Rješavanje problema putne torbe algoritmom diferencijalne evolucije

Šimić, Zvonimir

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

RJEŠAVANJE PROBLEMA PUTNE TORBE ALGORITMOM DIFERENCIJALNE EVOLUCIJE

Završni rad

Zvonimir Šimić

Osijek, 2020.

Sadržaj

1. Uvod

Problem putne torbe (engl. *knapsack problem*) predstavlja jedan od klasičnih NP-teških problema kombinatorne optimizacije u kojemu se pokušava povećati ukupna cijena predmeta unutar putne torbe bez premašivanja njezinog kapaciteta. S obzirom da je problem putne torbe NP-težak, uobičajeno je vrlo računalno zahtjevno pronaći optimalna rješenja za veće primjerke tog problema. Problem putne torbe je primjenjiv u raspodjeli resursa, investicijskom odlučivanju, financijskom upravljanju i slično. Za njegovo rješavanje predloženi su različiti algoritmi. Evolucijski algoritmi (engl. *evolutionary algorithms*, EAs), kao što je diferencijalna evolucija (engl. *differential evolution*, DE), prikladni su za rješavanje problema putne torbe. Algoritam diferencijalne evolucije jedan je od najpopularnijih i najučinkovitijih EA-ova koji ima mnoge prednosti kao što su brzina, jednostavnost i efikasnost. Algoritam DE izvorno je namijenjen za probleme kontinuirane optimizacije te nije izravno primjenjiv na probleme diskretne ili kombinatorne optimizacije kao što je problem putne torbe. Uzrok tome je operator mutacije koji je specifičan za DE. Kako bi algoritam DE bio primjenjiv na rješavanje problema putne torbe, potrebno je ugraditi neku od metoda diskretizacije. Metode diskretizacije vrše transformaciju realnih vektora/rješenja u binarne vektore te omogućuju primjenu DE na određene probleme diskretne optimizacije. S obzirom da postoje različite metode diskretizacije, različit je i njihov utjecaj na učinkovitost i ponašanje algoritma.

U drugom poglavlju dan je formalan opis problema putne torbe te je opisano nekoliko različitih varijanti tog problema. Zatim je detaljno obrađen algoritam diferencijalne evolucije te je prikazana ugradnja metoda diskretizacija kako bi algoritam DE bio prilagođen za rješavanje problema putne torbe. U trećem poglavlju predstavljeno je ostvareno programsko rješenje, detaljno je opisano što predstavlja ulaz i izlaz programskog rješenja te koje su njegove mogućnosti podešavanja. Zatim je prikazan izgled te način uporabe ostvarenog programskog rješenja. U četvrtom poglavlju prikazana je provedena eksperimentalna analiza na više različitih problema putne torbe koristeći tri različita načina diskretizacije pri odabranim parametrima algoritma. Grafički i tablično prikazane su razlike u ponašanju i učinkovitosti algoritma DE ovisno o korištenom načinu diskretizacije.

1.1. Zadatak završnog rada

Zadatak je završnog rada opisati problem putne torbe kao problem kombinatorne optimizacije, opisati moguće primjene i varijante problema te opisati algoritam diferencijalne evolucije kao

mogući postupak rješavanja problema. U praktičnom dijelu rada nužno je razviti programsko rješenje koje omogućuje rješavanje problema putne torbe pomoću algoritma diferencijalne evolucije. Također, potrebno je napraviti eksperimentalnu analizu na nekoliko različitih primjeraka problema.

2. Problem putne torbe i algoritam diferencijalne evolucije

Problem putne torbe konceptualno predstavlja problem optimizacije u kojemu se pokušava povećati ukupna cijena predmeta unutar putne torbe bez premašivanja njezinog kapaciteta. Postoji mnogo područja za primjenu problema putne torbe poput kriptografije za enkripciju javnog ključa, kontrolu budžeta ili tok mreže [1]. Neki od algoritama koji se mogu primijeniti za rješavanje navedenog problema su pohlepni algoritmi (engl. *greedy algorithm*), dinamičko programiranje (engl. *dynamic programming*), *branch & bound* algoritam te evolucijski algoritmi.

Evolucijski algoritmi mogu se koristiti za rješavanje problema putne torbe unutar razumne vremenske složenosti. Vremenska složenost tih algoritama je O(N) [1]. Ipak, ovi algoritmi ne mogu jamčiti pronalazak optimalnog rješenja, ali uobičajeno su sposobni pronaći rješenja blizu optimalnog, što je često dovoljno za različite primjene. U radu je detaljno obrađen jedan od najpopularnijih i najučinkovitijih EAs, algoritam diferencijalne evolucije, te je prikazano što je nužno dodati u njega kako bi bio primjenjiv za rješavanje razmatranog problema.

2.1. Problem putne torbe

Problem putne torbe formalno se može opisati kao problem gdje postoji skup od D predmeta, gdje svaki predmet ima svoju cijenu (engl. *profit*) p_j i masu (engl. *weight*) w_j za $j = 1, \dots, D$ te je zadan maksimalni kapacitet torbe *C*. Cilj je odabrati kombinaciju predmeta iz skupa *D*, tako da je ukupna cijena tih predmeta maksimalna uz ograničenje da ukupna masa ne prelazi maksimalni kapacitet *C*. Problem putne torbe predstavlja problem kombinatorne optimizacije koji je NP-težak [2]. Zbog toga je vrlo zahtjevno doći do optimalnih rješenja unutar razumnog vremena za velike primjerke problema. Formalni opis problema dan je jednadžbom (2-1).

$$maksimizirati \ \sum_{j=1}^{D} p_j x_j \ tako\ da \ \ \sum_{j=1}^{D} w_j x_j \ \leq C \ , x_j \in \{0,1\}, j = 1, \dots, D \ , \qquad (2\text{-}1)$$

gdje x_j iznosi 1 ako je predmet *j*-ti odabran, a u suprotnom iznosi 0. Navedeni problem je relevantan, što je opravdano zbog velikog broja problema iz stvarnog svijeta koji se mogu svesti na problem putne torbe.

Primjenom logike koja je prethodno opisana, u problemu investiranja kapitala, kapital odnosno novac *C* potrebno je raspodijeliti na *D* mogućih investicija, gdje je dan očekivan profit investicije te količinu kapitala koju zahtijeva [2]. Prema [3], jedan primjer stvarnog problema iz područja

ekonomije jest transport robe zrakoplovom od skladišta do odredišta. U ovom slučaju cilj je utovariti što više robe u zrakoplov kombinacijom artikala koji će prodajom ostvariti najveću dobit. Treba uzeti u obzir da roba mora zadovoljavati određeno ograničenje mase koju zrakoplov može podnijeti. Nadalje, prema [3], široka primjena analogije problema putne torbe može se vidjeti i u industrijskim pogonima. Primjerice, u tvornici metalnih cijevi u kojoj se cijevi režu na kraće te kao takve dalje šalju na tržište. Cijevi se režu na već definirane duljine, gdje se dobiva kraća cijev koja ima svoju prodajnu cijenu. Kako bi se maksimizirala zarada od početne cijevi, potrebno je pronaći optimalnu kombinaciju duljina cijevi koje će se rezati i njihovih pripadajućih cijena.

U stvarnom svijetu postoje dodatni parametri i ograničenja poput hitnosti, prioriteta ili vremenskog ograničenja u kojem radnja mora biti izvršena, što dovodi do raznih varijacija osnovnog problema putne torbe. Iz toga proistječe da problem putne torbe nije izravno moguće preslikati na neke stvarne probleme te da je potrebno proširiti osnovni problem uvođenjem dodatnih parametara i ograničenja. Neki od složenijih varijanti problema opisani su u nastavku.

2.1.1. Neke varijante problema putne torbe

Postoji mnogo varijacija problema putne torbe koje se smatraju zasebnim problemima kombinatorne optimizacije. Osnovni problem putne torbe, odnosno problem koji se detaljno obrađuje te daljnje koristi u eksperimentalnoj analizi je 0-1 problem putne torbe (engl. 0-1 *knapsack problem)*. Problem je opisan jednadžbom (2-1). Važno je napomenuti da se predmeti s kojima se raspolaže u problemu mogu samo odabrati ili ne odabrati te je neki predmet moguće odabrati samo jednom. Kako bi se primjeri iz stvarnog svijeta mogli svesti na problem putne torbe, potrebno je uzeti u obzir dodatne parametre koji utječu na formiranje problema.

Ograničeni problem putne torbe *(engl. bounded knapsack problem,* BKP*)* predstavlja varijaciju osnovnog problema koja se razlikuje po tome što svaki predmet ima ograničen broj identičnih kopija. Nadalje, neograničeni problem putne torbe (*engl. Unbounded knapsack problem,* UKP*)* razlikuje se od BKP po tome što je broj kopija svakog predmeta neograničen. Predloženi načini rješavanja BKP poput dinamičkog programiranja te približnih algoritama detaljno su opisani u [3]. Povećanjem broja kopija predmeta otežava se problem. Shodno tome, zbog neograničenog broja kopija predmeta, UKP je još složeniji i teži za riješiti [3]. Modeli problema BKP i UKP široko su rasprostranjeni u mnogim stvarnim situacijama gdje postoji određena ili praktički neograničena

količina kopija istog predmeta ili resursa. Primjena se može uočiti pri problemima izračunavanja maksimalnog dobitka utovara, kod problema rezanja koji se odvijaju u više dimenzija te u problemima raspoređivanja budžeta i slično.

Prema [3], još jednu varijantu problema predstavlja problem višedimenzionalne putne torbe *(*engl. *multidimensional Knapsack problem,* d-KP*)* koji se formira dodavanjem dodatnog ograničenja već postojećem ograničenju mase predmeta. Ovaj problem je široko primjenjiv u računarstvu, a njegova korist se pronalazi pri alokaciji procesora i bazi podataka u raspodijeljenim računalnim sustavima te u planiranju izvođenja računalnih programa. U ostalim područjima njegova se primjena uočava u problemima utovara, rezanja te grupiranja.

Problem višestrukih putnih torbi [3] *(*engl. *multiple knapsack problem,* MKP*)* generalizacija je standardnog problema putne torbe u kojem umjesto jedne torbe postoji određen broj torbi koje mogu imati različite maksimalne kapacitete. Cilj je problema rasporediti svaki predmet u torbe tako da broj predmeta u svakoj torbi ne premaši njihov maksimalni kapacitet te da je vrijednost raspoređenih predmeta maksimalna.

U dosad spomenutim varijacijama problema putne torbe cijena predmeta bila je neovisna o redoslijedu odabira ostalih predmeta. Kada u problemu putne torbe postoji međuzavisnost predmeta, formulira se kvadratni problem putne torbe (engl. *quadratic knapsack problem,* QKP*)* u kojem predmet ima svoju vlastitu vrijednost te dodatnu vrijednost koja se uračunava ako se predmet odabere zajedno s drugim određenim predmetom [3].

2.2. Algoritam diferencijalne evolucije

Algoritam diferencijalne evolucije je stohastički, optimizacijski algoritam zasnovan na populaciji rješenja te osmišljen za optimizaciju funkcija s realnim varijablama. Algoritam dijeli sličnosti s ostalim evolucijskim algoritmima, no uvelike se razlikuje time što se ugrađuje za njega specifičan operator mutacije [4]. Kao jedan od najpopularnijih i najučinkovitijih EAs, DE ima mnoge prednosti kao što su brzina, jednostavna ugradnja te visoka učinkovitost za probleme globalne kontinuirane optimizacije [5]. U konceptualnom opisu DE početna populacija $P_{x,g}$ sastoji se od vektora $x_{i,g}$ realnih komponenti koje je potrebno inicijalizirati. Nakon inicijalizacije populacije, DE mutacijom stvara donorski vektor $v_{i,g}$, nakon čega dolazi do rekombinacije, odnosno križanja u kojoj se stvara pokusni

vektor $u_{i,g}$ te se na kraju selekcijom između pokusnog vektora $u_{i,g+1}$ i ciljnog vektora $x_{i,g}$ formira ciljni vektor $x_{i.g+1}$ za iduću generaciju, odnosno iteraciju. Postupak je detaljnije opisan u nastavku.

Populacija rješenja/vektora koja se sastoji od *Np D*-dimenzionalnih vektora realnih parametara te je određena jednadžbom

$$P_{x,g} = (x_{i,g}) \quad i = 1, \dots, Np, g = 1, \dots, gmax, \tag{2-2}$$

gdje *Np* predstavlja veličinu populaciju, g predstavlja trenutnu generaciju, odnosno iteraciju, dok *gmax* predstavlja maksimalni broj generacija. Vektori $x_{i,g}$ od kojih se populacija sastoji određeni su jednadžbom

$$x_{i,g} = (x_{j,i,g}), \ j = 1, \dots, D, \tag{2-3}$$

gdje *D* predstavlja dimenziju vektora, odnosno broj njegovih komponenti.

Kako bi **inicijalizacija** bila moguća, potrebno je definirati gornje (engl. *upper bounds*) i donje granice (engl. *lower bounds*) za svaku komponentu vektora:

$$x_j L \leq x_{j,i,1} \leq x_j U. \tag{2-4}$$

Vrijednosti parametara se slučajnim odabirom inicijaliziraju na intervalu [x_jL, x_jU], gdje x_jU predstavlja gornju granicu, a x_jL donju granicu.

Nakon inicijalizacije DE **mutira** (engl. *mutation*) te rekombinira odabrane vektore populacije kako bi proizvela populaciju sastavljenu od *Np* pokusnih vektora (engl. *trial vectors*). Produkt mutacije je mutant ili donorski vektor (engl. *donor vector*) $v_{i,g}$ koji nastaje nasumičnim odabirom tri različita vektora te je određen jednadžbom

$$v_{i,g} = x_{r0,g} + F * (x_{r1,g} - x_{r2,g}), \tag{2-5}$$

gdje je $F \in (0,1+)$ faktor skaliranja (engl. *scale factor*), pozitivan realan broj, a $r0, r1$i $r2$ su indeksi nasumično odabranih vektora uz uvjet $r0 \neq r1 \neq r2 \neq i$. Za faktor skaliranja ne postoji određena gornja granica, no vrijednosti iznad 1 rijetko su korisne. Vrijednost 1 empirijski je izvedena granica [6].

Rekombinacija ili **križanje** (engl. *crossover*) obavlja se na način

$$u_{j,i,g} = \begin{cases} v_{j,i,g} & ako\ rand_j(0,1) \leq Cr\ ili\ j = j_{rand} \\ x_{j,i,g} & u\ suprotnom \end{cases},\ j = 1, \dots, D, \tag{2-6}$$

gdje *Cr* predstavlja stopu križanja (engl. *crossover-rate*), a j_{rand} predstavlja nasumično odabrani indeks. Stopa križanja korisnički je definirana vrijednost koja kontrolira udio vrijednosti parametara preuzetih od mutanta. Pokusni vektor $u_{\text{i,g}}$ formira se od ciljnog vektora $x_{\text{i,g}}$ i donorskog vektora $v_{\text{i,g}}$. Kako bi se odlučilo koja će komponenta mutanta prijeći u pokusni vektor $u_{\text{i,g}}$, *Cr* se uspoređuje s nasumično odabranim brojem $rand_{\text{j}}(0,1)$. Ako je nasumično odabrani broj manji ili jednak *Cr*-*u*, u pokusni vektor $u_{\text{i,g}}$ nasljeđuju se parametri iz $v_{\text{i,g}}$, a u suprotnome se uzimaju parametri iz vektora $x_{i,\text{g}}$. Uvjet $j = j_{rand}$ osigurava da pokusni vektor ne duplicira $x_{\text{i.g}}$.

Selekcijom (engl. *selection*) se određuje hoće li pokusni vektor prijeći u narednu iteraciju te se odvija na način

$$x_{i,g+1} = \begin{cases} u_{i,g} & ako\ f(u_{i,g}) \leq f(x_{i,g}) \\ x_{i,g} & u\ suprotnom \end{cases}, \tag{2-7}$$

gdje se ciljni vektor $x_{\text{i,g}}$ uspoređuje s pokusnim vektorom $u_{\text{i,g}}$. Ako je vrijednost funkcije cilja za pokusni vektor $u_{\text{i,g}}$ jednaka ili manja od iste za ciljni vektor, onda $u_{\text{i,g}}$ nadomješta $x_{\text{i,g}}$ te prolazi u sljedeću generaciju (bez smanjenja općenitosti, razmatran je problem minimiziranja). U suprotnome ciljni vektor $x_{\text{i,g}}$ ostaje te preživljava najmanje još jednu generaciju. Nakon formiranja populacije, proces mutacije, križanja i selekcije se ponavlja sve dok se ne pronađe optimum ili dok se ne zadovolji neki od uvjeta završetka. Na slici 2.1. je prikazan pseudokod DE.

```
//postavljanje parametara Np, F, Cr
//inicijalizacija populacije
for(i<Np) //i=0
{
       for(j<D) // j=0
       {
         population[i][j] = generiranje nasumičnih brojeva iz [xjL, xjU];
       }}
do // glavna petlja
{
       for (i<Np) // i = 0
       {
              // slučajno odaberi različite indekse r1,r2,r3, tako da r1!=r2!=r3!=i
              // generiranje pokusnog vektora
              for(j<D) // j = 0
              {
                     If(rand(0,1)<=Cr || j==j_rand)
                     {
                       u[j][i] = x[r1][j] + F *( x[r2][j]- x[r3][j]);  // donor vektor
                     }
                     else
                     {
                       u[j][i] = x[j][i];
                     }}}
// generiranje sljedeće generacije
       for (i<Np) // i=0
       {
              if ( f(u[i]) <= f(x[i]) ) // vrednovanje
              {
                     x[i]=u[i];
              }}
} while (uvjet završetka nije ispunjen)
```

Slika 2.1 Pseudokod DE

2.2.1. Primjena algoritma DE za rješavanje problema putne torbe

Algoritam diferencijalne evolucije izvorno je namijenjen za rješavanje problema kontinuirane optimizacije te nije izravno primjenjiv na probleme diskretne ili kombinatorne optimizacije. Uzrok tome je operator mutacije koji je specifičan za DE. Mutacija je ključni element njene učinkovitosti [7] te je primjenjiva isključivo na vektore realnih komponenti.

Kako bi se iskoristio potencijal DE u problemima diskretne ili kombinatorne optimizacije te kako bi ju primijenili na rješavanje problema putne torbe, potrebno je ugraditi metodu diskretizacije. Mnoge su metode diskretizacije predložene [8, 9, 10]. Metode diskretizacije potrebno je ugraditi u algoritam kako bi se ostvarila mogućnost odabira kombinacije predmeta koji će ispuniti torbu, odnosno potrebno je iz realnih vektora generirati binarne koji bi određivali koji predmeti ulaze, odnosno ne ulaze u torbu.

U ovom radu su odabrana tri različita načina diskretizacije. Prvi način diskretizacije, u algoritmu

pragDE, zasniva se na pragovanju (engl. *thresholding*) vrijednosti ovisno o tome je li broj veći ili manji od 0,5 koji predstavlja prag. Drugi način, u algoritmu *normDE*, [8], se sastoji od normalizacije vrijednosti te se zatim vrši pragovanje. Treći način, u algoritmu *mod2DE*, [14], se zasniva na modulo 2 operaciji.

Algoritam *pragDE* koristi najjednostavniji način diskretizacije kojim se provjerava vrijednost komponenti vektora, odnosno utvrđuje radi li se o broju većem ili manjem od 0,5. U slučaju ako je broj veći od ili jednak broju 0,5, generira se bit 1, u suprotnom generira se bit 0. Niz bitova se generira na način

$$\hat{y}_{ij}(t) = \begin{cases} 0 & ako\ x_{ij}(t) < 0{,}5 \\ 1 & u\ suprotnom \end{cases}, \quad i = 1, \dots, D \tag{2-8}$$

gdje je populacija inicijalizirana realnim vrijednostima iz [0,1] i održava se unutar tih granica tijekom izvođenja algoritma. Kod ugradnje drugog načina diskretizacije u algoritmu *normDE*, populacija se inicijalizira nasumičnim realnim vrijednostima iz proizvoljnog intervala ([-10,10] u konkretnom slučaju). Prvo se normalizira svaku komponentu vektora prema

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}, \tag{2-9}$$

gdje x_{min} i x_{max} predstavljaju najmanju i najveću komponentu *i*-tog vektora. Zatim se niz bitova generira prema

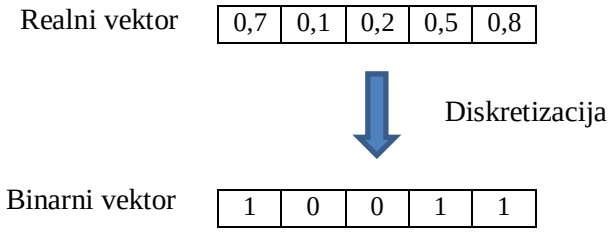
$$\hat{y}_{ij} = \begin{cases} 0 & ako\ {x'}_{ij} < 0{,}5 \\ 1 & u\ suprotnom \end{cases}, \quad i = 1, \dots, D \tag{2-10}$$

Treći način diskretizacije realnih komponenti vektora se odvija prema

$$\hat{y}_{ij} = floor\left(\left|x_{ij}\ mod\ 2\right|\right), \quad i = 1, \dots, D \tag{2-11}$$

gdje x_{ij} predstavlja vektor sa realnim parametrima, a $\hat{y}_{ij}$ predstavlja binarni, odnosno diskretizirani vektor. Nad realnim vektorom izvršava se funkcija mod2, zatim se njegova apsolutna vrijednost zaokružuje na manji cijeli broj pomoću funkcije floor(). Što znači, ako je apsolutna vrijednost broja između 0 i 0,999*, rezultat diskretizacije će biti 0. U slučaju kad je apsolutna vrijednost broja između 1 i 1,999*, rezultat diskretizacije će biti 1. Diskretizacija (Slika 2.2.) u algoritmu vrši se

samo tijekom vrednovanja rješenja kako bi bilo moguće odabrati kombinaciju predmeta koji ulaze u torbu te kako bi zatim selekcijom bilo moguće odrediti koja rješenja odlaze u sljedeću generaciju. Na slici 2.3. je prikazana diskretizacija vektora unutar algoritma gdje se vektori lokalno diskretiziraju samo tijekom vrednovanja rješenja. Vektori s kojima algoritam radi nakon vrednovanja nisu diskretizirani.



Slika 2.2. Primjer diskretizacije vektora

```
// postavljanje parametara Np, F, CR
// inicijalizacija populacije
do
{
        // generiranje sljedeće generacije
        for (i<Np) // i=0
        {
                // Diskretizacija ciljnog i pokusnog vektora x_i i u_i
                if ( f(û[i]) >= f(x̂[i]) ) // Diskretiziraj lokalno x_i i u_i
                {
                        x[i]=u[i]; // Spremljene vrijednosti nisu diskretizirane
                }
        }
} while (uvjet završetka nije ispunjen)
```

Slika 2.3. Diskretizacija u algoritmu DE

Nakon ugradnje načina diskretizacije u algoritam, omogućeno je vrednovanje rješenja. Sljedeći korak je implementacija funkcije cilja za vrednovanje rješenja kako bi se moglo odlučiti koja rješenja preživljavaju te ulaze u sljedeću generaciju. Treba napomenuti, kako bi se riješio problem putne torbe, potrebno je odabrati kombinaciju predmeta čija je ukupna cijena maksimalna te da njihova ukupna masa ne prelazi kapacitet torbe kao što je navedeno jednadžbom (2-1). Funkcija cilja *f* upravo računa tu ukupnu cijenu predmeta odabranih unutar trenutnog rješenja (Slika 2.4.). Zatim se veća cijena vrednuje kao bolje rješenje te odlazi u sljedeću generaciju.

```
// populacija rješenja je lokalno diskretizirana
ukupna_cijena = 0, ukupna_masa = 0;
for(i<D) // i=0
{
    ukupna_cijena += predmet[i] * pripadajuca_cijena[i];
    ukupna_masa   += predmet[i] * pripadajuca_masa[i];
}
if (ukupna_masa > kapacitet_torbe) return 0; // nevaljano rješenje vraća '0'
else return ukupna_cijena;
```

Slika 2.4. Pseudo kod funkcije cilja *za vrednovanje rješenja*

Kod vrednovanja treba voditi računa o rješenjima koja su nevaljana (engl. *infeasible*). To su ona koja premašuju kapacitet torbe. Takvim rješenjima treba rukovati na odgovarajući način. Najjednostavnije je kažnjavati takva rješenja tako što im se postavi vrijednost funkcije cilja na nulu (Slika 2.4.) ili čak negativnu vrijednost. Na taj način neće proći selekciju i u narednu generaciju/iteraciju. Nakon što se zadovolji uvjet završetka, algoritam se prekida. Zatim se pronalazi rješenje najvećeg iznosa unutar populacije rješenja, koje predstavlja potencijalno optimalno rješenje.

3. Ostvareno programsko rješenje

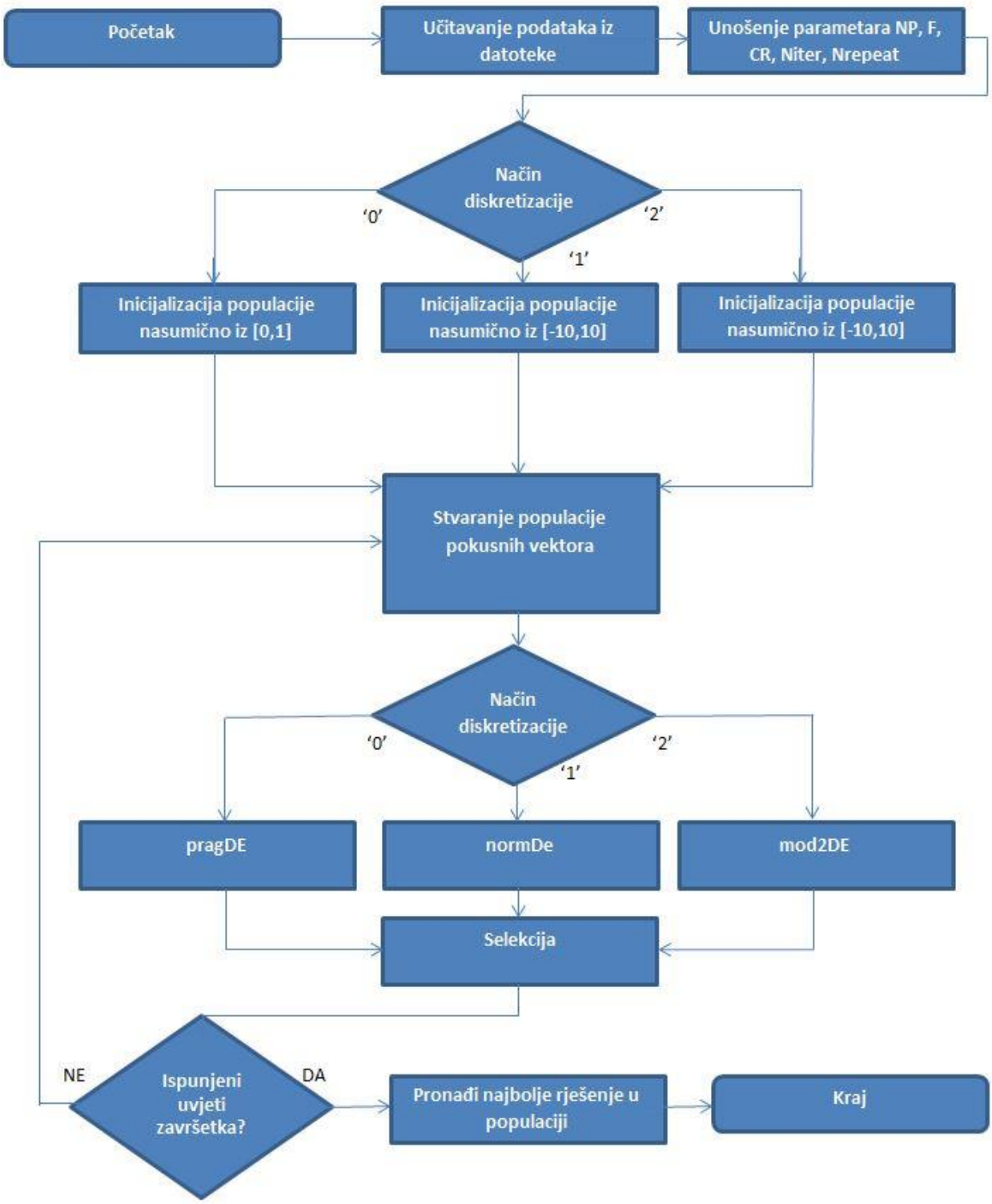
Programsko je rješenje ostvareno u programskom jeziku C#, koristeći integrirano razvojno okruženje Microsoft Visual Studio 2019. Programsko rješenje omogućava učitavanje datoteka s podacima o problemu. Omogućuje postavljanje načina diskretizacije te postavljanje parametara algoritma DE uz broj iteracija. Također, omogućuje postavljanje broja ponavljanja izvođenja cijelog postupka jer se radi o stohastičkom algoritmu pa je potrebno više izvođenja na istom problemu kako bi se dobio uvid u učinkovitost [11]. Kao izlaz programsko rješenje daje minimalnu vrijednost, maksimalnu vrijednost, odnosno potencijalnu optimalnu vrijednost funkcije cilja za svako od izvođenja na danom primjerku problema te računa standardnu devijaciju i odstupanje prosječne vrijednosti rješenja od optimuma radi prilagodbe eksperimentalnoj analizi. Korisnikova interakcija s programskim rješenjem tijekom izvođenja eksperimenta odvija se preko konzole.

3.1. Način rada programskog rješenja

Programsko rješenje učitava datoteku s podacima o problemu. Zatim korisnik unosi parametre algoritma DE: veličinu populacije *Np*, faktor skaliranja *F*, stopu križanja *Cr*, broj iteracija *Niter*, broj ponavljanja algoritma *Nrepeat*. Nakon toga korisnik odabire način diskretizacije. Kao što je prethodno opisano, koriste se tri načina diskretizacije ugrađene u algoritme *pragDE* (‘0’), *normDE* (‘1’) i *mod2DE* (‘2’). Nakon odabira načina diskretizacije, odnosno algoritma, program započinje s inicijalizacijom populacije. U slučaju da je odabran način ‘0’, vektori unutar populacije se popunjavaju nasumično odabranim realnim brojevima iz [0,1]. U slučaju da su odabrani načini ‘1’ ili ‘2’, vektori unutar populacije se popunjavaju nasumično odabranim realnim brojevima iz [-10, 10]. Nakon inicijalizacije populacije stvaraju se populacije pokusnih vektora. Zatim se vektori unutar algoritma lokalno diskretiziraju metodom koju je korisnik odabrao kako bi se rješenja mogla vrednovati te se vrši selekcija. Ako su uvjeti završetka ispunjeni, pronalazi se najbolje rješenje unutar populacije te se prekida izvođenje programa.

Nakon svakog ponavljanja *Nrepeat*, najveća vrijednost iz populacije, koja predstavlja najveću ukupnu cijenu predmeta, pohranjuje se u listu podataka. Ta vrijednost upravo je i potencijalni optimum. Radi prilagodbe programskog rješenja eksperimentalnoj analizi, iz liste podataka je računata minimalna vrijednost rješenja, maksimalna vrijednost rješenja, prosječna vrijednost

rješenja, standardna devijacija i odstupanje prosječne vrijednosti od optimuma te su predane na izlazu programa. Postupak je opisan dijagramom toka (Slika 3.1.).



Slika 3.1. Dijagram toka programskog rješenja

3.2. Prikaz i način uporabe programskog rješenja

Korisnikova interakcija s programskim rješenjem odvija se preko ugrađene konzole. Prije pokretanja programa potrebno je unutar koda navesti iz koje se datoteke uzimaju podaci (Slika 3.2.).

```
static void Main(string[] args)
{
    Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo("en-US");
    if (!loadFromFile("f8_l-d_kp_23_10000")) return;
```

Slika 3.2. Odabir datoteke za učitavanje podataka

Nakon što je datoteka uspješno učitana od korisnika se traži da unese parametre algoritma DE (veličina populacije, faktor skaliranja i stopa križanja) te broj ponavljanja izvođenja algoritma iz razloga što se radi o stohastičkom algoritmu, a kako bi se dobio uvid u učinkovitost algoritma. Nakon što korisnik unese broj ponavljanja, mora odabrati koji će se način diskretizacije koristiti u algoritmu DE. Ako se želi odabrati *pragDE*, korisnik unosi broj '0', za *normDE* broj '1' te za *mod2DE* broj '2' (Slika 3.3.).

```
Microsoft Visual Studio Debug Console
Unesi velicinu populacije:
50
Unesi faktor skaliranja:
0.5
Unesi stopu krizanja:
0.9
Unesi broj iteracija:
1000
Unesi broj ponavljanja algoritma:
30
Odaberi metodu diskretizacije:
0 : pragDE
1 : normDE
2 : mod2DE
```

Slika 3.3. Postavljanje parametara

Nakon što se algoritam izvrši, program kao izlaz za sva rješenja svih ponavljanja računa te ispisuje prosječnu vrijednost funkcije cilja, standardnu devijaciju, maksimalnu vrijednost funkcije cilja, najmanju vrijednost funkcije cilja te odstupanje prosječne vrijednosti funkcije cilja od optimuma ($(1 - \frac{\bar{X}}{Optimum}) * 100\%$), kao što je slikom 3.4. prikazano.

Slika 3.4. Prikaz izlaza programa

4. Eksperimentalna analiza

U ovom radu eksperimentima je pokazana važnost odabira metode diskretizacije u algoritmu DE. Odabir metode diskretizacije može imati osjetan učinak na ponašanje algoritma DE. Kao što je prethodno već navedeno, korištena su tri načina diskretizacije, ugrađena u algoritme *pragDE, normDE i mod2DE*. Eksperimentima je ispitana njihova učinkovitost, zatim su načini diskretizacije uspoređeni te su rezultati usporedbe prikazani tablično radi bolje preglednosti. U svrhu ovog istraživanja korišteni su primjerci problema manjih dimenzija.

Eksperimentirano je na deset problema manjih dimenzija koji su dostupni preko linka: http://artemisa.unicauca.edu.co/~johnyortega/instances_01_KP/ (označeni su kao "*Low-dimensional 0/1 knapsack problems*"). Tablicom 4.1. prikazani su problemi i osnovni podaci problema (naziv, broj predmeta, kapacitet torbe i vrijednost funkcije cilja u optimumu).

Tablica 4.1. Problemi korišteni za eksperimentalnu analizu

Problem	Naziv	Broj predmeta	Kapacitet	Optimum
P1	f1_l-d_kp_10_269	10	269	295
P2	f2_l-d_kp_20_878	20	878	1024
P3	f3_l-d_kp_4_20	4	20	35
P4	f4_l-d_kp_4_11	4	11	23
P5	f5_l-d_kp_15_375	15	375	481,0694
P6	f6_l-d_kp_10_60	10	60	52
P7	f7_l-d_kp_7_50	7	50	107
P8	f8_l-d_kp_23_10000	23	10000	9767
P9	f9_l-d_kp_5_80	5	80	130
P10	f10_l-d_kp_20_879	20	879	1025

Od deset problema korištenih za eksperimentalnu analizu, u njih devet cijene i mase predmeta su cjelobrojne vrijednosti, dok su u jednom problemu (P5) cijene i mase predmeta realne vrijednosti. Tablicom 4.2. prikazane su cijene i mase svih predmeta.

Tablica 4.2. Cijene i mase predmeta unutar problema

Problem	Cijene i mase predmeta (*pj,wj*)
P1	(55, 95), (10, 4), (10, 4), (47, 60), (5, 32), (50, 72), (8, 80), (61, 62), (85, 65), (87, 46)
P2	(44, 92), (46, 4), (90, 43), (72, 83), (91, 84), (40, 68), (75, 92), (35, 82), (8, 6), (54, 44), (78, 32), (40, 18), (77, 56), (15, 83), (61, 25), (17, 96), (75, 70), (29, 48), (75, 70), (29, 48),(75, 14), (63, 58)
P3	(9, 6), (11, 5), (13, 9), (15, 7)
P4	(6, 2), (10, 4), (12, 6), (13, 7)
P5	(0,125126, 56,358531), (19,330424, 80,874050), (58,500931, 47,987304), (35,029145, 89,596240), (82,284005, 74,660482), (17,410810, 85,894345), (71,050142, 51,353496), (30,399487, 1,498459), (9,140294, 36,445204), (14,731285, 16,589862), (98,852504, 44,569231), (11,908322, 0,466933), (0,891140, 37,788018), (53,166295, 57,118442), (60,176397, 60,716575)
P6	(20, 30), (18, 25), (17, 20), (15, 18),(15, 17), (10, 11), (5, 5), (3, 2), (1, 1), (1,1)
P7	(70, 31), (20, 10), (39, 20), (37, 19), (7, 4), (5, 3), (10, 6)
P8	(981, 983), (980, 982), (979, 981), (978, 980),(977, 979), (976, 978), (487, 488), (974, 976), (970, 972), (485, 486), (485, 486), (970, 972), (970, 972), (484, 485), (484, 485), (976, 969), (974, 966), (482, 483), (962, 964), (961, 963), (959, 961), (958, 958), (857, 959)
P9	(33, 15), (24, 20), (36, 17), (37, 8), (12, 31)
P10	(91, 84), (72, 83), (90, 43), (46, 4), (55, 44), (8, 6), (35, 82), (75, 92), (61, 25), (15, 83), (77, 56), (61, 25), (15, 83), (77, 56), (40, 18), (63, 58), (75, 14), (29, 48), (75, 70), (17, 96), (78, 32), (40, 68), (44, 92)

4.1. Postavke eksperimenta

Parametri korišteni za algoritam DE veličina populacije *Np*, faktor skaliranja *F*, stopa križanja *Cr* te parametri korišteni za eksperiment radi uvida u učinkovitost jesu broj iteracija *Niter* te broj ponavljanja izvođenja algoritma *Nrepeat*. Postavke parametara DE odabrane su iz literature, dok su postavke parametara eksperimenta *Niter* i *Nrepeat* odabrane empirijski preliminarnom analizom. Bitno je uzeti u obzir da su u eksperimentu korišteni problemi manjih dimenzija (Tablica 4.1), stoga je potrebno prilagoditi postavke parametara kako bi eksperiment bio valjan i pouzdan te kako bi rezultati što bolje odmjerili učinkovitost algoritma. Tablicom 4.3. su prikazane postavke parametara.

Tablica 4.3. Postavke parametara korištenih za eksperiment

Parametar	Vrijednost
Np	50
F	0,5
CR	0,9
$Niter_{1,2,3}$	250,500,1000
Nrepeat	30

Idealno, pronalaženje veličine populacije koja je najprikladnija za određeni algoritam omogućuje najbolju izvedbu danog algoritma [3]. Odabrana veličina populacije je *Np* =50 iz razloga što prema [3] taj je iznos davao najbolje rezultate za više primjeraka različitih problema sličnih ili većih dimenzija u odnosu na problem korišten u ovoj eksperimentalnoj analizi. Iz toga se može zaključiti da navedena veličina populacije neće negativno utjecati na provođenje eksperimenta. Iz istog razloga su ostali parametri DE faktor skaliranja *F* te stopa križanja *Cr* postavljeni na *F* = 0,5 te *CR* = 0,9, a koji su vrlo često korišteni u literaturi.

Broj iteracija *Niter* predstavlja broj generacija algoritma DE. Radi boljeg uvida u učinkovitost rada tri različita načina diskretizacije te kako bi se bolje istražilo njihove razlike, eksperiment proveden je na tri različite vrijednosti broja iteracija $Niter_1$ = 250, $Niter_2$ = 500 te $Niter_3$ =1000 za svaku metodu diskretizacije. Eksperiment proveden je na stohastičkom algoritmu pa je potrebno više izvođenja na istom problemu kako bi se dobio bolji uvid u učinkovitost [11]. Iz toga razloga broj ponavljanja algoritma je postavljen na *Nrepeat* = 30. To znači da je za svaku metodu diskretizacije i dani primjerak problema algoritam izvođen 30 puta, što predstavlja 90 ponavljanja po problemu.

Kako bi se mogla usporediti učinkovitost načina diskretizacije u algoritmima *pragDE*, *normDE* i *mod2DE*, izračunat je prosjek svih cijena $\bar{X}$ dobivenih unutar svih ponavljanja (*Nrepeat* = 30) za svaki način diskretizacije. Uzeto je najbolje i najlošije rješenje od 30 izvođenja X_{min} te X_{max} za svaki problem. Zatim je izračunata standardna devijacija σ i odstupanje prosjeka od optimalnog rješenja $\Delta D = (1 - \frac{\bar{X}}{Optimum}) * 100\%$ kako bi dobili uvid u raspršenost i preciznost rješenja koje pronalazi pojedini algoritam. Rezultati su predstavljeni tablično te su grafički prikazane razlike učinkovitosti metoda diskretizacije korištenih u algoritmima *pragDE*, *normDE* i *mod2DE* u ovisnosti o broju iteracija.

4.2. Rezultati

Tablicom 4.4. prikazani su rezultati izvođenja eksperimenta pri zadanom broju iteracija $Niter_1$=250. Primjećuje se da je DE učinkovita na zadanim problemima te da nije pronašla optimalno rješenje samo za problem P8 koristeći metodu diskretizacije ugrađenu u algoritmu *normDE*, iako je odstupanje prosjeka svih rješenja od optimuma iznosilo niskih ΔD=0,25%. Na problemima P3, P4 i P9 pronađeno je optimalno rješenje u svih 30 ponavljanja sa svim razmatranim metodama diskretizacije. Na ostalim problemima prosjek svih rješenja vrlo je blizu optimumu.

Tablica 4.4. Rezultati eksperimenta pri $Niter_1$= 250

Problem	**Optimum**	**Algoritam**	$\bar{X}$	σ	X_{max}	X_{min}	ΔD[%]
P1	295	*pragDE*	295	0	295	295	0
		normDE	294**,**8667	0,4269	295	293	0,05
		mod2DE	295	0	295	295	0
P2	1024	*pragDE*	1024	0	1024	1024	0
		normDE	985.1667	35,3677	1024	899	3,79
		mod2DE	1024	0	1024	1024	0
P3	35	*pragDE*	35	0	35	35	0
		normDE	35	0	35	35	0
		mod2DE	35	0	35	35	0
P4	23	*pragDE*	23	0	23	23	0
		normDE	23	0	23	23	0
		mod2DE	23	0	23	23	0
P5	481,0694	*pragDE*	481,0694	0	481,0694	481.0694	0
		normDE	463,7532	22,2339	481,0694	417.8686	3,60
		mod2DE	480,6966	1,3946	481,0694	475.4784	0,08
P6	52	*pragDE*	52	0	52	52	0
		normDE	51.8667	0,4989	52	50	0,26
		mod2DE	52	0	52	52	0
P7	107	*pragDE*	106,4667	0,8844	107	105	0,50
		normDE	106,9333	0,3590	107	105	0,06
		mod2DE	106,7333	0,6799	107	105	0,25
P8	9767	*pragDE*	9766,7	0,8622	9767	9763	0,01
		normDE	9742,6333	10,7098	9758	9720	0,25
		mod2DE	9763,5667	2,3760	9767	9760	0,04
P9	130	*pragDE*	130	0	130	130	0
		normDE	130	0	130	130	0
		mod2DE	130	0	130	130	0
P10	1025	*pragDE*	1025	0	1025	1025	0
		normDE	1000,8	22,8771	1025	924	2,36
		mod2DE	1025	0	1025	1025	0

Algoritam *normDE* se pokazao kao najgori odabir za 5 problema (P1, P2, P5, P6, P8) dok se algoritam *pragDE* pokazao kao najbolji odabir za 9 od 10 problema. Algoritam *mod2DE* se pokazao kao najbolji odabir za 7 od 10 problema.

Nadalje, sljedeći eksperiment je proveden na istim problemima, uz jednake postavke parametara algoritma DE, ali broj iteracija je povećan na $Niter_2$=500. Tablicom 4.5. prikazani su rezultati provođenja eksperimenta.

Tablica 4.5. Rezultati eksperimenta pri $Niter_2$=500

Problem	**Optimum**	**Algoritam**	$\bar{X}$	σ	X_{max}	X_{min}	ΔD[%]
P1	295	*pragDE*	295	0	295	295	0
		normDE	294,9667	0,1795	295	294	0,01
		mod2DE	295	0	295	295	0
P2	1024	*pragDE*	1024	0	1024	1024	0
		normDE	1018.8	11,2558	1024	978	0,51
		mod2DE	1024	0	1024	1024	0
P3	35	*pragDE*	35	0	35	35	0
		normDE	35	0	35	35	0
		mod2DE	35	0	35	35	0
P4	23	*pragDE*	23	0	23	23	0
		normDE	23	0	23	23	0
		mod2DE	23	0	23	23	0
P5	481,0694	*pragDE*	481,0694	0	481,0694	481,0694	0
		normDE	476,7532	13,1542	481,0694	431,7087	0,90
		mod2DE	481,0694	0	481,0694	481,0694	0
P6	52	*pragDE*	52	0	52	52	0
		normDE	52	0	52	52	0
		mod2DE	52	0	52	52	0
P7	107	*pragDE*	106,533	0,8459	107	105	0,44
		normDE	106,9333	0,3590	107	105	0,06
		mod2DE	106,6667	0,7454	107	105	0,31
P8	9767	*pragDE*	9767	0	9767	9767	0
		normDE	9760,5	4,4926	9767	9751	0,07
		mod2DE	9766,5667	0,0835	9767	9765	0,01
P9	130	*pragDE*	130	0	130	130	0
		normDE	130	0	130	130	0
		mod2DE	130	0	130	130	0
P10	1025	*pragDE*	1025	0	1025	1025	0
		normDE	1021,2	11,9175	1025	964	0,37
		mod2DE	1025	0	1025	1025	0

Algoritmi su uz povećan broj iteracija postigli nešto bolje rezultate, što je bilo i za očekivati s obzirom na povećani opseg pretrage. Iz rezultata provedenih eksperimenta vidi se da su algoritmi *pragDE*, *normDE* i *mod2DE* pronašli optimalno rješenje na svih 10 problema. Na problemima P3, P4, P6 i P9 DE pronašla je optimalno rješenje u svih 30 ponavljanja, bez obzira na korištenu metodu diskretizacije. Na ostalim problemima prosjek svih rješenja bliže je optimumu u odnosu na prethodni eksperiment pri manjem broju iteracija. Algoritam *pragDE* pokazao se kao najbolji odabir za 9 problema, dok se algoritam *normDE* pokazao kao najgori izbor.

Posljednji je eksperiment proveden na istim problemima, uz jednake postavke parametara algoritma DE, ali broj iteracija je dodatno povećan na $Niter_3$=1000. Tablicom 4.6. su prikazani rezultati ostvareni u navedenom eksperimentu.

Tablica 4.6. Rezultati eksperimenta pri $Niter_3$= 1000

Problem	**Optimum**	**Algoritam**	$\bar{X}$	σ	X_{max}	X_{min}	ΔD[%]
P1	295	*pragDE*	295	0	295	295	0
		normDE	295	0	295	295	0
		mod2DE	295	0	295	295	0
P2	1024	*pragDE*	1024	0	1024	1024	0
		normDE	1023,4	1,8	1024	1018	0,06
		mod2DE	1024	0	1024	1024	0
P3	35	*pragDE*	35	0	35	35	0
		normDE	35	0	35	35	0
		mod2DE	35	0	35	35	0
P4	23	*pragDE*	23	0	23	23	0
		normDE	23	0	23	23	0
		mod2DE	23	0	23	23	0
P5	481,0694	*pragDE*	481,0694	0	481,0694	481,0694	0
		normDE	481,0694	0	481,0694	481,0694	0
		mod2DE	481,0694	0	481,0694	481,0694	0
P6	52	*pragDE*	52	0	52	52	0
		normDE	52	0	52	52	0
		mod2DE	52	0	52	52	0
P7	107	*pragDE*	106,6667	0,7454	107	105	0,31
		normDE	106,8667	0,4989	107	105	0,12
		mod2DE	106,7333	0,6799	107	105	0,25
P8	9767	*pragDE*	9767	0	9767	9767	0
		normDE	9766,8667	0,4989	9767	9765	0,01
		mod2DE	9767	0	9767	9767	0
P9	130	*pragDE*	130	0	130	130	0
		normDE	130	0	130	130	0
		mod2DE	130	0	130	130	0
P10	1025	*pragDE*	1025	0	1025	1025	0
		normDE	1024,4333	2,2164	1025	1014	0,06
		mod2DE	1025	0	1025	1025	0

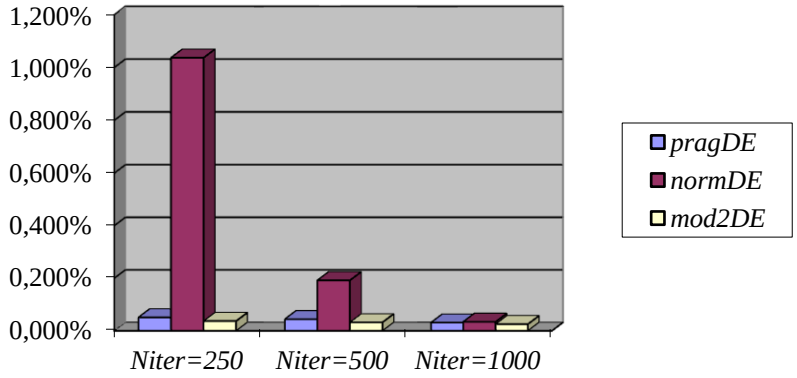
Kao što se moglo primijetiti kod prethodnog povećanja broja iteracija, i u ovom slučaju DE je postigla bolje rezultate s povećanjem broja iteracija na $Niter_3$=1000. Rezultati iz tablice 4.6. pokazuju da su algoritmi *pragDE, normDE* i *mod2DE* pronašli optimalno rješenje na svim problemima. Na problemima P1, P3, P4, P5, P6 i P9 razmatrani algoritmi pronašli su optimalno rješenje u svih 30 ponavljanja, što predstavlja dosad najbolji uspjeh u provedenim eksperimentima.

Tablicom 4.7. prikazani su ukupni rezultati bodovanja algoritama *pragDE*, *normDE* i *mod2DE* pri različitom broju iteracija. Format zapisivanja rezultata u obliku je „najbolji izbor/najgori izbor“, gdje „najbolji izbor“ predstavlja koliko se puta odabrani algoritam pokazao kao najbolji odabir u odnosu na ostale algoritme, analogno tome „najgori izbor“ predstavlja koliko se puta odabrani algoritam pokazao kao najgori odabir u odnosu na ostale algoritme, na zadanom problemu. Iz tablice 4.7. vidi se da je algoritam *pragDE* 27 puta bio najbolji izbor, a najgori izbor samo tri puta. Algoritam *normDE* 17 puta bio je najbolji izbor, a najgori 13 puta. Algoritam *mod2DE* 24 je puta bio najbolji izbor te nijednom nije bio najgori izbor.

Tablica 4.7. Ukupni rezultati bodovanja algoritama

Algoritam	*Niter*			Ukupni rezultat
	250	500	1000	
pragDE	9/1	9/1	9/1	27/3
normDE	4/5	6/5	7/3	17/13
mod2DE	7/0	8/0	9/0	24/0

Odnos performansi ovisno o broju iteracija prikazan je grafički slikom 4.2. na kojoj se vidi prosječno odstupanje prosječne vrijednosti od optimuma za svaki algoritam pri različitom broju iteracija. Iz grafa se može uočiti da povećanje broja iteracija pogoduje performansama DE, što je očekivano, jer broj iteracija izravno utječe na opseg pretrage prostora rješenja kroz broj vrednovanih rješenja (jednak je produktu veličine populacije i broja iteracija).



Slika 4.2. Prosječno odstupanje prosjeka od optimuma algoritama pri različitom broju iteracija

Razlike u odstupanju algoritama vrlo su malene te zbog malih dimenzija problema uglavnom nije potreban veći broj iteracija, ali potrebno je napomenuti da bi razlike u odstupanju vjerojatno bile

očitije pri većim/težim problemima. Iz rezultata eksperimentalne analize vidi se da je najjednostavnija metoda diskretizacije *pragDE* pokazala općenito najbolje performanse.

5. Zaključak

Zadatak je rada bio predstaviti diferencijalnu evoluciju (DE) kao algoritam za rješavanje problema putne torbe. S obzirom da je algoritam DE izvorno namijenjen za rješavanje problema kontinuirane optimizacije, nije izravno primjenjiv na probleme diskretne ili kombinatorne optimizacije. Stoga je nužno ugraditi neku od dostupnih metoda. U radu su opisane i ugrađene tri različite metode diskretizacije te je uspoređena njihova učinkovitost. Ostvareno je programsko rješenje koje ugrađuje metode diskretizacije u algoritam DE te omogućuje odabir željenih problema te parametara i omogućuje praktično izvođenje eksperimentalne analize. Eksperimentalnom analizom utvrđeno je da, ovisno o odabiru načina diskretizacije, izvedba algoritama DE rezultira različitim performansama. Rezultati analize pokazali su da je najjednostavnija od razmatranih metoda diskretizacije u najviše slučajeva bila najbolji odabir. Eksperimentalnom analizom dodatno je utvrđeno da povećanje broja iteracija pozitivno utječe na performanse algoritma, što je bilo za očekivati. Bitno je naglasiti da se u ovom radu eksperimentalna analiza vršila nad problemima manjih dimenzija te da povećanje broja iteracija nije imalo negativne učinke na vrijeme izvođenja ili iscrpljivanje računalnih resursa. U budućem radu moglo bi se razmotriti više metoda diskretizacije s obzirom da ih u literaturi postoji još nekoliko te je potrebno u eksperimentima razmotriti veće primjerke problema kako bi se dobio bolji uvid u performanse algoritma DE.

Literatura

[1] A. Shaheen, A. Sleit, Comparing between different approaches to solve the 0/1 knapsack problem, International Journal of Computer Science and Network Security, No. 7, Vol. 16, pp. 1–10, srpanj 2016.

[2] S. Martello, P. Toth, Knapsack Problems: Algorithms and Computer Implementations, John Wiley and Sons, Engleska, 1990.

[3] H. Kellerer, U. Pferschy, D. Pisinger, Knapsack Problems, Springer-Verlag Berlin Heidelberg, Berlin, 2004.

[4] A. P. Engelbrecht, Computational Intelligence: An Introduction, Second Edition. Wiley Publishing, Južna Afrika, 2007.

[5] H. Peng, Z. Wu, P. Shao, C. Deng, Dichotomous binary differential evolution for knapsack problems, Mathematical Problems in Engineering, Vol. 2016, prosinac 2016.

[6] K. Price, R. Storn, J. A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer-Verlag New York, Inc., Njemačka, 2005.

[7] D. Bajer, Adaptive k-tournament mutation scheme for differential evolution, Applied Soft Computing, Vol. 85, prosinac 2019.

[8] A. P. Engelbrecht, G. Pampara, Binary differential evolution strategies, Proceedings of the 2007 IEEE Congress on Evolutionary Computation Conference, pp. 1942– 1947, 25.–28. rujan, 2007.

[9] G. Martinović, D. Bajer, B. Zorić, A differential evolution approach to dimensionality reduction for classification needs, International Journal of Applied Mathematics and Computer Science, No. 1, Vol. 24, pp. 111–122, ožujak 2014.

[10] M. S. Kiran, The continuous artificial bee colony algorithm for binary optimization, Applied Soft Computing, Vol. 33, pp. 15-23, kolovoz 2015.

[11] S. Jun, L. Jian, Solving 0-1 knapsack problems via a hybrid differential evolution, Proceedings of the 2009 Third International Symposium on Intelligent Information Technology Application, pp. 134–137, 21. – 22. studeni, 2009.

Sažetak

U ovom se radu predstavlja algoritam diferencijalne evolucije (DE), jedan od najpopularnijih i najučinkovitijih evolucijskih algoritama, za rješavanje problema putne torbe. Navedeni problem predstavlja NP-težak problem kombinatorne optimizacije u kojemu se pokušava povećati ukupna cijena predmeta unutar putne torbe bez premašivanja njezinog kapaciteta. Algoritam diferencijalne evolucije (DE) izvorno je namijenjen za rješavanje problema kontinuirane optimizacije te nije izravno primjenjiv na problem putne torbe. Glavni je razlog za to za nju specifični operator mutacije koji je isključivo prikladan za vektore realnih komponenti. Naime, nužna je ugradnja neke od metoda diskretizacije. One diskretiziraju vektore unutar algoritma DE u svrhu njegovog korištenja u problemima diskretne kombinatorike. U ovom su radu korištene tri različite metode diskretizacije te su se njihove učinkovitosti usporedile u odgovarajućoj eksperimentalnoj analizi koja je obavljena pomoću ostvarenog programskog rješenja. Eksperimentalnom analizom pokazana je važnost odabira metode diskretizacije u svrhu ostvarivanja boljih performansi algoritma DE pri rješavanju diskretnih problema optimizacije.

Ključne riječi: diferencijalna evolucija, kombinatorna optimizacija, metode diskretizacije, problem putne torbe.

Abstract

Differential evolution algorithm for the knapsack problem

This paper presents the differential evolution algorithm (DE), one of the most popular and most efficient evolutionary algorithms, for the knapsack problem. The knapsack problem is an NP-hard problem in which the goal is to increase the total price of items within the knapsack without exceeding its capacity. The DE algorithm was originally proposed for solving continuous optimization problems and is not directly applicable to the knapsack problem. The main reason for this is the specific mutation operator that is only suitable for real-valued vectors. Namely, it is necessary to incorporate a method for discretization of the real-valued solutions the algorithm operates on. This enables the application of the algorithm to discrete problems. Three different discretization methods were used in this paper and their efficiencies were compared by an experimental analysis. The experimental analysis was performed using the developed software solution. The experimental analysis has proven the importance of choosing a proper discretization method in order to achieve better performance of the differential evolution algorithm.

Keywords: Differential evolution, combinatorial optimization, discretization methods, knapsack problem.

Životopis

Zvonimir Šimić rođen je 29. kolovoza 1997. godine u Osijeku. Pohađao je Osnovnu školu Franje Krežme u Osijeku. Nakon toga upisuje III. gimnaziju Osijek u Osijeku. Nakon završetka srednjoškolskog obrazovanja, upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku na Sveučilištu Josipa Jurja Strossmayera, smjer preddiplomski studij elektrotehnike. Zatim se nakon završetka prve godine prebacuje na smjer preddiplomski studij računarstva. Od programskih jezika učio je C, C++, C#, Javu, HTML, CSS i VHDL.

Prilozi /na CD-u/

1. Završni rad u doc, docx i PDF formatu

2. Projekt programskog rješenja

3. Podaci korišteni za eksperimentalnu analizu