(g)
244. vrijeme(it) = vrijeme(it-1)+TimePeriod;
245. [mf mi T N1]=vremenska_simulacija(PNtimes,T,TimePeriod,mf,mi,it);
246. N(:,it)=N1(:,1);
247. mf(13:19,1)=RD(1:7,it-1);
248. A=find(mi)';  it=it+1;
249. end
250. mf(13:19,1)=RD(1:7,it-1);
251. if mf(26)==4 && mf(27)==4
252. [mf';mi'];
253. end
254. end                          % kraj vanjske while petlje
255. end          % kraj if
256. vrijeme(it) = vrijeme(it-1)+TimePeriod;
257. if isempty(J)
258. mf(26)=5;  mf(27)=5;
259. break;end
260. end  % kraj FOR
261. Ft1=size(Ft,2);  Ft(1,Ft1)=max(Fx);
262. F_g=Fg; T1=vrijeme;
263. end
264. % end file
```

Prilog E. Datoteka 'konflikt2.m'

```
function y=konflikt2(mf,x,F,matriceCO)
1.  %  INPUT: x-vektor prijelaza koji su omogućeni
2.  % izracunava se vector conflict koji se dobije tako da se u
3.  % onim mjestima vectora mf koji prethode omogućenim prijelazima x
    oduzme broj tokena za 1
4.  conflict=mf-(F'*x);
5.  % provjera da li ima negativnog broja tokena u (17)K1
6.  if (conflict(17) < 0);   %da li  mjesto (17)K1 ima negativni broj
    tokena
7.  % Random dispatching - ova rutina po slucajnom izboru omogucuje
    okidanje jednog od prijelaza t1 ili t11, ukoliko su ovi prijelazi u
    konfliktu
8.  %             fprintf('konflikt t1, t11 \t');
9.      if (fix(2*rand(1))==0);
10.           mf(matriceCO.COa1)=1; mf(matriceCO.COb3)=0;
11.     else
12.            mf(matriceCO.COa1)=0; mf(matriceCO.COb3)=1;
13.     end
14. %         fprintf('nema konflikta t1, t11 \t');
15. end
16. % provjera da li ima negativnog broja tokena u (18)K2
17. if (conflict(18) < 0);   %da li  mjesto (18)K1 ima negativni broj
    tokena
18. %             fprintf('konflikt t3, t9\t');
19.     if (fix(2*rand(1))==0);
20.          mf(matriceCO.COa2)=1; mf(matriceCO.COb2)=0;
21.     else
22.             mf(matriceCO.COa2)=0; mf(matriceCO.COb2)=1;
23.      end
24. %             fprintf('nema konflikta t3, t9 \t');
25. end
26. % provjera da li ima negativnog broja tokena u (19)K3
27. if (conflict(19) < 0);   %da li  mjesto (19)K1 ima negativni broj
    tokena
28. %             fprintf('konflikt t5, t7\n')
29.     if (fix(2*rand(1))==0); %
30.           mf(matriceCO.COa3)=1; mf(matriceCO.COb1)=0;
31.     else
32.              mf(matriceCO.COa3)=0; mf(matriceCO.COb1)=1;
33.      end
34. %             fprintf('nema konflikta t5, t7\n');
35. end
36. y=mf; %KRAJ
```

Prilog F. Datoteka 'deadlock_avoidance.m'

```
function y=deadlock_avoidance(x,mf,m0,matrice,matriceCO)
% Deadlock Avoidance Subroutine
M=matrice.M;
% Broj tokena u kriticnom podsustavu 1 mora biti <=broja inicijalnih oznaka
      mtest=(mf)+(M'*x); % odrediti stanje mtest nakon okidanja prijelaza x
         % K1_A + B1A_A + K2_B + B1B_B <=(B1A + K2 + B1B + K1)-1
if(mtest(3)+mtest(4)+mtest(10)+mtest(11)) > (m0(13)+m0(16)+m0(17)+m0(18)-1)
      % forsirati izlaz iz kriticnog podsustava 1 i zabraniti ulaz u njega
                mf(matriceCO.COa1)=0;
                mf(matriceCO.COb2)=0;
                mf(matriceCO.COa2)=1;
                mf(matriceCO.COb3)=1;
 end
% ispitivanje kriticnog podsustava 2: K2_A+B2A_A+K3_B+B2B_B <=(B2A+K3+B2B+K2)-1
        x=multCO(matrice,mf,x);
     mtest=(mf)+(M'*x);  % odrediti stanje mtest nakon okidanja prijelaza x
if (mtest(5)+mtest(6)+mtest(8)+mtest(9)) > (m0(14)+m(15)+m(18)+m0(19)-1)
 % forsirati izlaz iz kriticnog podsustava 2 i zabraniti ulaz u njega
                mf(matriceCO.COa2)=0;
                mf(matriceCO.COb1)=0;
                mf(matriceCO.COa3)=1;
                mf(matriceCO.COb2)=1;
 end
 % osiguravanje da kljucni resurs K2 ne ostane zadnji slobodni resurs
  % K1_A + B1A_A + K3_B + B2B_B <=(B1A + K1 + B2B + K3)-1
    x=multCO(matrice,mf,x);  % odrediti koji su prijelazi omoguceni
     mtest=(mf)+(M'*x);  % odrediti stanje mtest nakon okidanja prijelaza x
if (mtest(3)+mtest(4)+ mtest(8)+mtest(9)) > (m0(13)+m0(15)+m0(17)+m0(19)-1)
      % forsirati izlaz iz skupa mjesta i zabraniti ulaz u njega
                mf(matriceCO.COa1)=0;     mf(matriceCO.COb1)=0;
                mf(matriceCO.COa2)=1;    mf(matriceCO.COb2)=1;
end
y=mf; % KRAJ
```

Prilog G. Datoteka 'vremenska_simulacija.m'

```
 function [mf mi T]=vremenska_simulacija(PNtimes,T,TimePeriod,mf,mi)
1.   % matrica T(j,k) prikazuje tablicu vremena pojedinih tokena za svako mjesto
2.   % matrica T(j,k) redci-mjesta, stupci-redni broj tokena k
3.   for j=1:length(mf);
4.   % rutina pregledava broj tokena u svakom mjestu vektora mi
5.    if mi(j)>0;
6.        for k= 1:mi(j); % za svaki token k u mjestu mi(j) tada treba
7.           if (T(j,k)>0) %ako je vrijeme za mjesto j i token k u tom mjestu > 0
8.                T(j,k)=T(j,k)-TimePeriod ;
9.            else
10.               % ako je isteklo pridruzeno vrijeme za pojedini token
11.             for kt=2:mi(j);
12.                T(j,kt-1)=T(j,kt);            end
13.       T(j,mi(j))=PNtimes(j); % vraća vrijeme za mjesto gdje je vrijeme isteklo
14.       mf(j)=mf(j)+1 %token prelazi u mf dio mjesta nakon protka vremena
15.       mi(j)=mi(j)-1         %broj tokena se smanjuje u mi dijelu mjesta
16.      end         % if petlja
17.     end          % for k
18.  end  % if
19.  end % for j
```

Prilog H. Datoteka 'selection_ga_elitist.m'

```
function [Pop_out,FObj_Pop_out,raspored_out,Efficiency,MO_Total_FObj_out]=
   selection_ga_elitist(Pop_in,Indiv1,Indiv2,FObj_Pop_in,FObj_Indiv1,FObj_In
   div2, raspored,MO_Total_FObj_in,MO_FObj_Indiv1,MO_FObj_Indiv2,param,XTop)
%  Ova funkcija izvodi elitisticku selekciju.
%  Biraju se najbolje jedinke iz skupa kojeg sacinjavaju roditelji i djeca.
% Ulazne varijable:
  %Pop_in - Pocetna populacija jedinki
  %Indiv1  - Prvi skup potomaka generirani pomocu krizanja + mutacija.
  %Indiv2 -  Drugi skup potomaka generirani pomocu krizanja + mutacija.
  %FObj_Pop_in - Vektor vrijednosti objektne funkcije za svaku jedinku iz Pop_in.
  %FObj_Indiv1 - Vektor vrijednosti objektne funkcije za svaku jedinku iz Indiv1.
  %FObj_Indiv2 - Vektor vrijednosti objektne funkcije za svaku jedinku iz Indiv2.
  %MO_Total_FObj_in-Matrica vrijednosti vise-objektne funkcije za svaku jedinku iz
        Pop_in.
 %MO_FObj_Indiv1 - Matrica vrijednosti vise-objektne funkcije za svaku jedinku iz
        Indiv1.
 % MO_FObj_Indiv2 - Matrica vrijednosti vise-objektne funkcije za svaku jedinku iz
        Indiv2.
    %param - lista parametera:  %'pressure': koeficijent pritiska selekcije
% Izlazne varijable:
   % Pop_out      - Odabrene jedinke u populaciji, velicine pop_size.
   % FObj_Pop_out - Sve vrijednosti objektne funkcije za svaku jedinku iz
                    skupa Pop_out.
   % Eficiency - Sve vrijednosti 'efficiency' za svaku jedinku iz Pop_out.
   % MO_Total_FObj_out - Sve vrijednosti vise-objektne funkcije za svaku
jedinku iz Pop_out.
mo_is_defined = isempty('MO_Total_FObj_in');
pressure = param.selek_pressure;
raspored_out=[];
Total_Pop  = [Pop_in; Indiv1; Indiv2];
Total_FObj = [FObj_Pop_in FObj_Indiv1 FObj_Indiv2]';
% Normalizacija ucinkovitosti
  FObj_Pop_Max = max(Total_FObj);
  FObj_Pop_Min = min(Total_FObj);
  Efficiency=[];
  Efficiency = (1-pressure)*(FObj_Pop_Max - Total_FObj)/max([FObj_Pop_Max -
        FObj_Pop_Min, eps])+pressure;
   [Efficiency, Index_sort] = sort(Efficiency,'descend');
  Efficiency = Efficiency(1:XTop);
% Ekstrakcija i Selekcija fenotipa
    Total_FObj   = Total_FObj(Index_sort);
    FObj_Pop_out = Total_FObj(1:XTop);
k=size(Total_Pop,1);
   for i=1:k
    Total_Pop_Novi(i,:)=Total_Pop(Index_sort(i,1),:);
    Total_raspored(1,i)=raspored(1,Index_sort(i,1));
   end
    Total_raspored;    Pop_out=Total_Pop_Novi(1:XTop,:);
      raspored_out=Total_raspored(1:XTop);
%Extraction of the multiobjective values (if defined)
if mo_is_defined
  % MO_Total_FObj for multiobjective function values
  MO_Total_FObj_out = [MO_Total_FObj_in' MO_FObj_Indiv1' MO_FObj_Indiv2']';
  MO_Total_FObj_out = MO_Total_FObj_out(Index_sort,:);
  MO_Total_FObj_out = MO_Total_FObj_out(1:XTop,:);
else
     MO_Total_FObj_out = [];
end   %KRAJ
```

Prilog I. Datoteka 'selection_ga_Fib.m'

```
function [Pop_out,FObj_Pop_out,raspored_out,Efficiency,MO_Total_FObj_out] =
      selection_ga_Fib(Pop_in,Indiv1,Indiv2,FObj_Pop_in,FObj_Indiv1,FObj_In
      div2,raspored,MO_Total_FObj_in,MO_FObj_Indiv1,MO_FObj_Indiv2,param)
% OPIS
%    Ova funkcija izvodi Fibonaccijevu selekciju.
%    Iz sortirane populacije izdvajaju se jedinke koje se nalaze na
mjestima koja odgovaraju Fibonaccijevim brojevima
  % Parametri
    % Pop_in - Pocetna populacija jedinki
    %Indiv1  - Prvi skup potomaka generirani pomocu crossover + mutation.
    %Indiv2 -  Drugi skup potomaka generirani pomocu crossover + mutation.
    %FObj_Pop_in - Vektor vrijednosti objektne funkcije za svaku jedinku iz Pop_in.
    %FObj_Indiv1 - Vektor vrijednosti objektne funkcije za svaku jedinku iz Indiv1.
    %FObj_Indiv2 - Vektor vrijednosti objektne funkcije za svaku jedinku iz Indiv2.
    %MO_Total_FObj_in - Matrica vrijednosti vise-objektne funkcije za svaku jedinku
iz Pop_in.
    %MO_FObj_Indiv1 - Matrica vrijednosti vise-objektne funkcije za svaku jedinku
iz Indiv1.
    %MO_FObj_Indiv2 - Matrica vrijednosti vise-objektne funkcije za svaku jedinku
iz Indiv2.
    %param - lista parametera:
              % 'pressure': koeficijent pritiska selekcije
% Pop_out       - Sve selektirane jedinke u populaciji velièine pop_size.
% FObj_Pop_out  - Sve vrijednosti objektne funkcije za svaku jedinku iz skupa
Pop_out.
% Eficiency - Sve vrijednosti 'efficiency' za svaku jedinku iz Pop_out.
% MO_Total_FObj_out - Sve vrijednosti vise-objektne funkcije za svaku jedinku iz
Pop_out.
pressure = 0.05 ;
raspored_out=[];
Total_Pop  = [Pop_in; Indiv1; Indiv2];
Total_FObj = [FObj_Pop_in FObj_Indiv1 FObj_Indiv2]';
% Normalizacija ucinkovitosti
  FObj_Pop_Max = max(Total_FObj);
  FObj_Pop_Min = min(Total_FObj);
Efficiency=[];
 Efficiency = (1 - pressure) * (FObj_Pop_Max -
Total_FObj)/max([FObj_Pop_Max - FObj_Pop_Min, eps]) + pressure;
 Efficiency = 1./Efficiency ;
[Efficiency, Index_sort] = sort(Efficiency,'ascend');

% Ekstrakcija i Selekcija fenotipa
         Total_FObj   = Total_FObj(Index_sort);
% Ekstrakcija i Selekcija genotipa
     k=size(Total_Pop,1);
    for i=1:k
     Total_Pop_Novi(i,:)=Total_Pop(Index_sort(i,1),:);
     Total_raspored(1,i)=raspored(1,Index_sort(i,1));
    end
      Pop_out=Total_Pop_Novi(1:2,:);
       FObj_Pop_out = Total_FObj(1:2);
      raspored_out=Total_raspored(1:2);
      fib(1)=1; fib(2)=2;fib(3)=3;i=3;
      while fib(i)<=k
             Pop_out(i,:)=Total_Pop_Novi(fib(i),:);
             FObj_Pop_out(i) = Total_FObj(fib(i));
              raspored_out(i)=Total_raspored(fib(i));
              i=i+1;fib(i)=fib(i-1)+fib(i-2);
      end   % KRAJ
```

Prilog J. Datoteka 'agv_pozovi.m'

```
function [N,i,mf,J,FinishTime,Tagv,y,free,WeitE0,Wa1,col,dizalice] =
        agv_pozovi(NcA,NcB,N,mf,m,D,crane,i,Tcrane,Idle,Tagv,J,r,PNtimes,Fi
        nishTime,DL,y,Wa1,Wa2,free,WeitE0,col,zas,dizalice)
1.   %  Odredivanje rasporeda AGVS
2.   global AGVs;global TimePeriod;
3.   global it;
4.   global vrijeme;global o;
5.   global Twait;  % vrijeme cekanja AGV na CR1
6.   NVAR=NcA;   c(1,1:NcA)=40;
7.   ss=[3 2 3]; %No skladišta
8.   if NcB>0
9.             flagB=1; end   % Ima li kontejnera za zeljeznicu?
10.  if (i<NVAR && m(17,it-1)>0 && (m(1,it-1)+m(2,it-1))<m(17,it-1) &&
     zas==1)
11.          flag=0;     projekti=[];
12.   projekti=round(1+rand(1,m(17,it-1)));
13.          diz=1;
14.  for j=1:(m(17,it-1)-(m(1,it-1)+m(2,it-1)))
15.      while crane(1,diz)==1       % Koja je dizalica CR1 slobodna?
16.           diz=diz+1; end
17.      i=i+1;crane(1,diz)=1;a=diz; diz=diz+1;Tcrane(a)=vrijeme(it-1);
18.  if ( projekti(j)==1 && NcB<=0)
19.       projekti(j)=2;
20.  else
21.      NcB=NcB-1;end    %odredi koliko je ostalo kontejnera za zeljeznicu
22.  if (N==1 && free(1,1)==1)                % ako se uvodi prvi AGV
23.     b(N)=ss(a);
24.   [y,t1,dx,Tdelay,Twait]= pozoviNovi(N,D,a,y);   %procedura koja poziva
                                                    i uvodi novi AGV
25.  l=N; Wa1(l)=0;
26.  AGVs(l).broj_vozila=l;
27.  AGVs(l).kontejner=i;Tagv(N)=0;
28.  free(1,l)=0;        % l-ti AGVagv je zauzet
29.    if flag==0
30.         for k=1:N
31.  % imali li medu AGV neki koji je obavio posao i ceka na CR1
32.            if free(1,k)&& WeitE0(k)<=Tcrane(a)&& WeitE0(k)>0
33.             Idle(k)=Tcrane(a)-WeitE0(k); %racuna koliko je AGV bez posla
34.            else
35.             Idle(k)=0;   %svi AGV su zauzeti i treba uvesti novi iz spremista
36.            end
37.  end
38.  if Idle==0
39.        N=N+1;        %uvedi novi AGV  N-ti po redu
40.         l=N;Wa1(l)=0;
41.              Tagv(l)=0;
42.      [y,t1,dx,Tdelay,Twait]= pozoviNovi(l,D,a,y);
43.          AGVs(l).broj_vozila=l;Wa1(l)=0;WeitE0(l)=0;
44.  else
45.     [MAXw,l]=max(Idle);[y,t1,dx,Tdelay,Twait]= pozoviNovi(l,D,a,y);
46.     V=AGVs(l).ukupno_vrijeme_voznje;Tagv(l)=V;Wa1(l)=Wa1(l)+Idle(l);
47.  end
48.    free(1,l)=0;Ac=AGVs(l).kontejner; AGVs(l).kontejner=[Ac i];
49.  end  % kraj if funkcije, linija 30
50.  end  % kraj if funkcije, linija 23
51.  AGVs(l).cekanje=Wa1(l);
52.  pom(a)=y(l);
```

```
53.  Tcrane(a)=Tcrane(a)+pom(a);col(:,i)=[projekti(j);
54.  round(Tcrane(a)/TimePeriod)+1];dizalice(:,i)=[i;a];
55.  yc(i)=Tcrane(a)+pom(a);  % start time za i-ti kontejner
56.  agv(l)=1;                % N-ti agv je zauzet
57.  m(l,i)=1;         % koje vozilo prenosi koji kontejner-posao
58.  Wa1(l)=Wa1(l)+Twait(l);
59.  pr(l)=t1+Twait(l);
60.  Tagv(l)=Tagv(l)+pr(l);
61.  if c(i)==20
62.  signal(l)=1; %AGV èeka na dizalicu
63.  WaitB(l)=Tagv(l);   %pocetak cekanja
64.  o(l)=a;
65.  free(1,l)=1;
66.  else
67.  signal(l)=0;
68.  end
69.  if projekti(j)==1
70.  FinishTime(i,1)=yc(i)+0.3*(i-1);  %#ok<AGROW>
71.  for j1=3:10
72.  J=cat(2,J,[i;j1]);
73.  FinishTime(i,j1)=yc(i)+0.3*(i-1)+r(1,j1)+PNtimes(j1,1);
74.  end
75.  J=cat(2,J,[i;33]);
76.  else
77.  FinishTime(i,2)=yc(i)+0.3*(i-1);  %#ok<AGROW>
78.  for j1=11:16
79.  J=cat(2,J,[i;j1]);
80.  end
81.  J=cat(2,J,[i;34]);
82.  end
83.  if i==NcA
84.  break;end   % ako su svi kontejneri iskrcani proces se prekida
85.  end
86.  end
87.  end
```

Prilog K. Datoteka 'pozoviNovi.m'

```
function [y,t1,dx,Tdelay,Twait]=pozoviNovi(N,D,a,y)
1.  %  uvodenje novog AGV-a;  N indeks vozila
2.  global Twait; global ve vf; % ve-brzina praznog AGV-a, vf-kad je AGV pun
3.  br=0;
4.  WaitB(N)=0;
5.  o(N)=1;
6.  l1=o(N);
7.  Tload=1.1;  % 66 sek=1.1min-  vrijeme zahvacanja kontejnera za CR1
8.  d1=D(l1,a+1); %udaljenost od trenutne lokacije do dizalice
9.  dx=d1;
10. t1=(d1/ve);
11. if t1 >Tload
12.         Tdelay(N)=t1 -Tload; % AGV kašnjenje i CR1 ga ceka
13.         Twait(N)=0;  %  AGV ceka na dizalicu
14. else
15.         Twait(N)=Tload-t1;     Tdelay(N)=0;
16. end
17.    y(N)=Tload+Tdelay(N);  %  kada je dizalica spremna za load na agv -
        to je relase time za projekt
18.    end  % KRAJ
```

Prilog L. Datoteka 'agv_wait.m'

```
function y=agv_wait(m,m_i,vrijeme)
1.   ny=size(m,2);
2.   time1=0;i1=2;y1=0;i2=2;y2=0;i3=2;y3=0;i4=2;y4=0;
3.   for i =2:ny
4.     if m(4,i)>m(4,i-1)
5.       time1=vrijeme(i-1);
6.        for j=i1:ny
7.          if m_i(5,j)>m_i(5,j-1)
8.               time2=vrijeme(j-1);
9.               i1=j+1;break;
10.           end
11.         end
12.      y1=y1+(time2-time1);
13.  end
14.  if m(12,i)>m(12,i-1)
15.   time3=vrijeme(i-1);
16.     for j=i2:ny
17.      if m_i(13,j)>m_i(13,j-1)
18.       time4=vrijeme(j-1);
19.          i2=j+1;break;
20.      end
21.     end
22.   y2=y2+(time4-time3);
23.  end
24.  if m(6,i)>m(6,i-1)
25.    time5=vrijeme(i-1);
26.      for j=i3:ny
27.        if m_i(7,j)>m_i(7,j-1)
28.             time6=vrijeme(j-1);
29.              i3=j+1;break;
30.         end
31.       end
32.     y3=y3+(time6-time5);
33.  end
34.  if m(8,i)>m(8,i-1)
35.  time7=vrijeme(i-1);
36.    for j=i4:ny
37.      if m_i(9,j)>m_i(9,j-1)
38.         time8=vrijeme(j-1);
39.         i4=j+1;break;
40.      end
41.     end
42.  y4=y4+(time8-time7);
43.  end
44.  end
45.  y=y1+y2+y3+y4;  % ukupno vrijeme cekanja AGV-a na dizalice CR2 i CR3 u
     oba smjera
```

INDEKSI

Q

R

S

T

U

V

W

Z