SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4845

RJEŠAVANJE PROBLEMA KROJENJA KORIŠTENJEM EVOLUCIJSKIH ALGORITAMA

Luka Abramušić

Zagreb, lipanj 2017.

Zahvaljujem se svojem mentoru, Marinu Golubu, te doktorandu Danijelu Domoviću na pruženoj pomoći pri istraživanju problematike i implementaciji rješenja.

SADRŽAJ

1. Uvod **1**

2. Problem krojenja **2**
2.1. Formalna definicija problema krojenja 3
2.2. Primjer jednodimenzijskog problema pakiranja 4
2.3. Dvodimenzijski problem krojenja 5

3. Evolucijski algoritmi **7**
3.1. Genetski algoritmi 8
3.2. Genetski algoritmi za rješavanje 2D problema krojenja 10
3.2.1. Operatori križanja 10
3.2.2. Operatori mutacije 12
3.2.3. Funkcija dobrote 12

4. Programska implementacija **15**
4.1. Ulazna i izlazna datoteka 15
4.2. Parametri algoritma 16
4.3. Evaluacijska funkcija 18
4.4. Upute za korištenje 21
4.5. Budući rad 22

5. Analiza rezultata **23**
5.1. Eksperimenti 23

6. Zaključak **26**

Literatura **27**

1. Uvod

Problem krojenja se ubraja među probleme kombinatoričke optimizacije te je vrlo teško pronaći optimalno rješenje u razumnom vremenu. Genetski algoritmi se u zadnje vrijeme zbog svojeg izuzetnog omjera brzine i kvalitete rješenja koriste za rješavanje velikog broja problema koje je teško riješiti deterministički. Ovaj rad analizira primjenu genetskih algoritama na problemu krojenja u dvodimenzionalnom prostoru.

Nakon uvoda slijedi opis problema krojenja. Daje se općenita formulacija problema, navodi se primjer krojenja i dodatni načini definiranja problema. Zatim se opisuje dvodimenzijski problem krojenja s naglaskom na tip problema koji se obrađuje u ovom radu. U trećem poglavlju se opisuje genetski algoritam. Definirani su algoritmi za rješavanje 2D problema krojenja, opisani su genetski operatori i funkcija dobrote. U četvrtom poglavlju opisan je ostvareni programski sustav. Opisuju se datoteke koje sustav koristi za rad i ostali parametri, a potom su dane korisničke upute koje opisuju način korištenja programskog sustava. U petom poglavlju se analiziraju rezultati za pojedine ulazne datoteke. Zadnje poglavlje je zaključak.

2. Problem krojenja

Problem krojenja (eng. Cutting Stock Problem, CSP) je problem rezanja raspoloživog materijala u potrebne manje komade (predmete) s ciljem da se minimizira količina otpada (neiskoristivi materijal). Problem krojenja je NP-potpun problem kombinatoričke optimizacije svediv na problem pakiranja. Problem pripada NP klasi ako je nedeterminističke polinomijalne vremenske složenosti. Problem je NP-potpun ako je NP i NP-težak. Zbog toga se smatra kako nema algoritma koji bi ga mogao riješiti u polinomijalnom vremenu pa se često koriste algoritmi koji daju aproksimativno rješenje.

Zajedno s problemom pakiranja predmeta (eng. Bin Packing Problem, BPP) pripada grupi problema pakiranja i rezanja (eng. Cutting and Packing, C&P). Problemi iz ove grupe nastoje minimizirati neiskorišteni ostatak korištenog materijala. Zastupljeni su u mnogim industrijama gdje se teži uštedjeti na upotrebljenim materijalima i smanjiti otpad, primjerice tekstilnoj, drvnoj, metalnoj i papirnoj industriji. Svi problemi iz te grupe problema se mogu rješavati istim algoritmima, ako imaju postavljena ista ograničenja. Iz tog razloga se za problem krojenja koriste isti algoritmi kao i za rješavanje problema pakiranja predmeta.

Za rješavanje se općenito koriste dva pristupa: fokusiranjem na objekte i fokusiranjem na uzorke. Prvi pristup pravi zahtjeve za izrezivanje redom po raspoloživim materijalima, dok drugi pristup kombinira raspoložive materijale iz čega se potom izrezuju predmeti. Prvi pristup je prikladniji za korištenje ako postoji više veličina materijala, dok su za jednu veličinu materijala oba pristupa jednako prikladna.

2.1. Formalna definicija problema krojenja

Standardna formulacija problema krojenja zadaje listu od m predmeta, od kojih je svaki potrebno izrezati u q_j komada, gdje je $j = 1, ..., m$. Konstruira se lista svih mogućih kombinacija izrezivanja koje nazivamo uzorcima. Neka je n broj takvih uzoraka. Svakom uzorku se pridruži pozitivan cijeli broj x_i koji označava koliko će se puta koristiti uzorak i, gdje je $i = 1, ..., n$. Formulacija problema tada glasi:

$$min \sum_{i=1}^{n} c_i x_i \tag{2.1}$$

$$t.d. \sum_{i=1}^{n} a_{ij} x_i \geq q_j, \forall j = 1, ..., m \tag{2.2}$$

$$x_i \geq 0, x_i \in \mathbb{Z} \tag{2.3}$$

gdje je a_{ij} broj pojavljivanja predmeta j u uzorku i, a c_i je cijena (ostatak) uzorka i. Navedena ograničenja su minimalna ograničenja te zahtijevaju izrezivanje barem zadane količine svakog predmeta. Nad osnovnom formulacijom problema mogu biti postavljeni dodatni zahtjevi, čime algoritam postaje specijaliziran za određeni problem. Međutim, u svakoj inačici algoritma se koristi količina ostatka kao mjera efikasnosti rješenja. Optimalno rješenje nije jedinstveno. Često postoji više rješenja s jednakom količinom ostatka za koja se naknadno može zahtijevati bolja zadovoljenost nekih od ostalih zahtjeva.

Prema tipologiji koju je postavio H. Dyckhoff, problemi krojenja se razlikuju prema četiri glavna obilježja [9]:

1. dimenzionalnost
   - (1) jednodimenzionalni
   - (2) dvodimenzionalni
   - (3) trodimenzionalni
   - (N) višedimenzionalni
2. način dodjeljivanja
   - (B) svi materijali i odabir predmeta
   - (V) odabir materijala i svi predmeti
3. izbor materijala
   - (O) jedan materijal
   - (I) materijali identičnih dimenzija
   - (D) materijali različitih dimenzija

4. izbor predmeta
    - (F) mali broj različitih predmeta
    - (M) veliki broj različitih predmeta
    - (R) veliki broj sličnih predmeta
    - (C) jednaki predmeti

Ova tipologija je kritizirana u posljednjih nekoliko godina jer se pokazala nedovoljno obuhvatnom za probleme koji su postali sve istraženiji, međutim u sklopu ovog rada će biti dovoljna za korištenje upravo zbog svoje jednostavnosti.

2.2. Primjer jednodimenzijskog problema pakiranja

Koristimo stroj koji može proizvesti neograničen broj velikih rola papira širine po 5600mm. Potrebno je izrezati 13 narudžbi prikazanih u tablici 2.1.

Tablica 2.1: Primjer narudžbe za rezanje

Širina	Broj rola
1380	22
1520	25
1560	12
1710	14
1820	18
1880	18
1930	20
2000	10
2050	12
2100	14
2140	16
2150	18
2200	20

Za ovakav mali primjer postoji 308 mogućih načina rezanja. Optimalno rješenje zahtijeva 73 role papira uz 0.401% ostatka. Računski se može pokazati da postoji 19 različitih rješenja s minimalnom količinom ostatka, od kojih je jedno prikazano u tablici 2.2.

Tablica 2.2: Jedno od mogućih rješenja primjera

Broj ponavljanja	Uzorak
2	1820 + 1820 + 1820
3	1380 + 2150 + 1930
12	1380 + 2150 + 2050
7	1380 + 2100 + 2100
12	2200 + 1820 + 1560
8	2200 + 1520 + 1880
1	1520 + 1930 + 2150
16	1520 + 1930 + 2140
10	1710 + 2000 + 1880
2	1710 + 1710 + 2150
73	

2.3. Dvodimenzijski problem krojenja

U dvodimenzijskom problemu krojenja potrebno je iz površine raspoloživog materijala izrezati zadane oblike tako da se površina što bolje iskoristi, odnosno da je površina ostatka što manja. Posebnost 2D problema krojenja je pojava problema razmještaja predmeta unutar materijala. Vrlo je bitan redoslijed izrezivanja predmeta, budući da je njime određena iskorištena površina materijala i pripadni ostatak. Predmeti također mogu biti fiksno orijentirani ili mogu biti rotirani za proizvoljan kut.

Specifičan problem za 2D problem krojenja je problem krojenja role. Poznata je samo širina materijala, dok se ukupna visina rezanih predmeta nastoji minimizirati na roli beskonačne duljine. Problem u ovom obliku se često susreće u tekstilnoj industriji. Otpad je definiran kao razlika zauzete pravokutne površine i ukupne površine izrezanih oblika. Budući da je unaprijed zadana, širina platna se često koristi umjesto stvarne zauzete širine za izračun zauzete površine. Zbog prirode materijala koji se koristi u tekstilnoj industriji, predmeti mogu biti rotirani jedino za $180°$. Budući da se ovaj rad ograničava na pravokutnike, rotacija neće biti uzeta u obzir jer ne čini nikakvu razliku u domeni ovog problema. Algoritmi definirani u idućem poglavlju usmjereni su upravo na ovakav tip problema.

Drugi tip problema za 2D problem krojenja je problem minimizacije površine. U ovom slučaju i širina i visina platna su varijabilne te se nastoji pronaći najmanja površina u koju stanu predmeti. Postoje i problemi drukčijih ograničenja gdje su dimenzije spremnika fiksne. U problemu pakiranja u spremnike (eng. bin packing problem), teži se minimizirati broj korištenih spremnika fiksnih dimenzija kako bismo smjestili sve predmete. U problemu pakiranja u naprtnjaču (eng. knapsack problem) cilj je naći podskup zadanih predmeta takav da maksimalno moguće popune spremnik (naprtnjaču) fiksnih dimenzija [6].

U ovom radu su obrađeni problemi tipa 2/V/O/F prema Dyckhoffovoj tipologiji. Obrađeni su dvodimenzionalni problemi krojenja gdje je potrebno ostvariti sve zahtjeve za rezanjem na zadanom materijalu. Koristi se jedan materijal za rezanje, a izrezuje se mali broj predmeta u puno različitih dimenzija. Dodatno, predmeti su ograničeni na pravokutnike zbog jednostavnosti implementacije algoritama.

3. Evolucijski algoritmi

Evolucijski algoritmi su potpodručje evolucijskog računanja, područja umjetne inteligencije koje se bavi općenitim metaheurističkim algoritmima optimizacije zasnovanim na populaciji. Metaheuristike su heuristike opće namjene čiji je zadatak usmjeravanje problemski specifičnih heuristika prema prostoru dobrih rješenja. Osnovni cilj evolucijskih algoritama je generiranje rješenja optimizacijskih problema pomoću tehnika inspiriranih biološkom evolucijom. Po načinu djelovanja ubrajaju se u metode usmjerenog slučajnog pretraživanja prostora rješenja u potrazi za globalnim optimumom. Razlikuje se više implementacija, od kojih su najznačajniji genetski algoritmi, genetsko programiranje i evolucijske strategije. Najpopularniji su genetski algoritmi (eng. genetic algorithms, GA), koji će i biti razmatrani u nastavku ovog rada.

Evolucija je neprekidan prirodni proces prilagodbe živih bića na svoju okolinu, odnosno na uvjete u kojima žive. U prirodi vlada nemilosrdna borba za opstanak u kojoj preživljavaju jedinke s najbolje prilagođenim osobinama, a odumiru jedinke s lošije prilagođenim osobinama. Danas se pretpostavlja da su sva svojstva jedinke zapisana u kromosomima. Kromosomi su lančaste tvorevine koje se nalaze u jezgri svake stanice, što znači da svaka stanica živih bića posjeduje sve informacije o svim svojstvima jedinke. Skup informacija koje obilježavaju jedno svojstvo zapisano je u jedan djelić kromosoma koji se naziva gen. Poistovjeti li se mjera sposobnosti pojedinca da preživi s nasljednim materijalom koji nosi u sebi, može se reći da geni dominantnih pojedinaca opstaju dok geni slabijih izumiru. Promatrajući samo gene, u svakoj novoj generaciji nastaje novi skup gena, od kojih su neki lošiji, a neki bolji od prethodne generacije.

Proces odabira jedinki koje će preživjeti i dalje se reproducirati iz populacije naziva se prirodna selekcija. Opstanak vrste stoga određuje njezin stupanj prilagodbe okolini u kojoj živi. Proces prilagođavanja je neprekidan jer se životni uvjeti i okolina neprekidno mijenjaju. Razmnožavanjem jedinki nastaju nove generacije jedinki koje nasljeđuju svojstva od roditelja. Tijekom tog procesa mogu nastati mutacije, koje stvaraju različitosti u genetskom materijalu. Nastaju jedinke koje su više ili manje prilagođene trenutnim životnim uvjetima, ovisno o svojstvima koja posjeduju. Jedinke s lošijim svojstvima imaju manju vjerojatnost preživljavanja te odumiru, sprječavajući širenje lošijih svojstava na sljedeću generaciju. Na taj način će većina jedinki posjedovati dobra svojstva, čime vrsta postaje prilagođenija iz generacije u generaciju.

Iako na prvi pogled ne postoji jasno vidljiva poveznica između spomenutih domena, lako se dolazi do zaključka da su principi darvinizma (bolji opstaje, gori izumire) idealni za pronalaženje dovoljno dobrog rješenja zadanog problema (ako ne i optimalnog). Simuliranje prirodnog evolucijskog procesa na računalu svodi se na grube aproksimacije rješenja.

3.1. Genetski algoritmi

Genetski algoritmi su skup algoritama zasnovanih na biološkoj evoluciji koji pronalaze rješenja zadanog problema. U GA pretraga započinje od cijelog niza rješenja koji predstavlja populaciju. Populacija potencijalnih rješenja se optimizira primjenjujući princip preživljavanja najsposobnijih, odnosno jedinki koje su zadovoljile kriterij uspješnosti više nego druge. Iteracije stvaranja novih generacija se ponavlja dok se ne pronađe zadovoljavajuće rješenje [4].

U genetskim algoritmima se izmjena gena pri reprodukciji naziva križanje, iako to nije sasvim u skladu s biologijom. Slučajna izmjena genetskog materijala pod djelovanjem vanjskih uzroka se naziva mutacija, no njena pojava je znatno manja. Križanje i mutacija se kod genetskih algoritama nazivaju genetski operatori, a proces izdvajanja najsposobnijih jedinki unutar svake generacije se naziva odabir ili selekcija. GA selekcijom odabire uglavnom bolje jedinke, ali i slabije imaju neku šansu preživljavanja. GA konvergira k optimumu zahvaljujući križanju, dok se mutacija brine da to ne bude lokalni optimum. Brzina i točnost konvergencije GA uvelike ovisi o parametrima algoritma kao što su veličina populacije, odabrani operatori križanja i mutacije te vjerojatnosti njihove uporabe.

Kao što evolucija u prirodi djeluje nad skupom jedinki tako i svaki evolucijski program održava populaciju jedinki u nekoj određenoj generaciji. Svaka jedinka predstavlja potencijalno rješenje problema koji se obrađuje. U genetskom algoritmu svaka je jedinka predstavljena jednakom podatkovnom strukturom — obično broj, niz, matrica ili stablo, ovisno o problemu koji se rješava. Te jedinke se nazivaju kromosomi.

Prilikom inicijalizacije generira se početna populacija jedinki. Obično se generira slučajnim odabirom rješenja iz domene ili usađenjem početnog rješenja dobivenog nekom drugom optimizacijskom metodom u početnu populaciju. Svakom rješenju se pridjeljuje određena mjera kvalitete izražena u postotcima koja se obično naziva dobrota (fitness), dok se funkcija koja tu kvalitetu određuje naziva funkcija cilja ili funkcija dobrote. Ona uvijek ovisi o problemu koji se rješava. Iz stare populacije se formira nova, izdvajajući bolje jedinke iz skupa postojećih po unaprijed definiranom postupku selekcije.

Pojedini članovi ove nove populacije podvrgnuti su utjecajima genetskih operatora koji iz njih formiraju nove jedinke. Genetske operatore dijelimo na unarne, koji stvaraju novu jedinku mijenjajući samo manji dio genetskog materijala (mutacije) i operatore višeg reda, koji stvaraju novu jedinku kombinirajući osobine nekoliko jedinki

(križanja). Križanje je binarni operator koji iz dvije jedinke (roditelja) stvara jednu ili dvije nove jedinke (djecu). Najvažnije svojstvo križanja je da djeca nasljeđuju svojstva svojih roditelja. Ako roditelji imaju visoku dobrotu, tada će najvjerojatnije i dijete imati visoku dobrotu, ako ne i višu od svojih roditelja. Križanje može biti definirano s proizvoljnim brojem prekidnih točaka.

Mutacija je unarni operator koji obavlja slučajnu promjenu jednog ili više gena nad jednom jedinkom. Rezultat je izmijenjena jedinka. Parametar p_m koji određuje vjerojatnost mutacije jednog bita je ujedno i parametar algoritma. Ako vjerojatnost mutacije teži ka jedinici, tada algoritam postaje algoritam slučajne pretrage prostora rješenja. S druge strane, ako vjerojatnost mutacije teži k nuli, postupak će vrlo vjerojatno već u početku optimizacije stati u lokalnom optimumu.

Nakon nekog broja izvršenih generacija, čitav postupak se zaustavlja kada se zadovolji uvjet zaustavljanja. Najbolja jedinka trenutne populacije predstavlja rješenje koje bi trebalo biti sasvim blizu optimuma.
Pseudokod opisanog algoritma dan je u nastavku.

Algorithm 1 Genetski algoritam

$t \leftarrow 0$
Generiraj početnu populaciju $P(0)$
repeat
  $t \leftarrow t + 1$
  Selektiraj generaciju $P'(t)$ iz $P(t-1)$
  Križaj jedinke iz $P'(t)$ i djecu spremi u $P(t)$
  Mutiraj jedinke iz $P(t)$
until nije zadovoljen uvjet zaustavljanja
Ispiši rješenje

Genetski algoritmi su podobni za optimizacijske probleme koji zahtijevaju pretragu izrazito velikog prostora stanja i vremenski su ovisni, a nije nužno pronaći optimalno rješenje. Primjenjivi su na velik broj problema, a algoritmi nude veliki broj stupnjeva slobode. Najveći nedostatak genetskih algoritama je masovna količina računalnih resursa koji su potrebni prije nego se mogu rješavati problemi iz stvarnog svijeta.

3.2. Genetski algoritmi za rješavanje 2D problema krojenja

Algoritme je moguće klasificirati prema načinu stvaranja novih jedinki i korištenom operatoru selekcije. Po ovoj podjeli genetski algoritmi mogu biti eliminacijski i generacijski.

Eliminacijski algoritmi

Križanjem dvaju roditelja nastaje dijete koje se ubacuje u populaciju, eliminirajući jednu od jedinki iz populacije. Roditelji se odabiru 3-turnirskom selekcijom (eng. steady state tournament selection), kojom se iz populacije nasumično odabiru 3 jedinke. Dvije bolje od njih se križaju, a njihovo dijete zamjenjuje najlošiju odabranu jedinku.

Generacijski algoritmi

U jednoj iteraciji se stvara broj novih jedinki jednak veličini populacije, odnosno nastaje cijela nova generacija rješenja. Roditelji se odabiru proporcionalnom selekcijom (eng. roulette wheel ili fitness proportionate selection), gdje je vjerojatnost odabira pojedine jedinke za križanje proporcionalna njenoj relativnoj dobroti. Kako je izbor jedinki potpuno slučajan, uvodi se elitizam radi očuvanja najbolje jedinke. Elitizam osigurava dodavanje najbolje jedinke u novu populacije prije selekcije.

Za zapis kromosoma koristi se permutacijski vektor niza brojeva od kojih svaki predstavlja jedan predmet koji je potrebno izrezati. Jedinke početne populacije se generiraju slučajnom permutacijom tog niza. Permutacijski zapis jedinke veličine n je lista n vrijednosti u rasponu $[1..n]$, od kojih se svaka pojavljuje točno jednom. Redoslijed predmeta u permutacijskom nizu određuje smještaj predmeta na platnu. Način smještanja predmeta na platnu ovisi o korištenoj funkciji dobrote.

3.2.1. Operatori križanja

Za permutacijski zapis problema postoji više operatora križanja, od kojih su u ovom radu obrađeni operatori navedeni u nastavku.

Križanje COSA:

Križanje COSA (Cooperative Simulated Annealing Crossover) Izvršava mutaciju zamjenom na prvom roditelju, zamijenivši dva odabrana gena. Pritom koristi drugog roditelja kako bi odredio s kojim genom treba zamijeniti nasumično odabrani gen.

Kružno križanje:

Kružno križanje (Cyclic Crossover, CX) pronalazi sve cikluse u roditeljima. Počevši od prve pozicije, odabire se gen od jednog od roditelja. Ciklus se stvara gledajući položaj u drugom roditelju na kojem se nalazi ista vrijednost kao ona odabrana u prvom roditelju. Postupak se ponavlja dok se ne zatvore svi ciklusi.

Kružno križanje 2:

Kružno križanje 2 (Cyclic Crossover 2, CX2) temeljeno je na CX, razlikuje se u tome što pronalazi samo jedan ciklus počevši od nasumično odabrane pozicije. Sve ostale pozicije se prepišu od drugog roditelja.

Križanje s očuvanjem udaljenosti:

Križanje s očuvanjem udaljenosti (Distance Preserving Crossover, DPX) kopira u dijete gen koji su jednaki u oba roditelja, a preostali geni se nasumično rasporede od preostalih vrijednosti.

Križanje s prekidnom točkom:

Kod križanja s prekidnom točkom (One Point Crossover, OPX) prekidna točka se bira nasumično u jednom od roditelja. Dijete sadrži kopiju gena od prve pozicije do prekidne točke, a ostatak gena je kopija od drugog roditelja, uz očuvanje redoslijeda.

Redno križanje:

Kod rednog križanja (Order Crossover, OX) dvije prekidne točke se biraju nasumično. Odabrani interval između točaka se prepiše od jednog roditelja na iste pozicije u djetetu. Ostatak neiskorištenih gena se kopira od drugog roditelja, počevši od kraja kopiranog intervala.

Redno križanje 2:

Redno križanje 2 (Order Crossover 2, OX2) koristi se istim postupkom kao OX, s razlikom da se ostatak neiskorištenih gena kopira od početka permutacije istim redoslijedom.

Križanje ovisno o poziciji:

Križanje ovisno o poziciji (Position Based Crossover, PBX) nasumično odabire pozicije u jednom roditelju s jednakom vjerojatnošću 0.5. Geni se prepišu u dijete na odgovarajuće pozicije, a ostatak se uzme od drugog roditelja.

Djelomično mapirano križanje:

Djelomično mapirano križanje (Partially Mapped Crossover, PMX) radi s dijelom kromosoma (mapirajući dio) između dvije prekidne točke. Mapirajući dio prvog roditelja zamijeni odgovarajuće gene drugog roditelja. Potom se izvrši inverzna zamjena izvan mapirajućeg dijela.

Križanje zamjenom puta:

Križanje zamjenom puta (Swap Path Crossover, SPX) izvršava mutaciju zamjenom na prvom roditelju. Mutirani geni se pronalaze na drugom roditelju i mijenjaju im se pozicije. Među mutiranim roditeljima odabire se jedinka bolje dobrote kao dijete.

Uniformno križanje:

Uniformno križanje (Uniform Like Crossover, ULX) koristi masku križanja duljine jedinke za odabir prekidnih točaka. Genetski materijal se uniformno odabire od jednog roditelja ako ga dijete ne sadrži, inače se koristi nasumična vrijednost.

Uniformno djelomično mapirano križanje:

Uniformno djelomično mapirano križanje (Uniform Partially Mapped Crossover, UPMX) je varijanta PMX koja radi s pozicijama umjesto dijelova kromosoma. Dijete je klon prvog roditelja. Odabire se nasumična pozicija u prvom roditelju. Druga pozicija se odabire u drugom roditelju tako da sadrži vrijednost kao i na nasumično odabranoj poziciji prvog roditelja. Ove dvije pozicije se zamjenjuju. Postupak se ponavlja tri puta veličine kromosoma.

3.2.2. Operatori mutacije

Za permutacijski zapis problema u ovom radu se koriste mutacija umetanjem i mutacija inverzijom.

Kod mutacije umetanjem nasumično se odabire pozicija gena i pozicija za umetanje. Odabrani gen se umeće nakon odabrane pozicije u jedinci.

Kod mutacije inverzijom se nasumično odabiru dvije pozicije u jedinci. Vrijednosti na pozicijama se zamjenjuju, a sve vrijednosti između njih su zapisane inverznim redoslijedom.

3.2.3. Funkcija dobrote

Kao što je ranije spomenuto, ostatak materijala je definiran kao razlika zauzete pravokutne površine i ukupne površine n izrezanih oblika. Pritom se u izračunu zauzete površine koriste zadana širina platna w i ukupna visina izrezanih predmeta H. Algoritam nastoji minimizirati ovako definirani ostatak, odnosno njegov udio u ukupnoj

zauzetoj površini. Problem minimizacije ostatka je ekvivalentan traženju maksimalne iskoristivosti ukupne površine. Tada dobrotu računamo po sljedećoj formuli:

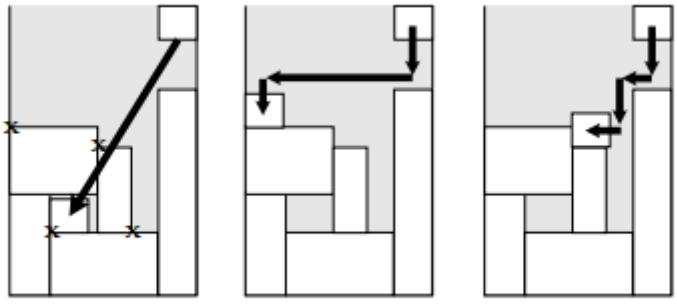
$$D = \frac{\sum_{i=1}^{n} w_i h_i}{Hw} \tag{3.1}$$

gdje je w_i širina i-tog predmeta, a h_i njegova visina.

Budući da ukupna površina predmeta i širina platna ne ovise o razmještaju predmeta i predstavljaju konstante, problem se svodi na minimizaciju ukupne visine izrezanih predmeta H.

Kako bi se izračunala dobrota jedinke, predmeti se moraju smjestiti na platno. Redoslijed smještanja određen je permutacijskim nizom, a postoji više implementacija algoritma smještanja predmeta, od kojih je najdokumentiraniji "dolje lijevo" (eng. bottom left) pristup. Prvi algoritam ovog tipa predložili su Baker, Coffman i Rivest te je u posljednjih par desetljeća predloženo nekoliko varijanti [2, 5]. Glavna značajka ovog algoritma je smještanje predmeta jedan po jedan na donje lijevo slobodno mjesto.

Baker, Coffman i Rivest su koristili pravilo koje smješta svaki predmet na najljevije mjesto među najnižim mogućim pozicijama. Algoritam je efikasniji od "dolje lijevo" pristupa jer pokušava ispuniti i rupe nastale prilikom smještanja predmeta. Ovaj pristup se naziva "dolje lijevo popuni" (eng. bottom left fill, BLF) strategijom i ilustriran je na slici 3.1 (a). Označene su slobodne donje lijeve točke. Postoje prirodni algoritmi vremenske složenosti $O(n^3)$ za ovu strategiju, no izveden je i efikasan algoritam s $O(n^2)$ vremenskom složenosti i $O(n)$ prostornom složenosti [5, 3].



Slika 3.1: Tri inačice *bottom left* algoritma

Algoritam sadrži popis točaka na koje se može smjestiti donji lijevi kut predmeta. Krenuvši od najniže najljevije točke, ispituje se može li se predmet smjestiti na trenutnu točku tako da ne izlazi izvan granica materijala i da se ne preklapa s drugim smještenim predmetima, i tako redom po listi svih mogućih točaka smještanja. Ako se pronađe točka na koju se predmet može smjestiti, predmet se smjesti, a lista točaka se

osvježi tako da se obriše iskorištena točka te dodaju gornji lijevi i donji desni kut predmeta smještenog na materijal. Ako se predmet ne može smjestiti ni u jednu točku iz liste, generira se nova točka na najljevijoj gornjoj poziciji na koju se smješta predmet.

Pseudokod algoritma BLF dan je u nastavku.

Algorithm 2 BLF

$x, y \leftarrow []$
$insertPoints \leftarrow [(0, 0)]$
for all rectangles **do**
  $chosenPoint$:= NULL
  for each $point$ in $insertPoints$ **do**
    if rectangle can be placed at $point$ **then**
      $chosenPoint \leftarrow point$
      break
    end if
  end for
  if $chosenPoint \neq$ NULL **then**
    remove$(chosenPoint, insertPoints)$
  else
    if $(w_i > maxWidth)$ **then**
      problem has no solution
    else
      $chosenPoint \leftarrow (0, y_{max})$
    end if
  end if
  update arrays x and y with $chosenPoint$
  insert$((x_i + w_i, y_i), insertPoints)$
  insert$((x_i, y_i + h_i), insertPoints)$
end for
solutions: arrays x and y with (x_i, y_i) as coordinates of rectangle i

Jakobs je koristio drukčiju *bottom left* metodu — svaki predmet se smješta najprije na gornju desnu lokaciju materijala, a potom se pomiče alternirajućim pokretima prema dolje i prema lijevo dokle god je moguće. Metoda je prikazana na slici 3.1 (b). Ova strategija se naziva *bottom left* (BL) i izvršava se u $O(n^2)$ vremenskoj složenosti [7, 5].

Liu i Teng razvili su *bottom left* heuristiku sličnu Jakobsovom algoritmu [8]. U njihovoj strategiji pokreti prema dolje imaju veći prioritet tako da se predmeti pomiču prema lijevo samo ako pokret prema dolje nije moguć. Ova strategija prikazana je na slici 3.1 (c). Ovaj algoritam se također izvršava u $O(n^2)$ vremenskoj složenosti.

4. Programska implementacija

U okviru ovog rada ostvaren je programski sustav za rješavanje osnovnog dvodimenzionalnog problema krojenja s pravokutnim elementima.

Programski sustav je napisan u programskom jeziku C++ zbog integracije s korištenim razvojnim okruženjima. Za implementaciju algoritama za rješavanje problema krojenja korišteno je razvojno okruženje za evolucijsko računanje (engl. Evolutionary Computation Framework, ECF), a za razvoj programskog sučelja korišteno je QT razvojno okruženje dostupno pod otvorenom licencom. Za olakšan rad s pokazivačima u jeziku C++ korištena je biblioteka boost, čije implementacije pokazivača samostalno upravljaju memorijom. Implementirani su algoritmi navedeni u 3. poglavlju za odabrane probleme 2/V/O/F te BLF algoritam smještanja za računanje funkcije dobrote. Algoritam se zaustavlja ako u posljednjih $5000/popSize$ generacija nije bilo nikakvih poboljšanja, gdje je $popSize$ veličina populacije.

Sustav je ostvaren kao sučelje preko kojeg je moguće zadati problem čije se rješenje traži, podešavati parametre algoritma te generirati rješenja koja je moguće sačuvati za daljnju analizu. Sustav također omogućava učitavanje prethodno generiranih rješenja te njihov vizualni prikaz.

4.1. Ulazna i izlazna datoteka

Problem je definiran širinom materijala i elementima koje treba izrezati. Zadaje se u ulaznoj datoteci s nastavkom `.tsp`. Format ulazne datoteke je preuzet sa službenih stranica problema rezanja i pakiranja gdje je definiran format datoteke za skupove poligona [1]. Ulazna datoteka sadrži ime skupa krojnih dijelova, širinu platna i redom popis krojnih dijelova. Krojni dijelovi se zadaju lokalnim koordinatama redoslijedom suprotnim od smjera kazaljke na satu, počevši od donje lijeve koordinate, te količinom istih predmeta koje treba izrezati. Zbog simulacijske prirode sučelja, koordinate predmeta kao i širina platna su izraženi u pikselima. Primjer ulazne datoteke dan je u tablici 4.1.

Pravokutni predmet za rezanje je predstavljen strukturom `Polygon` koja sadrži listu koordinata, kao i izračunatu širinu, visinu i površinu koju predmet zauzima. Prilikom implementacije se pokušao koristiti što općenitiji pristup kako bi bilo lakše problem proširiti na općenite poligone.

Tablica 4.1: Primjer ulazne datoteke

name:	TestRectangles					
size:	24					
object:	width: 200					
no.	quantity					
1	2	x	0	50	50	0
		y	0	0	50	50
2	2	x	0	50	50	0
		y	0	0	20	20
3	4	x	0	34	34	0
		y	0	0	22	22
4	4	x	0	20	20	0
		y	0	0	50	50
5	4	x	0	62	62	0
		y	0	0	33	33
6	4	x	0	17	17	0
		y	0	0	47	47
7	2	x	0	10	10	0
		y	0	0	80	80
8	2	x	0	45	45	0
		y	0	0	55	55

Nakon završetka algoritma, rješenje se upisuje u izlaznu datoteku s nastavkom `.out`, koja se naknadno može učitati. Zapis jedinke u izlaznoj datoteci je jednostavno permutacijski brojčani niz pokazivača na predmete ulazne datoteke. Ukoliko je algoritam pokrenut s većim brojem iteracija, u jednu datoteku se zapisuju sva dobivena rješenja, sa svakim rješenjem u svojem retku.

4.2. Parametri algoritma

Parametri algoritma se učitavaju iz konfiguracijske XML datoteke, čiju strukturu definira ECF. Datoteku je moguće mijenjati ručno, ali i preko programskog sučelja koje omogućuje direktno zapisivanje u datoteku.

U `<Algorithm>` dijelu podešava se korišteni algoritam i njegovi parametri. Korišteni su `SteadyStateTournament` (eliminacijski algoritam) i `RouletteWheel` (generacijski algoritam) koji su analizirani u poglavlju 3.2. U konfiguracijskoj datoteci je potrebno navesti jedan od ova dva algoritma, a ako ih je navedeno više, koristi se prvi koji je naveden. Za svaki algoritam je moguće podesiti specifične postavke. Za `SteadyStateTournament` jedini promjenjiv parametar je veličina turnirske selekcije. Za `RouletteWheel` moguće je promijeniti vjerojatnost operatora križanja `crxprob` te `selpressure` koji određuje koliko je najbolja jedinka "bolja" od

najgore.

U `<Genotype>` dijelu podešava se veličina permutacijskog niza koja odgovara ukupnom broju elemenata koje treba izrezati, kao i vjerojatnostima korištenja operatora križanja i mutacije opisanih u 3.2.1 i 3.2.2.

U `<Registry>` dijelu podešavaju se parametri vezani za tijek izvođenja algoritma, poput ulazne datoteke, veličine populacije i broj iteracija algoritma. Primjer konfiguracijske datoteke dan je u nastavku.

```
<ECF>
    <Algorithm>
        <SteadyStateTournament>
            <Entry key="tsize">3</Entry>
        </SteadyStateTournament>
        <RouletteWheel>
            <Entry key="crxprob">0.5</Entry>
            <Entry key="selpressure">10</Entry>
        </RouletteWheel>
    </Algorithm>
    <Genotype>
        <Permutation>
            <Entry key="size">24</Entry>
            <Entry key="crx.COSA">0</Entry>
            <Entry key="crx.DPX">0</Entry>
            <Entry key="crx.OBX">0.5</Entry>
            <Entry key="crx.OPX">0</Entry>
            <Entry key="crx.OX">0.3</Entry>
            <Entry key="crx.OX2">0</Entry>
            <Entry key="crx.PBX">0</Entry>
            <Entry key="crx.PMX">0.1</Entry>
            <Entry key="crx.SPX">0</Entry>
            <Entry key="crx.ULX">0</Entry>
            <Entry key="crx.UPMX">0</Entry>
            <Entry key="crx.cyclic">0.1</Entry>
            <Entry key="crx.cyclic2">0</Entry>
            <Entry key="mut.ins">0.5</Entry>
            <Entry key="mut.inv">0</Entry>
            <Entry key="mut.toggle">1</Entry>
        </Permutation>
    </Genotype>
    <Registry>
        <Entry key="tsp.infile">./TestRectangles.tsp</Entry>
        <Entry key="randomizer.seed">0</Entry>
        <Entry key="population.size">100</Entry>
        <Entry key="mutation.indprob">0.45</Entry>
        <Entry key="term.stagnation">50</Entry>
        <Entry key="batch.repeats">3</Entry>
        <Entry key="batch.singlemilestone">1</Entry>
    </Registry>
</ECF>
```

4.3. Evaluacijska funkcija

U sljedećem isječku koda dana je implementacija evaluacijske funkcije u radnom okviru ECF. Prikazane su i pojedine relevantne pomoćne funkcije. Nakon programskog isječka implementirani algoritam je dodatno pojašnjen.

```
FitnessP TSPEvalOp::evaluate(IndividualP individual) {
    FitnessP fitness (new FitnessMax);
    // dohvati permutacijski genotip jedinke
    PermutationP perm =
      boost::static_pointer_cast<Permutation::Permutation> (individual->getGenotype());
    const uint& size = (uint) perm->variables.size();
    // lista smjestenih predmeta
    vector<Polygon> placedElements;
    // skup slobodnih tocaka za smjestanje poligona
    set<Polygon::Point, Polygon::pointCompare> insertPoints;
    insertPoints.insert(Polygon::Point{ 0, 0 });
    int fitnessV = 0;

    uint ypos = 0, ymax = 0;
    for (uint i = 0; i < size; i++){
        Polygon p = elements[perm->variables[i]];
        boost::scoped_ptr<Polygon::Point> chosenPoint;

        foreach(Polygon::Point point, insertPoints) {
            if (!canBePlaced(p, point, placedElements)) continue;
            chosenPoint.reset(new Polygon::Point);
            *chosenPoint = point;
            break;
        }

        if (chosenPoint != NULL) {
            insertPoints.erase(*chosenPoint);
        } else {
            // nema rjesenja
            if (p.width > width) {
                fitness->setValue(0);
                return fitness;
            }
            chosenPoint.reset(new Polygon::Point);
            *chosenPoint = Polygon::Point{ 0, ymax };
        }
        p.adjust(chosenPoint->x, chosenPoint->y);
        insertPoints.insert(Polygon::Point{ chosenPoint->x + p.width, chosenPoint->y });
        insertPoints.insert(Polygon::Point{ chosenPoint->x, chosenPoint->y + p.height });

        ypos = p.coordinates[0].y + p.height;
        if (ypos > ymax) ymax = ypos;
        placedElements.push_back(p);
    }
```

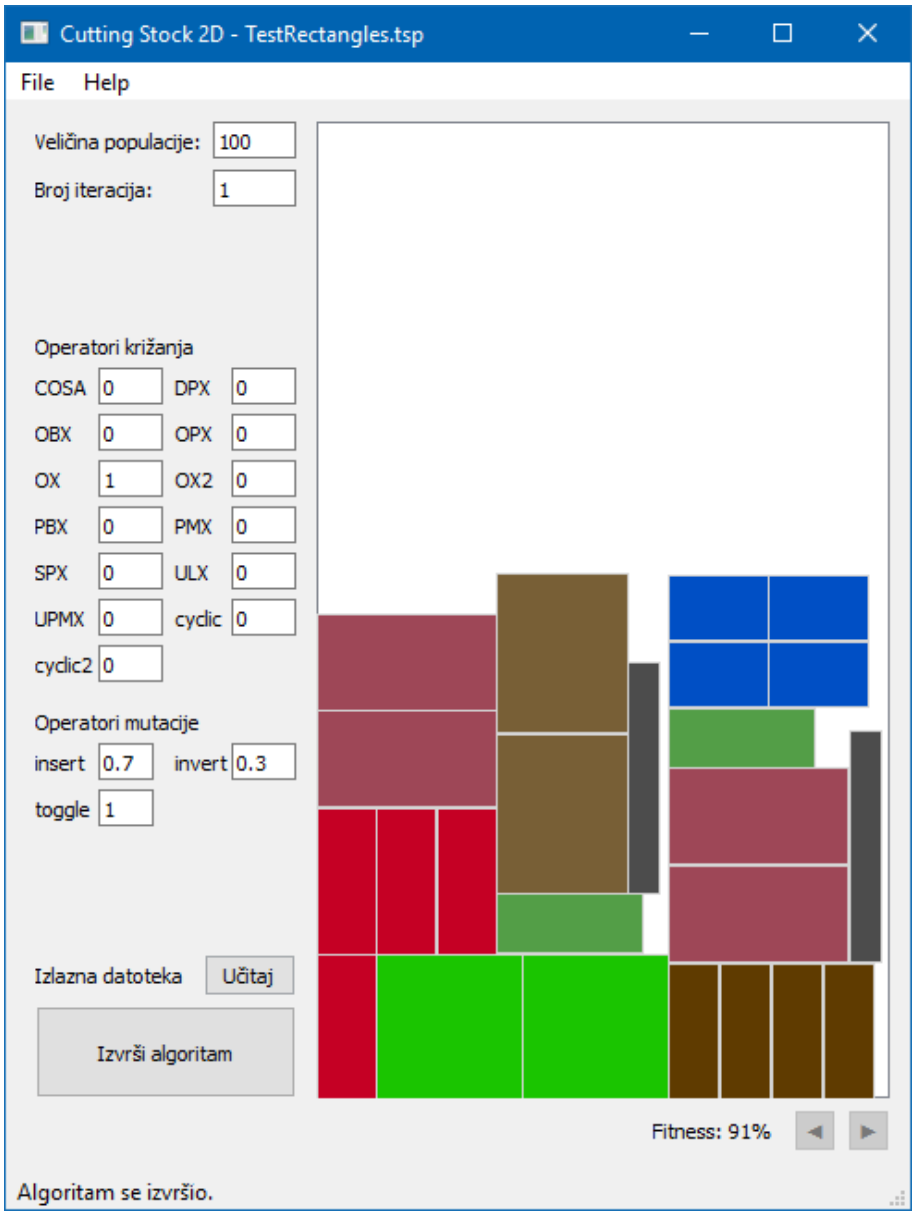
```
    // postotak iskoristivosti materijala
    fitnessV = (int) ((double) totalArea / (width * ymax) * 100);
    fitness->setValue(fitnessV);
    return fitness;
}

// provjerava moze li se poligon smjestiti na zadanu tocku
bool TSPEvalOp::canBePlaced(Polygon p,
  const Polygon::Point &point, const vector<Polygon> &placedElements) {
    p.adjust(point.x, point.y);
    // predmet ne smije izlaziti izvan granica materijala
    if (p.coordinates[0].x + p.width > width) return false;
    // predmet ne smije prelaziti preko vec smjestenih predmeta
    const uint& size = (uint)placedElements.size();
    for (uint i = 0; i < size; ++i) {
        const Polygon& placedP = placedElements[i];
      if (rectanglesOverlap(p, placedP)) return false;
    }
    return true;
}

// provjerava preklapaju li se dva pravokutnika
bool TSPEvalOp::rectanglesOverlap(const Polygon &rectA, const Polygon &rectB) {
    const Polygon::Point& pA1 = rectA.coordinates[0];
    const Polygon::Point& pA3 = rectA.coordinates[2];
    const Polygon::Point& pB1 = rectB.coordinates[0];
    const Polygon::Point& pB3 = rectB.coordinates[2];
    if (pA3.x < pB1.x || pB3.x < pA1.x || pA3.y < pB1.y || pB3.y < pA1.y) return false;
    return true;
}

// pozicionira predmet na zadane koordinate
void Polygon::adjust(const int &x, const int &y) {
    uint size = (uint)coordinates.size();
    for (uint i = 0; i < size; ++i) {
        coordinates[i] = { coordinates[i].x + x, coordinates[i].y + y };
    }
}
```

Na početku evaluacijske funkcije definira se vrsta funkcije dobrote. Kao što je prethodno definirano, dobrota je izražena postotkom iskorištenosti materijala koji nastojimo maksimizirati te stoga koristimo `FitnessMax` vrstu. Nakon inicijalizacije vrijednosti, dodaje se točka $(0, 0)$ u skup slobodnih točaka. Potom se iterira po predmetima redoslijedom zapisanom u permutacijskom genotipu. Predmet se redom pokušava smjestiti na jednu od slobodnih točaka dok se ne nađe točka na koju je moguće smjestiti predmet. Skup slobodnih točaka sortiran je tako da veći prioritet imaju točke manje y vrijednosti, a potom manje x vrijednosti na koordinatnom sustavu. Pomoćna metoda `canBePlaced` smješta predmet na zadanu točku i potom provjerava izlazi li

izvan granica materijala i prelazi li preko već smještenih predmeta. Ako u oba slučaja odgovor nije potvrdan, predmet je moguće smjestiti na zadanu točku. Ako je pronađena odgovarajuća točka, lista točaka se osvježi tako da se obriše iskorištena točka te dodaju gornji lijevi i donji desni kut smještenog pravokutnika. Lokalno zadane koordinate predmeta prevedu se u koordinate globalnog koordinatnog sustava materijala pomakom za koordinate točke na koju se predmet smješta. Time je predmet smješten na materijal i algoritam nastavlja s iteriranjem. Ako se predmet ne može smjestiti ni u jednu točku iz skupa, najprije se provjerava je li širina predmeta veća od samog materijala. Ako je predmet preširok, zadani problem nema rješenja jer ne može smjestiti sve predmete na zadani materijal, funkcija dobrote vraća vrijednost 0 i algoritam prestaje s radom. Inače, generira se nova točka na najljevijoj gornjoj poziciji na koju se smješta predmet. U svakoj iteraciji ažurira se visina do koje se predmeti protežu. Nakon što su svi predmeti smješteni, računa se dobrota jedinke prema formuli definiranoj u odjeljku 3.2.3.

4.4. Upute za korištenje

Programsko rješenje ostvaruje grafičko korisničko sučelje za unos parametara algoritma i prikaz rezultata optimiranja. Grafičko korisničko sučelje je prikazano na slici 4.1.



Slika 4.1: Ostvareno grafičko sučelje

Pri pokretanju programa automatski se učitava ulazna datoteka zapisana u konfiguracijskoj datoteci. Ulaznu datoteku je moguće učitati preko izborničke trake putem naredbe File → Load, čime se otvara izbornik za učitavanje nove ulazne datoteke. Naziv trenutno učitane datoteke vidljiv je na naslovnoj traci pored imena prozora. S desne strane nalazi se prostor za grafički prikaz rješenja. Prije pokretanja algoritma, predmeti su raspoređeni redoslijedom kojim su navedeni u ulaznoj datoteci. Nakon što se algori-

tam izvrši, prikazano je dobiveno rješenje. Ispod prostora za grafički prikaz ispisana je dobrota trenutnog rješenja. Ako je odabrano više iteracija algoritma, moguće je listati kroz rješenja u iteracijama pomoću gumba na kojima su iscrtane strelice.

Na samom dnu prozora nalazi se statusna traka gdje se ispisuje poruka o trenutnom stanju programa. Pri pokretanju programa ispisuje se poruka o uspješnoj inicijalizaciji algoritma. Tijekom izvršavanja algoritma ispisuje se poruka o statusu izvršavanja algoritma zajedno s brojem generacije koja se trenutno evaluira. Nakon što se algoritam izvrši, poruka se također prikladno ažurira. Ako se tijekom inicijalizacije ili izvršavanja algoritma dogodi pogreška, poruka pokazuje gdje je došlo do pogreške.

S lijeve strane prozora nalaze se polja za unos. Moguće je promijeniti veličinu populacije, broj iteracija algoritma te vjerojatnosti korištenja pojedinih operatora križanja i mutacije. Ispod polja za unos nalazi se gumb za učitavanje izlazne datoteke, prethodno generiranog rješenja. Rješenje mora odgovarati veličini skupa za krojenje iz trenutno učitane ulazne datoteke. Konačno, veliki gumb "Izvrši algoritam" učitava sve podešene parametre i izvršava algoritam za zadani broj iteracija.

4.5. Budući rad

Ostvareni programski sustav bi se dalje mogao nadograditi kako bi podržao općenitiji zapis predmeta za rezanje u obliku poligona. U skladu s tim, trebali bi se implementirati dodatni algoritmi pomicanja i preklapanja poligona kako bi se minimizirao korišteni prostor. Također bi se trebala uzeti u obzir i rotacija predmeta. Sustav bi se također mogao nadograditi tako da podržava lakšu definiciju problema, primjerice grafičko sučelje za crtanje željenih poligona koji bi se potom prevodili u koordinate.

5. Analiza rezultata

Algoritmi su pokrenuti na nekoliko primjeraka problema te su evaluirane njihove performanse. Svaki algoritam je pokrenut 10 puta, nakon čega su zabilježeni najbolji i prosječni rezultat. Prvo su dani rezultati za probleme koji koriste mali broj pravokutnika različitih dimenzija, a potom problemi koji koriste veliki broj pravokutnika različitih dimenzija. U oba slučaja su pokrenuti eliminacijski i generacijski algoritmi te su uspoređeni njihovi rezultati.

Parametri algoritama su u svim iteracijama bili fiksirani kako ne bi utjecali na interpretaciju rezultata. Veličina populacije je bila podešena na 100, križanje je korišteno s vjerojatnošću 0.5, a mutacija s vjerojatnošću 0.45. Kao operator križanja je korišten isključivo OX operator, jer je pokazano da u praksi daje najbolje rezultate.

5.1. Eksperimenti

Za eksperimentalnu analizu uzeti su problemi s malim brojem različitih pravokutnika. Ovakvi problemi su tipa 2/V/O/F prema Dyckhoffovoj tipologiji. Korištena su tri skupa problema nad kojima su ispitani algoritmi. Obilježja problema navedena su u tablici 5.1

Tablica 5.1: Obilježja jednostavnih problema

Broj problema	Broj predmeta	Širina platna
1	24	200
2	30	400
3	50	500

Rezultati algoritama dani su u tablicama 5.2 i 5.3.

Tablica 5.2: Rezultati eliminacijskog algoritma

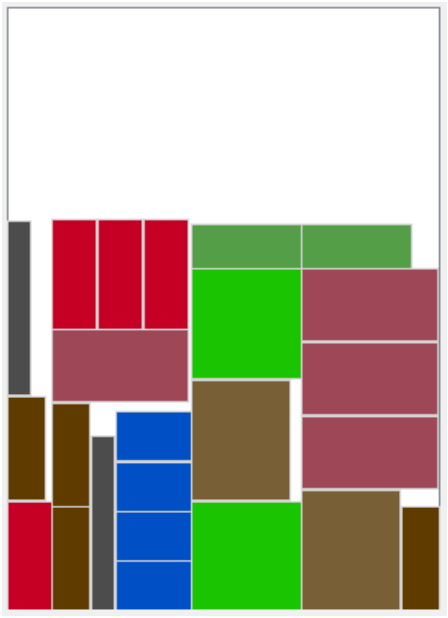
Broj problema	Prosječno rješenje[%]	Najbolje rješenje[%]
1	90.5	93
2	92.3	94
3	90.8	92

Tablica 5.3: Rezultati generacijskog algoritma

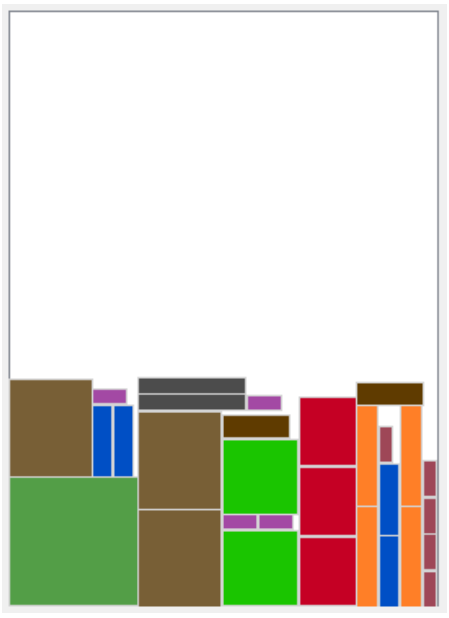
Broj problema	Prosječno rješenje[%]	Najbolje rješenje[%]
1	89.9	92
2	90.1	93
3	89.8	92

U prvom stupcu je označen broj problema, u drugom stupcu je dobrota prosječnog rješenja, a u trećem stupcu je navedena najbolja postignuta vrijednost dobrote u 10 iteracija algoritma.

Grafički prikazi najboljih rješenja dobivenih eliminacijskim algoritmom za probleme 1, 2 i 3 prikazani su na slikama 5.1, 5.2 i 5.3, respektivno. Predmeti su skalirani tako da stanu u prostor za rezanje uzevši u obzir njihove zadane dimenzije. Iako se predmeti na slici 5.2 čine manji, većih su dimenzija i čine se manjima upravo zbog potrebnog skaliranja. U svakom rješenju su korištene različite boje za svaku vrstu predmeta zbog njihovog lakšeg razlikovanja.



Slika 5.1: Rješenje problema 1



Slika 5.2: Rješenje problema 2

Slika 5.3: Rješenje problema 3

Evidentno je da je algoritam uspješan u oba slučaja te da su za mali broj predmeta razlike između rezultata evolucijskog i generacijskog algoritma neznatne.

6. Zaključak

Problem krojenja pripada kategoriji NP-potpunih problema, te pronalazi mnoge primjene u industriji. Cilj problema krojenja je smanjiti industrijski višak koji nastaje pri obradi materijala, kako bi se uštedjeli materijali i smanjili troškovi.

U ovom radu se analizira pristup rješavanja dvodimenzijskog problema krojenja uporabom genetskih algoritama. Genetski algoritmi su pogodni za rješavanje optimizacijskih problema za koje ne postoje deterministički algoritmi koji mogu pronaći optimalno rješenje u razumnom vremenu.

Programski je ostvareno sučelje za rješavanje problema krojenja i podešavanje pripadnih parametara. Za implementaciju evolucijskih algoritama korišteno je razvojno okruženje za evolucijsko računanje (ECF). Programski sustav je potrebno nadograditi kako bi se mogao koristiti kao ozbiljan alat za analizu algoritama kod problema krojenja, no već sada predstavlja izvrstan potencijal za optimiranje problema opisanih u ovom radu, upravo zbog mogućnosti radnog okruženja ECF.

LITERATURA

[1] Službene stranice problema rezanja i pakiranja. https://neos-guide.org/content/cutting-stock-problem.

[2] B.S. Baker, E.G. Coffman Jr., i R.L. Rivest. Orthogonal packing in two dimensions. SIAM J. on Comput., 1980.

[3] B. Chazelle. The bottom-left bin-packing heuristic: an efficient implementation. IEEE Trans. on Computers, 1983.

[4] M. Golub. Genetski algoritam. http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf, 1997.

[5] E. Hopper i B.C.H. Turton. An empirical investigation of metaheuristic and heuristic algorithms for a 2d packing problem. Eur. J. of Oper. Res., 2001.

[6] S. Imahori, M. Yagiura, i H. Nagamochi. Practical algorithms for two-dimensional packing. http://www.keisu.t.u-tokyo.ac.jp/research/techrep/data/2006METR06-19.pdf, 2006.

[7] S. Jakobs. On genetic algorithms for the packing of polygons. Eur. J. of Oper. Res., 1996.

[8] D. Liu i H. Teng. An improved bl-algorithm for genetic algorithm of the orthogonal packing of rectangles. Eur. J. of Oper. Res., 1999.

[9] G. Wäscher, H. Haußner, i H. Schumann. An improved typology of cutting and packing problems. https://pdfs.semanticscholar.org/39e8/e265d6ba678968339813671d1d64d98d7a64.pdf, 2005.

RJEŠAVANJE PROBLEMA KROJENJA KORIŠTENJEM EVOLUCIJSKIH ALGORITAMA

Sažetak

U ovom radu opisan je i definiran problem krojenja u dvodimenzionalnom prostoru. Navedeni su evolucijski algoritmi pogodni za rješavanje problema krojenja i opisane su njihove značajke. Posebna pažnja posvećena je prikazu rješenja i specifičnim evolucijskim operatorima. Opisan je ostvareni programski sustav za rješavanje osnovnog problema krojenja s pravokutnim elementima koristeći razvojno okruženje za evolucijsko računanje (engl. Evolutionary Computation Framework, ECF). Opisane su upute za korištenje grafičkog korisničkog sučelja za unos parametara i prikaz rezultata optimiranja. Ispitana je uspješnost nekoliko odabranih evolucijskih algoritama u rješavanju problema krojenja te su uspoređeni dobiveni rezultati.

Ključne riječi: problem krojenja, genetski algoritam, grafičko sučelje

Solving 2D Cutting-Stock Problem Using Evolutionary Algorithms

Abstract

Describe and define 2D cutting stock problem. Examine evolutionary algorithms appropriate for solving cutting stock problem and describe their features. Focus on solution representation and specific evolutionary operators. Describe implemented software for solving basic 2D cutting stock problem with rectangular items using Evolutionary Computation Framework (ECF). Describe instructions for using implemented graphical user interface for parameter input and optimization result display. Analyze successfulness of a few selected evolutionary algorithms in solving cutting stock problem and compare obtained results.

Keywords: cutting stock problem, genetic algorithm, graphical user interface