SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1732

UPORABA ALGORITAMA ZASNOVANIH NA INTELIGENCIJI ROJA ZA RJEŠAVANJE PROBLEMA KROJENJA

Filip Domazet

Zagreb, lipanj 2011.

Sadržaj

Uvod

Problem krojenja je važan problem koji se javlja u industriji, gdje se teži što više uštedjeti na upotrijebljenom materijalu i smanjiti stvoreni otpad. To je NP-težak problem, što znači da je velike instance teško rješavati egzaktnim metodama. Zbog toga se pribjegava određenim aproksimativnim algoritmima kojima se želi postići relativno dobro rješenje, blizu optimuma, ali u puno kraćem vremenu.

Algoritmi zasnovani na inteligenciji roja su neki od optimizacijskih algoritama koji se uspješno primjenjuju na brojne NP-teške probleme, te se u ovom radu razmatra kako ih primijeniti na problem krojenja.

Rad je strukturiran na sljedeći način:

- u 1. poglavlju opisuje se problem krojenja, i ukratko opisuje tipologija problema
- u 2. poglavlju daje se općeniti opis algoritama koji se koriste te opis kako se točno primjenjuju na konkretan problem
- u 3. poglavlju opisuje se programska implementacija ostvarena u okviru rada i sustav korišten za ispitivanje algoritama
- u 4. poglavlju navode se eksperimentalni rezultati primjene algoritama na skup problema čija su rješenja poznata

1. Problem krojenja 1D

Problem 1D krojenja, također poznat kao problem rezanja ili pakiranja, je NP težak problem. To znači da se smatra kako nema algoritma koji bi ga mogao riješiti u polinomijalnom vremenu pa se zbog toga često koriste razni algoritmi koji ga aproksimativno rješavaju. Postoje razne kategorije navedenog problema. Dyckhoff u [31] kategorizira probleme prema dimenzionalnosti, namjeri da se iskoriste svi izvorni predmeti ili dobiju svi traženi, zatim da li se radi o jednom izvornom predmetu, više identičnih ili više različitih te prema odnosima među traženim predmetima. Prema Dyckhoffovoj klasifikaciji, ovaj rad se bavi problemima tipa 1/V/D/R, 1/V/D/M, te 1/V/I/M i 1/V/I/R, pri čemu 1 označava da se radi o jednodimenzionalnom problemu, V označava da je potrebno zadovoljiti potraživanje svih zahtijevanih rezova, D odnosno I označava da je raspoloživo više različitih veličina izvornog materijala odnosno više materijala identičnih veličina, R označava više zahtjeva za rezanje s relativno malo različitih veličina, a M označava više zahtjeva za rezanje s više različitih veličina. Matematički formulirano tražimo minimizaciju

$$\sum_{i=1}^{M}(L_i - \sum_{j=1}^{m} x_{ij} \cdot lj) \tag{1.1}$$

Uz uvjete:

$$\sum_{j=1}^{m} x_{ij} \cdot l_j \leq L_i \tag{1.2}$$

$$\sum_{i=1}^{M} x_{ij} = N_j \tag{1.3}$$

pri čemu je L_i duljina i-tog materijala za rezanje, l_j duljina j-tog tipa traženog reza, x_{ij} broj rezova duljine l_j u planu rezanja predmeta L_i, N_j ukupna potražnja za predmetom duljine l_j. Ako su svi izvorni materijali jednake duljine, problem se može svesti na minimizaciju broja upotrebljenih materijala. Ovo je samo najosnovnija inačica problem, u stvarnosti se mogu pojaviti dodatni zahtjevi ili uvjeti. Cijena po jedinici duljine svakog tipa izvornog materijala

ne mora biti ista. Dodatni zahtjevi mogu uključivati minimalan broj različitih planova rezanja, minimizaciju broja otvorenih narudžbi pri čemu se traži da se narudžba čim prije završi kad se jednom počnu realizirati stavke te narudžbe, minimalnu promjenu položaja noževa za rezanje i sl. Također je često poželjnije imati manje većih ostataka koji se mogu dalje upotrebljavati nego puno manjih. Usporedba različitih algoritama stoga može biti relativno teška budući da su često specijalizirani za različite vrste potproblema, no minimizacija samog otpada je u pravilu najvažnija te daje mjeru koja može služiti za usporedbu.

Primjer jednog problema dan je u nastavku. Pretpostavimo da na raspolaganju imamo samo izvorne materijale duljine 1000, te da je narudžba opisana tablicom 1.1.

Tablica 1.1 Primjer narudžbe

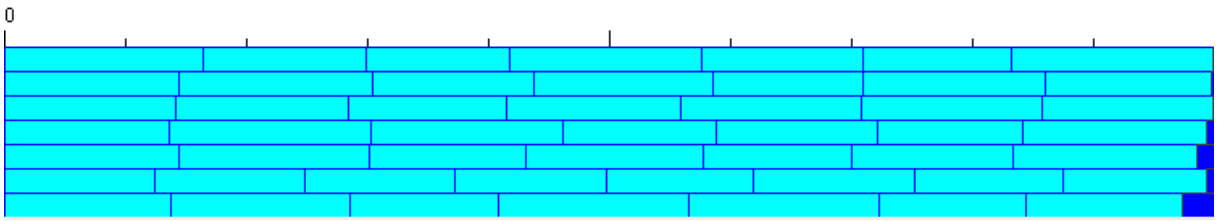
Duljina	Količina
167	2
164	1
160	1
158	4
157	1
152	2
150	2
149	2
148	1
146	1
144	3
142	2
141	1
137	2
136	1
135	1
134	1
133	4
131	1
129	2
127	1
125	2
124	3
123	3
122	2
121	2
119	1
118	1

Traži se takav plan rezanja da je što manje potrošenog materijala. Za zadani problem optimalno rješenje dano je tablicom 1.2, gdje je u prvom stupcu dana duljina korištenog izvornog materijala, koji je za zadani problem uvijek 1000, a u drugom stupcu je dan plan rezanja pojedinog materijala gdje su pojedini predmeti odvojeni znakom „+“.

Tablica 1.2 Primjer optimalnog rješenje

Duljina zalihe	Plan rezanja
1000	164 + 135 + 119 + 158 + 133 + 123 + 167
1000	144 + 160 + 133 + 148 + 125 + 150 + 137
1000	142 + 131 + 144 + 142 + 149 + 150 + 141
1000	136 + 167 + 158 + 127 + 133 + 121 + 152
1000	144 + 158 + 129 + 146 + 123 + 134 + 152
1000	124 + 125 + 122 + 133 + 124 + 123 + 124 + 118
1000	137 + 149 + 122 + 157 + 158 + 121 + 129

Optimalno rješenje nije jedinstveno. Često postoji više rješenja s jednakom količinom potrošenog materijala za koja se potom traži da što bolje zadovoljavaju neki od drugih navedenih zahtjeva. Grafički prikaz navedenog rješenja dan je slikom Sl. 1.1. Svijetlo plavom bojom na slici su označeni predmeti koje treba rezati, a tamno plavom je označen ostatak nakon rezanja pojedinog materijala.



Sl. 1.1 Grafički prikaz optimalnog rješenja

2. Algoritmi

Među algoritme zasnovane na inteligenciji rojeva spadaju brojni algoritmi. Njihova osnovna odlika je imitacija određenog ponašanja iz prirode u kojem više jedinki (agenata) surađuje bez centralizirane kontrole kako bi savladali neki problem. Često su to jednostavne interakcije pojedinih agenata među sobom i svojim okruženjem preko kojih se u konačnici manifestira kompleksno ili „inteligentno“ ponašanje na razini zajednice i sposobnost učinkovitog rješavanja određenog problema.

Među ove algoritme spadaju i optimizacija kolonijom mrava (engl. Ant Colony Optimization, skraćeno ACO) i optimizacija rojem čestica (engl. Particle Swarm Optimization, skraćeno PSO). Radi se o metaheurističkim algoritmima, što znači da nastoje naći što bolje rješenje za zadani problem, međutim bez garancije optimalnosti. Heuristički i metaheuristički algoritmi se koriste za rješavanje velikih instanci problema koji su NP teški ili NP potpuni. Za takve probleme trenutno ne postoje algoritmi koji bih ih optimalno rješavali u polinomijalnom vremenu. Heuristike koriste neko znanje usko vezano uz problem kako bi aproksimirale ili poboljšale rješenja. Metaheuristike su skup algoritamskih koncepata koji mogu biti primijenjeni kako bi rješavali širok skup problema. Često u sebi sadrže stohastičke elemente.

2.1. Optimizacija kolonijom mrava (ACO)

2.1.1. Općenito o ACO

Optimizacija kolonijom mrava se temelji na promatranju mravljeg učinkovitog načina rješavanja problema najkraćeg puta između mravinjaka i izvora hrane. Mravi imaju relativno slab vid, no imaju dobra osjetila za određenu vrstu kemijskog spoja pomoću kojeg komuniciraju - feromone. Kako se kreću do izvora hrane i nazad za sobom ostavljaju feromonski trag. Preko traga indirektno komuniciraju s drugima, tj. direktna interakcija se zbiva između mrava i okoline, što zatim posredno utječe na puteve kojima se kreću ostali mravi. Svaki mrav bira put kojim će ići stohastički, s time da ima veću vjerojatnost da krene putem gdje je trag jači. Feromonski trag predstavlja određeno znanje stečeno od strane kolonije.

Jedna od prvih verzija algoritma prema [2] je Ant System primijenjen na problem trgovačkog putnika. Rješenje se gradi postepeno – prvi grad se bira nasumično, a svaki sljedeći stohastički prema sljedećem pravilu:

Ako se mrav nalazi u čvoru *i* u koraku *k*, vjerojatnost da odabere čvor *j* kao sljedeći je dana izrazom:

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l \in N_i^k} \tau_{il}^{\alpha}\eta_{il}^{\beta}}, ako\ j \in N_i^k \\ 0, \quad inače \end{cases} \tag{2.1}$$

gdje je p_{ij}^{k} vjerojatnost odabira čvora *j* iz čvora *i* u koraku *k*, N_i^k je dopustivo susjedstvo čvora *i* , tj. svi čvorovi koji su susjedni i još nisu posjećeni, τ_{ij} je vrijednost feromonskog traga između čvorova *i* i *j*, η_{ij} je apriorna heuristička vrijednost odabira čvora *j* iz čvora *i*, α i β određuju važnost feromonskih tragova odnosno heurističke informacije.

Nakon što su svi mravi izgradili svoja rješenja, slijedi postavljanje feromonskih tragova. Prvo se primijeni isparavanje na sve feromonske tragova, dano sljedećim izrazom:

$$\tau_{ij} \leftarrow \rho \cdot \tau_{ij} \tag{2.2}$$

gdje je ρ konstanta isparavanja. Zatim slijedi dodavanje traga svakog mrava:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{n} \Delta\tau_{ij}^{k} \tag{2.3}$$

Pri čemu

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{1}{C^k}, ako\ brid\ (i,j) pripada\ putu\ T^k \\ 0, \quad inače \end{cases} \tag{2.4}$$

gdje je C^k duljina, odnosno cijena puta T^k napravljenog od strane *k-tog* mrava.

U međuvremenu su razvijene brojne varijante algoritama koji se temelje na optimizaciji kolonijom mrava. Jedan od njih, koji je u ovom radu prilagođen na problem krojenja, je Max-Min Ant System (skraćeno MaxMinAS). Razlikuje se po tome što nakon svake iteracije pravo

ostavljanja tragova ima samo jedan mrav i to samo ili globalno najbolji ili najbolji u iteraciji. Najčešće se izmjenjuje koji od njih ostavlja feromonski trag, određeno parametrom γ, tako da svakih γ iteracija globalno najbolji mrav ostavlja trag. Također, jedna od izmjena, po kojoj je algoritam dobio ime, je uvođenje minimalnih, odnosno maksimalnih vrijednosti (τ_{min}, *i* τ_{max}) koje tragovi mogu poprimiti kako bi se postigla bolja ravnoteža između eksploracije i eksploatacije prostora rješenja. Treba primijetiti da je vrijednost tragova ograničena odozgo vrijednošću $1 / \rho C^*$, gdje je C^* cijena optimalnog rješenja. Algoritam koristi procjenu ove vrijednosti tako se τ_{max} postavlja prema sljedećem izrazu:

$$\tau_{max} = \frac{1}{(1-\rho)C^{bs}} \tag{2.5}$$

gdje je C^{bs} cijena trenutno najboljeg rješenja. To znači da se vrijednost osvježava svaki put kada se pronađe bolje rješenje. Minimalna vrijednost traga se obično zadaje preko vrijednosti parametra a u odnosu na τ_{max}, tako da

$$\tau_{min} = \frac{\tau_{max}}{a} \tag{2.6}$$

Kako bi se dobilo bolje istraživanje prostora stanja u početku, početne vrijednosti svih tragova se postave na τ_{max}.

2.1.2. Primjena ACO na problem krojenja

U [3] je predstavljen ACO algoritam, preciznije MaxMinAS, za problem rezanja i pakiranja, no samo za slučaj kada su izvorni materijali jednakih duljina. Algoritam prezentiran u ovom poglavlju se oslanja na njihov, ali s nekoliko izmjena kako bi mogao raditi za slučaj kada ima više različitih tipova izvornih materijala.

Prvo je potrebno definirati značenje tragova. Trag predstavlja povoljnost kombiniranja traženog predmeta *i* s predmetom *j* za neki tip izvornog materijala (zalihe). Prema tome, u određenom koraku algoritma vjerojatnost odabira predmeta *i* u plan rezanja određene zalihe glasi:

$$p(i,s) = \frac{\tau_b(i)^\alpha \cdot \eta(i,s)^\beta}{\sum_{g \in J(x,s)} \tau_b(g)^\alpha \cdot \eta(g,s)^\beta} \tag{2.7}$$

ako je *i* element *J(x, s),* pri čemu je *J(x, s)* skup svih predmeta koji još stanu u plan rezanja trenutne zalihe *s*, u protivnom je *p(i, s) = 0.*

η(i,s) je heuristička informacija koja se računa na sljedeći način:

$$\eta(i,s) = \frac{1}{L(s) - l_i - \sum_{j \in s} l_j} \tag{2.8}$$

gdje je *L(s)* duljina materijala *s*, l_j duljina materijala tipa *j*.

Dakle, heuristička informacija je inverzno proporcionalna ostatku koji ostane kada se doda predmet tipa *i* u plan rezanja trenutne zalihe *s*. U [3] se kao heuristička informacija koristi duljina predmeta vođena istom idejom, što je dulji predmet, to će ostatak biti manji, međutim, ovako izražena heuristička informacija ima učinak da postaje važnija što je plan rezanja zalihe popunjeniji. Dok je plan rezanja prazan, relativni odnosi između ostataka će biti manji nego kad već sadrži određen broj predmeta.

$\tau_b(i)$ je prosječna vrijednost tragova između predmeta tipa *i* i svih predmeta trenutno u planu rezanja materijala:

$$\tau_b(i) = \begin{cases} \frac{\sum_{j \in s} \tau(i,j)}{|s|}, & ako\ s \neq \emptyset \\ 1, & inače \end{cases} \tag{2.9}$$

Vjerojatnost odabira određenog tipa materijala određena je izrazom:

$$p(x) = \frac{\tau(x)\eta(x)}{\sum_{y \in O(k)} \tau(y)\eta(y)} \tag{2.10}$$

gdje *O(k)* predstavlja skup raspoloživih tipova materijala nakon *k* odabira. Heuristička informacija može se koristiti primjerice za veću vjerojatnost odabira onih materijala koji su ostali od prijašnjih rezova, no u ovom radu se ne koristi jer se gleda isključivo minimizacija otpada.

Kao dobrota rješenja uzima se postotak iskorištenosti materijala

$$f(S) = \frac{\sum_{i=1}^{m} l_i}{\sum_{s_j \in G} s_j} \tag{2.11}$$

gdje je gornji zbroj označava ukupnu duljinu traženih predmeta, a donji ukupnu duljinu upotrijebljenih predmeta. Tragovi među predmetima postavljaju se posebno za svaki tip izvornog materijala prema sljedećem izrazu

$$\tau_{ij} \leftarrow \rho \cdot \tau_{ij} + t_{ij} \cdot f(S) \tag{2.12}$$

gdje t_{ij} predstavlja broj pojavljivanja predmeta duljine *i* i *j* u istom planu rezanja, promatrajući samo planove rezanja tog materijala. Zbog ovako definiranog traga i načina postavljanja tragova nije moguće postaviti gornju granicu na vrijednost traga. Minimalni trag se prema [3] aproksimira prema gruboj procjeni vjerojatnosti stvaranja najboljeg rješenja, no ovdje će se zadavati kao parametar algoritma s time da traženi predmeti i izvorni materijali koriste zasebne vrijednosti za minimalni trag. Trag materijala se postavlja prema recipročnoj vrijednosti srednje količine otpada za planove rezanja tog materijala.

Funkcija dobrote prezentirana u [3] uzima u obzir prosječan relativan ostatak na određenu potenciju *k*, te je zanimljiva jer nudi mogućnost određivanja koliko su povoljna rješenja koja imaju malo velikih i puno manjih ostataka, naspram onih u kojima je duljina ostataka ravnomjernije raspoređena, međutim ovdje nije bila primjenjiva jer želimo minimizirati ukupnu količinu otpada, a rješenja koja koriste više većih predmeta sa malo više ostatka mogu prema toj formuli imati jednaku dobrotu kao ona s više predmeta manje duljine s nešto manjim ostacima.

Grubi pseudokod algoritma dan je na Sl. 2.1.

```
postavi početne tragove
dok nije kraj
      za i = 0 do broj_mrava
            dok ima_još_narudžbi
                  odabrei materijal
                  popuni plan rezanja materijala
            kraj
            evaluiraj rješenje
            usporedi rješenje s najboljim i najboljim u iteraciji
                  ako je bolje, postaje novo najbolje
      kraj
      ako (broj_iteracije % γ == 0)
            postavi tragove prema najboljem mravu
      inače
            postavi tragove prema najboljem mravu u iteraciji
kraj
```

Sl. 2.1 Pseudokod algoritma MaxMinAS

2.2. Optimizacija rojem čestica (PSO)

2.2.1. Općenito o PSO

Algoritam roja čestica se zasniva na međuutjecaju jedinki (čestica) u populaciji. Ideja se temelji na promatranju ponašanja određenih jata ptica u potrazi za hranom. Svaka jedinka pamti svoj najbolji nađeni položaj i zna za najbolji od njenih susjeda, bilo da se radi o čitavoj populaciji ili samo o nekoliko jedinki. Također ima određeno povjerenje prema svojem najboljem rješenju i prema onome iz susjedstva te prema tome korigira svoju putanju. Sljedeća dva izraza opisuju promjenu brzine *i-te* čestice i novi položaj u koraku *t+1*:

$$v_{t+1,i} = \omega \cdot v_{t,i} + c_1 \cdot U(0,1) \cdot \left(P_i^{pBest} - P_i\right) + c_2 \cdot U(0,1) \cdot \left(P_i^{lBest} - P_i\right) \tag{2.14}$$

$$x_{t+1,i} = x_{t,i} + v_{t+1,i} \tag{2.15}$$

Pri tome je ω faktor inercije, c_1 „mjera povjerenja“ u vlastito rješenje, a c_2 mjera povjerenja u najbolje rješenje iz susjedstva, *U(0,1)* označava slučajno odabrani broj iz jednolike razdiobe na intervalu [0,1].

2.2.2. Primjena PSO na problem krojenja

Algoritam roja čestica je očito predviđen za pretraživanje kontinuiranog prostora, a ne rješavanje kombinatornih problema, stoga je potrebno uvesti nekoliko promjena. U [4] je opisan jedan način definiranja značenja točke i brzine u diskretnom prostoru za probleme koji se mogu prikazati kao slijed cijelih brojeva (ili simbola), što jest slučaj ako se ograničimo na samo jednu duljinu izvornog predmeta za rezanje. Također je dano nekoliko varijanti algoritma.

Jedna točka predstavlja jedno rješenje, što znači jedan redoslijed svih traženih predmeta ako svaki predmet tretiramo individualno. Razliku dvaju točaka P_2 - P_1 možemo definirati kao slijed zamjena potreban da bi se točka P_1 transformirala u P_2. Formira se na sljedeći način - ako je na indeksu l vektora P_2 predmet *i*, a u vektoru P_1 nije, onda se dodaje pokret *(i/l)*. Primjena tog pokreta ima sljedeći učinak – u vektoru P_1 nađe se pozicija *i-tog* predmeta i on zamijeni mjesta sa predmetom na poziciji l.

Množenje vektora brzine sa konstantom se definira na sljedeći način - označimo konstantu s *c*, te neka ona poprima vrijednost na intervalu [0,1]. Množenjem vektora s *c*, generira se jedan slučajni broj u intervalu [0,1] za svaki element vektor, ako je on veći od *c*, element se miče iz vektora.

Treba primijetiti dvije posljedice ovakve definicije, prvo parametri ω, c_1 i c_2 moraju poprimati vrijednosti u intervalu [0,1], drugo, vrijednost parametra ω mora biti manja od 1, jer će u protivnom vektor brzine neograničeno rasti s brojem iteracija.

Zbroj dvaju vektora brzina definira se kao dodavanje liste pokreta drugog vektora na kraj prvog. Zbroj točke i vektora brzine definira se tako da se svaki pokret vektora brzine primjeni na tu točku.

Najjednostavniji algoritam koji koristi samo prethodno navedene formule s novim definicijama točke i brzine naziva se *PSO_M1F*. U sklopu ovoga rada razmotren je također još jedan algoritam kojim se nastoji čuvati heterogenost populacije, konkretno radi se o algoritmu koji je u [4] nazvan DPSO*_poi_cpdyn*. U diskretnom DPSO*_poi* algoritmu se

nakon primjena dvije osnovne formule za dobivanje nove točke za svaki element točke generira slučajan broj. Ako je generirani broj manji od parametra c_p element zamjena mjesta s nasumično odabranim drugim elementom. Ova operacija se naziva disipacija. DPSO_*poi_cpdyn* algoritam koristi mjeru heterogenosti populacije kako bi dinamički odredio vrijednost parametra c_p. Budući da su vektori brzine proporcionalni broju elemenata na kojima su točke različite, njihova duljina može dati procjenu heterogenosti populacija. Mjera heteregenosti se računa prema sljedećem izrazu

$$het(t) = \frac{\sum_{i \in P} |v_{t,i}|}{D \cdot |P|} \tag{2.16}$$

gdje P označava populaciju, prema tome $|P|$ označava veličinu populacije, D je ukupan broj traženih predmeta. Budući da je cilj da c_p bude veći što je populacija homogenija, c_p treba biti manji što je mjera heterogenosti veća. Sljedeća funkcija se koristi u [4]

$$c_p = e^{-K \cdot het(t)} \tag{2.17}$$

gdje je K pozitivna konstanta koja se zadaje kao parametar algoritma.

Kako DPSOpo_cp-dyn ima ugrađeno sredstvo za održavanje heterogenosti populacije, koristit će se globalno najbolje rješenje pri računanju brzina. PSO_M1F međutim ne sadrži takav mehanizam te se stoga koristi lokalno najbolje rješenje. Za ostvarivanje susjedstva koristi se jednostavna prstenasta topologija. Tu se javlja razlika u odnosu na [4] gdje se uvijek koristi globalno najbolje rješenje.

Dobrota rješenja definirana je kao i kod algoritma kolonije mrava, kao omjer iskorištenosti, tj. ukupna duljina predmeta podijeljena ukupnom duljinom korištenog materijala.

Pseudokod algoritma DPSO_poi_pcdyn dan je na Sl. 2.2.

```
inicijaliziraj populaciju
dok nije kraj
      za i = 0 do vel_pop
            v_i = ω*v_i + c1*(P_i^pBest - P_i) + c2*(P^gBest-P_i)
            P_i = P_i + v_i
      kraj
      izracunaj heterogenost populacije
      izracunaj c_p
      za i = 0 do vel_pop
            provedi disipaciju nad P_i
            evaluiraj rješenje
            ako je rješenje bolje od trenutno osobno najboljeg
                  zamijeni osobno najbolje
      kraj
      nađi globalno najboljeg
kraj
```

Sl. 2.2 Pseudokod algoritma DPSO_*poi_codyn*

Pseudokod algoritma PSO_M1F dan je na Sl. 2.3.

```
inicijaliziraj populaciju
dok nije kraj
      za i = 0 do vel_pop
            v_i = ω*v_i + c1*(P_i^pBest - P_i) + c2*(P_i^lBest-P_i)
            P_i = P_i + v_i
            evaluiraj rješenje
      kraj
      za i = 0 do ve_pop
            nađi lokalno P_i^lBest
      kraj
kraj
```

Sl. 2.3 Pseudokod algoritma PSO_M1F

3. Programsko ostvarenje

U okviru ovog rada programski su ostvareni algoritmi navedeni u 2. poglavlju. Također se koristi programski sustav za ispitivanje algoritama. Svi algoritmi su napravljeni tako da poštuju pravila komunikacije koja sustav od njih zahtijeva, stoga prvo kreće opis sustava. Nad svakim algoritmom je napravljena paralelizacija, a detaljniji opis nalazi se u potpoglavlju pojedinog algoritma.

3.1. Programski sustav za ispitivanje

Za ispitivanje algoritama koristi se programski sustav [5]. Sustav je napravljen kao sučelje za rješavanje problema krojenja 1D. Može se koristiti na dva načina – iz perspektive krajnjeg korisnika koji unosi zadatak i traži njegovo rješenje, ili za ispitivanje algoritama koji se bave navedenim problemom, kao što je slučaj u ovome radu.

3.1.1. Pravila komunikacije

Pravila komunikacije nalažu da algoritam kao prvi i jedini argument komandne linije prima ime zadatka te očekuje da je zadatak naveden u datoteci „ime_zadatka.in“ u direktoriju „algoritmi“. Također očekuje u tom istom direktoriju datoteku po imenu „ime_algoritma.conf“ u kojoj su zadani parametri algoritma. Format ulazne datoteke je strogo propisan, no format datoteke s parametrima može svaki algoritam definirati za sebe. Sustav nema direktne veze s datotekom vezanom uz parametre te bi ona mogla biti izostavljena, no radi lakše modifikacije istih preoručuje se koristiti je. Format ulazne datoteke je sljedeći – u prvom retku nalaze se 3 broja, prvi predstavlja vrijeme izraženo u sekundama koje algoritam smije provesti u pokušaju traženja rješenja, drugi broj predstavlja debljinu rezne ploče, a treći maksimalnu veličinu prihvatljivog ostatka, tj. onog koji se smatra dovoljno malim da se nagradi rješenje koje ih više sadrži većom dobrotom, no zadnja dva broja se ne koriste u sklopu ovoga rada.

Sljedeći reci definiraju duljine zaliha i koliko ima svakog tipa na raspolaganju, u sljedećem formatu:

```
duljina   broj
```

pri čemu su duljina i broj odvojeni tabulatorom. Ako je broj -1, smatra se da zalihe ima u neograničenim količinama, što nije nerazumna pretpostavka ako se radi o materijalima proizvodne duljine koji se uvijek mogu dodatno nabaviti.

Nakon nabrajanja jednog ili više tipa zaliha slijedi redak koji se sastoji od 10 znakova '-', koji služi kao graničnik. Zatim slijedi jedan ili više redaka koji opisuju duljine predmeta koje treba rezati i potražnju pojedinih duljina navedenih u istom formatu kao za izvorne materijale. Treba napomenuti da svi algoritmi mogu pretpostavljati da će uvijek imati dovoljno materijala za izradu rješenja. Dužnost sustava za ispitivanje je da se brine o ispravnosti zadatka. Kako bi se mogli koristiti određeni primjeri zadataka za ispitivanje bilo je potrebno učiniti modifikacije u dijelu koda koji se brine o ispravnosti zadatka.

Nakon što završi s radom, što će za svaki od ovdje implementiranih algoritama biti ili istekom vremena ili dosegom određenog broja iteracija, algoritam će ispisati plan rezanja u izlaznu datoteku imena „ime_algoritma_ime_zadatka.out“ u direktoriju „rjesenja“, npr. za algoritam „MaxMinAS“ i ime zadatka „test“ izlazna datoteka će se zvati „MaxMinAS_test.out“. Važno je napomenuti da algoritam neće prebrisati datoteku, već će nadodati rješenje na kraj datoteke. Prvo se na kraj datoteke nadodaje niz od 10 znakova '#' koji služi kao graničnik među rješenjima. Potom se redak po redak zapisuje plan rezanja svakog izvornog materijala u formatu:

```
duljina_materijala  duljina_predmeta1+...+duljina_predmetaN+
```

Duljina materijala je tabulatorom odvojena od duljina predmeta, i redak završava sa znakom '+'.

Bilo koji algoritam koji implementira zadana pravila može se koristiti u sklopu sustava.

3.1.2. Datoteke koje sustav koristi

Algoritam se dodaje u sustav tako da se u datoteci „config.txt“ doda jedan redak za algoritam u kojem su navedeni redom ime algoritma, kategorija algoritma i staza, koja može biti i relativno zadana s obzirom na root direktorij do mjesta na kojem se algoritam nalazi, međusobno odvojeni tabulatorom. Npr. za MaxMinAS, redak je sljedeći

```
MaxMinAS  ACO  algoritmi\MaxMinAS.jar
```

Jedina ograničavajuća okolnost sustava u slučaju kad se koristi za ispitivanje je da se tipovi zaliha i njihove dostupne količine moraju unaprijed zadati u bazi podataka, međutim, kako se

isti mogu učitati iz datoteke, to nije veći problem. U nastavku su dani opisi formata datoteke u kojoj se definiraju vrste zaliha, u smislu imena, materijala i širine, zatim datoteke za navođenje duljina zaliha i količina za neku vrstu, te na koncu format zadatka u tekstualnoj datoteci. Datoteka u kojoj se definiraju tipovi zaliha sastoji se od jednog ili više redaka pri čemu je svaki redak formata:

```
ime_tipa_zalihe     širina    materijal boja
```

Sve stavke su međusobno odvojene tabulatorom. Datoteka u kojoj se navode zalihe i njihove količine koje se dodaju u bazu u prvom retku deklarira ime tipa zalihe o kojem se radi. Svaki sljedeći redak je oblika:

```
duljina   količina
```

Pri tome su stavke odvojene tabulatorom. Datoteka koja definira zadatak u prvom retku također deklarira tip zaliha koji se koristi u zadatku praćeno jednim ili više redaka koji definiraju tražene rezove formata:

```
duljina   potražnja
```

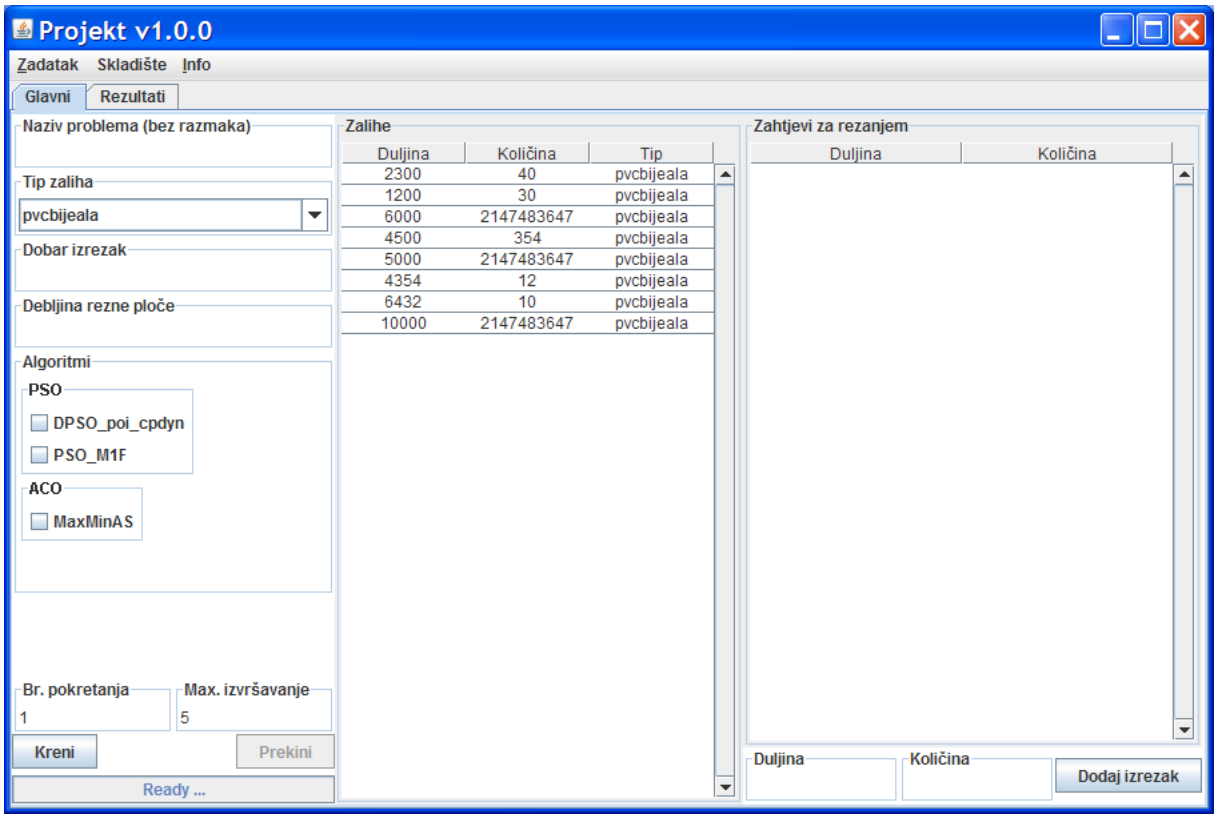
pri čemu su stavke odvojene tabulatorom. Alternativno ili dodatno se mogu definirati površine koje treba pokriti. Iza zadnjeg retka u kojem se eksplicitno definira duljina i potražnja (ili odmah iza deklaracije tipa zaliha) potrebno je navesti redak od 10 znakova '#' koji služi kao graničnik. Svi reci koji slijede su formata:

```
duljina   visina    broj_površina
```

Pri tome su stavke odvojene tabulatorom. Svaki redak definira površinu koju treba pokriti i broj takvih površina na temelju čega se računa koliko je određenih rezova potrebno.

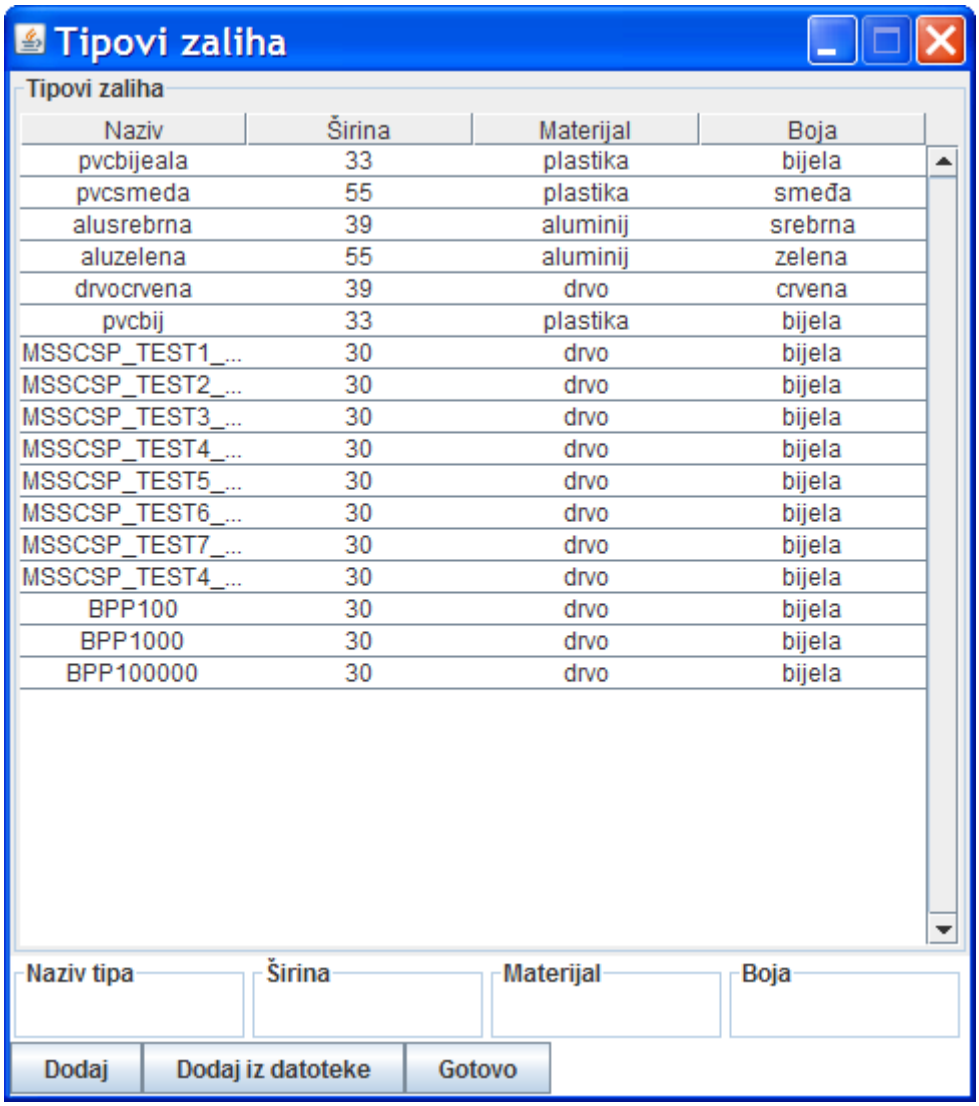
3.1.3. Upute za pokretanje

Za svaki problem se definira nova vrsta zalihe, datoteka koja za tu vrstu zalihe definira raspoložive duljine u količine, te datoteka sa zadatkom. Nakon toga potrebno je učitati zalihe u bazu, otvoriti zadatak i pokrenuti željene algoritme.



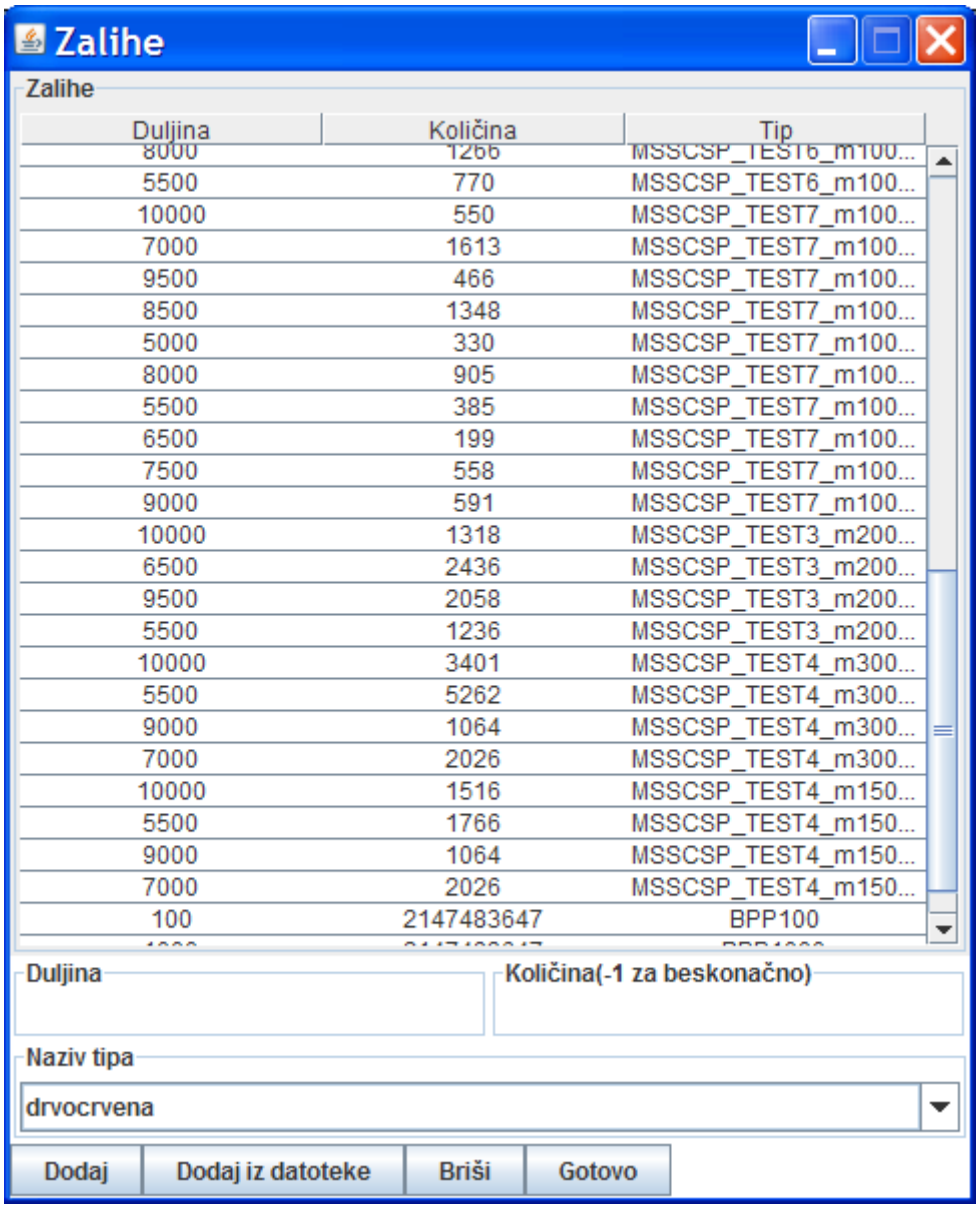
Sl. 3.1 Programski sustav za ispitivanje algoritama

Slika Sl. 3.1 prikazuje programski sustav za ispitivanje algoritama. Prvo je potrebno u bazu podataka unijeti vrstu zalihe, to se radi odabirom *„Skladište“* te zatim *„Tip zaliha“* nakon čega se otvara prozor prikazan slikom u nastavku(Sl. 3.2).



Sl. 3.2 Prozor za unošenje vrste zaliha

Preko prozora potrebno je ili upisati tip i odabrati *„Dodaj“* ili odabrati *„Dodaj iz datoteke“* te odabrati datoteku u kojoj se nalaze opisi vrsta zaliha prema specificiranom formatu. Zatim je potrebno učitati koje su duljine i u kojim količinama raspoložive za vrstu zalihe. To se radi odabirom *„Skladište“* te *„Zalihe“* čime se otvara prozor prikazan slikom u nastavku (Sl. 3.3).



Sl. 3.3 Prozor za unošenje zaliha

Zatim je potrebno ili za svaku zalihu upisati duljinu, količinu i naziv vrste zalihe te odabrati na „*Dodaj*“ ili odabrati „*Dodaj iz datoteke*“ te odabrati datoteku u kojoj se nalazi popis zaliha.

Dodavanje vrsta zaliha i njihovih raspoloživih duljina i količina potrebno je napraviti samo jednom. Svakim sljedećim pokretanjem programa one se čitaju iz baze podataka.

Zadatak se unosi odabirom „*Zadatak*“ iz glavnog izbornika, zatim „*Otvori*“ te odabirom datoteke u kojoj je naveden popis predmeta koje treba rezati. Alternativno, zadatak se može unijeti i preko samog programa te spremiti odabirom „*Zadatak*“ i „*Spremi*“.

Kad je zadatak učitan, potrebno je odabrati algoritme koji se žele pokrenuti odabirom polja pored imena algoritma da se pojavi kvačica. U polje *„Br. pokretanja“* upisuje se koliko puta se želi pokrenuti svaki algoritam, a u polje *„Max. izvršavanje“* najveće dopušteno vrijeme izvršavanja u sekundama. Za svrhe ovog rada se u polje *„Dobar izrezak“* i *„Debljina rezne ploče“* upisuje 0, njihovo značenje je objašnjeno ranije pri opisu komunikacijskih datoteka. Odabirom *„Kreni“* algoritmi se slijedno pokreću te se obrađuju njihovi rezultati kako svaki završava s izvođenjem. Odabirom *„Prekini“* može se prekinuti daljnje pokretanje algoritama. Rezultati se mogu vidjeti odabirom kartice *„Rezultati“* (Sl. 3.4).

Sl. 3.4 Prikaz rezultata

Odabir algoritma za kojeg se žele vidjeti rezultati vrši se preko padajućeg izbornika. Za svaki algoritam se ispisuje ukupna količina otpada, korištenog materijala i iskorištenost najboljeg rješenja te prosječna iskorištenost materijala.

3.2. MaxMinAS

Format datoteke s parametrima je sljedeći – u svakom retku se nalazi jedan parametar, s time da se ime parametra nalazi s lijeve strane znaka '=' a vrijednost parametra s desne. Redoslijed navođenja parametara nije bitan. Neki ili svi parametri mogu biti izostavljeni u kojem slučaju poprimaju podrazumijevane vrijednosti kako je definirano unutar programskog koda. Imena parametara i njihovo značenje dano je tablicom 3.1.

Tablica 3.1 MaxMinAS – imena i značenje parametara

Ime parametra	**Značenje**
evaporation constant	ρ, konstanta isparavanja
gama	γ, učestalost postavljanja tragova od strane najboljeg mrava
alpha	α, važnost vrijednosti tragova
beta	β, važnost heurističke informacije
minimum trail	τ_{min}, minimalni iznos feromonskog traga za tražene predmete
minimum trail stock	τ_{min}, minimalni iznos feromonskog traga za izvorne materijale
max iterations	najveći dopušteni broj iteracija algoritma
max ants	broj mrava koji svaku iteraciju grade rješenja

Paralelizacija algoritma je ostvarena na razini iteracije, tako da je svaki mrav predstavljen jednim poslom, pronalaskom rješenja, te svaki zasebno pronalazi rješenje. Promatrajući pseudokod iz poglavlja 2, pronalazak rješenja je paraleliziran, dok postavljanje tragova nije. Cijena sinkronizacije je stavljanje poslova u red poslova, uzimanje istih od strane radnih dretvi te stavljanje rješenja u red rješenja i uzimanje istih od glavne dretve. Nad svim globalnim podacima obavlja se isključivo čitanje, a svaka dretva ima svoje lokalne pomoćne strukture koje se koriste pri pronalasku rješenja.

3.3. PSO_M1F

Format datoteke s parametrima je sličan formatu koji koristi algoritam kolinije mrava, dakle navodi se jedan parametar po redu tako da je s lijeve strane znaka '=' ime parametra, a s desne vrijednost koju treba poprimiti. Također se ne moraju navesti svi parametri. Parametri koji se ne navedi poprimit će podrazumijevane vrijednosti vrijednosti. Imena parametara i njihovo značenje dano je tablicom 3.2.

Tablica 3.2 PSO_M1F – imena i značenje parametara

Ime parametra	**Značenje**
population size	veličina populacije, broj čestica
inertia weight	ω, faktor inercije, određuje utjecaj trenutne brzine pri stvaranju nove
personal best coef	c_1, koeficijent sklonosti prema vlastitom najbolje nađenom rješenju
local best coef	c_2, koeficijent sklonosti prema lokalno najbolje nađenom rješenju
max iterations	najveći dopušteni broj iteracija algoritma
neighbour count	broj susjeda s lijeva i desna od čestice

Algortiam će iz datoteke zadatka uzeti najveću zalihu u neograničenim količinama ili najveću zalihu ako takve nema.

Paralelizacija je u ovom algoritmu također provedena na razini iteracije. Promatrajući pseudokod iz poglavlja 2, jedna iteracije se dijeli na dvije vrste poslova. Prva vrsta uključuje izračun nove brzine, izračun nove pozicije točke i evaluaciju iste, dok druga vrsta poslova uključuje pronalazak lokalno najboljeg rješenje za i-tu točku.

Cijena sinkronizacije je stavljanje prvih poslova u red poslova, uzimanje istih od strane radnih dretvi te čekanje da se izvrše, stavljanje drugih poslova u red poslova, uzimanje istih od strane radnih dretvi te čekanje da se izvrše.

3.4. DPSO_*poi_cpdyn*

Format datoteke s parametrima je sličan formatu koji koristi PSO_M1F algoritam s razlikom da se umjesto „local best coef“ koristi „global best coef“, te dodatnim parametrom *K*. Imena parametara i njihovo značenje dano je tablicom 3.3.

Tablica 3.3 DPSO_*poi_cpdyn* – imena i značenje parametara

Ime parametra	**Značenje**
population size	veličina populacije, broj čestica
inertia weight	ω, faktor inercije, određuje utjecaj trenutne brzine pri stvaranju nove
personal best coef	c_1, koeficijent sklonosti prema vlastitom najbolje nađenom rješenju
global best coef	c_2, koeficijent sklonosti prema globalno najbolje nađenom rješenju
diversity coef factor	K, konstanta kojom se regulira vrijednost funkcije koja određuje c_p
max iterations	najveći dopušteni broj iteracija algoritma

Algortiam će iz datoteke zadatka uzeti najveću zalihu u neograničenim količinama ili najveću zalihu ako takve nema.

Paralelizacija se provodi na razini iteracije. Promatrajući pseudokod iz poglavlja 2, jedna iteracija se dijeli na dvije vrste poslova. Prva vrsta poslova uključuje izračun nove brzine i nove pozicije točke, a druga vrsta poslova uključuje disipaciju točke i evaluaciju iste. Izračun heterogenosti populacije kao ni pronalazak najboljeg rješenja nije paraleliziran.

Cijena paralelizacije je slična kao kod algoritma PSO_M1F, jedina razlika je u definiciji pojedinih poslova.

4. Eksperimentalni rezultati

Algoritmi su pokrenuti na nekoliko primjeraka problema čije je rješenje poznato te su njihove performanse evaluirane. Za svaki problem svaki algoritam je dobio 10 sekundi vremena te je pokrenut 30 puta, nakon čega su zabilježeni najbolji i prosječni rezultat. Računalo na kojem su se pokretali algoritmi sadrži dvojezgreni Intel Core 2 Duo 2GHz procesor. Prvo su dani rezultati na primjerima koji koriste više različitih duljina zaliha, zatim rezultati na primjerima koji koriste samo jednu duljinu zaliha.

4.1. Različite duljine zaliha

Korišten je skup problema koji je riješen je u sklopu rada [6]. Primjeri koji su korišteni dostupni su preko http://www.math.tu-dresden.de/~capad/ te su od tamo preuzeti. Iz navedenog skupa problema odabrano je šest na kojima je ispitan algoritam MaxMinAS. Karakteristike problema u obliku broja različitih duljina zaliha(M) i traženih predmeta(m) dane su tablicom 4.1.

Tablica 4.1 Karakteristike ispitnih problema

Broj problema	**M**	**m**	**ukupno predmeta**
1	2	98	5299
2	4	96	5299
3	4	198	10 294
4	4	150	7070
5	7	98	5299
6	10	98	5299

Korišteni parametri algoritma dani su tablicom 4.2.

Tablica 4.2 Vrijednosti parametara algoritma MaxMinAS

Ime parametra	**Vrijednost**
evaporation constant	0.95
gama	4
alpha	1.3
beta	1.9
minimum trail	0.3
minimum trail stock	0.000001
max ants	50

Rezultati algoritma dani su tablicom 4.3. Prvi stupac predstavlja broj problema, drugi ukupnu količinu materijala u optimalnom rješenju, treći ukupnu količinu materijala u najboljem nađenom rješenju, četvrti postotak iskorištenosti najboljeg rješenja te peti prosječni postotak iskorištenosti.

Tablica 4.3 Rezultati algoritma MaxMinAS

#	**Opt. mat.**	**Najbolji mat.**	**Najbolji %**	**Prosjek %**
1	20 674 000	21 062 000	97.35	97.28
2	4 101 500	4 107 000	99.85	99.84
3	40 854 500	41 657 000	97.78	97.74
4	26 402 000	26 693 500	98.84	98.53
5	20 534 500	20 950 500	97.87	97.73
6	20 529 000	20 911 000	98.06	97.92

Tablica 4.4 prikazuje odstupanje algoritma od optimuma. Prvi stupac predstavlja broj problema, drugi prikazuje postotak viška materijala naspram optimalnog rješenja, računato kao razlika u količini materijala podijeljena optimalnom količinom - u najboljem slučaju, a treći stupac prikazuje isti postotak za prosječni slučaj. U četvrtom stupcu je dana razlika između postotka iskorištenosti materijala najboljeg i optimalnog rješenja, a u petom stupcu između najboljeg i prosječnog rješenja.

Tablica 4.4 Odstupanje od optimuma MaxMinAS

#	**Višak materijala najbolji (%)**	**Višak materijala prosjek (%)**	**Razlika iskorištenosti najbolji**	**Razlika iskorištenosti prosjek**
1	1.88	1.96	1.83	1.90
2	0.13	0.15	0.14	0.15
3	1.96	2.01	1.92	1.96
4	1.10	1.42	1.09	1.40
5	2.03	2.18	1.99	2.13
6	1.86	2.00	1.82	1.96

Najveće odstupanje od optimuma u prosječnom slučaju izraženo kao postotak viška materijala je 2.18%, a kao razlika naspram iskorištenosti u najboljem rješenju 2.13%.

4.2. Iste duljine zaliha

U [7] je korišten skup problema za usporedbu algoritama za rješavanje problema rezanja dostupan na http://www.wiwi.uni-jena.de/Entscheidung/binpp/, otkud su preuzeti i problemi za ispitivanje korišteni u ovom radu. Problemi su grupirani u tri težinske kategorije, u lake, srednje i teške probleme. Iz svake kategorije su odabrana dva problema nad kojima se provodilo ispitivanje. Svi problemi imaju samo jednu duljinu zaliha u neograničenim količinama, te su traženi predmeti u velikom broju različitih duljina s malim potražnjama za svaku duljinu, no razmak između dvije duljine ako ih sortiramo po veličini je relativno mali.

Tablica 4.5 Vrijednosti parametara algoritma DPSO_*poi_cpdyn*

Ime parametra	**Vrijednost**
population size	13
inertia weight	0.7
personal best coef	0.13
global best coef	0.7
diversity coef factor	4.7

Tablica 4.5 prikazuje vrijednosti parametara korištenih za algoritam DPSO_*poi_cpdyn*, dok tablica 4.6 prikazuje vrijednosti parametara algoritma PSO_M1F.

Tablica 4.6 Vrijednosti parametara algoritma PSO_M1F

Ime parametra	**Vrijednost**
population size	13
inertia weight	0.4
personal best coef	0.87
global best coef	0.37
neighbour count	2

Tablica 4.7 Rezultati algoritma DPSO_*poi_cpdyn*

#	Opt	Najbolji	Najbolji %	Prosjek %
1	25	26	93.62	91.88
2	82	82	83.01	82.38
3	18	19	89.13	89.13
4	22	22	96.34	92.29
5	57	60	90.67	89.98
6	55	58	93.09	90.60

Rezultati algoritama DPSO_*poi_cpdyn* i PSO_M1F su dani tablicama 4.7, odnosno 4.8. Optimalno rješenje je izraženo kao minimalni broj predmeta koje je potrebno rezati. Sukladno tome, u trećem stupcu se umjesto ukupne količine materijala navodi broj korištenih zaliha.

Tablica 4.8 Rezultati algoritma PSO_M1F

#	Opt	Najbolji	Najbolji %	Prosjek %
1	25	26	93.62	91.88
2	82	82	83.01	82.38
3	18	19	89.13	89.13
4	22	23	92.15	92.15
5	57	60	90.67	89.58
6	55	59	91.51	90.19

Može se vidjeti da je algoritam DPSO_*poi_cpdyn* u dva slučaja našao optimalno rješenje, dok je PSO_M1F našao samo u jednom slučaju. Ukupno u 2 slučaja algoritam DPSO_*poi_cpdyn* je našao bolje rješenje od PSO_M1F, međutim, u prosjeku oba algoritma daju približno jednak postotak iskorištenosti.

Zaključak

U ovom radu opisana je primjena algoritama zasnovanih na inteligenciji rojeva na problem krojenja 1D. Dan je opis problema krojenja te su napravljena tri algoritma za dva slučaja problema. Općenitiji slučaj može sadržavati zalihe različitih duljina, dok drugi slučaj sadrži samo zalihe jedne duljine. Algoritam MaxMinAS, temeljen na optimizaciji kolonijom mrava, preciznije na MaxMinAS algoritmu, može raditi za općenitiji slučaj. Algoritmi DPSO_*poi_cpdyn* i PSO_M1F temelje se na optimizaciji rojem čestica i prilagođeni da rade na kombinatornom problemu, no u trenutnoj implementaciji mogu se primijeniti samo na probleme koji imaju jednu duljinu zalihe. DPSO_*poi_cpdyn* koristi procjenu heterogenosti populacije na temelju koje pojačava eksploraciju, dok PSO_M1F isto nastoji postići uporabom lokalno najboljeg rješenja pri osvježavanju brzine.

Pronađeni su ispitni problemi čija su egzaktna rješenja poznata te su na njima ispitani algoritmi. Ispitivanje se vršilo uporabom programskog sustava opisanog u poglavlju 3. Pokazalo se da algoritmi DPSO_*poi_cpdyn* i PSO_M1F daju u prosjeku približno jednake rezultate, iako je DPSO_*poi_cpdyn* u dva slučaja dao bolji rezultat od PSO_M1F.

Algoritam MaxMinAS ne uspijeva niti u jednom slučaju naći optimalno rješenje, međutim po ukupnoj količini materijala nalazi rješenja s najvećim odstupanjem od optimuma od 2.18% u prosječnom slučaju, te s najvećom razlikom u iskorištenosti naspram optimuma od 2.13% za isti problem.

Ideje za daljnji rad uključuju modifikacije PSO algoritama za općenitiji slučaj problema te dodavanje lokalne pretrage u algoritme.

Literatura

[1] Dyckhoff, H: „A typology of cutting and packing problems“, European Journal of Operational Research 44, 145-149, Elsevier Science Publishers B. V., 1990.

[2] Dorigo, M.; Stützle, T: „Ant Colony Optimization“, A Bradford book, The MIT Press, Cambridge, Massachusets, 2004.

[3] Levine, J; Ducatelle F: „Ant colony optimization and local search for bin packing and cutting stock problems“, Journal of the Operational Research Society 55, 705-716, 2004.

[4] Garcia-Villoria, A; Pastor R: „Introducing dynamic diversity into a discrete Particle Swarm Optimization“, Computers & Operations Research, Volume 36, Issue 3, ožujak 2009.

[5] Modrić, A; Domazet, F; Maržić, M: „Problem krojenja 1D“, programski projekt

[6] Belov, G; Scheithauer, G: „A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths“, European Journal for Operational Research 141, 274-294, 2002.

[7] Bingul, Z; Oysu, C: „Comparison of Stochastic and Approximation Algorithms for One-Dimensional Cutting Problems“, Advances in Intelligent Computing: International Conference on Intelligent Computing, ICIC 2005, Proceedings, Hefei, Kina, 2005.

[8] Trkman, P; Gradisar M: „One-dimensional cutting stock optimization in consecutive time periods“, European Journal of Operational Research, 179, 291-301, 2007.

Uporaba algoritama zasnovanih na inteligenciji roja za rješavanje problema krojenja

Sažetak

Problem krojenja je važan problem koji se javlja u industriji, gdje se teži što više uštedjeti na upotrijebljenom materijalu i smanjiti stvoreni otpad. Problem je NP-težak zbog čega se pribjegava metodama koje ga aproksimativno rješavaju. Algoritmi zasnovani na inteligenciji roja su neki od takvih metoda. Temelje se na jednostavnim interakcijama između agenata iz čega izranja inteligentno ponašanje na razini populacije. U ovom radu su na 1D problem krojenja primijenjeni algoritam kolonije mrava i algoritam roja čestica. Napravljena je programska implementacija opisanih algoritama, te je korišten programski sustav za ispitivanje algoritama na skupu problema čija su egzaktna rješenja poznata.

Ključne riječi: 1D problem krojenja, algoritam kolonije mrava (ACO), algoritam roja čestica (PSO), inteligencija roja

Applying Swarm Intelligence Based Algorithms to Solve the Tailoring Problem

Abstract

The tailoring problem is an important problem which occurs in industry, where it is important to minimize the amount of used material and generated waste. It is an NP-hard problem, because of which approximation methods are often used to solve it. Swarm intelligence based algorithms are some of those methods. They are based on simple interactions between agents from which an intelligent behaviour emerges at the population level. In this work, Ant Colony Optimization and Particle Swarm Optimization algorithms are applied to the 1D tailoring problem. The algorithms have been implemented and a software system was used to test them on a set of problems with known exact solutions.

Key words: 1D Tailoring Problem, Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), Swarm Intelligence