UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Aleksandra Đorđević

REŠAVANJE PROBLEMA MINIMALNOG ZBIRNOG BOJENJA GRAFA EGZAKTNIM I HEURISTIČKIM METODAMA

master rad

Beograd, 2022.

Mentor:

dr Zorica Dražić, docent
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

prof. dr Aleksandar Savić, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

prof. dr Zorica Stanimirović, redovni profesor
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: 09.09.2022.

Naslov master rada: Rešavanje problema minimalnog zbirnog bojenja grafa egzaktnim i heurističkim metodama

Rezime: Bojenje grafova je jedan od najpoznatijih i najviše rešavanih NP-teških problema kombinatorne optimizacije i ima veliki praktičan značaj. Primenjuje se kod problema rasporeda, upravljanja skladištem, alokacije frekvencija u mobilnoj mreži, alokacije registara u optimizaciji kompajlera. U ovom radu razmatrana je varijanta problema bojenja čvorova grafova pod nazivom Problem minimalnog zbirnog bojenja (engl. Minimum Sum Coloring Problem, MSCP). Zadatak ovog problema je pronaći pravilno bojenje čvorova grafa koristeći boje predstavljene prirodnim brojevima $\{1, 2, ...\}$, tako da zbir boja dodeljenih čvorovima bude minimalan.
Za navedeni problem, u ovom radu su izložena dva ekvivalentna matematička modela preuzeta iz literature: jedan model koji je formulisan kao problem celobrojnog nelinearnog programiranja, dok je drugi linearan i korišćen je u ovom radu u okviru CPLEX rešavača. U sekciji sa eksperimentalnim rezultatima prikazane su najbolje teorijske i računske granice rešenja problema koje su preuzete iz literature. U ovom radu su dizajnirane i implementirane metoda promenljivih okolina i Gausovska metoda promenljivih okolina za rešavanje navedenog problema. Obe predložene metode su testirane na 87 instanci malih, srednjih i velikih dimenzija. Svi dobijeni rezultati su prikazani i upoređeni sa rezultatima egzaktnog rešavača, kao i sa postojećim rezultatima iz literature.

Ključne reči: bojenje grafova, heuristike, egzaktne metode, optimizacija, metoda promenljivih okolina

Title: Solving the minimum sum coloring problem using exact and heuristic methods

Abstract: Graph coloring is one of the fundamental and mostly solved NP-hard problems in combinatorial optimization and has a great practical importance. It has application in scheduling, warehouse management, frequency assignment in cellular networks, register allocation in compilers. In this paper, a variant of a vertex coloring problem is presented, called the Minumum Sum Coloring Problem (MSCP). The goal is to find a proper vertex coloring using colors that are represented by natural numbers {1, 2, ...}, such that the sum of all colors is minimal.
In this paper, two equivalent mathematical models from the literature are presented for the considered problem: the first one is an integer nonlinear mathematical formulation, while the second formulation, which is linear, is used within the CPLEX exact solver. Theoretical and computational bounds proposed in the literature are also presented. In order to solve the mentioned problem, the Variable Neighborhood Search and the Gaussian Variable Neighborhood Search are designed and implemented in this paper. Both methods are tested on 87 instances of small, medium and large dimensions. The obtained results are presented and compared with the results of an exact solver, as well as with the results from the literature.

Keywords: graph coloring, heuristics, exact methods, optimization, variable neighborhood search

Sadržaj

Glava 1

Uvod

1.1 Problemi diskretne i kontinualne optimizacije

Optimizacija je grana matematike i računarstva koja se bavi pronalaženjem najboljeg rešenja nekog problema pri određenim uslovima. U opštem slučaju, razlikujemo probleme minimizacije i probleme maksimizacije. Problemi tipa maksimizacije se mogu svesti na problem tipa minimizacije, i obrnuto. Potreba za rešavanjem ovakvih problema javlja se u mnogim naučnim disciplinama, kao što su operaciona istraživanja, menadžment, inženjerstvo, ekonomija, fizika, hemija, itd. Problem minimizacije može se formulisati na sledeći način: odrediti minimum date funkcije na zadatom skupu, odnosno rešiti:

$$\min_{x \in X} f(x), \quad X \subseteq S. \tag{1.1}$$

Skup S se naziva prostor rešenja, X skup dopustivih rešenja, $f : S \to \mathbb{R}$ funkcija cilja. Za tačku $x \in X$ se kaže da je dopustivo rešenje problema (1.1). Ovakva formulacija problema podrazumeva nalaženje dopustivog rešenja x^* (ili više njih) takvog da je $f(x^*) = \min_{x \in S} f(x)$. Rešenja x^* nazivamo optimalna rešenja.

U slučaju da je pronađeno rešenje $x^* \in X$ takvo da je $f(x^*) \leq f(x)$, za svako $x \in X$, kažemo da je x^* tačka globalnog minimuma funkcije f, tj. optimalno rešenje problema (1.1). Vrednost $f(x^*)$ je globalni minimum, odnosno vrednost funkcije cilja koja odgovara optimalnom rešenju. Ako postoji $\varepsilon > 0$ takvo da za neku tačku x^* važi da je $f(x^*) \leq f(x)$, za svako $x \in X$ i $||x^* - x|| \leq \varepsilon$, gde je sa $|| \cdot ||$ definisana neka ℓ_p norma u prostoru $\mathbb{R}^n$, tada x^* nazivamo tačka lokalnog minimuma, a $f(x^*)$ lokalni minimum. Ukoliko postoji samo jedan lokalni minimum, on je ujedno i globalni minimum za dati problem.

Budući da je $\max_{x \in X} f(x) = -\min_{x \in X}(-f(x))$, problem maksimizacije se može svesti na problem minimizacije, pa je dovoljno posmatrati samo problem nalaženja minimuma funkcije f. U zavisnosti od skupa S, optimizacione probleme možemo podeliti u dve kategorije. Ukoliko je skup S konačan ili prebrojiv, onda govorimo o kombinatornoj (diskretnoj) optimizaciji, a ako je $S \subseteq \mathbb{R}^n$ neprebrojiv, onda imamo problem kontinualne optimizacije.

Cilj optimizacije je konstrukcija efikasnih algoritama za nalaženje globalnog minimuma problema (1.1), čiji uspeh zavisi od težine problema koji se rešava. U praksi se često javljaju problemi kod kojih postoji mnogo (nekad i beskonačno mnogo) lokalnih minimuma. Egzaktne metode, koje garantuju nalaženje globalnog minimuma diskretne optimizacije, često su nepraktične iz razloga što njihove performanse zavise od veličine skupa dopustivih rešenja X. Ukoliko se poveća broj promenljivih i ograničenja, problem može postati previše vremenski i memorijski zahtevan za rešavanje. Iz tog razloga su razvijene heurističke metode, koje ne garantuju da će dobijeno rešenje biti i globalni minimum, ali za relativno kratko vreme izvršavanja dolaze do približnog, a nekad i do optimalnog rešenja.

1.2 Metode za rešavanje problema optimizacije

Algoritmi za rešavanje problema optimizacije se u opštem slučaju mogu podeliti u dve kategorije: egzaktne i približne algoritme.

Egzaktne metode garantuju ili optimalnost dobijenog rešenja, uz pružanje dokaza o optimalnosti, ili dokazuju da ono ne postoji. U nekim slučajevima, egzaktne metode rešavaju problem u polinomijalnom vremenu. Međutim, ovo ne važi i za NP-teške probleme, za koje nisu poznati algoritmi polinomijalne složenosti, već je potrebno eksponencijalno vreme kako bi se rešile egzaktnim metodama. U praksi se takođe dešava da rešavanje čak i NP-teških problema malih dimenzija zahteva dosta vremena i memorije. Još jedan nedostatak je što često nije moguće primeniti egzaktne metode jer ne postoji matematički model ili je taj model previše složen.

Sa druge strane, približne metode, među koje spadaju i heurističke metode, ne garantuju optimalnost dobijenog rešenja problema, ali u praksi često dolaze do dovoljno dobrog, neretko i optimalnog rešenja dovoljno brzo. Iako su u pitanju najmanje pouzdane metode, njihova primena je od velikog značaja u poslednjih 30 godina jer rešavaju najveći broj problema kombinatorne i kontinualne optimizacije.

1.3.1 Egzaktne metode

Neke od najpoznatijih egzaktnih metoda diskretne optimizacije su metoda grananja i ograničavanja, metoda implicitnog prebrojavanja, Gomorijeva metoda odsecanja i metoda grananja i vrednovanja. Metoda grananja i ograničavanja (engl. Branch-and-Bound) zasniva se na pretrazi samo onih delova skupa rešenja u kojima se po proceni nalazi bolje rešenje od već nađenog ili unapred poznatog. Glavna ideja ove metode je grananje početnog problema na manje potprobleme, gde je cilj da se mnogi potproblemi ne rešavaju, već odbace jer nemaju bolja rešenja od trenutnog rekorda, koji predstavlja vrednost funkcije cilja u do tada najboljem nađenom dopustivom rešenju [13]. Grananje se vrši tako što se dopustiv skup X predstavi kao unija nekih svojih podskupova X_k, koji ne moraju biti disjunktni. Potom se iz skupa potproblema izabere jedan, koji nije nužno lakši za rešavanje. Od njega se formira relaksacioni problem ili izostavljenjem nekih ograničenja ili zamenom skupa X_k nekim drugim skupom Y_k, tako da je potproblem lakši za rešavanje.

Metoda implicitnog prebrojavanja je specijalan slučaj metode grananja i ograničavanja i primenjuje se na probleme oblika

$$\begin{aligned} \min \quad & f(x_1, x_2, ..., x_n) \\ g_i(x_1, x_2, ..., x_n) \leq 0, \quad & i = 1, 2, ..., m \\ x_j \in K_j, \quad & j = 1, 2, ..., n, \end{aligned} \tag{1.2}$$

gde su K_j konačni brojevni skupovi. Fiksiranjem vrednosti promenljive x_j iz skupa K_j svakom potproblemu odgovara delimično rešenje, kod koga su promenljive x_{j_1}, x_{j_2}, ... x_{j_k} fiksirane vrednosti, dok su ostale nefiksirane. Zatim se obavljaju test neoptimalnosti i test nedopustivosti kojima se utvrđuje da li potproblem treba skinuti sa spiska ili nastaviti njegovo dalje grananje.

Grananje sa odsecanjem (engl. Branch-and-Cut) je egzaktna metoda za rešavanje problema diskretne optimizacije kod koje postoji uslov da neke od promenljivih (možda i sve) budu celobrojne. Kod ove metode se relaksacija potproblema vrši dodavanjem određenog broja novih ograničenja, tzv. odsecajućih površi (engl. cutting planes). Svaka od odsecajućih površi mora biti izabrana tako da sva dopustiva rešenja polaznog problema budu takođe dopustiva rešenja novog problema. Pored dodavanja novih ograničenja, metodu karakteriše i grananje na osnovu vrednosti određenih promenljivih kojim se polazni problem deli na dva ili više jednostavnijih problema.

Gomorijeva metoda, poznata i kao Gomorijeva metoda odsecanja, primenjuje se samo za one probleme diskretne optimizacije kod kojih postoji uslov celobrojnosti promeljivih. Kod ove metode se relaksacija potproblema vrši tako što se izostavi uslov celobrojnosti promenljivih. Takav problem se rešava dualnom simpleks metodom, pri čemu svaki put kada se njegovim rešavanjem dođe do necelobrojnog rešenja, treba dodati tzv. Gomorijev rez, tj. novo ograničenje na početni problem. Ovim postupkom se odsecaju necelobrojna rešenja i problem se dalje rešava dok sve krajnje vrednosti ne budu celobrojne.

Grananje i vrednovanje (engl. Branch-and-Price) je egzaktna metoda kombinatorne optimizacije za rešavanje problema celobrojnog linearnog programiranja i mešovitog celobrojnog linearnog programiranja. Predstavlja hibridizaciju metode grananja i ograničavanja i metode generisanja kolona (engl. column generation). Svakom čvoru stabla pretrage zadatog problema se mogu dodati kolone relaksaciji linearnog programiranja (LP relaksacija). Na početku algoritma, skupovi kolona se isključuju iz LP relaksacije da bi se smanjili računski i vremenski zahtevi, a zatim se po potrebi kolone dodaju nazad u LP relaksaciju. Ideja potiče od toga da će za složene probleme većina kolona biti nebazična (nebazisna) i da će im odgovarajuća promenljiva biti jednaka nuli u bilo kom optimalnom rešenju. Dakle, većinu kolona nije potrebno rešavati da bi se došlo do rešenja problema.

1.3.2 Približne metode

Klasi približnih metoda (engl. Approximate Methods) pripadaju aproksimativni (engl. Approximation Algorithms) i heuristički algoritmi. Za razliku od heuristika, koje obično dolaze do kvalitetnih rešenja za prihvatljivo vreme izvršavanja, aproksimativni algoritmi pružaju dokazivost kvaliteta rešenja i granice za vreme rada.

Kažemo da algoritam ima aproksimativni faktor ϵ ako je njegova vremenska složenost polinomijalna i ako se za svaki ulaz generiše rešenje a takvo da je

$$\begin{aligned} a \leq \epsilon \cdot s, \quad \epsilon > 1, \\ \epsilon \cdot s \leq a, \quad \epsilon < 1, \end{aligned} \tag{1.3}$$

gde je s globalni optimum, a faktor ϵ može biti neka kontanta ili funkcija koja zavisi od dimenzije ulaza. Nazivamo ga i relativna performansa garancije. ϵ–aproksimativni algoritam ima apsolutnu performansu garancije ili graničnu grešku ϵ, ako je dokazano da važi:

$$(s - \epsilon) \leq a \leq (s + \epsilon). \tag{1.4}$$

Jedna od najpoznatijih klasa aproksimativnih algoritama je PTAS klasa, gde svaki problem može biti aproksimiran bilo kojim faktorom većim od 1. Formalno rečeno, problem pripada klasi šema aproksimacije polinomijalnog vremena (engl. Polynomial-Time Approximation Scheme, PTAS) ako postoji $(1+\epsilon)$–aproksimativni algoritam polinomijalnog vremena za svako fiksirano $\epsilon > 0$.

Glavni cilj aproksimativnih algoritama je pronalaženje dovoljno bliskih donjih i gornjih granica optimalnog rešenja. Takođe, ovi algoritmi pružaju uvid u težinu problema koji se rešava i time pomažu u dizajniranju efikasnih heuristika. Međutim, njihova implementacija zavisi od konkretnog problema, čime se ograničava njihova primena. Još jedna mana je što se u praksi često dobija aproksimacija koja je prilično daleko od optimalnog rešenja.

Heuristike su se poslednjih decenija naglo razvile i igraju važnu ulogu u rešavanju složenih optimizacionih problema u oblasti biznisa, trgovine, inženjestva, industrije i mnogim drugim [60]. Pojavile su se prvi put tokom osamdesetih godina i njihov uspeh u praksi je doveo ne samo do razvoja čitavog niza najrazličitijih metoda, već i do njihovog daljeg teorijskog ispitivanja kako bi se objasnila njihova efikasnost. Ovakve metode podrazumevaju određeni nivo apstrakcije i daju opšta uputstva kako treba rešiti neki problem, ali ne garantuju optimalnost dobijenog rešenja. Konkretna interpretacija heuristike će zavisiti od strukture i specifičnosti problema koji se rešava.

1.3.3 Heurističke i metaheurističke metode

Heuristika implementirana za jedan problem često nije primenjiva za rešavanje drugih problema, tj. za dati problem se projektuje i implementira konkretna heuristika. Sa druge strane, metaheuristike su tehnike koje su nezavisne i mogu se primeniti na širok spektar optimizacionih problema. Delimo ih na S-metaheuristike (engl. Single-solution based), zasnovane na iterativnom poboljšanju jednog rešenja, i P-metaheuristike (engl. Population based), zasnovane na iterativnom poboljšanju skupa rešenja. Postoji još podela metaheuristika na osnovu različitih kriterijuma: evolutivni ili neevolutivni algoritmi, metaheuristike koje koriste ili ne koriste memoriju radi pamćenja rezultata tokom pretraživanja, deterministički ili stohastički algoritmi, iterativni ili pohlepni (konstruktivni) algoritmi. Heuristike i metaheuristike se izvršavaju iterativno sve dok se ne ispuni neki od unapred zadatih kriterijuma zaustavljanja, na primer dostignut maksimalan broj iteracija, dostignuto maksimal-

no vreme izvršavanja, dostignut određeni kvalitet rešenja i drugi. Sve podele, kao i više detalja, mogu se naći u [66] i [29]. U nastavku ce biti dat kratak pregled nekoliko najpoznatijih metaheuristika.

Lokalno pretraživanje

Lokalno pretraživanje (engl. Local Search, LS) je jedna od najstarijih i najjednostavnijih metoda za rešavanje problema optimizacije. Osnovna ideja ove metode je iterativno pretraživanje različitih okolina trenutnog rešenja u cilju nalaženja globalnog optimuma. Definisanjem neke strukture okolina u prostoru dopustivih rešenja, za svako $x \in X$ dobija se podskup $N(x) \subseteq X, x \notin N(x)$, čije članove zovemo susedima rešenja x.

Na početku algoritma se na slučajan ili neki drugi način generiše početno rešenje x i izračuna njegova vrednost funkcije cilja. Zatim se ažurira trenutno najbolje rešenje x^* sa $x^* := x$ i vrednost najboljeg rešenja x^* se inicijalizuje na vrednost početnog, tj. $f(x^*) = f(x)$. Potom se algoritam iterativno ponavlja, gde se u svakom koraku razmatra rešenje x' (dobijeno ili po nekom unapred zadatom kriterijumu ili na slučajan način) u okolini x^*. Ukoliko je njegova vrednost funkcije cilja bolja od vrednosti funkcije cilja trenutno najboljeg rešenja, odnosno ako važi $f(x') < f(x^*)$, ažurira se trenutno najbolje rešenje kao i najbolja vrednost funkcije cilja, tj. $x^* := x'$ i $f(x^*) = f(x')$. Algoritam se ponavlja sve dok ne bude ispunjen neki od unapred zadatih kriterijuma zaustavljanja.

Mana ove metode je često zaglavljivanje u lokalnom optimumu, što onemogućava nalaženje globalnog optimuma. Iz tog raloga se LS najčešće koristi kao deo neke druge metaheuristike. Takođe, razvijene su razne modifikacije kojima se prevazilazi ovaj nedostatak. Jedna mogućnost je da se na kontrolisan način dozvoljavaju prelasci u lošija rešenja (simulirano kaljenje), a druga da se ne dozvoljavaju prelasci u rešenja koja su već posećena tokom pretrage ili ne vode ka boljim rešenjima (tabu pretraživanje). Dodatno poboljšanje se može dobiti ukoliko se lokalna pretraga u svakoj iteraciji primenjuje na pametno izabrano početno rešenje koje se razlikuje dovoljno od trenutnog lokalnog minimuma. Ukoliko se generiše više početnih rešenja na slučajan način i nad svakim primeni LS, onda se radi o multistartnoj lokalnoj pretrazi (engl. Multistart Local Search, MLS). Ako se na kraju MLS iteracije izvrši perturbacija rešenja i taj rezultat prosledi kao početno rešenje sledeće MLS iteracije, dobija se iterativna lokalna pretraga (engl. Iterated Local Search, ILS). Detaljnije o lokalnom pretraživanju može se naći u [1].

Simulirano kaljenje

Simulirano kaljenje (engl. Simulated Annealing, SA) je S-metaheuristika koja koristi principe lokalnog pretraživanja i predstavlja kombinaciju determinističkog i stohastičkog pristupa. Pod time se podrazumeva da tačka koja se generiše kao slučajni sused u okolini trenutnog rešenja može da se prihvati sa izvesnom verovatnoćom kao novo trenutno rešenje, čak i u slučaju pogoršanja vrednosti funkcije cilja.

Ova metoda je prvi put predložena u radu [43] i nastala je kao simulacija procesa kaljenja čelika. Na početku algoritma se definiše okolina rešenja. Zatim se na slučajan ili neki drugi način generiše početno rešenje x i izračuna njegova vrednost funkcije cilja. Vrednost najboljeg rešenja x^* se najpre inicijalizuje na vrednost početnog, tj. $x^* := x$, i vrši se ažuriranje $f(x^*) = f(x)$. Zatim se algoritam iterativno ponavlja, gde se u svakom koraku razmatra rešenje x' izabrano na slučajan način u okolini trenutno najboljeg x^*. Ukoliko je njegova vrednost funkcije cilja bolja od vrednosti funkcije cilja trenutno najboljeg rešenja, odnosno ako je $f(x^*) > f(x')$, ažurira se trenutno najbolje rešenje na $x^* := x'$, kao i vrednost najboljeg rešenja $f(x^*) = f(x')$. U slučaju da vrednost funkcije cilja novog rešenja x' nije bolja od vrednosti funkcije cilja trenutno najboljeg rešenja x^*, ono se prihvata sa određenom verovatnoćom p_n.

Vrednost parametra p_n nazivamo verovatnoćom prihvatanja lošijeg rešenja u iteraciji n. Njegova vrednost zavisi od trenutne temperature t_n i najčešće se računa po sledećoj formuli

$$p_n = e^{\frac{f(x_n)-f(x)}{t_n}}. \tag{1.5}$$

Niz vrednosti koje predstavljaju temperature t_1, t_2, ..., t_n je takav da važi $t_1 \geq t_2 \geq$ ... i $\lim_{n\to\infty} t_n = 0$. Od ovog niza zavise brzina i način hlađenja tokom iteracija i nazivamo ga šemom hlađenja. Na početku algoritma temperatura je visoka, što obezbeđuje visok stepen diverzifikacije, tj. pretraživanje većeg dela prostora, čime se izbegava zaglavljivanje u lokalnom optimumu. Hlađenjem se smanjuje verovatnoća prihvatanja lošijeg rešenja, što znači da se pri samom kraju izvršavanja algoritma prihvataju uglavnom ona rešenja sa boljom funkcijom cilja. Algoritam se ponavlja sve dok se ne ispuni unapred zadati kriterijum zaustavljanja. Detaljan opis ove metaheuristike može se naći u [18] i [4].

Tabu pretraživanje

Tabu pretraživanje (engl. Tabu Search, TS) je S-metaheuristika koja se bazira na lokalnom pretraživanju, pri čemu se zaglavljivanje u lokalnim optimumima prevazi-

lazi uvođenjem kratkoročne memorije procesa pretrage i skupa zabranjenih poteza T, koga zovemo tabu lista. Ona podrazumeva pamćenje informacija o prethodnim iteracijama pretraživanja u cilju sužavanja izbora novog rešenja iz unapred definisane okoline tekućeg rešenja. Osnovni koncepti ove metode su prvi put predloženi u [28].

Na početku algoritma definiše se struktura okoline i vrši generisanje početnog rešenja x na slučajan način ili nekom drugom metodom uz ažuriranje tabu liste na prazan skup. Vrednost najboljeg rešenja x^* se inicijalizuje na vrednost početnog, tj. vrši se ažuriranje $x^* := x$ i $f(x^*) = f(x)$. Na početno rešenje x se primenjuje lokalna pretraga i dobija novo rešenje x'. Ukoliko je njegova vrednost funkcije cilja bolja od vrednosti funkcije cilja trenutno najboljeg rešenja x^*, ažurira se x^* na $x^* := x'$. Takođe, pronađeno bolje rešenje se ubacuje u tabu listu, kako bi bilo isključeno iz okoline u kojoj se vrši dalja pretraga u narednih nekoliko iteracija. Potom se dalje primenjuje lokalna pretraga na modifikovanu okolinu.

Performanse tabu pretraživanja dosta zavise od ulaznih parametara. Jedan od parametara ove metaheuristike je dužina tabu liste T od koje zavisi efikasnost algoritma. Loš izbor dužine tabu liste dovodi ili do pojačane diverzifikacije, (ukoliko se izabere predugačka tabu lista) ili do pojave ciklusa (ako je tabu lista suviše mala). Još jedan parametar je tzv. tabu vreme, koje predstavlja broj iteracija nakon koga se određeni element briše iz liste i ponovo uključuje u proces lokalnog pretraživanja.

Kako bi se uštedelo na memorijskom prostoru, često se umesto čuvanja celih rešenja u tabu listi čuvaju samo neki njegovi atributi ili potezi koji dovode do tih rešenja. Međutim, ovo dovodi do problema jer se tako zabrane i neka rešenja koja mogu biti od interesa. Ponekad se desi da korišćenje samo kratkoročne memorije ne može obezbediti dovoljan stepen diverzifikacije i intenzifikacije, pa se zato uvode i srednjoročna i dugoročna memorija. Ovakva modifikacija se zove Reaktivno tabu pretraživanje (engl. Reactive Tabu Search).

Genetski algoritam

Genetski algoritam (engl. Genetic Algorithm, GA) je P-metaheuristika koja je prvobitno kreirana da simulira proces biološke evolucije jedne populacije jedinki pod dejstvom različitih faktora okruženja i genetskih operatora, tako da unutar jedne populacije najčešće opstaju najbolje prilagođene jedinke. Prve ideje o ovom algoritmu izložene su u radu [36], i to u okviru teorije adaptivnih sistema.

U procesu prirodne evolucije, svaka jedinka je okarakterisana hromozomom koji

predstavlja njen genetski kod. One jedinke iz populacije koje su u većoj meri prilagođene okruženju međusobno se i dalje reprodukuju (primenom genetskih operatora na njihove hromozome) i tako se stvara nova generacija jedinki, prilagođenija od prethodne. Ovaj proces se ponavlja, pri čemu se iz generacije u generaciju prosečna prilagođenost članova populacije povećava.

Skup rešenja genetskog algoritma se naziva populacija, a jedno rešenje predstavlja jedinku unutar te populacije. Kôd svake jedinke predstavlja skup njenih gena. Kodiranje jedinke zavisi od prirode problema i može biti binarno, celobrojno, realno ili predstavljeno nekim drugim nizom simbola. Za svaku jedinku se definiše i funkcija prilagođenosti, na osnovu koje se određuje vrednost njenog koda.

Nakon inicijalizacije početne populacije, nove generacije jedinki se dobijaju uzastopnom primenom genetskih operatora. Smena generacija se vrši sve dok ne bude ispunjen unapred zadati kriterijum zaustavljanja. Početna generacija može biti generisana na slučajan način ili primenom neke heuristike. Najčešći genetski operatori su selekcija, ukrštanje i mutacija.

Selekcija predstavlja izbor jedinki iz trenutne populacije koje će biti korišćene za dobijanje naredne generacije. Jedinke mogu biti izabrane na osnovu njihove vrednosti funkcije prilagođenosti, pri čemu se jedinkama sa boljom funkcijom prilagođenosti daje veća verovatnoća da budu izabrane. Ovakva selekcija se često naziva i ruletska selekcija, gde se pretpostavlja da na krugu veći isečak pripada jedinkama sa većom funkcijom prilagođenosti. U literaturi se često koristi i turnirska selekcija. Kod nje se na slučajan način bira nekoliko jedinki, nakon čega se organizuje tzv. turnir, čiji je pobednik jedinka sa najboljom funkcijom prilagođenosti.

Ukrštanje predstavlja kombinovanje gena dva roditelja, pri čemu se dobijaju dva nova potomka. U literaturi se najčešće javljaju jednopoziciono, dvopoziciono i uniformno ukrštanje. Kod jednopozicionog se uniformno bira gen i, gde prvi potomak nasleđuje gene prvog roditelja od prvog do i-tog i gene drugog roditelja od $i + 1$ do poslednjeg, a drugi potomak gene drugog roditelja od prvog do i-tog i prvog roditelja od $i + 1$ do poslednjeg. Kod dvopozicionog ukrštanja se uniformno biraju dva gena, i i j, takvi da je $i < j$. Prvi potomak nasleđuje gene pre i-tog i nakon j-tog od prvog roditelja, a između gena i i j od drugog roditelja. Nasuprot tome, drugi potomak nasleđuje gene pre i-tog i nakon j-tog od drugog roditelja, a gene između i-tog i j-tog od prvog roditelja. Kod uniformnog ukrštanja, svaki gen se bira od oba roditelja sa jednakom verovatnoćom $p = 0.5$. U nekim slučajevima je moguće definisati i vektor koji će za svaki gen sadržati verovatnoću (koja može biti različita

za svaki gen) da će baš on biti nasleđen od prvog roditelja. Ovakvo ukrštanje se naziva p–uniformno ukrštanje, a odgovarajući vektor se naziva maska.

Mutacija predstavlja slučajnu promenu nekih gena određene jedinke. Ona se na svaku jedinku primenjuje sa unapred zadatom malom verovatnoćom. Ukoliko je verovatnoća mutacije premala, ili se uopšte ne primenjuje, može doći do preuranjene lokalne konvergencije. Ako je vrednost velika, genetski algoritam će prostor rešenja pretraživati bez jasnog fokusa. U literaturi se najčešće javljaju prosta, normalna i eksponencijalna mutacija. Kod proste mutacije se svaki gen posmatra posebno i menja sa malom verovatnoćom p_m. U slučajevima kada geni nisu međusobno ravnopravni, tj. kada je neke gene potrebno mutirati sa većim ili manjim nivoom mutacije, koristi se normalna mutacija sa normalnom raspodelom ili eksponencijalna mutacija, kod koje broj gena u jedinki koji mutiraju eksponencijalno raste.

Iako je pretpostavka da će kvalitetni geni sa velikom verovatnoćom preći u narednu generaciju, to ne mora uvek da bude slučaj. Tako se može desiti da neki potomci budu lošiji od roditelja ili čak i da najbolja jedinka ne pređe u narednu generaciju. Jedan od najlošijih ishoda je da se u nekom trenutku dostigne rešenje koje je dobrog kvaliteta, a da se zatim izgubi primenom operatora ukrštanja i mutacije. Zato se često primenjuje princip elitizma, koji se zasniva na ideji da se najbolja ili više najboljih jedinki prenesu direktno u narednu generaciju.

Optimizacija mravljim kolonijama

Optimizacija mravljim kolonijama (engl. Ant Colony Optimization, ACO) je P-metaheuristika koja je prvi put predložena u radu [12] i spada u klasu konstruktivnih metaheuristika. Osnovna ideja ove metode je razvijanje kolektivnog ponašanja kolonije mrava u procesu pronalaženja hrane sa ciljem obavljanja složenih zadataka, kao što su transport i nalaženje najkraćeg puta do hrane.

Kretanje mrava se obavlja pomoću jednostavnog mehanizma komunikacije baziranom na određenoj količini hemijske supstance, koju svaka jedinka kolonije ostavlja za sobom. Ovakav trag se naziva feromon. Svaki mrav sa određenom verovatnoćom preferira praćenje putanje na kojoj postoji jači sloj feromona. Na primer, kada naiđu na prepreku, mravi slučajno biraju način da je zaobiđu tako što skreću ulevo ili udesno. U početku će oko polovina mrava preferirati jednu, a druga polovina drugu stranu. Mravi koji krenu kraćim putem će brže formirati jači trag feromona nego oni koji idu dužim putem. Time će sve više i više mrava krenuti za jačim feromonskim tragom. Iako jači trag privlači više mrava, uvek ima onih koji u međuvremenu idu

svojim putem bez obzira na trag i tako istražuju nove putanje. Feromonski trag se postepeno gubi usled isparavanja, ali ga novi mravi istovremeno i pojačavaju. Pravci koji se manje koriste konačno se eliminišu, budući da feromon koji je na njih položen brže isparava. Više detalja o ovoj metaheuristici može se naći u [63] i [15].

U početnoj fazi potrebno je odrediti matricu feromona $L = [\lambda_{ij}]$, čiji su elementi unapred zadate početne vrednosti feromona za svaku komponentu rešenja, kao i parametar koji predstavlja broj mrava u koloniji. Vrednost λ_{ij} predstavlja poželjnost i-te komponente rešenja na j-tom mestu. Algoritam se sastoji iz dva osnovna koraka – konstrukcija rešenja i ažuriranje tragova feromona. Svaki mrav se posmatra kao jedna stohastička pohlepna struktura i konstruiše se iterativno dodavanjem odgovarajuće komponente rešenja na već izgrađeno parcijalno rešenje, koja se bira sa određenom verovatnoćom u zavisnosti od odgovarajućih vrednosti trenutne matrice feromona. Karakteristika feromona je da čuva dobra parcijalna ili kompletna rešenja, čime doprinosi konstrukciji kvalitetnijih rešenja.

Ažuriranje tragova feromona se sastoji iz dve faze – evaporacije i pojačavanja (osvežavanja) tragova feromona. Faza evaporacije je prirodni proces isparavanja feromona i ona podrazumeva promenu svake komponente matrice feromona primenom sledeće formule:

$$\lambda_{ij} = (1 - \rho)\lambda_{ij}, \quad \rho \in (0, 1], \tag{1.6}$$

gde je parametar ρ izabran na slučajan način iz intervala $(0, 1]$ i on najviše utiče na stepen diverzifikacije algoritma.

U fazi pojačavanja (osvežavanja) se na osnovu konstruisanog rešenja (delimičnog ili potpunog) jednog ili više mrava pojačava feromon dodavanjem neke vrednosti, koja zavisi od kvaliteta kontruisanog (kompletnog ili parcijalnog) rešenja. Strategija pojačavanja feromona zavisi od konkretnog problema koji se rešava i postoje razne strategije. Jedna od mogućih strategija je da svaki mrav osvežava trag feromona λ_{ij} u svakom koraku konstrukcije svog rešenja proporcijalno kvalitetu parcijalnog rešenja koje je već konstruisao (engl. Online step-by-step feromone update). Takođe, svaki mrav može da osveži trag feromona λ_{ij} tek kada konstruiše svoje kompletno rešenje proporcijalno kvalitetu kontruisanog rešenja (engl. Online delayed feromone update). Postoji i strategija kod koje se trag feromona λ_{ij} osvežava tek kada svi mravi generišu kompletno rešenje (engl. Offline feromone update).

Koraci konstrukcije rešenja i ažuriranja tragova feromona se smenjuju sve dok ne bude ispunjen neki od unapred zadatih kriterijuma zaustavljanja.

Optimizacija kolonijom pčela

Optimizacija kolonijom pčela (engl. Bee Colony Optimizaton, BCO) je P-metaheuristika, takođe konstruktivnog tipa, iz klase algoritama zasnovanih na inteligenciji roja, čije su prve ideje potekle iz posmatranja ponašanja pčela u potrazi za hranom i iznete su u radu [53]. Pčele u prirodi nasumično traže hranu i donose uzorke nektara u košnicu, gde se ispituje njegov kvalitet. Pčele koje ukazuju na pravac i daljinu izvora kvalitetnog nektara plešu u obliku broja osam (engl. waggle dance) i tako dele informaciju o lokaciji hrane sa ostalim pčelama u košnici. Pravac u kome sve pčele nastavljaju svoju potragu za hranom određen je na osnovu prikupljenih informacija.

Svaka veštačka pčela u BCO algoritmu predstavlja jedno rešenje. Prvi korak se sastoji od inicijalizacije kolonije u kojoj se svakoj pčeli dodeli početno (prazno) rešenje. Tokom iterativnih koraka naizmenično se primenjuju dve faze – let unapred i let unazad. Najbolje rešenje pronađeno u prvoj iteraciji se pamti, a proces traganja za boljim rešenjem pčele nastavljaju iterativno.

Neka parametar B predstavlja broj veštačkih pčela, a NC broj međusobnih interakcija svake pčele sa ostatkom roja. Tokom leta unapred, na svakom koraku u, $u = 1, 2, ..., NC$, svaka pčela b, $b = 1, 2, ..., B$ razmatra moguće poteze i odabira sledeći pravilom ruletske selekcije. Verovatnoća odabira je veća za one poteze koji vode do boljih rešenja. Potom se ocenjuje kvalitet dobijenog rešenja i pčela odlučuje da li da odustane od rešenja do kojeg je došla poslednjim letom unapred i postane neopredeljena ili da ostane lojalna tom rešenju. Ovu fazu zovemo letom unazad. Lojalnost pčele b svom rešenju se određuje na osnovu verovatnoće koja se računa po formuli:

$$p_b^{u+1} = e^{-\frac{O_{\max} - O_b}{u}}, \tag{1.7}$$

gde je O_b normalizovana vrednost funkcije cilja rešenja određenog pčelom b, a $O_{\max}$ maksimalna normalizovana vrednost funkcije cilja svih rešenja. Ako se rešava problem minimizacije funkcije, normalizovana vrednost funkcije cilja se računa kao

$$O_b = \frac{C_{\max} - C_b}{C_{\max} - C_{\min}}, \quad b = 1, 2, ..., B, \tag{1.8}$$

gde je C_b vrednost funkcije cilja određeno pčelom b, a $C_{\max}$ i $C_{\min}$ su maksimalna i minimalna vrednost funkcije cilja čitavog roja. Iz ove formule se može videti da će kvalitetnija rešenja imati i bolju normalizovanu funkciju, čime će i verovatnoća pčele da ostane lojalna svom rešenju biti veća. Pčele koje ostaju lojalne svom rešenju

nazivamo regruteri. Ako imamo R pčela regrutera, neopredeljena pčela će izabrati rešenje b pravilom ruletske selekcije

$$p_b = \frac{O_b}{\sum_{k=1}^{R} O_k}, \tag{1.9}$$

što znači da kvalitetnija rešenja imaju veću verovatnoću da budu izabrana. Posle NC izvršenih letova unapred i unazad se ažurira najbolje rešenje i koraci se iterativno ponavljaju sve dok se ne ispuni neki od unapred zadatih kriterijuma zaustavljanja. Detaljnije o samoj metodi se može naći u radovima [54] i [67].

Optimizacija rojevima čestica

Optimizacija rojevima čestica (engl. Particle Swarm Optimization, PSO) je P-metaheuristika zasnovana na inteligenciji roja. Ideja je razvoj kolektivne inteligencije koja se dešava usled međusobne interakcije i razmenjivanja znanja svake jedinke međusobno. Ova metoda je prvi put predložena u radu [42], dok se više detalja može naći u [38] i [62].

Algoritam radi nad skupom jedinki, koji se naziva rojem. Elementi ovog skupa se nazivaju česticama i svaka od njih odgovara jednom potencijalnom rešenju. Čestice se na unapred definisan način kreću po pretraživačkom prostoru pokušavajući da poprave svoje pozicije (rešenja). Svaka čestica $i = 1, 2, ..., N$ iz roja sadrži sledeće informacije:

- trenutna pozicija čestice, u oznaci $X_i = (x_{i_1}, x_{i_2}, ..., x_{i_d})$;
- brzina, tj. gradijent ili pravac u kome bi se čestica kretala bez drugih uticaja, u oznaci $V_i = (v_{i_1}, v_{i_2}, ..., v_{i_d})$;
- najbolja pozicija čestice (engl. local best solution), u oznaci $p_i = (p_{i_1}, p_{i_2}, ..., p_{i_d})$;
- najbolje rešenje celog roja (engl. global best solution) ili najbolje rešenje okoline čestice roja (engl. neighborhood best solution), u oznaci $p_g = (p_{g_1}, p_{g_2}, ..., p_{g_d})$.

U fazi inicijalizacije se generiše početni roj, i to obično na slučajan način. U svakoj iteraciji se ažuriraju vektori brzina i položaja čestica, najbolja vrednost rešenja za svaku česticu, kao i za roj u celini. Naredna pozicija čestice zavisi od njenog trenutnog vektora brzine, iskustva same čestice i iskustva ostalih čestica. Formula po kojoj se vrši ažuriranje vektora brzine čestice roja i u iteraciji j je

$$v_{ik}^j = w v_{ik}^{j-1} + c_1 r_1 (p_{ik}^j - x_{ik}^j) + c_2 r_2 (p_{gk}^j - x_{gk}^j), \quad k = 1, ..., d, \tag{1.10}$$

gde w nazivamo parametar inercije, koji služi da bi se kontrolisao uticaj prethodne brzine čestice na njenu trenutnu brzinu. Faktor kognitivnog učenja, u oznaci c_1, kontroliše uticaj iskustva čestice. Faktorom socijalnog učenja, u oznaci c_2, kontroliše se uticaj iskustva okolnih čestica. Parametri r_1 i r_2 su izabrani uniformnom raspodelom $\mathcal{U}(0,1)$. Formula kojom se vrši promena položaja čestice i u iteraciji j je

$$x_{ik}^{j+1} = x_{ik}^{j} + v_{ik}^{j}, \quad k = 1, ..., d. \tag{1.11}$$

Zbog potencijalnog izlaska vrednosti v_{ik}^{j} iz prostora dopustivih rešenja, ove vrednosti se ograničavaju na unapred definisan opseg $[V_{\min}, V_{\max}]$. Stepen intenzifikacije i diverzifikacije zavisi od izbora parametra w – za veće vrednosti w imaćemo jaču diverzifikaciju, a za manje vrednosti intenzifikaciju pretrage trenutne oblasti. Proces se ponavlja dok se ne ispuni unapred zadati kriterijum zaustavljanja.

Metoda promenljivih okolina

Metoda promenljivih okolina (engl. Variable Neighborhood Search, VNS) je S-metaheuristika koja predstavlja uopštenje lokalne pretrage i jedna je od najefikasnijih metoda za rešavanje problema kombinatorne i kontinualne optimizacije. Prvi put je predložena u radu [57] i bazira se na sledećim principima:

- Lokalni optimum za jedan tip okoline ne mora nužno da bude lokalni optimum za drugi tip okoline.
- Globalni optimum predstavlja lokalni optimum za sve tipove okolina.
- Za mnoge probleme u praksi, lokalni optimumi u više tipova okolina su relativno blizu jedan drugog.

U literaturi (npr. [33] i [34]) postoji više različitih verzija VNS metode, među kojima se najčešće sreću redukovana, osnovna, uopštena i iskošena metoda promenljivih okolina.

Redukovana metoda promenljivih okolina (engl. Reduced Variable Neighborhood Search, RVNS) je stohastička metoda zbog izostavljene lokalne pretrage, pri čemu se umesto lokalnog minimuma bira proizvoljno rešenje iz odgovarajuće okoline trenutno najboljeg rešenja u cilju traženja poboljšanja. Karakteriše je kraće vreme izvršavanja ali i manji kvalitet dobijenog rešenja. Na početku algoritma, početno rešenje se generiše na slučajan način (ili nekom drugom metodom) i dodeljuje se trenutno najboljem rešenju x^*. Takođe, biraju se i okoline $N_k, k = 1, 2, ..., k_{\max}$ i k se postavlja

na 1. U svakoj iteraciji izabere se $x' \in N_k(x^*)$ na slučajan način i ispituje se da li je ta tačka bolja od tekućeg rešenja. Ukoliko jeste, x' postaje trenutno najbolje rešenje i pretraživanje se nastavlja u njegovoj prvoj okolini, tj. u $N_1(x')$. U suprotnom, k se ažurira na $k+1$ i nastavlja se pretraga u široj okolini nepromenjenog trenutnog rešenja $N_{k+1}(x^*)$. U slučaju da k postane veće od $k_{\max}$, njegova vrednost se ponovo postavlja na 1 i opisani postupak se ponavlja sve dok se ne ispuni jedan od unapred zadatih kriterijuma zaustavljanja.

Osnovna metoda promenljivih okolina (engl. Basic Variable Neighborhood Search, BVNS) predstavlja kombinaciju determinističkog i stohastičkog pristupa da bi se napravio balans između stepena diverzifikacije i intenzifikacije. U fazi inicijalizacije, na slučajan način bira se početno rešenje, koje se dodeljuje trenutno najboljem rešenju x^*, vrši se odabir okolina kao i kod redukovane metode promenljivih okolina, i k se postavlja na 1. Ovu metodu karakterišu dve faze – faza razmrdavanja (engl. shaking) i faza lokalne pretrage. Fazom razmrdavanja počinje svaka nova iteracija algoritma, gde se na slučajan način bira rešenje $x' \in N_k(x^*)$, nakon čega se primenjuje neki tip lokalne pretrage, počevši od rešenja x'. Lokalna pretraga ne mora da koristi istu okolinu kao faza razmrdavanja. Ukoliko je vrednost funkcije cilja za rešenje x'' dobijeno lokalnom pretragom bolje u odnosu na vrednost funkcije cilja trenutno najboljeg rešenja x^*, vrši se ažuriranje x^* na $x^* := x''$, k se postavlja na 1 i nova iteracija počinje fazom razmrdavanja u okolini $N_1(x^*)$. U suprotnom, vrednost k se postavlja na $k+1$ i algoritam se nastavlja pretragom u narednoj okolini nepromenjenog trenutno najboljeg rešenja $N_{k+1}(x^*)$. Kada se desi $k > k_{\max}$, vrednost k se vrati na 1 i pretraga se nastavlja u okolini $N_1(x^*)$. Iteracije se izvršavaju sve dok se ne ispuni unapred zadati kriterijum zaustavljanja.

Uopštena metoda promenljivih okolina (engl. General Variable Neighborhood Search, GVNS) dobija se iz osnovne metode promenljivih okolina zamenom lokalne pretrage metodom promenljivog spusta. Metoda promenljivog spusta (engl. Variable Neighborhood Descent, VND) je deterministička varijanta VNS metode. Imajući u vidu činjenicu da lokalni optimum u odnosu na jednu okolinu ne mora biti lokalni optimum u odnosu na neku drugu, cilj VND metode je naći rešenje koje će biti lokalni optimum u odnosu na sve okoline iz unapred definisanog skupa okolina. Takođe, za GVNS mogu se koristiti dva tipa okolina – $N_k, k = 1, 2, ..., k_{\max}$ za fazu razmrdavanja i $N_l, l = 1, 2, ..., l_{\max}$ za fazu lokalne pretrage.

Iskošenu metodu promenljivih okolina (engl. Skewed Variable Neighborhood Search, SVNS) karakteriše pretraga prostora koji su daleko od tekućeg rešenja, tako

što se dozvoljava i prihvatanje lošijih rešenja pod određenim uslovima. Ideja potiče iz činjenice da kada se nađe najbolje rešenje u velikoj okolini, potrebno je otići prilično daleko da bi se dobilo bolje rešenje. Upravo iz tog razloga se ova metoda najčešće koristi za optimizacione probleme kod kojih su lokalni optimumi na velikoj udaljenosti jedni od drugih. Na početku algoritma treba definisati rastojanje $\rho(x, x')$ između dva rešenja i parametar α, kojim se kontroliše prihvatanje lošijih rešenja u pogledu funkcije cilja. Da bi se izbeglo često prihvatanje rešenja bliskih trenutno najboljem, parametar α bi trebalo da ima veću vrednost kada je rastojanje ρ malo. Jedina razlika u odnosu na ostale algoritme VNS-a je u situaciji kada se lokalnom pretragom dođe do rešenja koje zadovoljava $f(x'') - \alpha \cdot \rho(x^*, x'') < f(x^*)$, gde je sa x^* označeno trenutno najbolje rešenje, a sa x'' rezultat lokalne pretrage. U ovom slučaju x'' se prihvata kao trenutno najbolje rešenje i u zavisnosti α, pretraga se nastavlja u odgovarajućoj okolini novog rešenja.

Iako je najčešće korišćena za probleme kombinatorne optimizacije, metoda promenljivih okolina se u poslednjih decenija uspešno primenjuje i na probleme kontinualne optimizacije. Prvi kontinualni problem na kome je korišćena VNS metoda poznat je u literaturi kao Višeizvorni Veberov problem (engl. Multisource Weber Problem) ili Lokacijsko-alokacijski problem (engl. Location-Alocation Problem) [7, 6]. Drugi problem kontinualne optimizacije za čije rešavanje je primenjen VNS je Uopšteni problem bilinearnog programiranja (engl. General Bilinear Programming Problem) [31]. Oba navedena problema jesu kontinualna, međutim, za njihovo rešavanje je uzet pristup iz kombinatorne optimizacije.

U radu [58] je VNS prvi put primenjen bez elemenata diskretizacije, gde je rešavan nelinearan problem bez ograničenja za dizajn rasutog spektra polifaznog radara (engl. Spread Spectrum Radar Polyphase Code Design Problem). Nekoliko godina kasnije, razvijen je i softverski paket GLOB koji koristi VNS metodu za rešavanje kontinualnih problema nelinearne globalne optimizacije [46, 56]. GLOB je usavršen u radovima [20, 52, 55, 19] tako što je dodato više različitih varijanti VNS-a. Za mnoge probleme, kako diskretne, tako i kontinualne optimizacije, VNS se pokazao kao najuspešnija metoda u odnosu na druge heuristike [46, 32, 58].

Gausovska metoda promenljivih okolina

Gausovska metoda promenljivih okolina (engl. Gaussian Variable Neighborhood Search, GaussVNS), izložena prvi put u radu [9], predstavlja modifikaciju metode

promenljivih okolina za rešavanje problema kontinualne optimizacije. Ukoliko se problem (1.1), gde je $S = \mathbb{R}^n$, rešava klasičnim VNS pristupom, pošto je broj $k_{\max}$ konačan, ne bi bilo moguće polazeći od trenutno najbolje tačke dostići svaku tačku iz dopustivog skupa i zbog toga se možda neće ni doći do oblasti gde se nalazi globalni minimum. Zbog toga se često u praksi posmatra sledeći nelinearan problem globalne kontinualne optimizacije bez ograničenja

$$global \quad \min_{x \in S} f(x), \tag{1.12}$$

gde je $f : \mathbb{R}^n \to \mathbb{R}$ neprekidna funkcija, koja u opštem slučaju ne mora biti ni konveksna ni glatka, i $S = \{x \in \mathbb{R}^n | a_i \leq x_i \leq b_i, i = 1, 2, ..., n\}$.

Ovakvi problemi često imaju veliki broj lokalnih minimuma, čiji broj može eksponencijalno da raste ukoliko se poveća dimenzija problema n. Takođe, drugi problem se javlja čak i za rešavanje problema manjih dimenzija, za koje je potrebna velika količina vremena da bi se došlo do rešenja. U opštem slučaju, kod problema kontinualne optimizacije, moguće je konstruisati aproksimativne algoritme koji će sa nekom unapred zadatom greškom ε naći rešenje, koje je aproksimacija tačnog rešenja.

Kao što je već pomenuto ranije, VNS metoda je bazirana na sistematskoj promeni unapred definisanih okolina $N_1(x), N_2(x), ..., N_{k_{\max}}(x)$ trenutno najboljeg rešenja x u cilju izbegavanja zaglavljivanja u lokalnim optimumima. Prilikom izbora okolina $N_k, k = 1, 2, ..., k_{\max}$, neophodno je definisati izbor geometrije $\mathcal{G}$ strukture okolina. Jedan od uslova koji okoline treba da ispunjavaju je da njihova unija pokriva ceo skup S, kako bi se dobilo bilo koje dopustivo rešenje x tog skupa. Najčešće se okoline definišu na sledeći način:

$$N_k(x) = \{y | \rho(x, y) \leq \rho_k\} \tag{1.13}$$

ili

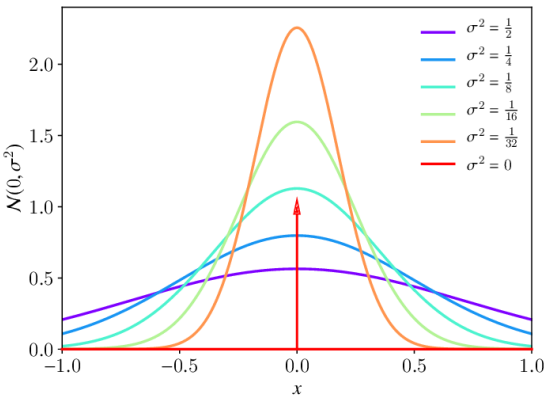
$$N_k(x) = \{y | \rho_{k-1} < \rho(x, y) \leq \rho_k\}, \tag{1.14}$$

gde je $\rho(\cdot, \cdot)$ funkcija rastojanja, tj. metrika u prostoru $\mathbb{R}^n \times \mathbb{R}^n$. Za metriku se obično uzima neko l_p rastojanje između dve tačke, gde je $1 \leq p \leq \infty$. Na ovaj način, geometrija okolina je određena izborom metrike $\rho(\cdot)$, a sama okolina $N_k(x)$ je određena sa $\mathcal{G}$ i ρ_k. Brojeve ρ_k nazivamo radijusima i oni se menjaju uvećavanjem k, čije se vrednosti izračunavaju u procesu optimizacije ili se unapred definišu.

Ukoliko bismo želeli da pretražujemo šire okoline, morali bismo izabrati veće $k_{\max}$. Neretko se za $N_{k_{\max}}$ uzima okolina takva da obuhvati ceo prostor pretrage S. Međutim, tako definisane okoline prouzrokuju dosta manju efikasnost metode.

Nalaženje slučajne tačke x' iz okoline $N_k(x)$ u fazi razmrdavanja zavisi od definisane geometrije okolina, ali i od raspodele slučajnih brojeva P, koja se koristi za generisanje slučajne tačke. Najčešće se u literaturi koristi uniformna raspodela. Međutim, u radu [9] je pokazano da generisanje slučajnih tačaka ovom raspodelom ne mora uvek biti najbolji izbor. Na osnovu Centralne granične teoreme, odnosno činjenice da niz slučajnih veličina (pod određenim pretpostavkama) konvergira ka slučajnoj promenljivoj koja ima normalnu raspodelu, pojavila se ideja za nastanak GaussVNS metode.

Da bi se prevazišle navedene mane klasičnog VNS pristupa, u fazi razmrdavanja GaussVNS algoritma su sve okoline jednake celom prostoru, pri čemu se odabir slučajnih tačaka vrši na osnovu unapred definisanog niza raspodela verovatnoća $P_1(x), P_2(x), ..., P_{k_{\max}}(x)$. Pretpostavimo da je $P_k(x)$ višedimenzionalna Gausova raspodela sa očekivanjem x i matricom kovarijansi Σ_k. Drugim rečima, svaki element $P_k(x)$ definisanog niza je n-dimenzionalna normalna raspodela centrirana u tački x. Na taj način teorijski je moguće doći do bilo koje tačke prostora dopustivih rešenja. U slučaju da se za niz P_k izabere uniformna raspodela, dobila bi se klasična VNS metoda. Na slici 1.1 prikazan je grafik krive Gausove raspodele centrirane u nuli za različite vrednosti varijanse.



Slika 1.1: Gausova raspodela za različite vrednosti varijanse σ^2 [45].

U fazi inicijalizacije potrebno je izabrati početno rešenje x na slučajan način, dodeliti ga trenutno najboljem rešenju x^*, izabrati matricu kovarijansi Σ_k, $k = 1, 2, ..., k_{\max}$, i postaviti k na 1. Dokle god je $k \leq k_{\max}$, izvršavaju se naizmenično faza razmrdavanja i faza lokalne pretrage. U fazi razmrdavanja se generiše novo rešenje x' koristeći Gausovu raspodelu slučajnih brojeva sa očekivanjem x^* i

matricom kovarijansi preko formule za funkciju gustine

$$\phi(x') = \frac{1}{(2\pi)^{\frac{n}{2}}\sqrt{|\Sigma_k|}} e^{-\frac{1}{2}(x'-x^*)^T {\Sigma_k}^{-1}(x'-x^*)}, \quad |\Sigma_k| = det\Sigma_k. \tag{1.15}$$

Sledi faza lokalne pretrage nakon čega se ispituje da li je nova dobijena vrednost rešenja bolja od trenutno najboljeg. Ukoliko jeste, vrši se ažuriranje trenutno najboljeg rešenja na novo nađeno rešenje dobijeno nakon lokalne pretrage i vrednost k se postavlja na 1. Inače, k se uveća na $k+1$ i postupak se nastavlja. Ako se desi da je $k > k_{\max}$, k se ponovo postavlja na 1 i algoritam nastavlja sa traženjem boljeg rešenja sve do ispunjenja zadatog kriterijuma zaustavljanja.

Glava 2

Problem minimalnog zbirnog bojenja grafa

2.1 Istorijat bojenja grafova

Problem bojenja grafova (engl. Graph Coloring Problem) spada u važne probleme računarstva i operacionih istraživanja i bio je jedan među prvima za koji se pokazalo da je NP-težak. Teorija bojenja grafova započeta je izučavanjem problema bojenja država na geografskoj mapi, pri čemu svake dve susedne države ne treba da budu obojene istom bojom. Ukoliko bismo države zamenili sa čvorovima grafa i spojili sve susedne države granama, dobili bismo planarni graf. U radu iz 1852. godine, pod nazivom Problem četiri boje, prvi put je izložen opisani problem od strane autora Augustusa de Morgana i Vilijama Rovana Hamiltona, i glasio je ovako:

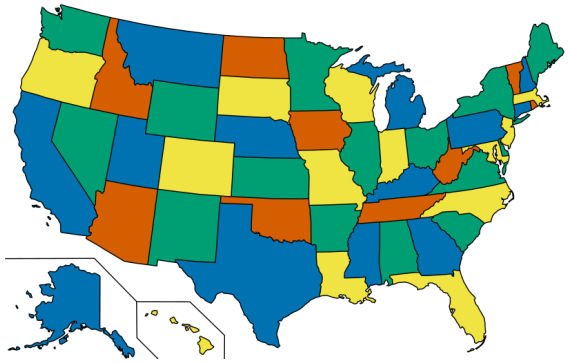
Da li se države na svakoj mapi mogu obojiti sa najviše četiri boje tako da su svake dve države sa zajedničkom granicom obojene različitim bojama?

Problem je ostao nerešen sve do 1879. godine, dok autor Alfred Kempe nije izneo prvi dokaz i dao jako koristan doprinos izloženom problemu. Međutim, čitavu deceniju kasnije, tačnije 1890. godine, Hivud je uočio grešku i uz pomoć Kempeovih argumenata uspeo je da dokaže da se svaka mapa može obojiti sa pet boja. Nakon nekoliko beuzpešnih pokušaja njegovog rešavanja, Problem četiri boje postaje jedan od najpoznatijih problema diskretne matematike dvadesetog veka [69].

Godine 1976. u radu Hakena i Apela javlja se nova formulacija u terminima grafova:

Da li se čvorovi svakog planarnog grafa mogu obojiti sa najviše četiri boje tako da su svaka dva čvora koja su povezana granom obojena različitim bojama?

Tada je prvi put u istoriji priznat dokaz koji uključuje samo rad računara, koji je radeći 1200 sati potvrdio da se može izvesti pravilno bojenje sa četiri boje. Na slici 2.1 je primer jednog takvog bojenja.



Slika 2.1: Mapa Sjedinjenih Američkih Država obojena sa četiri boje.

Bojenje grafova u praksi ima značajnu i široku primenu u različitim problemima iz realnog života, kao što su problemi raspoređivanja (engl. scheduling), rasporeda časova ili ispita (engl. timetabling), upravljanja skladištem (engl. warehouse management), dodele frekvencija u mobilnim mrežama (engl. frequency assignment in cellular networks), alokacije registara u optimizaciji kompajlera (engl. register allocation in compilers), dizajna i rada prilagodljivih proizvodnih sistema itd.

2.2 Osnovni pojmovi

Uvedimo par osnovnih pojmova i definicija o grafovima.

Definicija 2.2.1. Graf G je uređen par $(V(G), E(G))$ (ili samo (V, E)), gde je $V(G)$ konačan neprazan skup elemenata koje nazivamo čvorovima, dok je $E(G)$ jedan skup dvoelementnih podskupova skupa $V(G)$, čije elemente nazivamo granama.

Ukoliko je grana određena čvorovima u i v, tada umesto $\{u, v\}$ možemo pisati uv, ili samo e, gde je $e = \{u, v\}$. Za granu e kažemo još i da spaja čvor u i v, kao i da se u i v krajevi grane e. Takve čvorove u i v zovemo susednim. Skup suseda svakog čvora u datog grafa označavamo sa $N(u)$. Na slici 2.2 su prikazane dve različite geometrijske interpretacije grafa $G = (V, E)$, pri čemu je sa $V = \{a, b, c, d, e\}$ označen skup čvorova, a sa $E = \{\{a, b\}, \{a, c\}, \{a, e\}, \{b, c\}, \{c, d\}, \{c, e\}, \{d, e\}\}$ skup grana grafa G.

Slika 2.2: Primer grafa.

Definicija 2.2.2. Grane koje spajaju isti par čvorova se nazivaju paralelne grane. Grana koja spaja čvor sa samim sobom se naziva petlja. Graf je prost ukoliko nema petlji i paralelnih grana.

Definicija 2.2.3. Za dva čvora u i v grafa G, uv–put u G je niz čvorova koji počinje sa u, završava se sa v i takav je da su svaka dva uzastopna čvora na tom putu susedni u G. Put u grafu G u kome nema ponavljanja čvorova nazivamo prostim putem. Put kod koga su prvi i poslednji čvor jednaki nazivamo zatvorenim putem. Zatvoreni prost put koji se sastoji od bar 3 čvora nazivamo ciklus. Ciklus dužine k, za $k \geq 3$, označavamo sa C_n.

Definicija 2.2.4. Dva čvora u i v grafa G su povezana ukoliko u G postoji uv–put.

Definicija 2.2.5. Graf G je povezan ukoliko su svaka dva čvora u G međusobno povezana.

Definicija 2.2.6. Graf $G' = (V', E')$ je podgraf grafa $G = (V, E)$, ako je $V' \subseteq V$ i za sve $uv \in E'$ važi $uv \in E$.

Definicija 2.2.7. Komplement grafa $G = (V, E)$ je graf $\overline{G}$ sa istim skupom čvorova kao i G, pri čemu su dva različita čvora u $\overline{G}$ susedna ako i samo ako oni nisu susedni u G.

Definicija 2.2.8. Kompletan graf ili klika K_n je prost graf sa n čvorova u kome su svaka dva čvora susedna.

Definicija 2.2.9. Dekompozicija grafa G na klike je particija skupa čvorova V sa odgovarajućim granama iz skupa grana E na klike, odnosno podskupove gde su svaka dva čvora međusobno susedna.

Definicija 2.2.10. Graf G je bipartitan ako postoje neprazni skupovi A i B takvi da važi:

- $A \cap B = \emptyset$ i $A \cup B = V$

- za sve $uv \in E$ je tačno jedan element skupa $\{u, v\}$ u A i tačno jedan u B.

Ukoliko je prethodno ispunjeno, kažemo da je (A, B) particija bipartitnog grafa G.

Definicija 2.2.11. Stepen čvora u prostog grafa G je broj suseda tog čvora, i označavamo ga sa $d_G(u) = |N_G(u)|$. Minimalan stepen svih čvorova grafa obeležavamo sa $\delta(G)$ a maksimalan sa $\Delta(G)$. Dakle, $\delta(G) = \min_{u \in V(G)} d_G(u)$ i $\Delta(G) = \max_{u \in V(G)} d_G(u)$.

Definicija 2.2.12. Neka je $G = (V, E)$ graf. Utapanje grafa G u ravan $\mathbb{R}^2$, u oznaci G', zadato je različitim tačkama $x_1, x_2, ..., x_n$, $x_i \in \mathbb{R}^2$, $i = 1, 2, ..., n$, gde je $|V| = n$ (tačke $x_1, x_2, ..., x_n$ odgovaraju čvorovima grafa), i skupom prostih krivih, takvim da za tačke x_i i x_j postoji prosta kriva sa krajevima u x_i i x_j ako i samo ako su čvorovi grafa G koji odgovaraju tačkama x_i i x_j susedni. Za G' kažemo da je planarno utapanje ukoliko se opisane krive seku jedino u njihovim krajevima i svaka kriva sadrži tačno dva čvora iz skupa $\{x_1, x_2, ..., x_n\}$.

Definicija 2.2.13. Graf G je planaran ako postoji planarno utapanje grafa G.

2.3 Problemi bojenja grafova

Neki od najpoznatijih problema u teoriji bojenja grafova su: problem bojenja čvorova grafa, problem bojenja grana grafa, nalaženje hromatskog broja grafa i nalaženje intervalskog bojenja i intervalskog hromatskog broja grafa.

Definicija 2.3.1. Neka je $G = (V, E)$ graf i S proizvoljan skup čiji su elementi „boje". Svako preslikavanje $c : V \to S$ nazivamo bojenje čvorova grafa G (engl. Vertex Coloring Problem, VCP) bojama iz S. Za ovo bojenje kažemo da je pravilno ako za sve $uv \in E$ važi $c(u) \neq c(v)$, što znači da su susedni čvorovi obojeni različitim bojama. Uz to, ako je $|S| = k$, tada ovo pravilno bojenje zovemo k–bojenje čvorova grafa G.

Ovakvo preslikavanje definiše particiju skupa V na k nezavisnih skupova $V_1, V_2, ..., V_k$ (među čvorovima iz skupova V_i, za $i = 1, 2, ..., k$, nema susednih), tako da za svaki par $u, v \in V_i, i = 1, 2, ..., k$, važi $(u, v) \notin E$. Drugim rečima, c možemo posmatrati kao particiju od V na k međusobno disjunktnih i nezavisnih skupova $V_1, V_2, ..., V_k$, pri čemu mora da važi $\bigcup_{i=1}^{k} V_i = V$ i $v \in V_i$ ako i samo ako $c(v) = i$. Skupove $V_1, V_2, ..., V_k$ nazivamo klasama obojenosti.

Prvi model VCP problema je prvi put predložen u radu [8]. U [26] je dokazano da je VCP NP-težak problem, nakon čega je postao jako značajan u literaturi, kako zbog praktičnih primena u mnogim oblastima, kao što su problem raspoređivanja [50], problem rasporeda časova [14], alokacija registara [10] i dodela frekvencija [25], tako i zbog svojih teorijskih aspekata.

Jedan od osnovnih postupaka za nalaženje pravilnog bojenja čvorova grafova, koje nije obavezno optimalno (tj. sa najmanjim brojem boja), zasnovan je na sledećem: da bismo dobili pravilno bojenje čvorova grafa dovoljno je svaki njegov čvor u obojiti bojom koja nije upotrebljena za bojenje čvorova skupa $N(u)$. Iz ovoga je nastao algoritam za dobijanje pravilnog bojenja čvorova grafa G koji nazivamo pohlepni (engl. greedy) algoritam za bojenje čvorova grafa. Postupak je sledeći:

1. numerisati čvorove grafa G, kao i svaku od boja;
2. prvi čvor grafa G obojiti prvom bojom;
3. sledećem čvoru u nizu dodeliti boju sa najmanjim indeksom koja nije već iskorišćena za bojenje njegovih susednih čvorova;
4. ponavljati korak 3 sve dok se ne oboje svi čvorovi iz niza.

Primenom ovog algoritma se može dobiti $(\Delta(G)+1)$–bojenje čvorova grafa G, jer je kod $G\backslash\{u\}$ za bojenje čvorova iz $N(u)$ upotrebljeno najviše $|N(u)| \leq \Delta(G)$ boja, pa je preostala barem jedna boja koju možemo dodeliti čvoru u.

Definicija 2.3.2. Hromatski broj grafa $G = (V, E)$, u oznaci $\chi(G)$, je najmanji prirodan broj k za koji postoji k–bojenje čvorova grafa G. Svako $\chi(G)$–bojenje čvorova grafa G naziva se optimalno bojenje ovog grafa.

Za neke specijalne klase grafova lako se određuje hromatski broj. Na primer, svaki bipartitan graf je 2–hromatski. Ciklus sa parnim brojem čvorova je 2–hromatski, a sa neparnim brojem čvorova 3–hromatski. Ako je graf proizvoljan, onda je hromatski broj teško odrediti. Analitički izraz za izračunavanje hromatskog broja do danas nije nađen, pa je od značaja dati što bolja ograničenja za $\chi(G)$. Jedan od očiglednih ograničenja, koje se koristi kao gruba ocena hromatskog broja kod heuristika, je da za svaki graf G važi $\chi(G) \leq \Delta(G)+1$. U literaturi se mogu naći razne teorijske ocene donje i gornje granice hromatskog broja [21, 16, 30]. Problem određivanja hromatskog broja i optimalnog bojenja grafa G poznat je kao NP-težak, što je dokazano u radu [27].

Definicija 2.3.3. Neka je $G = (V, E)$ graf i S proizvoljan skup, čiji su elementi „boje". Svako preslikavanje $c : E \to S$ nazivamo bojenje grana grafa G (engl. Edge Coloring Problem, ECP) bojama iz S. Za ovo bojenje kažemo da je pravilno ako za sve $uv, uw \in E, v \neq w$ važi $c(uv) \neq c(uw)$, što znači da su grane koje imaju zajednički čvor obojene različitim bojama. Uz to, ako je $|S| = k$, tada ovo pravilno bojenje nazivamo k–bojenjem grana grafa G.

Prvi rad koji se bavio ECP je objavljen 1889. godine, dok je u [37] dokazana njegova NP-kompletnost. Značajan je zbog svoje primene u raznim problemima raspoređivanja, kao što su prenos datoteka i računarske mreže.

Definicija 2.3.4. Hromatski indeks grafa $G = (V, E)$, u oznaci $\chi'(G)$, je najmanji prirodan broj k za koji postoji k–bojenje grana grafa G.

Kao i u slučaju hromatskog broja, hromatski indeks proizvoljnog grafa je teško odrediti. Ipak, interesantno je da se za hromatski indeks grafa mogu naći mnogo bolja ograničenja nego za hromatski broj. Pre svega, ako čvor v ima k suseda, možemo zaključiti da je za bojenje grana koje sadrže v neophodno upotrebiti barem k boja. Dakle, važi $\chi'(G) \geq \Delta(G)$.

Definicija 2.3.5. Neka je $G = (V, E, W)$ prost težinski graf. W je skup težina čvorova i svaki element $w_i \in W$ predstavlja prirodan broj pridružen čvoru $v_i \in V$. Pridruživanje koje svakom čvoru v_i iz grafa G dodeljuje w_i uzastopnih brojeva (boja) iz skupa $\{1, 2, ..., k\}$, tako da su iskorišćene sve boje iz ovog skupa i ne postoje dva susedna čvora koja imaju neku zajedničku boju, nazivamo intervalsko k–bojenje grafa G. Graf G je intervalski k–obojiv ako postoji neko intervalsko k–bojenje ovog grafa.

Definicija 2.3.6. Intervalski hromatski broj $\chi_{int}(G)$ grafa G je najmanji broj k za koji postoji neko intervalsko k–bojenje ovog grafa. Svako intervalsko $\chi_{int}(G)$–bojenje grafa naziva se optimalno intervalsko bojenje.

Problemi intervalskog bojenja i nalaženja intervalskog hromatskog broja takođe spadaju u klasu NP-teških problema [65], i za njihovo rešavanje razvijene su mnoge metode, kao na primer sekvencijalne heuristike [11] i algoritmi bazirani na implicitnoj enumeraciji [73].

U ovom radu ćemo posmatrati pravilno bojenje čvorova neusmerenih prostih grafova, što znači da graf nema petlji i da je redosled čvorova u skupu $\{u, v\}$ nebitan za sve čvorove $u, v \in V(G)$.

2.4 Pregled i definicija problema MSCP

Problem minimalnog zbirnog bojenja grafova (engl. Minimum Sum Coloring Problem, MSCP) je nastao od standardnog problema bojenja čvorova grafa, koji je jedan od najpoznatijih NP-teških problema optimizacije. Sam problem su predložili Kubicka [49], u oblasti teorije grafova, i Supovit [64], u oblasti VLSI (engl. Very Large Scale Integration) dizajna. VLSI projektovanje se odnosi na izradu integrisanih kola i predstavlja povezivanje hiljade tranzistora u jedinstveno elektronsko kolo (engl. chip).

Problem raspoređivanja se može pojaviti u praksi u velikom broju formi. Kada govorimo o problemu raspoređivanja, onda se u uopštenom smislu misli na alokaciju resursa tako da se optimizuje jedna ili više funkcija cilja. Raspoređivanje operacija na mašinama predstavlja za mašinske inženjere jako bitnu formu problema raspoređivanja. Rezervacija piste za poletanje i sletanje aviona takođe spada u kategoriju problema raspoređivanja. Da bi se problem mogao rešavati, u literaturi je definisano više matematičkih notacija i modela, što se može naći u radu [61]. Problem se može formulisati na sledeći način [13]:

Zadati su skup $R = \{r_1, r_2, ..., r_m\}$ od m resursa (ljudi, mašine, sirovine, itd.) i skup $P = p_1, p_2, ..., p_n$ od n poslova. Svaki posao $p_i, i = 1, 2, ..., n$ zahteva angažovanje nekih resursa iz R koji ga realizuju. Svi poslovi imaju isto vreme realizacije koje se smatra vremenskom jedinicom, tj. dužinom jednog vremeskog perioda. Zadat je takođe i skup $T = \{t_1, t_2, ..., t_k\}$ vremenskih perioda predviđenih za kompletno izvršenje svih poslova iz P.

Odrediti jedan mogući raspored poslova u vremenu, tj. svaki posao iz P treba dodeliti nekom periodu iz T, tako da svi poslovi budu obavljeni bez prekida, a nijedan resurs nije angažovan na dva ili više poslova istovremeno.

Ovako formulisanom problemu pridružujemo graf G sa n čvorova, gde svakom poslu iz P odgovara jedan čvor, a dva čvora su spojena granom ukoliko odgovarajući poslovi angažuju isti resurs. Očigledno je da je bilo kojim pravilnim bojenjem čvorova grafa G brojem boja ne većim od k zadat jedan mogući raspored poslova, pri čemu svaka boja tog bojenja odgovara nekom vremenskom periodu iz T. Ukoliko treba

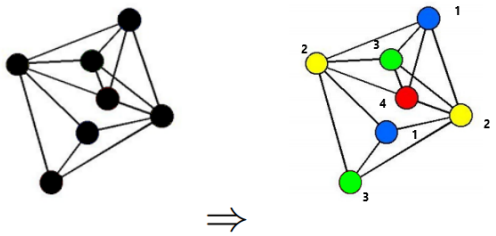
naći onaj raspored poslova kod koga je ukupan broj vremenskih perioda potrebnih za njihovu realizaciju minimalan, tada se problem svodi na nalaženje hromatskog broja $\chi(G)$ i svako optimalno bojenje čvorova grafa G definiše ovakav jedan raspored poslova.

Kao specijalan slučaj opisanog problema raspoređivanja definišu se problemi rasporeda časova ili ispita, čiji su matematički modeli dati u [47]. Cilj problema raspoređivanja ispita je dodeliti ispite (poslove) predviđenim danima (vremenskim periodima) tako da ne postoji student koji polaže dva ili više predmeta istovremeno. Pridruživanjem grafa opisanom problemu, čvorovi bi predstavljali ispite a između dva čvora bi postojala grana ako za odgovarajuće ispite postoji bar jedan student koji treba da ih polaže.

Slično važi i za problem rasporeda časova – potrebno je dodeliti lekcije (poslove) školskim časovima (vremenskim periodima), koji su tog dana predviđeni za izvođenje nastave, tako da nijedan nastavnik, razred ili učionica (resursi) nisu raspodeljeni na dve ili više lekcija istovremeno. Ovde su čvorovi grafa lekcije i između njih postoji grana ako odgovarajuće lekcije angažuju istog nastavnika, razred ili učionicu.

Neka je dat prost neusmeren graf $G = (V, E)$ sa skupom čvorova $V = \{v_1, v_2, ..., v_n\}$ i skupom grana $E \subseteq V \times V$, $|V| = n$ i $|E| = m$. Promenljive $x_i \in \mathbb{N}$ predstavljaju boju čvora v_i za $i = 1, 2, ..., n$. Cilj MSCP je naći pravilno bojenje čvorova grafa G tako da je zbir boja dodeljenih čvorovima v_i minimalan, tj. potrebno je minimizovati funkciju $\sum_{i=1}^{n} x_i$, uz uslov da svaka dva susedna čvora ne budu obojena istom bojom.

Na slici 2.3 prikazan je jedan primer bojenja grafa. Sa leve strane je polazni graf a sa desne je ilustrovano optimalno rešenje, pri čemu njegova vrednost iznosi 16. Intuitivno je jasno da čvorove sa najvećim brojem suseda treba obojiti bojama koje imaju najmanju vrednost.



Slika 2.3: Primer minimalnog zbirnog bojenja grafa.

Tokom poslednje dve decenije, problem je najviše teorijski izučavan. Međutim, zbog svog praktičnog značaja u problemima iz realnog života, uspešno su razvijeni

razni aproksimativni i polinomijalni algoritmi za specijalne grafove, kao na primer za stabla, bipartitne i intervalske grafove [70]. Takođe su tokom poslednjih par godina na opisanom problemu testirane i heurističke metode kao što su paralelni genetski algoritam [44], pohlepni algoritami [51], tabu pretraživanje [5], lokalno pretraživanje [35] i mnogi hibridni algoritmi [17, 71].

2.5 Teorijske i računske granice problema MSCP

MSCP se smatra teškim i izazovnim problemom za rešavanje i većina postojećih algoritama za njegovo rešavanje je eksponencijalne složenosti. Sa druge strane, polinomijalni algoritmi postoje samo za neke specijalne slučajeve grafova. Stoga se u poslednjih nekoliko godina i dalje traga za algortimima manje složenosti i radi na razvoju različitih aproksimativnih i praktičnih metoda za nalaženje rešenja.

Ukoliko prilikom rešavanja optimizacionog problema dobijemo rešenje za koje se zna da nije optimalno, za njega kažemo da je suboptimalno rešenje. Cilj aproksimativnih algoritama je obezbeđivanje rešenja dokazivog kvaliteta, dok praktični algoritmi pokušavaju da nađu što je moguće bolja suboptimalna rešenja za prihvatljivo vremensko izvršavanje. Važnu ulogu kod ispitivanja kvaliteta dobijenog rešenja i samog algoritma koji je korišćen takođe imaju i teorijski dokazi donjih i gornjih ocena rešenja koji su prikazani u radu [39].

Podsetimo se da je za svaki prost neusmeren graf $G = (V, E)$, sa $n = |V|$ čvorova i $m = |E|$ grana, hromatski broj $\chi(G)$ najmanji broj potrebnih boja za bojenje grafa G tako da postoji pravilno k–bojenje čvorova. Uvedimo i pojam hromatske sume $\sum(G)$ koja predstavlja minimalni zbir boja dodeljenih čvorovima od svih mogućih pravilnih k–bojenja čvorova grafa G. Sledeće teorijske donje i gornje granice za problem MSCP izvedene su u radovima [44, 59, 68]:

$$\sum(G) \leq n + m, \tag{2.1}$$

$$\left\lceil \sqrt{8m} \right\rceil \leq \sum(G) \leq \left\lfloor \frac{3(m+1)}{2} \right\rfloor, \tag{2.2}$$

$$n + \frac{\chi(G)(\chi(G)-1)}{2} \leq \sum(G) \leq \left\lfloor \frac{n(\chi(G)+1)}{2} \right\rfloor. \tag{2.3}$$

Iz nejednakosti (2.1) – (2.3) lako se dobija da su najbolja teorijska donja i gornja granica za MSCP, redom:

$$LB_t = \max\left\{\left\lceil\sqrt{8m}\right\rceil, n + \frac{\chi(G)(\chi(G)-1)}{2}\right\},$$

$$UB_t = \min\left\{n+m, \left\lfloor\frac{3(m+1)}{2}\right\rfloor, \left\lfloor\frac{n(\chi(G)+1)}{2}\right\rfloor\right\}.$$

Označimo sa $\mathcal{T}(G)$ skup svih pravilnih k–bojenja čvorova grafa G. S obzirom na to da je cilj MSCP određivanje pravilnog k–bojenja uz minimizaciju zbira boja dodeljenih čvorovima, izraz $\sum_{i=1}^{n} c(v_i)$ daje računsku gornju granicu za rešenje problema, gde $c \in \mathcal{T}(G)$.

Neka je $G' = (V, E')$, $(E' \subseteq E)$, proizvoljan podgraf grafa $G = (V, E)$. Očigledno je $\sum(G')$ donja granica za $\sum(G)$ jer je svako pravilno boje grafa G ujedno i pravilno bojenje grafa G'. Dakle, imamo da važi $\sum(G) \geq \sum(G')$. Podgrafovi koji su korišćeni u literaturi za dobijanje računske donje granice f_{LB} su bipartitni [27, 48] i klike [59, 72]. Najbolje računske donje i gornje granice dobijene su iz dekompozicije grafa na klike [59].

Neka je c dekompozicija grafa G na klike. Sledeća jednačina daje računsku donju granicu za MSCP jer se svaka klika S_l može obojiti na tačno jedan način (sa $|S_l|$ boja) i zbir svih korišćenih boja za S_l iznosi $|S_l|(|S_l|+1)/2$:

$$f_{LB}(c) = \sum_{l=1}^{k} \frac{|S_l|(|S_l|+1)}{2}. \tag{2.4}$$

Jedan od načina za dobijanje dekompozicije grafa na klike prikazan je u radu [40] gde je korišćen pristup traženja pravilnog bojenja komplementarnog grafa $\overline{G}$ grafa G.

2.6 Matematički modeli celobrojnog linearnog i nelinearnog programiranja za MSCP

Neka je dat prost neusmeren graf $G = (V, E)$ sa skupom čvorova $V = \{v_1, v_2, ..., v_n\}$ i skupom grana $E \subseteq V \times V$, $|V| = n$ i $|E| = m$. Promenljive $x_i \in \mathbb{N}$ označavaju boju čvora v_i za $i = 1, 2, ..., n$. Cilj problema minimalnog zbirnog bojenja je naći pravilno bojenje čvorova grafa G tako da je zbir boja dodeljenih čvorovima v_i minimalan.

U radu [22] autori su MSCP formulisali kao problem celobrojnog nelinearnog programiranja (engl. Nonlinear Programming, NLP) na sledeći način:

$$\min \sum_{i=1}^{n} x_i \tag{2.5}$$

$$|x_i - x_j| \geq 1, \qquad \forall (i,j) \in E \tag{2.6}$$

$$1 \leq x_i \leq n, \qquad i = 1, 2, ..., n \tag{2.7}$$

$$x_i \in \mathbb{Z}^+, \qquad i = 1, 2, ..., n. \tag{2.8}$$

Funkcija cilja je suma svih boja čvorova zadatog grafa i zadatak je odrediti njen minimum, što je zadato sa (2.5). Ograničenja (2.6), kojih ima ukupno m, nam garantuju da nikoja dva susedna čvora nisu obojena istom bojom. Sa (2.7) date su granice za promenljive, a (2.8) definiše tip promenljivih $x_i, i = 1, 2, ..., n$. Iako je formulacija intuitivno jasna, zbog svoje nelinearnosti je model teško rešiti egzaktnim rešavačima, posebno u slučaju velikog broja čvorova. Postoje egzaktni rešavači koji mogu da reše ovaj nelinearan model (npr. Lingo, Gurobi itd.), ali za instance manjih dimenzija.

Sledeća formulacija MSCP, koja je izložena u radu [24], linearna je i u njoj su uvedene binarne promeljive x_v^i za koje važi:

$$x_v^i = \begin{cases} 1, & \text{čvor } v \text{ je obojen bojom } i \\ 0, & \text{inače.} \end{cases}$$

MSCP kao problem celobrojnog linearnog programiranja (engl. Integer Linear Programming, ILP) može biti formulisan na sledeći način [24]:

$$\min \sum_{i \in \{1,...,k\}} \sum_{v \in V} i \cdot x_v^i \tag{2.9}$$

$$x_u^i + x_v^i \leq 1, \qquad (u,v) \in E, i \in \{1, ..., k\}, \tag{2.10}$$

$$\sum_{i \in \{1,...,k\}} x_v^i = 1, \qquad v \in V \tag{2.11}$$

$$x_v^i \in \{0, 1\}, \qquad v \in V, i \in \{1, ..., k\}. \tag{2.12}$$

Za parametar k se uzima $k = 1 + \triangle(G)$, gde je $\triangle(G)$ maksimalan stepen grafa G (najveći broj suseda po svim čvorovima). Funkcija cilja je zbir svih boja čvorova zadatog grafa i zadatak je odrediti njen minimum, što je zadato sa (2.9). Ograničenja (2.10) obezbeđuju da nijedan par susednih čvorova ne bude obojen istom bojom, dok ograničenja (2.11) znače da svakom čvoru treba dodeliti tačno jednu boju. Drugi model ima ukupno $k \cdot n$ promenljivih i $m \cdot k + n$ ograničenja. Sa (2.12) definisane su promenljive x_v^i kao binarne.

2.7 Pregled relevantne literature

Ovo poglavlje sadrži pregled metoda korišćenih u literaturi za rešavanje problema MSCP. U radu [35] iz 2011. godine za rešavanje MSCP problema je korišćen novi LS algoritam baziran na VNS-u i ILS-u, kojim su dostignuta najbolja rešenja za 27 od testiranih 38 instanci (15 malih sa do 100 čvorova, 16 srednjih sa do 500 čvorova i 7 velikih sa preko 500 čvorova). Dobijena rešenja su poređena sa predstavljenim rezultatima iz radova [5, 17, 44, 51], pri čemu je za 27 instanci prezentovano bolje rešenje, dok za preostalih 11 instanci nije dostignuto bolje rešenje od dotadašnjih rezultata iz literature. U predloženoj implementaciji iz rada [35] se dozvoljavaju i nepravilna bojenja, tj. moguće je dobiti nedopustiva rešenja, ali uz kasniju modifikaciju na dopustivo. Ukoliko bi bilo prihvatljivo duže vreme izvršavanja, autori rada [35] predlažu promenu inicijalnog koraka za generisanje boja početnog rešenja na slučajan način.

EXSCOL, koji predstavlja hibridni algoritam zasnovan na nezavisnom izdvajanju skupa, predložen je u radu [71]. Ideja ovog algoritma je da se pronađe što više nezavisnih (disjunktnih) skupova u svakoj iteraciji. Tako se formira više velikih skupova čime se i veći broj čvorova može obojiti sa što manjom bojom, vodeći tako ka minimalnom zbiru svih boja. Za potrebe računanja velikih nezavisnih skupova, EXSCOL koristi heuristiku zasnovanu na tabu pretraživanju. Testiran je na 52 instance (11 malih sa do 100 čvorova, 22 srednje sa do 500 čvorova i 19 velikih sa preko 500 čvorova), pri čemu se za 28 instanci dostižu bolji rezultati za gornje granice od do tada poznatih.

Tokom 2012. godine objavljen je rad [3] gde je korišćen algoritam Breakout lokalna pretraga (engl. Breakout Local Search, BLC) koji kombinuje više osnovnih metaheuristika. Prostor dopustivih rešenja se pretražuje kombinovanim korišćenjem lokalne pretrage i adaptivne perturbacione strategije. Testirano je ukupno 27 instanci (11 malih sa do 100 čvorova, 11 srednjih sa do 500 čvorova i 5 velikih sa preko 500 čvorova), a od toga je za 4 dostignuto poboljšano najbolje rešenje u odnosu na dotadašnje rezultate iz literature, za 15 su dobijena već poznata najbolja rešenja, dok je za 8 algoritam došao do lošijih rezultata.

Autori rada [72] iz 2013. godine su predložili novi pristup rešavanja problema MSCP dekompozicijom zadatog grafa na disjunktne klike za izračunavanje donjih granica. Predloženi pristup pri svakoj iteraciji pronalazi maksimalan broj klika iste veličine, pri čemu se teži ka tome da njihova veličina bude najveća moguća. Testirane

su 62 instance (15 malih sa do 100 čvorova, 27 srednjih sa do 500 čvorova i 20 velikih sa preko 500 čvorova) i dobijeno je da se za 14 instanci dolazi do poboljšanja donjih granica i da se po prvi put pokazala optimalnost za 4 instance. Takođe, dobijene su jos 24 donje granice za instance koje do tada nisu bile testirane u literaturi.

Kasnije 2014. godine je u radu [41] izložen memetski algoritam baziran na tabu pretraživanju sa dva susedstva i operatoru ukrštanja sa više roditelja. Dobijeni eksperimentalni rezultati na ukupno 77 instanci (17 malih sa do 100 čvorova, 39 srednjih sa do 500 čvorova i 21 velika sa preko 500 čvorova) pokazuju da je na 17 instanci dobijeno bolje rešenje od do tada poznatih, kao i zbog toga što je za 18 instanci po prvi put dobijena gornja granica.

Godine 2016. je objavljen rad [40] gde je predstavljen stohastički hibridni evolutivni algoritam pretrage za dobijanje donjih i gornjih granica za problem MSCP. Predloženi algoritam se oslanja na zajedničku upotrebu dva operatora ukrštanja za generisanje rešenja potomaka, koja se poboljašavaju iterativnim dvofaznim postupkom tabu pretrage. Prilikom testiranja ovog algoritma na 94 instance (17 malih sa do 100 čvorova, 51 srednja sa do 500 čvorova i 26 velikih sa preko 500 čvorova), dobijeno je poboljšanje za donje granice 24 instance i za gornje granice 27 instanci.

SVNS za rešavanje problema MSCP je prvi put predložen u radu [22], gde je testirano ukupno 23 instance (15 malih sa do 100 čvorova i 7 srednjih sa do 500 čvorova), od čega se za 19 dobilo već poznato optimalno rešenje, a za 4 algoritam je uspeo da dođe samo do približnog rešenja u odnosu na postojeće rezultate iz literature. Značaj rada [22] je u tome što je prvi put za implementaciju SVNS-a korišćena nelinearna matematička formulacija zadata sa (2.5) – (2.8).

Glava 3

Metoda promenljivih okolina za MSCP

U ovom poglavlju dat je opis VNS metode koja je razvijena za rešavanje MSCP. Iako je MSCP problem dosta rešavan metaheurističkim metodama, pregledom postojeće literature nije pronađen pristup koji je predložen u ovom radu.

3.1 Reprezentacija rešenja

Pre same implementacije VNS metode za MSCP, potrebno je najpre definisati skup dopustivih rešenja X, odnosno reprezentaciju svakog rešenja $x \in X$. Neka je n broj čvorova, a m broj grana zadatog grafa G. Skup dopustivih rešenja X je skup svih n-dimenzionalnih vektora čije su sve koordinate međusobno različite celobrojne vrednosti iz skupa $\{1, 2, ..., n\}$, tj. za svako i važi $1 \leq x_i \leq n$, $x_i \neq x_j$, za svako $i \neq j$. Dakle, svaki vektor $x \in X$ iz dopustivog skupa je jedna permutacija brojeva $1, 2, ..., n$ na osnovu koje se određuje jedinstveni redosled po kome će se čvorovima dodeljivati boje.

Od vektora x formira se redosled čvorova na sledeći način: ako je $x(i) = k$, to znači da je čvor v_i k-ti po redu za dodelu boje. Prednost ovakve reprezentacije rešenja je u tome što se nakon faze razmrdavanja i faze lokalne pretrage, implementiranih u ovom radu, ne mora voditi računa o dopustivosti rešenja, tj. ni u jednom koraku se ne mora vršiti ispitivanje da li dobijeno rešenje pripada skupu dopustivih rešenja X, uz eventualnu popravku do dopustivosti.

Na primer, posmatrajmo graf G sa 5 čvorova, $v_1, v_2, ..., v_5$. Neka je vektor rešenja oblika $x = [3, 5, 2, 1, 4]$. Pošto je $x(4) = 1$, to znači da će prvo čvoru v_4 biti dodeljena

boja 1. Nakon toga slede redom čvorovi v_3, v_1, v_5, v_2, kojima će se dodeljivati najmanja moguća boja tako da bojenje čvorova bude pravilno, tj. susednim čvorovima treba dodeliti različite boje.

3.2 Funkcija cilja

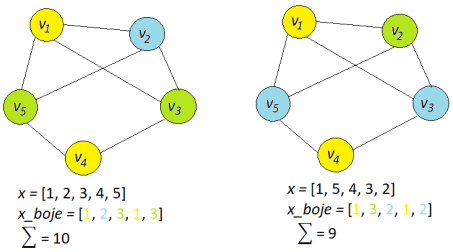
Vrednost funkcije cilja za proizvoljno rešenje $x \in X$ izračunava se na sledeći način:

- Inicijalizuje se vektor x_boje dužine n, čiji su svi elementi nule (nula označava da čvoru još uvek nije dodeljena boja);
- Za čvor v_i koji je na redu za bojenje na osnovu zadatog vektora rešenja, određuje se vektor $zauzete\_boje$, koji sadrži boje koje su već zauzete njegovim obojenim susedima;
- Nakon određivanja vektora $zauzete\_boje$, koji sadrži već zauzete boje, čvoru v_i se dodeljuje boja jednaka $\min\{q : q \in \mathbb{Z}^+, q \notin zauzete\_boje\}$;
- Ovaj postupak se ponavlja za sve čvorove v_i grafa G, sve dok se ne ažurira ceo vektor x_boje;
- Vrednost funkcije cilja za rešenje x se dobija tako što se saberu svi elementi vektora x_boje, tj. $f(x) = \sum_{i=1}^{n} x\_boje(i)$.

Ideja potiče iz pohlepnog algoritma za bojenje čvorova grafa. Složenost izračunavanja funkcije cilja će najviše zavisiti od broja grana grafa koji se testira. To znači da će vreme izvršavanja za grafove sa jako velikim brojem grana biti znatno duže od grafova sa manjim brojem grana.

Na slici 3.1 je prikazana razlika između ovakve vrste bojenja sa dva različita redosleda dodeljivanja boja. Na grafu levo bojenje je vršeno prateći redosled $x = [1, 2, 3, 4, 5]$. To znači da se najpre čvoru v_1 (jer je $x(1) = 1$) dodeli prva boja (1). Zatim se bira boja za čvor v_k takav da je $x(k) = 2$. Kako je $x(2) = 2$, čvor v_2 se boji sledeći. Kako je v_2 sused čvoru v_1 koji je već obojen, njemu se dodeljuje prva sledeća slobodna boja (2). Sada se bira boja za čvor v_3 jer je $x(3) = 3$. Kako su njegovi susedi i v_1 i v_2 koji su obojeni redom bojama 1 i 2, čvor v_3 se boji prvom sledećom slobodnom bojom, a to je 3. Postupak se nastavlja analogno do poslednjeg čvora v_5 i vektor dobijenih boja je $x\_boje = [1, 2, 3, 1, 3]$. Vrednost funkcije cilja $f(x)$ jednaka

je zbiru elemenata iz vektora x_boje, tj. 10. Na grafu desno prikazano je bojenje po istom principu, samo prateći drugačiji redosled $x = [1, 5, 4, 3, 2]$. U ovom slučaju će čvor v_1 prvi dobiti boju 1 pošto je $x(1) = 1$, pa zatim čvor v_5 jer je $x(5) = 2$. Kako je v_1 obojen bojom 1, a v_5 je njegov sused, čvoru v_5 treba dodeliti boju 2. Zatim sledi v_4 ($x(4) = 3$) koji se boji najmanjom dostupnom bojom 1, iz razloga što jedini njegov do sada obojeni sused v_5 ima boju 2. Potom se redom boje čvorovi v_3 i v_2, sve dok se ne dobije vektor boja $x\_boje = [1, 3, 2, 1, 2]$. Vrednost funkcije cilja $f(x)$ za ovakav vektor boja je 9, što je manje od vredosti funkcije cilja za levi graf sa slike.



Slika 3.1: Poređenje rezultata MSCP sa različitim reprezentacijama rešenja.

3.3 Faza razmrdavanja

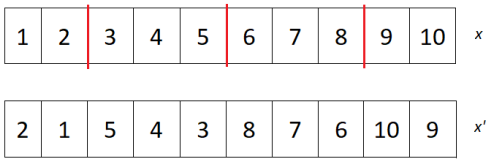
Razmrdavanje je jedan od ključnih koraka u VNS metodi jer pravi balans između malih i velikih okolina i utiče najviše na stepen diverzifikacije. Ovime je moguće stići do rešenja koja su daleko od trenutno najboljeg, čime se rešava problem zaglavljivanja u lokalnim minimumima.

U implementaciji VNS metode za rešavanje MSCP problema, korak razmrdavanja se realizuje na sledeći način. Okolina $N_k(x)$ rešenja x, za $1 \leq k \leq k_{\max}$, se definiše kao skup svih rešenja koja se od x dobijaju sledećim postupkom. Na slučajan način se izabere k različitih pozicija iz skupa $\{1, 2, ..., n\}$. Zatim se vektor rešenja x prelomi na tih k mesta i svakom delu se obrne redosled elemenata. Primenom koraka razmrdavanja dobija se novi vektor rešenja $x' \in N_k(x)$.

Ovakvom fazom razmrdavanja će se očuvati dopustivost rešenja jer će svaki novi vektor x' biti jedna nova permutacija brojeva $1, 2, ..., n$ na osnovu koje se određuje jedinstveni redosled dodeljivanja boja čvorovima. Ideja za ovakvu implementaciju razmrdavanja je da se na neki način očuva redosled dodeljivanja boja, ali i da se

„razdrma" toliko da se ispitaju i neke dalje oblasti dopustivog prostora X. Zbog toga se prekidanjem veze na k mesta dobijeni lanci spajaju sa drugim lancima vektora rešenja, čime se obezbeđuje jači stepen diverzifikacije, dok se sa druge strane unutar jednog lanca rotiranjem redosleda obezbezbeđuje slabija diverzifikacija.

Neka je $k = 3$ i neka su na slučajan način izabrane pozicije 2, 5 i 8. Na slici 3.2 je prikazan postupak razmrdavanja za vektor rešenja $x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$. Najpre se vektor x preseče na 3 mesta – između pozicija 2 i 3, 5 i 6, 8 i 9. Zatim se rotiraju elementi sva 4 dobijena dela vektora, odnosno obrtanje elemenata se izvršava 4 puta i to između sledećih pozicija: 1 i 2, 3 i 5, 6 i 8, 9 i 10. Novo rešenje x' dobijeno ovakvim razmrdavanjem je $x' = [2, 1, 5, 4, 3, 8, 7, 6, 10, 9]$.



Slika 3.2: Primer dobijanja vektora x' od x u koraku razmrdavanja.

Koraci faze razmrdavanja VNS metode prikazani su sledećim pseudokodom:

Algoritam 1: $izaberi\_resenje\_iz\_okoline\_N(x, k)$

1: $indeksiPrelamanja \leftarrow$ na slučajan način izabrati k različitih prirodnih brojeva od 0 do $n - 1$
2: Sortiranje vektora $indeksiPrelamanja$
3: **for** $i \leftarrow 1$ to k do
4: obrtanje redosleda elemenata vektora x od pozicije $indeksiPrelamanja[i - 1] + 1$ do indeksa $indeksiPrelamanja[i]$
5: **end for**
6: **return** x

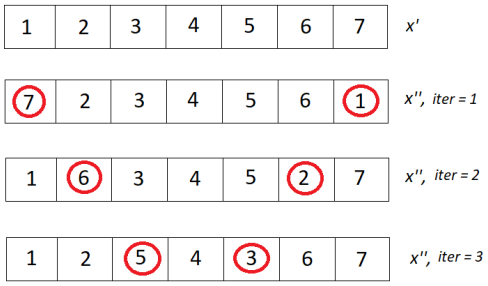
3.4 Faza lokalne pretrage

Iako se fazom ramrdavanja obezbeđuje diverzifikacija rešenja kako bi se izbeglo zaglavljivanje u lokalnom minimumu, odnosno pretraga delova prostora koji su daleko od tekućeg rešenja, u većini slučajeva dobijeno rešenje x' nije tačka lokalnog minimuma. Zato je potrebno izvršiti neku metodu lokalne pretrage kako bi se došlo do kvalitetnijih rešenja.

U implementiranoj fazi lokalne pretrage se za okolinu uzima skup svih vektora dobijenih iz x' zamenom elemenata na poziciji i i $n-i+1$, za $i = 1, 2, ..., \lfloor n/2 \rfloor$ (npr. prvi i poslednji, drugi i pretposlednji itd.). Korišćena je strategija prvog poboljšanja (engl. first improvement strategy). Ova strategija podrazumeva da kada se tokom iterativne pretrage neke okoline rešenja x' naiđe na prvo poboljšanje x'', tj. $f(x'') < f(x')$, lokalna pretraga u okolini rešenja x' se prekida i postupak se nastavlja daljom pretragom u okolini rešenja x''. U slučaju da se rešenje ne može popraviti, lokalna pretraga se prekida i rezultat je vektor x'.

Ideja za ovako implementiranu lokalnu pretragu je da se vrši ispitivanje manje okoline i to samo zamenom mesta dva elementa.

Neka je dato rešenje $x' = [1, 2, 3, 4, 5, 6, 7]$. Par iteracija lokalne pretrage za vektor x' prikazane su na slici 3.3. U prvoj iteraciji ($iter = 1$) vrši se rotacija prvog i poslednjeg elementa vektora x', čime se dobija $x'' = [7, 2, 3, 4, 5, 6, 1]$. Pretpostavimo da nije došlo do poboljšanja vrednosti funkcije cilja, tj. ne važi $f(x'') < f(x')$. Tada prelazimo na sledeći vektor i u iteraciji $iter = 2$ poredimo vrednosti funkcije cilja vektora x' i $x'' = [1, 6, 3, 4, 5, 2, 7]$. Ukoliko i tada nije ispunjeno $f(x'') < f(x')$, u iteraciji $iter = 3$ prelazimo na novi vektor $x'' = [1, 2, 5, 4, 3, 6, 7]$ i nastavljamo analogno prethodnim koracima. Ako bi se sada desilo da je $f(x'') < f(x')$, lokalna pretraga bi se prekinula i zatim bi se krenulo od vektora x''. Prvi sledeći vektor koji bi se ispitivao bio bi $[7, 2, 5, 4, 3, 6, 1]$ za $iter = 1$.



Slika 3.3: Rešenje x i njegova okolina za fazu LS.

Koraci faze lokalne pretrage VNS metode prikazani su sledećim pseudokodom:

Algoritam 2: *lokalna_pretraga*(x')

1: $pozicija = 0$
2: **while** $pozicija < n/2$ **do**
3: $x'' := x'$
4: zameniti elemente vektora x'' sa indeksima $pozicija$ i $n - pozicija - 1$
5: **if** $f(x'') < f(x')$
6: $x' := x''$
7: $pozicija = 0$
8: **else**
9: $pozicija = pozicija + 1$
10: **end if**
11: **end while**
12: **return** x'

3.5 VNS algoritam za MSCP

Nakon definisanja reprezentacije rešenja, određivanja vrednosti funkcije cilja, definisanja faze razmrdavanja i faze lokalne pretrage, može se preći na konkretnu implementaciju VNS metode za problem MSCP. Na početku algoritma vrši se učitavanje ulaznih podataka. To su broj čvorova n, broj grana m i matrica susedstva za graf G.

U inicijalnom koraku generiše se početno rešenje x_0, tako što se na slučajan način formira permutacija celobrojnih vrednosti iz skupa $\{1, 2, ..., n\}$. Ovako dobijena početna permutacija postaje trenutno najbolje rešenje x^* i vrši se ažuriranje najbolje vrednosti funkcije cilja na $f^* = f(x^*)$. Takođe, promenljiva k se postavlja na 1.

Svaka iteracija VNS algoritma počinje opisanom fazom razmrdavanja, nakon čega sledi faza lokalne pretrage. Razmrdavanjem se iz okoline $N_k(x^*)$ tekućeg rešenja x^* dobija novo rešenje x', koje se prosleđuje kao početno rešenje lokalne pretrage. Rezultat lokalne pretrage x'' se poredi sa trenutno najboljim rešenjem x^* i ukoliko važi da je $f(x'') < f(x^*)$, novo rešenje x'' se prihvata kao trenutno najbolje, tj. $x^* := x''$, i vrši se ažuriranje najbolje vrednosti funkcije cilja na $f^* = f(x'')$, kao i parametra k na 1. U ovom slučaju, algoritam nastavlja pretragu razmrdavanjem u okolini novog najboljeg rešenja $N_1(x^*)$.

U slučaju da nakon lokalne pretrage nije pronađeno bolje rešenje od trenutno najboljeg, parametar k se povećava za 1 i prelazi se na ispitivanje naredne okoline trenutno najboljeg rešenja $N_{k+1}(x^*)$. Ako k postane veće od $k_{\max}$, njegova vrednost

se ponovo postavlja na 1 i postupak se nastavlja razmrdavanjem u okolini $N_1(x^*)$. Opisani koraci se smenjuju sve dok se ne ispuni kriterijum zaustavljanja, koji je u ovoj VNS implementaciji određen maksimalnim brojem dozvoljenih iteracija, zadatim parametrom max_{iter}, bez poboljšanja rešenja.

Koraci implementirane verzije VNS metode prikazani su sledećim pseudokodom:

Algoritam 3: VNS za MSCP

Učitavanje ulaznih podataka n, m i matrice susedstva
Generisanje početnog rešenja x
Ažuriranje $x^* := x$ i $f^* = f(x)$
while $broj\_iteracija < max_{iter}$
    $k = 1$
    while $k \leq k_{\max}$ **do**
        Faza razmrdavanja: $x' = izaberi\_resenje\_iz\_okoline\_N(x^*, k)$
        Faza lokalne pretrage: $x'' = lokalna\_pretraga(x')$
        if $f(x'') < f^*$
            $x^* := x''$
            $f^* = f(x'')$
            $k = 1$
            $broj\_iteracija = 0$
        else
            $k = k + 1$
    end while
end while

Ulazni parametri VNS metode su maksimalni broj okolina koje se pretražuju, $k_{\max}$, kao i maksimalni broj iteracija algoritma, max_{iter}.

Glava 4

Gausovska metoda promenljivih okolina za MSCP

Iako je GaussVNS namenjen za rešavanje problema kontinualne optimizacije, u nekim slučajevima je moguće prilagoditi ga i za rešavanje problema kombinatorne optimizacije. Mogućnost prilagođavanja zavisi od samog problema koji se rešava, reprezentacije rešenja, implementacije faze razmrdavanja i lokalne pretrage.

Reprezentacija rešenja u VNS algoritmu opisanog u glavi 3 je realizovana tako da ju je moguće adaptirati i za GaussVNS u kom će se raditi sa realnim promenljivim umesto sa celobrojnim. Problem opisan u 2.4 dosta podseća na probleme kontunualne optimizacije, u smislu da ima jako puno lokalnih minimuma. Osnovni cilj je bio upoređivanje sa običnom VNS metodom iz glave 3, da bi se najpre ispitalo koliko će promena distribucije (sa uniformne na normalnu), koja se koristi za generisanje slučajnih tačaka u koraku razmrdavanja, uticati na performanse algoritma i dobijene krajnje rezultate.

Umesto okolina $N_1(x), N_2(x), ..., N_{k_{\max}}$, potrebno je definisati niz raspodela slučajnih tačaka $P_1(x), P_2(x), ..., P_{k_{\max}}(x)$, gde je $P_k(x)$ n-dimenzionalna Gausova raspodela slučajnih brojeva centrirana u tački x, sa matricom kovarijansi Σ_k. Na ovaj način je od tačke x moguće dobiti bilo koju tačku iz dopustivog prostora. Kako bi se GaussVNS algoritam primenio za rešavanje diskretnog problema MSCP, jedina okolina koja se koristi za proces pretrage je čitav prostor dopustivih rešenja.

4.1 Reprezentacija rešenja

U kontinualnoj optimizaciji skup dopustivih rešenja je neprebrojiv, dok je u slučaju problema MSCP konačan. Da bi se metoda za rešavanje problema kontinualne optimizacije mogla adaptirati za rešavanje diskretnog problema MSCP neophodno je najpre izvršiti preslikavanje iz neprebrojivog skupa u konačan skup. Reprezentacija rešenja u implementaciji VNS metode iz glave 3 je takva da rešenje x sadrži celobrojne vrednosti x_i kojima je određen redosled čvorova po kom će im se dodeljivati boje. Takva reprezentacija se na jednostavan način može adaptirati za primenu GaussVNS metode tako što će se dozvoliti da vektor rešenja x sadrži i realne vrednosti x_i. Ovo je moguće realizovati jer se za sve koordinate vektora rešenja međusobno porede njihove vrednosti kako bi se dobio određeni raspored, bilo da su one celobrojne ili realne. Skup dopustivih rešenja X sada je skup svih n-dimenzionalnih vektora koji sadrže realne vrednosti.

Sa ovakvom reprezentacijom rešenja za GaussVNS metodu (gde dopuštamo da vrednosti x_i rešenja x budu necelobrojna, tj. realna) omogućavamo korišćenje normalne raspodele čiji rezultat će biti realne vrednosti. Opravdanje za ovakav prelazak sa diskretnog problema je to što vektor rešenja sa realnim vrednostima i dalje zadaje jedan redosled dodeljivanja boja čvorovima, jer se njegove koordinate mogu međusobno uporediti operatorom $<$. Svaki vektor rešenja $x \in X$ u predloženoj GaussVNS metodi je vektor koji sadrži međusobno različite realne brojeve, na osnovu čijih vrednosti se određuje jedinstveni redosled po kome će se čvorovima dodeljivati boje. Princip dobijanja redosleda čvorova ostaje nepromenjen: čvor v_i ima prednost za dodelu boja u odnosu na čvor v_j ukoliko je $x(i) < x(j)$.

Glavni nedostatak ovakvog pristupa je što se može desiti da se kroz iteracije menjaju vektori rešenja x, ali da zapravo ni u jednom koraku ne dolazi do promene samog redosleda čvorova. Na primer, koristeći ovakvu reprezentaciju rešenja, jasno je da vektori $x_1 = [2.9, 4.5, 1.3]$ i $x_2 = [3.1, 4.4, 2.3]$ zadaju isti redosled dodeljivanja boja čvorovima grafa. Kako je $1.3 < 2.9 < 4.5$, odnosno $2.3 < 3.1 < 4.4$, u oba slučaja će se najpre obojiti čvor v_3, a zatim i čvorovi v_1 i v_2, respektivno.

4.2 Faza razmrdavanja

Glavna razlika GaussVNS metode u odnosu na klasičan VNS iz glave 3 je u fazi razmrdavanja. Kako je sada skup dopustivih rešenja drugačiji jer sadrži vektore

realnih vrednosti, sada neće postojati različite okoline N_k, već je ceo prostor rešenja jedna okolina, u kojoj će verovatnoća odabira slučajne tačke zavisiti od „debljine" zvona Gausove krive. Ovo je analogno veličinama okolina N_k, u smislu da će se sa većom verovatnoćom birati tačke koje su blizu x (poput pretrage u malim okolinama N_k), dok je sa druge strane teorijski moguće (sa malom verovatnoćom) doći i do bilo koje tačke u prostoru pretrage, što sa ograničenim okolinama N_k nije moguće osim u slučaju da je $N_{k_{\max}}$ ceo prostor pretrage.

U VNS metodi opisanoj u glavi 3 se koristi uniformna raspodela za generisanje slučajnih vrednosti, na osnovu kojih se u fazi razmrdavanja menja vektor x (videti podsekciju 3.3). Za razliku od ovog pristupa, u GaussVNS se korišćenjem n-dimenzionalne Gausove raspodele određuju slučajne vrednosti koje će uticati na promenu vektora x u koraku razmrdavanja. Parametar k, $k = 1, 2, ..., k_{\max}$, sada ima ulogu indeksa koji označava koji element iz niza unapred definisanih disperzija $VNS\_Radijus$ će se koristiti u datoj iteraciji. Kako bi se od x dobio novi vektor x' u koraku razmrdavanja, najpre je potrebno formirati n nezavisnih vrednosti $z_1, z_2, ..., z_n$ dobijenih Gausovom raspodelom sa očekivanjem 0 i disperzijom 1. Svaki od ovih brojeva z_i pomnožimo sa $VNS\_Radius(k)$, koje određuje „debljinu" zvona u tekućoj iteraciji GaussVNS metode. Vektor x' će se dobiti kao $x' = x + z \cdot VNS\_Radijus(k)$.

Koraci faze razmrdavanja GaussVNS metode prikazani su sledećim pseudokodom, gde funkcija $RandNA$ predstavlja Arens-Dieter algoritam za generisanje slučajne tačke sa normalnom raspodelom $N(0, 1)$:

Algoritam 4: *Gausova_raspodela*$(x, VNS\_Radijus[k])$

```
1: //generisanje novog vektora x' na sledeći način:
2: for i ← 0 to n
3:     rdist = RandNA() * VNS_Radijus[k]
4:     x'[i] = x[i] + rdist
5: end for
6: return x'
```

4.3 Generator slučajnih brojeva korišćenjem Gausove raspodele

Važno je da se prilikom implementacije koristi efikasan generator slučajnih brojeva sa Gausovom normalnom raspodelom. Neka je $P_k(x)$ višedimenzionalna Gausova raspodela sa očekivanjem x i matricom kovarijansi Σ_k. Slučajne veličine sa gustinom raspodele $P_k(x)$ mogu se lako dobiti od n nezavisnih vrednosti $z_1, z_2, ..., z_n$ koje su dobijene Gausovom raspodelom sa matematičkim očekivanjem 0 i disperzijom 1, za koju postoje efikasni generatori slučajnih brojeva kao što je Arens-Dieter algoritam [2, 23] koji je korišćen u ovom radu.

Ako je $\Sigma_k = L_k L_k^T$ Čoleski dekompozicija simetrične i pozitivno definitne matrice Σ_k, onda će slučajan vektor u fazi razmrdavanja biti dobijen od x kao $x' = x + L_k z$, gde je $z = (z_1, z_2, ..., z_n)$, vektor sa nezavisnim slučajnim veličinama sa raspodelom $N(0, 1)$. Na osnovu ovoga se generisanje slučajnog vektora korišćenjem $P_k(x)$ svodi na računanje Čoleski dekompozicije matrice Σ_k, a takođe i na generisanje n nezavisnih brojeva korišćenjem normalne raspodele sa očekivanjem 0 i disperzijom 1. Za Čoleski dekompoziciju potrebno je $c \cdot n^3$ računskih operacija u najgorem slučaju.

Ukoliko bi se za matricu kovarijansi Σ_k uzeo umnožak jedinične matrice I, odnosno

$$\Sigma_k = VNS\_Radijus(k)^2 \cdot I, \quad k = 1, 2, ..., k_{\max}, \tag{4.1}$$

postupak bi se dodatno olakšao jer je u ovom slučaju Čoleski dekompozicija $\Sigma_k = (VNS\_Radijus(k) \cdot I)(VNS\_Radijus(k) \cdot I)^T$. Na ovaj način, složenost određivanja matrice L_k za Čoleski dekompoziciju je značajno smanjena jer se postupak svodi samo na računanje umnoška jedinične matrice skalarom $VNS\_Radijus(k)$. Tada se vektor x' u koraku razmrdavanja dobija od x na sledeći način: $x' = x + L_k \cdot z = x + VNS\_Radijus(k) \cdot I \cdot z = x + VNS\_Radijus(k) \cdot z$.

4.4 GaussVNS za MSCP

Inicijalizacija, računanje vrednosti funkcije cilja i lokalna pretraga identično su implementirane kao kod VNS algoritma iz glave 3, jer promena reprezentacije rešenja, kao ni načina generisanja slučajne tačke u koraku razmrdavanja, ne zahteva i modifikaciju ovih ostalih koraka VNS metode.

Posle faze razmrdavanja i dobijanja slučajne tačke iz celog prostora $\mathbb{R}^n$, sledi faza lokalne pretrage, identična onoj opisanoj u poglavlju 3.4, da bi se dobilo poboljšano

rešenje. Donošenje odluke o prelasku u novo rešenje se realizuje po istom principu kao što je opisano za VNS metodu u glavi 3. Koraci faze razmrdavanja i lokalne pretrage se, isto kao u VNS metodi, primenjuju naizmenično sve dok se ne ispuni zadati kriterijum zaustavljanja, koji je u ovom slučaju takođe maksimalan broj iteracija max_{iter} bez poboljšanja rešenja.

Ulazni parametri GaussVNS algoritma su najveći indeks okolina koje treba pretraživati, $k_{\max}$, i maksimalni broj iteracija algoritma bez poboljšanja rešenja, max_{iter}. Dodatni parametri su elementi vektora $VNS\_Radijus$ koji su unapred fiksirani u ovoj implementaciji metode i ne menjaju se u zavisnosti od problema. Koraci implementirane verzije GaussVNS metode prikazani su sledećim pseudokodom:

Algoritam 5: GaussVNS za MSCP

Učitavanje ulaznih podataka n, m i matrice susedstva
Određivanje vektora disperzije $VNS\_Radijus$ dužine $k_{\max}$
Generisanje početnog rešenja x
Ažuriranje $x^* := x$ i $f^* = f(x)$
while $broj\_iteracija < max_{iter}$
    $k = 1$
    while $k \leq k_{\max}$ **do**
        Faza razmrdavanja: $x' = Gausova\_raspodela(x^*, VNS\_Radijus[k])$
        Faza lokalne pretrage: $x'' = lokalna\_pretraga(x')$
        if $f(x'') < f^*$
            $x^* := x''$
            $f^* = f(x'')$
            $k = 1$
            $broj\_iteracija = 0$
        else
            $k = k + 1$
    end while
end while

Glava 5

Eksperimentalni rezultati

5.1 Opis instanci

Za potrebe testiranja preuzeto je ukupno 87 instanci. Od toga, 32 instance su generisane za DIMACS izazov, a 55 instanci korišćene su za COLOR 2002–2004 takmičenje. U odnosu na DIMACS, grafovi koji spadaju u grupu COLOR instanci dosta su jednostavniji u pogledu topologije i gustine. Među ovim instancama se nalaze grafovi koji su slučajno generisani (DSJC.n.m, $n \in \{125, 250, 500, 1000\}$, $m \in \{1, 5, 9\}$), grafovi koji predstavljaju probleme alokacije registara (fpsol2_i_a, inithx_i_a, zeroin_i_a, mulsol_i_b, $a \in 1, 2, 3$, $b \in \{1, 2, 3, 4, 5\}$), grafovi za probleme raspoređivanja (school1 i school1_nsh), grafovi za koje je teško odrediti dekompoziciju na klike (mugn_a, $n \in \{88, 100\}$, $a \in \{1, 25\}$) i drugi.

Pomenute instance dostupne su na `https://mat.tepper.cmu.edu/COLOR02/`. Dimenziju instance određuje broj čvorova n zadatog grafa G, pa je izvršena podela preuzetih instanci na 3 grupe:

- male instance, sa brojem čvorova do 100;
- srednje instance, sa brojem čvorova od 100 do 500;
- velike instance, sa brojem čvorova preko 500.

Malih instanci ima ukupno 17, srednjih 51 i velikih 19. Svaka instanca sadrži redom podatke o broju čvorova n i broju grana m, nakon čega slede uređeni parovi (v_i, v_j) susednih čvorova v_i i v_j. Kako se za rešavanje MSCP koriste neusmereni grafovi, na osnovu podataka o susednim čvorovima formira se simetrična matrica

susedstva $A = (a_{ij})$, za koju važi sledeće:

$$a_{ij} = \begin{cases} 1, & (v_i, v_j) \in E(G) \\ 0, & (v_i, v_j) \notin E(G). \end{cases}$$

5.2 Eksperimentalni rezultati egzaktne metode

Matematički model celobrojnog linearnog programiranja (2.9) – (2.12), predložen u radu [24], implementiran je u programskom jeziku C a rešavan je pomoću IBM ILOG CPLEX 12.8 egzaktnog rešavača. U tabelama 5.1 – 5.3 su prikazani eksperimentalni rezultati dobijeni za instance malih, srednjih i velikih dimenzija, respektivno. Testiranja su vršena na računaru sa procesorom Intel Core i5-8265U 1.8 GHz i 8GB RAM memorije. Ukupno vreme izvršavanja rešavača ograničeno je na 2 sata po instanci i prikazano je u sekundama.

Tabele 5.1 – 5.3 su organizovane na sledeći način: u prvoj koloni dat je naziv instance, druga i treća kolona, označene sa n i m, sadrže broj čvorova i broj grana grafa odgovarajuće instance, respektivno. U četvrtoj koloni se nalazi dobijen rezultat, peta kolona sadrži vreme (u sekundama) za koje je CPLEX rešavač došao do odgovarajućeg rezultata, a poslednje tri kolone sadrže podatke o broju čvorova, iteracija i *gap*-u (u procentima). Kod instanci za koje ni za 2 sata nije pronađeno optimalno rešenje, *gap* vrednost je različita od nule. Ukoliko je program prekinuo sa radom usled nedostatka memorije, u odgovarajućoj vrsti svuda stoji oznaka '–'.

Tabela 5.1: Rezultati CPLEX rešavača dobijeni za male instance ($n < 100$).

Instanca	n	m	opt_{sol}	Vreme(s)	Broj čvorova	Broj iteracija	gap(%)
2Insertions3.txt	37	72	62	0.1820	12	1118	0.0000
3Insertions3.txt	56	110	92	0.2280	19	1747	0.0000
david.txt	87	812	237	0.1820	0	419	0.0000
huck.txt	74	602	243	0.0820	0	186	0.0000
jean.txt	80	508	217	0.0590	0	316	0.0000
mug88_1.txt	88	146	178	0.7550	1523	13910	0.0000
mug88_25.txt	88	146	178	0.9250	2185	21847	0.0000
myciel3.txt	11	20	21	0.0360	7	347	0.0000
myciel4.txt	23	71	45	0.1560	57	1522	0.0000
myciel5.txt	47	236	93	1.7831	365	12161	0.0000
myciel6.txt	95	755	189	169.9531	13792	575261	0.0000
queen5_5.txt	25	320	75	0.0430	0	231	0.0000
queen6_6.txt	36	580	138	203.2141	97512	5192259	0.0000
queen7_7.txt	49	952	196	0.2400	0	1016	0.0000
queen8_8.txt	64	1456	–	–	–	–	–
queen8_12.txt	96	2736	624	10.9461	661	76352	0.0000
queen9_9.txt	81	2112	–	–	–	–	–

Dobijeni rezultati u tabeli 5.1 pokazuju da se za skoro sve male instance na ovaj način mogu dobiti optimalna rešenja. CPLEX dostiže optimalna rešenja za

15 od 17 instanci ($gap = 0\%$), dok za dve instance rešenje nije dostignuto usled nedostatka memorije. Za 12 instanci ukupno vreme za dobijanje optimalnih rešenja je manje od 2s, dok je za preostale 3 instance (queen8_12, myciel6 i queen6_6) bilo potrebno dosta više vremena (10.946s, 169.953s i 203.214s, respektivno), kao i dosta više iteracija.

Tabela 5.2: Rezultati CPLEX rešavača dobijeni za srednje instance ($100 \leq n < 500$).

Instanca	n	m	opt_{sol}	Vreme(s)	Broj čvorova	Broj iteracija	$gap(\%)$
anna.txt	138	986	276	0.2380	0	336	0.0000
DSJC125.1.txt	125	736	–	–	–	–	–
DSJC125.5.txt	125	3891	1322	7105.1251	1450	3481392	0.5843
DSJC125.9.txt	125	6961	1891.7430	7173.3861	3388	4866550	0.1806
DSJC250.1.txt	250	3218	1184	7144.6611	1334	2236577	0.4965
DSJC250.5.txt	250	15668	4587	7152.5981	0	396037	0.7287
DSJC250.9.txt	250	27897	–	–	–	–	–
flat300_20_0.txt	300	21375	–	–	–	–	–
flat300_26_0.txt	300	21633	–	–	–	–	–
flat300_28_0.txt	300	21695	–	–	–	–	–
fpsol2_i_1.txt	496	11654	3403	33.5891	0	8637	0.0000
fpsol2_i_2.txt	451	8691	1668	41.4571	0	2836	0.0000
fpsol2_i_3.txt	425	8688	1636	40.0581	0	2599	0.0000
games120.txt	120	1276	443	1.3500	539	17202	0.0000
le450_5a.txt	450	5714	1414	7117.1440	239	1473655	0.1171
le450_5b.txt	450	5734	1371	7161.5500	594	3118105	0.0926
le450_5c.txt	450	9803	1350	7172.8230	105	1367022	0.0115
le450_5d.txt	450	9757	1351	7122.2550	27	545213	0.0158
le450_15a.txt	450	8168	–	–	–	–	–
le450_15b.txt	450	8169	3436	7127.1891	729	1501112	0.2809
le450_15c.txt	450	16680	5302	7170.5811	81	1062334	0.4505
le450_15d.txt	450	16750	5421	7171.1231	85	1139524	0.4606
le450_25a.txt	450	8260	–	–	–	–	–
le450_25b.txt	450	8263	–	–	–	–	–
le450_25c.txt	450	17343	5897	7103.1841	138	695084	0.3138
le450_25d.txt	450	17425	6096	7176.3171	378	1731999	0.3333
miles250.txt	128	774	325	0.1980	8	1407	0.0000
miles500.txt	128	2340	705	19.1340	1102	113785	0.0000
miles750.txt	128	4226	1173	191.0750	2839	724194	0.0000
miles1000.txt	128	6432	1666	137.3530	523	404464	0.0000
miles1500.txt	128	10396	3260	137.5040	484	130061	0.0000
mug100_1.txt	100	166	202	0.8500	2249	23622	0.0000
mug100_25.txt	100	166	202	1.8100	4726	53458	0.0000
mulsol_i_1.txt	197	3925	1957	2.5110	0	3096	0.0000
mulsol_i_2.txt	188	3885	1191	3.3930	0	2280	0.0000
mulsol_i_3.txt	184	3916	1187	3.6390	0	2173	0.0000
mulsol_i_4.txt	185	3946	1189	3.5290	0	2141	0.0000
mulsol_i_5.txt	186	3973	1160	3.8160	0	2155	0.0000
myciel7.txt	191	2360	–	–	–	–	–
queen10_10.txt	100	2940	–	–	–	–	–
queen11_11.txt	121	3960	–	–	–	–	–
queen12_12.txt	144	5192	–	–	–	–	–
queen13_13.txt	169	6656	–	–	–	–	–
queen14_14.txt	196	8372	–	–	–	–	–
queen15_15.txt	225	10360	1944	7170.4850	25897	14899730	0.0740
queen16_16.txt	256	12640	2270	7175.9680	10685	11853178	0.0410
school1.txt	385	19095	–	–	–	–	–
school1-nsh.txt	352	14612	–	–	–	–	–
zeroin_i_1.txt	211	4100	1822	3.0281	0	4215	0.0000
zeroin_i_2.txt	211	3541	1004	3.4881	0	2226	0.0000
zeroin_i_3.txt	206	3540	998	3.3621	0	2405	0.0000

Eksperimenti pokazuju da se vreme izvršavanja i memorijski zahtevi CPLEX rešavača vrlo brzo povećavaju sa povećavanjem dimenzije problema. Iz tabele 5.2 se može uočiti da je za instance srednjih dimenzija CPLEX rešavač dostigao optimalna rešenja za 20 od ukupno 51 instance ($gap = 0\%$). Za 15 instanci vrednost gap je različita od nule, što znači da je za 2h CPLEX rešavač uspeo da dođe samo do dopustivih rešenja. Za preostalih 16 instanci usled nedostatka memorije nije uspeo da stigne ni do dopustivih rešenja za te primere.

Tabela 5.3: Rezultati CPLEX rešavača dobijeni za velike instance ($n \geq 500$).

Instanca	n	m	sol	Vreme(s)	Broj čvorova	Broj iteracija	gap(%)
DSJC500.1.txt	500	12458	4110	7169.2391	20	817439	0.6889
DSJC500.5.txt	500	62624	–	–	–	–	–
DSJC500.9.txt	500	224874	–	–	–	–	–
DSJC1000.1.txt	1000	49629	–	–	–	–	–
DSJC1000.5.txt	1000	249826	–	–	–	–	–
DSJC1000.9.txt	1000	449449	–	–	–	–	–
DSJR500.1.txt	500	3555	2142	7171.1511	27484	20678046	0.0074
DSJR500.1c.txt	500	121275	–	–	–	–	–
DSJR500.5.txt	500	58862	–	–	–	–	–
flat1000_50_0.txt	1000	245000	–	–	–	–	–
flat1000_60_0.txt	1000	245830	–	–	–	–	–
flat1000_76_0.txt	1000	246708	–	–	–	–	–
homer.txt	561	3258	1150	38.1061	579	45014	0.0000
inithx_i_1.txt	864	18707	–	–	–	–	–
inithx_i_2.txt	645	13979	–	–	–	–	–
inithx_i_3.txt	621	13969	–	–	–	–	–
latin_square_10.txt	900	307350	–	–	–	–	–
qg.order30.txt	900	26100	13950	484.6691	15	108196	0.0000
qg.order40.txt	1600	62400	–	–	–	–	–

Za instance velikih dimenzija se egzaktni rešavać nije pokazao uspešnim. Iz tabele 5.3 vidi se da je CPLEX rešavač uspeo da dođe do optimalnih rešenja za samo 2 od 19 instanci, dok je takođe za 2 instance nakon 2h izvršavanja došao samo do dopustivog rešenja. Za preostalih 15 instanci izvršavanje je prekinuto usled nedostatka memorijskog prostora.

5.3 Eksperimentalni rezultati dobijeni predloženim VNS i GaussVNS metodama za MSCP

U ovom odeljku su prikazani rezultati VNS i GaussVNS algoritma, koji su upoređeni sa optimalnim rezultatima dobijem CPLEX-om, kao i sa rezultatima iz literature za različite metaheuristike. Opisani algoritmi iz poglavlja 3 i 4 implementirani su u programskom jeziku C++. Sva testiranja izvršena su na računaru sa procesorom Intel Core i5-8265U 1.8 GHz i 8GB RAM memorije.

Zbog svoje stohastičke prirode, oba algoritma su za svaku instancu pokretana po 20 puta. Jedina razlika u više pokretanja za isti test primer je u vrednostima koje će se koristiti za generisanje slučajnih brojeva (seed-ovima). U tabelama 5.4 – 5.9 prikazani su rezultati dobijeni primenom VNS i GaussVNS metode dobijeni za instance malih, srednjih i velikih dimenzija MSCP problema. Kolone tabela 5.4, 5.6 i 5.8 (samo za VNS) sadrže redom sledeće vrednosti:

- instanca – naziv instance;
- n – broj čvorova zadatog grafa;
- m – broj grana zadatog grafa;
- CPLEX – optimalno rešenje dobijeno egzaktnim rešavačem; ukoliko nakon dva sata program nije došao do optimalnog rešenja sa *gap*–om jednakim 0, pored rezultata stoji '*', a ako je izvršavanje prekinuto usled nedostatka memorije, stoji '–';
- CPLEX_t – ukupno vreme koje je bilo potrebno CPLEX rešavaču da dođe do optimalnog rešenja; ukoliko nakon dva sata program nije došao do optimalnog rešenja sa *gap*–om jednakim 0 ili je izvršavanje prekinuto usled nedostatka memorije, stoji '–';
- $best_{sol}$ VNS – vrednost funkcije cilja najboljeg nađenog rešenja u 20 izvršavanja algoritma VNS; ukoliko algoritam dostiže optimalno rešenje koje se nalazi u koloni CPLEX, umesto vrednosti funkcije cilja stoji oznaka „opt";
- t_{tot} VNS – prosečno vreme izvršavanja potrebno da bi se odredilo najbolje rešenje VNS-om (u sekundama);

- t_{best} VNS – prosečno vreme za koje se prvi put dobilo najbolje rešenje $best_{sol}$ VNS, tzv. srednje početno vreme nalaženja najboljeg rešenja algoritma VNS (u sekundama);

- $iter_{best}$ VNS – prosečan broj iteracija za koje se dobilo prvi put najbolje rešenje $best_{sol}$ VNS;

- $agap$ VNS – prosečno procentualno odstupanje vrednosti nađenih rešenja u odnosu na vrednost sol, koja predstavlja ili optimalno rešenje (ako je poznato) ili najbolje nađeno rešenje $best_{sol}$ VNS-a (u procentima)
  $agap = \frac{1}{k}\sum_{i=1}^{k} gap_i$, gde je $gap_i = 100 \cdot \frac{|sol_i - best_{sol}|}{|best_{sol}|}$;

- σ VNS – standardna devijacija prosečnog odstupanja vrednosti nađenih rešenja algoritma VNS (u procentima)
  $\sigma = \sqrt{\frac{1}{k}\sum_{i=1}^{k}(gap_i - agap)^2}$.

Tabele 5.5, 5.7 i 5.9 organizovane su na isti način, samo se odnose na rezultate dobijene GaussVNS metodom.

Parametri VNS i GaussVNS algoritama su $k_{\max}$ i max_{iter}, i oni su eksperimentalnim putem određeni u zavisnosti od dimenzije instance. Kod malih instanci je uzeto da je $k_{\max} = 5$ i $max_{iter} = 100$, kod srednjih je $k_{\max} = 5$ i $max_{iter} = 200$, a kod velikih instanci je $k_{\max} = 7$ i $max_{iter} = 200$. Kao kriterijum zaustavljanja kod metoda VNS i GaussVNS korišćen je maksimalni broj dozvoljenih iteracija max_{iter} bez poboljšanja rešenja. Informacija o prosečnom ukupnom broju iteracija za obe metode je izostavljena iz tabela jer se ona može dobiti kao zbir $iter_{best} + max_{iter}$ za svaki test primer.

Za generisanje slučajnih tačaka uniformnom raspodelom kod VNS metode u i-tom pokretanju algoritma ($i = 1, 2, ..., 20$) korišćen je isti seed kao i za generisanje slučajnih tačaka Gausovom raspodelom kod GausVNS metode u i-tom pokretanju. Za vektor disperzija kod Gauss-VNS-a korišćen je vektor $VNS\_Radijus = [0.05, 0.1, 0.2, 0.5, 1, 2, 3]$, pri čemu u slučaju da je $k_{\max} = 5$ koriste se samo prvih 5 vrednosti, a u slučaju $k_{\max} = 7$ svih sedam vrednosti. Akcenat je bio na poređenju ove dve metode, te su iz tog razloga za testiranje istih instanci kod obe korišćeni isti parametri.

Iz tabele 5.4 može se videti da je VNS metoda dostigla optimalno rešenje za 14 od 15 poznatih optimalnih rešenja za instance malih dimenzija za relativno kratko vreme izvršavanja. Budući da za dve instance CPLEX nije dobio optimalno rešenje,

Tabela 5.4: Poređenje rezultata CPLEX-a i VNS-a za male instance ($n < 100$).

Instanca	n	m	CPLEX	CPLEX_ $t(s)$	$best_{sol}$ VNS	t_{tot} VNS(s)	t_{best} VNS(s)	$iter_{best}$ VNS	$agap$ VNS(%)	σ VNS(%)
2Insertions3.txt	37	72	62	0.1820	opt	0.1062	0.0087	14.8000	0.0000	0.0000
3Insertions3.txt	56	110	92	0.2280	opt	0.1774	0.0512	35.3000	0.0033	0.0078
david.txt	87	812	237	0.1820	opt	1.2928	0.6672	103.0500	0.0152	0.0085
huck.txt	74	602	243	0.0820	opt	0.4472	0.0815	20.8000	0.0002	0.0009
jean.txt	80	508	217	0.0590	opt	0.5824	0.2237	60.4500	0.0018	0.0023
mug88_1.txt	88	146	178	0.7550	opt	0.4824	0.1021	25.3500	0.0020	0.0027
mug88_25.txt	88	146	178	0.9250	opt	0.4922	0.1209	31.7500	0.0022	0.0028
myciel3.txt	11	20	21	0.0360	opt	0.0065	0.0003	3.4500	0.0000	0.0000
myciel4.txt	23	71	45	0.1560	opt	0.0274	0.0036	10.4500	0.0000	0.0000
myciel5.txt	47	236	93	1.7831	opt	0.1754	0.0473	32.4000	0.0005	0.0023
myciel6.txt	95	755	189	169.9531	opt	1.4186	0.4888	51.3000	0.0011	0.0021
queen5_5.txt	25	320	75	0.0430	opt	0.0556	0.0020	2.2500	0.0000	0.0000
queen6_6.txt	36	580	138	203.2141	opt	0.2265	0.0988	73.1000	0.0123	0.0119
queen7_7.txt	49	952	196	0.2400	opt	0.5049	0.1915	68.0000	0.0538	0.0216
queen8_8.txt	64	1456	–	–	307	1.0245	0.4517	77.5000	0.0194	0.0110
queen8_12.txt	96	2736	624	10.9461	626	2.2882	0.8330	56.2500	0.0062	0.0017
queen9_9.txt	81	2112	–	–	426	1.8109	0.7572	71.1500	0.0353	0.0107

ne može se zaključiti ništa o optimalnosti rešenja za instance queen8_8 i queen9_9. Samo za jednu instancu, queen8_12, VNS je uspeo da pronađe rešenje blisko optimalnom. Vreme izvršavanja VNS-a je kod 10 instanci kraće, ali i kod 5 instanci duže od CPLEX-a. Najduže vreme VNS metode bilo je potrebno za instancu queen8_12 i iznosi 2.2882 s. Vrednosti *agap* i σ govore o kvalitetu dobijenih rešenja i za sve instance *agap* je manja od 0.0538%, dok je vrednost σ manja od 0.0216%. Za 4 instance ove obe vrednosti su 0, što znači da je u svih 20 izvršavanja algoritam dostigao isto (optimalno) resenje.

Kod GaussVNS metode se iz tabele 5.5 može takođe videti da je za instance malih dimenzija dostignuto optimalno rešenje za 14 od 15 poznatih optimalnih rešenja za relativno kratko vreme izvršavanja. Za instance queen8_8 i queen9_9 se takođe ne može zaključiti ništa o optimalnosti, ali je GaussVNS dostigao bolja rešenja, 304 i 424, u odnosu na VNS, 307 i 426, respektivno. U slučaju instance queen8_12, VNS je uspeo da pronađe rešenje 626 a GaussVNS rešenje 625, što je bliže optimalnom rešenju koje iznosi 624. Na osnovu vremena izvršavanja primećuje se da GaussVNS zanemarljivo sporije od VNS-a dolazi do rešenja, dok je u poređenju sa CPLEX-om za 10 instanci kraće, a za 5 instanci duže od CPLEX-a, kao i kod VNS-a. Najduže vreme GaussVNS metode bilo je potrebno za instancu queen8_12 i iznosi 3.3490 s. Vrednost *agap* je svuda manja od 0.0270%, dok je vrednost σ manja od 0.0235%. Za 5 instanci je $agap = 0$ i $\sigma = 0$.

Kod srednjih instanci, iz tabele 5.6 može se primetiti da je VNS metoda našla optimalno rešenje za 11 od 20 instanci za koje su poznata optimalna rešenja. Za 9 instanci nije dostignuto optimalno rešenje, međutim, svih 9 rezultata jesu bliska optimalnom. Pošto CPLEX nije dobio optimalno rešenje za 31 instancu srednjih dimenzija, ne može se zaključiti ništa o optimalnosti rešenja dobijenih VNS metodom. Vreme izvršavanja VNS-a je kod 5 instanci kraće, ali i kod 15 instanci duže od CPLEX-a. Najduže vreme VNS metode bilo je potrebno za instancu le450_5d i iznosi 1678.09 s. Vrednost *agap* je svuda manja od 0.2283%, dok je vrednost σ manja od 0.0765%.

Iz tabele 5.7 vidi se da je GaussVNS metoda našla optimalno rešenje za 12 od 20 instanci za koje su poznata optimalna rešenja, što je za 1 instancu više u odnosu na VNS. Za 8 instanci, za koje je CPLEX došao do optimalnog rešenja a GaussVNS nije, primećuje se da je našao rešenja bliža optimalnom u poređenju sa rezultatima VNS metode iz tabele 5.6. Ovde se takođe ništa ne može zaključiti o optimalnosti ukupno 31 rešenja, međutim, za 30 od 31 instance je GaussVNS dostigao bolja

Tabela 5.5: Poređenje rezultata CPLEX-a i GaussVNS-a za male instance ($n < 100$).

Instanca	n	m	CPLEX	CPLEX_ $t(s)$	$best_{sol}$ Gauss	t_{tot} Gauss(s)	t_{best} Gauss(s)	$iter_{best}$ Gauss	$agap$ Gauss(%)	σ Gauss(%)
2Insertions3.txt	37	72	62	0.1820	opt	0.1211	0.0112	17.7500	0.0032	0.0097
3Insertions3.txt	56	110	92	0.2280	opt	0.1807	0.0304	17.7500	0.0049	0.0087
david.txt	87	812	237	0.1820	opt	1.3674	0.5729	81.8000	0.0095	0.0077
huck.txt	74	602	243	0.0820	opt	0.4498	0.0492	12.3000	0.0000	0.0000
jean.txt	80	508	217	0.0590	opt	0.6088	0.1869	43.8500	0.0012	0.0020
mug88_1.txt	88	146	178	0.7550	opt	0.6117	0.1775	41.8000	0.0000	0.0000
mug88_25.txt	88	146	178	0.9250	opt	0.5429	0.1195	29.4000	0.0000	0.0000
myciel3.txt	11	20	21	0.0360	opt	0.0121	0.0008	4.4000	0.0000	0.0000
myciel4.txt	23	71	45	0.1560	opt	0.0395	0.0070	17.9500	0.0044	0.0089
myciel5.txt	47	236	93	1.7831	opt	0.2408	0.0738	41.8000	0.0075	0.0108
myciel6.txt	95	755	189	169.9531	opt	1.4579	0.5239	52.3000	0.0069	0.0084
queen5_5.txt	25	320	75	0.0430	opt	0.0606	0.0031	4.3500	0.0000	0.0000
queen6_6.txt	36	580	138	203.2141	opt	0.2733	0.1199	76.9500	0.0043	0.0074
queen7_7.txt	49	952	196	0.2400	opt	0.6159	0.2645	74.2500	0.0199	0.0235
queen8_8.txt	64	1456	–	–	304	1.4891	0.7055	90.3500	0.0215	0.0113
queen8_12.txt	96	2736	624	10.9461	625	3.3490	1.4308	76.5000	0.0065	0.0021
queen9_9.txt	81	2112	–	–	424	2.6416	1.1816	80.0500	0.0270	0.0132

Tabela 5.6: Poređenje rezultata CPLEX-a i VNS-a za srednje instance ($100 \leq n < 500$).

Instanca	n	m	CPLEX	CPLEX_$t(s)$	$best_{sol}$ VNS	t_{tot} VNS(s)	t_{best} VNS(s)	$iter_{best}$ VNS	$agap$ VNS(%)	σ VNS(%)
anna.txt	138	986	276	0.2380	opt	6.4871	2.8882	159.6500	0.0065	0.0054
DSJC125.1.txt	125	736	–	–	337	9.8884	5.4317	238.9500	0.0338	0.0147
DSJC125.5.txt	125	3891	1322*	–	1146	22.9935	10.5837	171.5000	0.0208	0.0087
DSJC125.9.txt	125	6961	1891.7430*	–	2632	52.6321	27.0921	209.0000	0.0243	0.0109
DSJC250.1.txt	250	3218	1184*	–	1112	79.8267	34.8133	155.6000	0.0172	0.0083
DSJC250.5.txt	250	15668	4587*	–	4042	233.8051	117.5511	202.1000	0.0098	0.0077
DSJC250.9.txt	250	27897	–	–	9404	562.5581	277.1971	188.2500	0.0260	0.0124
flat300_20_0.txt	300	21375	–	–	5104	453.5731	209.1161	169.8500	0.0377	0.0121
flat300_26_0.txt	300	21633	–	–	5423	352.1651	156.8121	159.9500	0.0172	0.0093
flat300_28_0.txt	300	21695	–	–	5325	319.1851	121.6651	120.7500	0.0314	0.0098
fpsol2_i_1.txt	496	11654	3403	33.5891	3419	1308.350	691.0710	231.0000	0.0119	0.0038
fpsol2_i_2.txt	451	8691	1668	41.4571	1680	856.6420	465.4600	239.2500	0.0147	0.0056
fpsol2_i_3.txt	425	8688	1636	40.0581	1646	999.3550	575.2321	266.6500	0.0117	0.0032
games120.txt	120	1276	443	1.3500	445	6.2890	2.6428	137.2000	0.0129	0.0049
le450_5a.txt	450	5714	1414*	–	1493	774.9371	451.9461	281.4500	0.2283	0.0765
le450_5b.txt	450	5734	1371*	–	1594	943.6631	568.8421	304.4000	0.1609	0.0682
le450_5c.txt	450	9803	1350*	–	1443	1429.3600	806.5671	252.1000	0.0690	0.0490
le450_5d.txt	450	9757	1351*	–	1413	1678.0900	1093.9900	369.7000	0.0691	0.0504
le450_15a.txt	450	8168	–	–	3060	513.5440	246.3890	182.7500	0.0129	0.0054
le450_15b.txt	450	8169	3436*	–	3080	527.6660	237.3340	155.2000	0.0110	0.0062
le450_15c.txt	450	16680	5302*	–	4797	780.6880	324.6750	144.9500	0.0106	0.0062
le450_15d.txt	450	16750	5421*	–	4830	670.3280	279.7600	140.6500	0.0093	0.0040
le450_25a.txt	450	8260	–	–	3557	814.7630	494.3710	305.5000	0.0100	0.0067
le450_25b.txt	450	8263	–	–	3732	694.6280	356.5830	205.3500	0.0214	0.0076
le450_25c.txt	450	17343	5897*	–	5426	762.8220	391.4090	211.1000	0.0101	0.0045
le450_25d.txt	450	17425	6096*	–	5457	752.0860	374.6410	199.0000	0.0132	0.0058
miles250.txt	128	774	325	0.1980	334	6.8280	2.9212	162.1500	0.0372	0.0051
miles500.txt	128	2340	705	19.1341	736	14.6275	7.3176	197.5500	0.0173	0.0091
miles750.txt	128	4226	1173	191.0751	1224	26.9406	16.0326	288.9500	0.0599	0.0078
miles1000.txt	128	6432	1666	137.3531	1775	32.9201	16.9561	207.3500	0.0827	0.0079
miles1500.txt	128	10396	3260	137.5041	3371	47.9351	26.7901	251.0000	0.0432	0.0042
mug100_1.txt	100	166	202	0.8500	opt	1.5638	0.5067	90.1000	0.0007	0.0018
mug100_25.txt	100	166	202	1.8101	opt	1.4603	0.4127	76.4500	0.0017	0.0024
mulsol_i_1.txt	197	3925	1957	2.5111	opt	47.1143	16.2130	108.5500	0.0003	0.0004
mulsol_i_2.txt	188	3885	1191	3.3931	opt	45.9897	16.4563	107.7000	0.0005	0.0007
mulsol_i_3.txt	184	3916	1187	3.6391	opt	44.3265	17.2283	124.9500	0.0010	0.0007
mulsol_i_4.txt	185	3946	1189	3.5291	opt	44.0703	15.6500	105.9500	0.0007	0.0010
mulsol_i_5.txt	186	3973	1160	3.8161	opt	50.8190	19.8853	126.4000	0.0007	0.0008
myciel7.txt	191	2360	–	–	381	23.3614	8.9593	119.7500	0.0035	0.0035
queen10_10.txt	100	2940	–	–	588	7.6932	3.6438	179.7000	0.0272	0.0119
queen11_11.txt	121	3960	–	–	795	10.0934	3.4015	101.2000	0.0140	0.0067
queen12_12.txt	144	5192	–	–	1023	19.4236	8.6706	161.1500	0.0158	0.0069
queen13_13.txt	169	6656	–	–	1287	36.1536	19.3608	227.4500	0.0222	0.0081
queen14_14.txt	196	8372	–	–	1619	46.7990	21.7190	169.3500	0.0128	0.0064
queen15_15.txt	225	10360	1944*	–	1971	76.1423	38.7828	205.2000	0.0147	0.0067
queen16_16.txt	256	12640	2270*	–	2397	97.9359	45.1923	167.9500	0.0115	0.0053
school1.txt	385	19095	–	–	3343	1312.1100	728.0941	252.0500	0.1403	0.0573
school1-nsh.txt	352	14612	–	–	3490	720.2130	397.6480	248.0000	0.0797	0.0402
zeroin_i_1.txt	211	4100	1822	3.0281	opt	37.6065	15.4623	130.2000	0.0002	0.0004
zeroin_i_2.txt	211	3541	1004	3.4881	opt	44.3000	21.9909	195.7500	0.0020	0.0015
zeroin_i_3.txt	206	3540	998	3.3621	opt	47.4201	24.8874	225.4000	0.0019	0.0019

Tabela 5.7: Poređenje rezultata CPLEX-a i GaussVNS-a za srednje instance ($100 \leq n < 500$).

Instanca	n	m	CPLEX	CPLEX_$t(s)$	$best_{sol}$ Gauss	t_{tot} Gauss(s)	t_{best} Gauss(s)	$iter_{best}$ Gauss	$agap$ Gauss(%)	σ Gauss(%)
anna.txt	138	986	276	0.2380	opt	4.1120	1.3762	105.8500	0.0043	0.0037
DSJC125.1.txt	125	736	–	–	329	10.6122	5.6591	227.3500	0.0327	0.0155
DSJC125.5.txt	125	3891	1322*	–	1096	34.1242	17.6851	220.1000	0.0355	0.0157
DSJC125.9.txt	125	6961	1891.7430*	–	2594	61.1606	34.0003	256.3000	0.0198	0.0117
DSJC250.1.txt	250	3218	1184*	–	1064	136.0190	82.9612	320.5500	0.0231	0.0112
DSJC250.5.txt	250	15668	4587*	–	3869	250.9640	119.3430	184.4000	0.0288	0.0116
DSJC250.9.txt	250	27897	–	–	9010	508.0940	266.9950	225.9000	0.0296	0.0148
flat300_20_0.txt	300	21375	–	–	4549	708.5460	464.5820	386.5000	0.0631	0.0302
flat300_26_0.txt	300	21633	–	–	5242	421.6990	192.9860	174.6500	0.0218	0.0095
flat300_28_0.txt	300	21695	–	–	5239	471.1430	215.1090	169.1500	0.0203	0.0104
fpsol2_i_1.txt	496	11654	3403	33.5891	3408	575.8070	318.5130	257.7500	0.0065	0.0040
fpsol2_i_2.txt	451	8691	1668	41.4571	1670	440.0720	273.806	327.1000	0.0071	0.0033
fpsol2_i_3.txt	425	8688	1636	40.0581	1639	315.4580	162.6520	221.6000	0.0086	0.0086
games120.txt	120	1276	443	1.3501	opt	6.1215	2.9576	190.8000	0.0064	0.0044
le450_5a.txt	450	5714	1414*	–	1438	1237.6600	774.0840	341.2500	0.0542	0.0310
le450_5b.txt	450	5734	1371*	–	1432	1013.6500	588.1520	279.0000	0.0799	0.0441
le450_5c.txt	450	9803	1350*	–	1376	957.6150	517.4770	239.1000	0.0210	0.0122
le450_5d.txt	450	9757	1351*	–	1375	847.7760	395.8770	185.0000	0.0307	0.0227
le450_15a.txt	450	8168	–	–	2969	633.0920	315.6160	200.6500	0.0235	0.0082
le450_15b.txt	450	8169	3436*	–	3020	600.4980	283.2420	184.7500	0.0147	0.0068
le450_15c.txt	450	16680	5302*	–	4641	803.9280	349.4390	157.4000	0.0198	0.0062
le450_15d.txt	450	16750	5421*	–	4567	840.5340	353.2050	145.2000	0.0437	0.0125
le450_25a.txt	450	8260	–	–	3464	718.8500	383.9210	227.9000	0.0160	0.0067
le450_25b.txt	450	8263	–	–	3669	614.7280	320.9130	216.5000	0.0156	0.0066
le450_25c.txt	450	17343	5897*	–	5319	798.3780	365.1050	169.0000	0.0131	0.0051
le450_25d.txt	450	17425	6096*	–	5339	713.9610	280.6570	131.3500	0.0185	0.0072
miles250.txt	128	774	325	0.1980	328	5.3867	2.3600	154.3000	0.0245	0.0079
miles500.txt	128	2340	705	19.1341	724	16.8636	9.6533	273.4000	0.0472	0.0074
miles750.txt	128	4226	1173	191.0751	1219	25.0295	13.9780	258.1000	0.0512	0.0059
miles1000.txt	128	6432	1666	137.3531	1747	40.6259	25.3085	339.3000	0.0636	0.0071
miles1500.txt	128	10396	3260	137.5041	3363	36.3190	20.5628	265.3500	0.0376	0.0026
mug100_1.txt	100	166	202	0.8500	opt	1.0458	0.1544	37.2000	0.0005	0.0015
mug100_25.txt	100	166	202	1.8101	opt	1.0047	0.1405	34.2500	0.0012	0.0021
mulsol_i_1.txt	197	3925	1957	2.5111	opt	23.8637	10.9198	172.1500	0.0035	0.0042
mulsol_i_2.txt	188	3885	1191	3.3931	opt	21.0337	8.6186	145.4500	0.0031	0.0034
mulsol_i_3.txt	184	3916	1187	3.6391	opt	21.6667	10.3396	174.0000	0.0045	0.0095
mulsol_i_4.txt	185	3946	1189	3.5291	opt	19.8550	7.5931	127.4500	0.0032	0.0037
mulsol_i_5.txt	186	3973	1160	3.8161	opt	22.7308	9.1103	129.7500	0.0047	0.0050
myciel7.txt	191	2360	–	–	381	15.3424	5.8423	126.2500	0.0062	0.0072
queen10_10.txt	100	2940	–	–	588	9.7485	5.1394	222.4500	0.0201	0.0098
queen11_11.txt	121	3960	–	–	772	48.5748	7.0378	195.8500	0.0259	0.0093
queen12_12.txt	144	5192	–	–	1012	24.2122	11.9909	198.8000	0.0134	0.0085
queen13_13.txt	169	6656	–	–	1283	35.5394	16.0480	167.7000	0.0122	0.0056
queen14_14.txt	196	8372	–	–	1602	48.5565	19.4733	134.0500	0.0103	0.0055
queen15_15.txt	225	10360	1944*	–	1967	77.0356	32.9361	158.5500	0.0083	0.0050
queen16_16.txt	256	12640	2270*	–	2353	114.5720	50.9947	164.2500	0.0191	0.0062
school1.txt	385	19095	–	–	3001	1795.5800	1091.8500	320.5500	0.1317	0.0511
school1_nsh.txt	352	14612	–	–	2936	957.1620	553.8330	287.3500	0.1282	0.0513
zeroin_i_1.txt	211	4100	1822	3.0281	opt	35.2581	14.8717	152.6500	0.0012	0.0021
zeroin_i_2.txt	211	3541	1004	3.4881	opt	26.9322	10.0669	116.2500	0.0077	0.0065
zeroin_i_3.txt	206	3540	998	3.3621	opt	29.6229	13.5709	175.3500	0.0051	0.0056

rešenja u odnosu na VNS, a za jednu instancu (myciel7) su obe metode našla isto rešenje koje iznosi 381. Vreme izvršavanja GaussVNS-a je kod 5 instanci kraće, ali i kod 15 instanci duže od CPLEX-a. Takođe, može se zaključiti da je GaussVNS došao do rešenja za 26 instanci brže, a za 25 instanci sporije u poređenju sa VNS-om. Najduže vreme GaussVNS metode bilo je potrebno za instancu school1 i iznosi 1795.58 s. Vrednost *agap* je uvek manja od 0.1317%, dok je vrednost σ manja od 0.0513%, što je manje u odnosu na VNS.

U tabeli 5.8 su prikazani rezultati VNS metode na instancama velikih dimenzija. Može se primetiti da je VNS metoda našla samo rešenje približno optimalnom za 2 od 19 instanci za koje je poznato optimalno rešenje. Kako CPLEX nije dobio optimalno rešenje za 17 velikih instanci, ne može se zaključiti ništa o optimalnosti rešenja dobijenih VNS metodom. Primećuje se i da je dosta više vremena bilo potrebno kako bi se došlo do najboljeg rešenja, i to je najveće vreme izvršavanja kod instance DSJC1000.9 i iznosi 36714.3 s. Vrednost *agap* je svuda manja od 0.0241%, dok je vrednost σ manja od 0.0069%.

Tabela 5.9 sadrži rezultate GaussVNS metode na instancama velikih dimenzija. Za 2 od 19 instanci za koje je poznato optimalno rešenje može se primetiti sledeće: kod instance homer je GaussVNS dobio bolje rešenje (1159) od VNS-a (1168), pri čemu su oba bliska optimalnom, dok su qg.order30 i qg.order40 jedine 2 instance od ukupno 87 za koje je VNS metoda bila bolja od GaussVNS metode. Manje vremena je trebalo GaussVNS-u da dođe do najboljeg rešenja u odnosu na VNS kod 13, a više kod 6 instanci. Kako CPLEX nije dobio optimalno rešenje za 17 velikih instanci, ne može se zaključiti ništa o optimalnosti rešenja dobijenih GaussVNS metodom, ali važi da je u svih 17 slučajeva GaussVNS dobio bolja rešenja od VNS-a. Primećuje se i da je kao i kod VNS-a dosta više vremena bilo potrebno kako bi se došlo do najboljeg rešenja kod instanci velikih dimenzija, i to najveće vreme izvršavanja kod instance DSJC1000.9 iznosi 59342.8 s. Vrednost *agap* je manja od 0.0251%, dok je vrednost σ manja od 0.0098%, što je zanemarljivo veće nego kod VNS-a.

Poređenjem rezultata izloženih u tabelama može se zaključiti da su performanse GaussVNS metode bolje od VNS-a u vidu boljeg rešenja, manjeg vremena izvršavanja, manjeg *agap* i σ kod instanci malih i srednjih dimenzija. Samo se kod 2 velike instance desilo da je VNS dao bolje rešenje, ali je za jednu od te 2 instance vreme izvršavanja bilo manje u odnosu na GaussVNS.

Tabela 5.8: Poređenje rezultata CPLEX-a i VNS-a za velike instance ($n \geq 500$).

Instanca	n	m	CPLEX	CPLEX_$t(s)$	$best_{sol}$ VNS	t_{tot} VNS(s)	t_{best} VNS(s)	$iter_{best}$ VNS	$agap$ VNS(%)	σ VNS(%)
DSJC500.1.txt	500	12458	4110*	–	3537	685.9910	252.3020	116.9000	0.0132	0.0069
DSJC500.5.txt	500	62624	–	–	14518	1727.6800	659.9640	122.9000	0.0101	0.0055
DSJC500.9.txt	500	224874	–	–	36229	4681.6700	2204.8500	181.5000	0.0135	0.0054
DSJC1000.1.txt	1000	49629	–	–	12004	8674.0800	4281.8700	199.9500	0.0067	0.0038
DSJC1000.5.txt	1000	249826	–	–	52350	19683.8000	9441.3700	183.8500	0.0118	0.0046
DSJC1000.9.txt	1000	449449	–	–	137075	36714.3000	13712.3000	118.6000	0.0112	0.0044
DSJR500.1.txt	500	3555	2142*	–	2340	976.1650	508.2430	218.5500	0.0113	0.0049
DSJR500.1c.txt	500	121275	–	–	17125	16246.1000	9375.0600	275.6000	0.0104	0.0060
DSJR500.5.txt	500	58862	–	–	28871	3048.2700	1519.0600	190.5500	0.0156	0.0062
flat1000_50_0.txt	1000	245000	–	–	51193	18745.4000	8612.6400	165.1000	0.0182	0.0049
flat1000_60_0.txt	1000	245830	–	–	51934	18307.1000	8321.1900	161.5000	0.0059	0.0031
flat1000_76_0.txt	1000	246708	–	–	52078	17206.9000	6882.3600	135.5500	0.0087	0.0041
homer.txt	561	3258	1150	38.1061	1168	426.5130	213.1450	198.0000	0.0241	0.0052
inithx_i_1.txt	864	18707	–	–	3699	7435.0100	3973.7200	221.7000	0.0100	0.0049
inithx_i_2.txt	645	13979	–	–	2070	2352.2300	1204.8700	212.6500	0.0064	0.0042
inithx_i_3.txt	621	13969	–	–	2001	2151.9000	1129.9000	222.7000	0.0064	0.0045
latin_square_10.txt	900	307350	–	–	48677	23977.1000	10960.1000	166.8000	0.0112	0.0052
qg.order30.txt	900	26100	13950	484.6691	13979	1771.9600	630.0170	109.3000	0.0044	0.0008
qg.order40.txt	1600	62400	–	–	32937	13992.3000	6028.2600	151.5500	0.0010	0.0005

Tabela 5.9: Poređenje rezultata CPLEX-a i GaussVNS-a za velike instance ($n \geq 500$).

Instanca	n	m	CPLEX	CPLEX_$t(s)$	$best_{sol}$ Gauss	t_{tot} Gauss(s)	t_{best} Gauss(s)	$iter_{best}$ Gauss	$agap$ Gauss(%)	σ Gauss(%)
DSJC500.1.txt	500	12458	4110*	–	3395	1009.4600	384.0540	187.6500	0.0233	0.0081
DSJC500.5.txt	500	62624	–	–	14149	1714.8400	711.7700	144.7500	0.0105	0.0056
DSJC500.9.txt	500	224874	–	–	34353	3961.3500	1723.5700	159.3500	0.0251	0.0098
DSJC1000.1.txt	1000	49629	–	–	11632	6553.4300	2575.8200	133.6500	0.0121	0.0047
DSJC1000.5.txt	1000	249826	–	–	51599	18626.5000	8309.1000	163.3500	0.0095	0.0040
DSJC1000.9.txt	1000	449449	–	–	132163	59342.8000	27546.3000	178.3000	0.0149	0.0060
DSJR500.1.txt	500	3555	2142*	–	2302	559.8210	238.657	155.6500	0.0107	0.0051
DSJR500.1c.txt	500	121275	–	–	16927	9092.8200	4035.6500	221.3500	0.0084	0.0057
DSJR500.5.txt	500	58862	–	–	28404	3582.0900	2094.5600	234.5500	0.0101	0.0057
flat1000_50_0.txt	1000	245000	–	–	49766	16654.3000	6471.1000	130.8500	0.0242	0.0072
flat1000_60_0.txt	1000	245830	–	–	51024	13551.5000	4450.4900	99.2000	0.0064	0.0032
flat1000_76_0.txt	1000	246708	–	–	51057	19017.1000	7580.7600	141.9000	0.0101	0.0047
homer.txt	561	3258	1150	38.1061	1159	284.9230	145.6930	211.5500	0.0150	0.0042
inithx_i_1.txt	864	18707	–	–	3683	3531.1600	2265.2000	366.5000	0.0046	0.0028
inithx_i_2.txt	645	13979	–	–	2054	1019.1900	520.8020	222.0500	0.0060	0.0056
inithx_i_3.txt	621	13969	–	–	1991	1735.5000	671.6410	299.8500	0.0029	0.0023
latin_square_10.txt	900	307350	–	–	47229	27148.0000	13024.8000	186.6000	0.0132	0.0062
qg.order30.txt	900	26100	13950	484.6691	14020	2194.8300	913.6330	143.9500	0.0060	0.0005
qg.order40.txt	1600	62400	–	–	32992	10596.1000	5371.2900	211.0000	0.0006	0.0004

5.4 Poređenje sa rezultatima iz literature

U ovom poglavlju prikazano je poređenje rezultata implementiranih metoda VNS i GaussVNS sa rezultatima iz literature. Tabele 5.10, 5.11 i 5.12 sadrže podatke o dobijenim rešenjima na malim, srednjim i velikim instancama, respektivno. Prva kolona sadrži naziv instance, druga optimalno rešenje dobijeno CPLEX-om, treća i četvrta najbolje rešenje dobijeno VNS i GaussVNS metodom. U petoj i šestoj koloni nalaze se podaci o najboljim poznatim teorijskim donjim (f^b_{LB}) i gornjim (f^b_{UB}) granicama rešenja problema iz rada [39] opisanim u podsekciji 2.5, a preostalih pet kolona sadrže najbolje rešenje dobijeno različitim algoritmima iz radova [35, 71, 3, 41, 22], respektivno.

Ukoliko u nekom od navedenih radova ne postoji rezultat za navedenu instancu, u odgovarajućoj vrsti stoji oznaka "–". Različite metode iz literature koristile su drugačije kriterijume zaustavljanja, puštane različit broj puta za svaku instancu, pisane su u drugačijim programskim jezicima i testirane su na računarima sa različitim performansama. U nekim od navedenih radova podaci poput vremena izvršavanja i/ili broja iteracija nisu ni prezentovani. Zbog toga, jedino poređenje koje je moguće izvršiti je na osnovu dobijenih najboljih vrednosti. Od 87 instanci, za 3 velike i 12 srednjih instanci ne postoje rezultati dobijeni heurističkim metodama u literaturi. Za njih postoje samo gornja i donja granica, te su one izostavljene iz ovih tabela i njihovi rezultati dobijeni VNS i GaussVNS metodama se mogu videti u tabelama iz prethodne podsekcije.

Za instance malih dimenzija su metode VNS i GaussVNS dobile skoro uvek optimalna rešenja, koja se poklapaju i sa najboljim rezultatima iz literature. Na osnovu podataka o poslednje tri instance iz tabele 5.10, primećuje se sledeće: za queen8_8 se u navedena četiri od pet radova dostigla gornja granica, dok su VNS i GaussVNS metode našle lošije rešenje; kod queen8_12 je samo jedan rad iz literature dostigao optimalno rešenje (u ostalim nije testirana), dok su rešenja VNS i GaussVNS metoda približna optimalnom; za queen9_9 nije nađeno optimalno rešenje CPLEX-om, ali se u jednom radu iz literature rešenje poklapa sa gornjom granicom (ostala četiri rada je nisu testirala), dok su rešenja VNS i GaussVNS metoda lošija od te gornje granice.

Kod instanci srednjih dimenzija, kao što se može videti iz tabele 5.11, VNS metoda je došla do optimalnog rešenja za 11 od 17 instanci za koje postoji optimalno rešenje, a GaussVNS metoda za 12 od 17 instanci. Za jednu od ukupno

Tabela 5.10: Poređenje rezultata dobijenih za male instance ($n < 100$).

Instanca	opt_{sol}	VNS	GaussVNS	f^b_{LB}	f^b_{UB}	[35]	[71]	[3]	[41]	[22]
2Insertions3.txt	62	opt	opt	55	62	opt	–	–	opt	opt
3Insertions3.txt	92	opt	opt	84	92	opt	–	–	opt	opt
david.txt	237	opt	opt	234	237	opt	opt	opt	opt	opt
huck.txt	243	opt	opt	243	243	opt	opt	opt	opt	opt
jean.txt	217	opt	opt	216	217	opt	opt	opt	opt	opt
mug88_1.txt	178	opt	opt	164	178	opt	–	–	opt	opt
mug88_25.txt	178	opt	opt	162	178	opt	–	–	opt	opt
myciel3.txt	21	opt	opt	16	21	opt	opt	opt	opt	opt
myciel4.txt	45	opt	opt	34	45	opt	opt	opt	opt	opt
myciel5.txt	93	opt	opt	70	93	opt	opt	opt	opt	opt
myciel6.txt	189	opt	opt	142	189	opt	opt	opt	opt	opt
queen5_5.txt	75	opt	opt	75	75	opt	opt	opt	opt	opt
queen6_6.txt	138	opt	opt	126	138	opt	150	opt	opt	opt
queen7_7.txt	196	opt	opt	196	196	opt	opt	opt	opt	opt
queen8_8.txt	–	307	304	288	291	291	291	291	291	294
queen8_12.txt	624	626	625	624	624	–	–	–	opt	–
queen9_9.txt	–	426	424	405	409	–	–	–	409	–

34 instance, za koje CPLEX rešavač nije dobio optimalno rešenje, rešenja dobijena VNS i GaussVNS metodama se poklapaju, dok je kod preostale 33 instance rezultat GaussVNS-a bolji u odnosu na VNS. Takođe, može se primetiti da su za 5 instanci za koje postoji optimalno rešenje, rešenja dobijena GaussVNS-om bliska optimalnim.

Poredeći sa rezultatima iz literature, VNS i GaussVNS nisu uspeli u slučaju instanci srednjih dimenzija da dostignu kvalitet postojećih rezultata iz literature. Za rad [35] gde su autori prezentovali rezultate za 15 od ovih 39 instanci, GaussVNS je dostigao samo 6, a VNS samo 5 istih rezultata. U [71] testirane su 22 od ovih 39 instanci i VNS i GaussVNS su samo za jednu dostigli bolja rešenja od njihovih, za jednu je VNS imao isto rešenje, a GaussVNS za tri. U [3] postoje rezultati za samo 11 od 39 instanci, VNS je dobio isto rešenje za dve, a GaussVNS za tri. U radu [41] prikazani su rezultati za svih 39 instanci i oba implementirana algoritma su dobila rešenja koja se poklapaju sa njihovim za 12 instanci. Konacno, u [22] je testirano samo 7 od 39 i VNS je dobio ista resenja kao oni za 4, a GaussVNS za 5 instanci.

Instance velikih dimenzija imaju optimalna rešenja samo u 2 od 16 slučajeva i može se primetiti sledeće: za homer instancu su obe metode došle do rešenja bliskog optimalnom, dok se kod instanci qg.order30 i qg.order40 može videti da je VNS metoda jedino u ta dva slučaja dobila bolje rešenje (13979 i 32937) u odnosu na

Tabela 5.11: Poređenje dobijenih rezultata za srednje instance ($100 \leq n < 500$).

Instanca	opt_{sol}	VNS	GaussVNS	f^b_{LB}	f^b_{UB}	[35]	[71]	[3]	[41]	[22]
anna.txt	276	opt	opt	273	276	opt	283	opt	opt	opt
DSJC125.1.txt	–	337	329	247	326	326	326	326	326	–
DSJC125.5.txt	–	1146	1096	549	1012	1015	1017	1012	1012	–
DSJC125.9.txt	–	2632	2594	1691	2503	2511	2512	2503	2503	–
DSJC250.1.txt	–	1112	1064	570	970	977	985	973	974	–
DSJC250.5.txt	–	4042	3869	1287	3210	3281	3246	3219	3230	–
DSJC250.9.txt	–	9404	9010	4311	8277	8412	8286	8290	8280	–
flat300_20_0.txt	–	5104	4549	1531	3150	–	3150	–	3150	–
flat300_26_0.txt	–	5423	5242	1548	3966	–	3966	–	3966	–
flat300_28_0.txt	–	5325	5239	1547	4238	–	4282	–	4238	–
fpsol2_i_1.txt	3403	3419	3408	3403	3403	opt	–	–	opt	–
fpsol2_i_2.txt	1668	1680	1670	1668	1668	–	–	–	opt	–
fpsol2_i_3.txt	1636	1646	1639	1636	1636	–	–	–	opt	–
games120.txt	443	445	opt	442	443	opt	opt	opt	opt	opt
le450_5a.txt	–	1493	1438	1193	1350	–	–	–	1350	–
le450_5b.txt	–	1594	1432	1189	1350	–	–	–	1350	–
le450_5c.txt	–	1443	1376	1278	1350	–	–	–	1350	–
le450_5d.txt	–	1413	1375	1282	1350	–	–	–	1350	–
le450_15a.txt	–	3060	2969	2331	2632	–	2632	–	2706	–
le450_15b.txt	–	3080	3020	2348	2632	–	2642	–	2724	–
le450_15c.txt	–	4797	4641	2610	3487	–	3866	–	3491	–
le450_15d.txt	–	4830	4567	2628	3505	–	3921	–	3506	–
le450_25a.txt	–	3557	3464	3003	3153	–	3153	–	3166	–
le450_25b.txt	–	3732	3669	3305	3365	–	3366	–	3366	–
le450_25c.txt	–	5426	5319	3657	4515	–	4515	–	4700	–
le450_25d.txt	–	5457	5339	3698	4544	–	4544	–	4722	–
miles250.txt	325	334	328	318	325	opt	328	327	opt	329
miles500.txt	705	736	724	686	705	712	709	710	opt	719
mug100_1.txt	202	opt	opt	188	202	opt	–	–	opt	opt
mug100_25.txt	202	opt	opt	186	202	opt	–	–	opt	opt
mulsol_i_1.txt	1957	opt	opt	1957	1957	–	–	–	opt	–
mulsol_i_2.txt	1191	opt	opt	1191	1191	–	–	–	opt	–
mulsol_i_3.txt	1187	opt	opt	1187	1187	–	–	–	opt	–
mulsol_i_4.txt	1189	opt	opt	1189	1189	–	–	–	opt	–
mulsol_i_5.txt	1160	opt	opt	1160	1160	–	–	–	opt	–
myciel7.txt	–	381	381	286	381	381	381	381	381	381
zeroin_i_1.txt	1822	opt	opt	1822	1822	–	–	–	opt	–
zeroin_i_2.txt	1004	opt	opt	1004	1004	opt	–	–	opt	–
zeroin_i_3.txt	998	opt	opt	998	998	opt	–	–	opt	–

Tabela 5.12: Poređenje rezultata dobijenih za velike instance ($n \geq 500$).

Instanca	opt_{sol}	VNS	GaussVNS	f^b_{LB}	f^b_{UB}	[35]	[71]	[3]	[41]	[22]
DSJC1000.1.txt	–	12004	11632	2762	8991	10123	9003	9520	8995	–
DSJC1000.5.txt	–	52350	51599	6708	37575	43614	37598	40661	37594	–
DSJC1000.9.txt	–	137075	132163	26557	103445	112749	103464	–	103464	–
DSJC500.1.txt	–	3537	3395	1250	2836	2951	2850	2882	2841	–
DSJC500.5.txt	–	14518	14149	2923	10886	11717	10910	11187	10897	–
DSJC500.9.txt	–	36229	34353	11053	29862	30818	29912	30097	28896	–
flat1000_50_0.txt	–	51193	49766	6601	25500	–	25500	–	25500	–
flat1000_60_0.txt	–	51934	51024	6640	30100	–	30100	–	30100	–
flat1000_76_0.txt	–	52078	51057	6632	37164	–	37167	–	37167	–
homer.txt	1150	1168	1159	1129	1150	–	–	–	1123	–
inithx_i_1.txt	–	3699	3683	3676	3676	3676	–	–	3676	–
inithx_i_2.txt	–	2070	2054	2050	2050	–	–	–	2050	–
inithx_i_3.txt	–	2001	1991	1986	1986	–	–	–	1986	–
latin_square_10.txt	–	48677	47229	40950	41444	–	42223	–	41444	–
qg.order30.txt	13950	13979	14020	13950	13950	–	opt	–	12581	–
qg.order40.txt	–	32937	32992	32800	32800	–	32800	–	32800	–

GaussVNS metodu (14020 i 32992). Takođe, rezultat VNS-a za instancu qg.order30 (13979) je čak blizak optimalnom (13950). Na osnovu ostalih 14 rešenja VNS i GaussVNS metode, jasno se vidi da je GaussVNS bio uspešniji od VNS-a. Za tri velike instance, inithx_i_1, inithx_i_2 i inithx_i_3, donja i gornja granica rešenja problema se poklapaju i iznose redom 3676, 2050 i 1986. Za pomenute instance je GaussVNS metoda došla do rešenja bliskih donjim, tj. gornjim granicama (3683, 2054 i 1991, respektivno).

Poredeći sa rezultatima iz literature, ni jedna od dve implementirane metode nije dostigla najbolje rezultate prezentovane u literaturi za instance velikih dimenzija. Međutim, samo u radu [41] su prikazani rezultati za svih 16 instanci iz ove grupe. U radovima [35], [71] i [3] su prezentovani rezultati za 7, 12 i 5 instanci iz ove grupe, dok u radu [22] nije testirana ni jedna instanca iz skupa velikih.

Na osnovu prikazanih rezultata, VNS i GaussVNS metode za skoro sve male i neke srednje instance dolaze do optimalnog rešenja. Problem nastaje kod većih instanci, gde se nije dalo previše prostora (u smislu broja iteracija) zbog predugog vremena izvršavanja.

U literaturi postoji više radova koji različitim heuristikama rešavaju ovaj problem. Međutim, ni u jednom radu nije prezentovano svih 87 instanci, koje su testirane VNS i GaussVNS algoritmom. U radovima [35], [71], [3], [41], [22] je testirano 38, 52, 27, 77 i 23 instanci, respektivno. Takođe, način na koji su autori birali podskup skupa instanci na kojima prezentuju svoje rezultate nije naveden. Najviše testirane su one instance za koje se zna optimalno rešenje.

U radu [41], koji ima najbolje rezultate od svih pet navedenih radova, uočene su neke greške: za instancu homer (najmanja velika instanca) prezentovali su rešenje 1123 kao najbolje, dok je CPLEX rešavač dao optimalno rešenje 1150. Takođe, teorijske granice su 1129–1150, što znači da vrednost 1123 ispada iz ovog okvira. Kod instance qg.order30 su gornja i donja granica 13950, dok rešenje koje su oni prezentovali iznosi 12581. Na ostalim većim instancama nije moguće utvrditi da li je došlo do još neke greške, kako zbog nedostatka CPLEX rezultata, tako i zbog prevelikog raspona donjih i gornjih granica.

Vremenski najzahtevniji korak u VNS i GaussVNS metodi bio je određivanje pravilnog bojenja grafa. U ostalim radovima se za to koriste neki drugi već postojeći algoritmi koji su se pokazali uspešnim. Na primer, u radu [35] se za generisanje ispravnog bojenja koriste DSATUR i MDSAT algoritmi, u [41] TABUCOL, a u [71] MACOL.

Sa efikasnijim algoritmom za generisanje pravilnog bojenja grafa, VNS i GaussVNS metode bi se drastično ubrzale, čime se pretpostavlja da bi se povećanjem maksimalnog broja iteracija stiglo do boljih rešenja. Maksimalan ukupan broj iteracija koji je odrađen kod VNS-a je 569.7 u slučaju instance le450_5d, odnosno 586.5 kod GaussVNS-a za instancu flat300_20_0. Ovo je relativno mali broj iteracija metode u odnosu na 50000 iteracija algoritma bez poboljšanja koje su koristili u radu [41] i 10^8 iteracija u radu [71].

Glava 6

Zaključak

U ovom radu razmatrana je varijanta problema bojenja grafa, čiji je cilj pronalaženje pravilnog bojenja čvorova bojama predstavljenim prirodnim brojevima $\{1, 2, ...\}$, tako da zbir boja dodeljenih čvorovima grafa bude minimalan. Opisani problem su predložili Kubicka [49], u oblasti teorije grafova, i Supovit [64], u oblasti VLSI (engl. Very Large Scale Integration) dizajna, i on spada u klasu NP-teških problema. MSCP ima veliki značaj u praksi jer se koristi za rešavanje problema raspoređivanja, rasporeda časova, alokacije registara, i mnogim drugim.

Za rešavanje problema MSCP korišćene su egzaktna metoda i heuristički pristup. Egzaktnim CPLEX rešavačem testiran je model celobrojnog linearnog programiranja (2.9) – (2.12) i dobijena su optimalna rešenja za ukupno 37 od 87 instanci. Ovo ukazuje na to da usled ograničenog vremena izvršavanja i memorijskog prostora nije moguće primeniti ovaj pristup za rešavanje problema sa strukturno kompleksnim grafovima.

Iz tog razloga, implemetirane su dve heurističke metode, čije su performanse međusobno upoređene. Takođe, dobijeni rezultati za 50 instanci za koje CPLEX rešavač nije našao optimalno rešenje poređeni su sa teorijski gornjim i donjim granicama iz rada [39]. Predložene heurističke metode su metoda promenljivih okolina (engl. Variable Neighborhood Search, VNS) i Gausovska metoda promenljivih okolina (engl. Gaussian Variable Neighborhood Search, GaussVNS).

Metoda VNS rešava MSCP kao problem kombinatorne optimizacije, dok je ideja metode GaussVNS bila prelazak na kontinualan problem zbog postojanja velikog broja lokalnih minimuma. Ovo je bilo moguće uraditi zbog same reprezentacije rešenja, koja predstavlja jedinstveni redosled po kome će se čvorovima grafa dodeljivati boje.

Izvršeno je testiranje oba algoritma na istom skupu instanci, sa istim ulaznim parametrima. Na osnovu dobijenih rezultata može se zaključiti da su implementirane heurističke metode za većinu instanci malih i za neke instance srednjih dimenzija dobile optimalno rešenje. Međutim, problem nastaje kod instanci velikih dimenzija, gde nije dobijena čak ni poznata gornja granica najboljeg rešenja.

Upoređivanjem VNS i GaussVNS metode primećuje se da je GaussVNS za sve osim dve testirane instance dao bolja rešenja. Kraće vreme izvršavanja je bilo potrebno GaussVNS metodi kod 40, a duže kod 47 instanci u odnosu na VNS. Nedostaci navedenih metoda mogu se rešiti korišćenjem nekog efikasnijeg algoritma za generisanje pravilnog bojenja, čime bi se vremenski zahtevi smanjili i time omogućilo povećanje broja iteracija, u cilju dobijanja boljih rešenja. Prezentovani rezultati takođe zavise i od strukture testiranih grafova, ali se zaključuje da navedene metode imaju potencijal za dalje usavršavanje.

Dalja istraživanja i unapređenje rada se mogu realizovati u više pravaca:

- Probati ovakvu ideju VNS metode sa algoritmom koji brže i efikasnije pronalazi pravilno bojenje grafa.
- Modifikovati korak lokalne pretrage proširivanjem okoline koja će se koristiti u ovoj fazi.
- Promene u koraku razmrdavanja i lokalne pretrage takve da novodobijena rešenja ne zahtevaju bojenje celog grafa nego samo nekog njegovog (promenjenog) dela.

Bibliografija

[1] E. Aarts, E. H. L. Aarts, and J. K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[2] J. H. Ahrens and U. Dieter. Efficient table-free sampling methods for the exponential, cauchy, and normal distributions. *Communications of the ACM*, 31(11):1330–1337, 1988.

[3] U. Benlić and J. K. Hao. A study of breakout local search for the minimum sum coloring problem. In *Asia-Pacific conference on simulated evolution and learning*, pages 128–137. Springer, 2012.

[4] D. Bertsimas and J. Tsitsiklis. Simulated annealing. *Statistical science*, 8(1):10–15, 1993.

[5] H. Bouziri and M. Jouini. A tabu search approach for the sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36:915–922, 2010.

[6] J. Brimberg. A variable neighborhood algorithm for solving the continuous location-allocation problem. *Studies in Locational Analysis*, 10:1–12, 1996.

[7] J. Brimberg, P. Hansen, N. Mladenović, and E. D. Taillard. Improvements and comparison of heuristics for solving the uncapacitated multisource weber problem. *Operations research*, 48(3):444–460, 2000.

[8] J. R. Brown. Chromatic scheduling and the chromatic number problem. *Management science*, 19(4-part-1):456–463, 1972.

[9] E. Carrizosa, M. Dražić, Z. Dražić, and N. Mladenović. Gaussian variable neighborhood search for continuous optimization. *Computers & Operations Research*, 39(9):2206–2213, 2012.

[10] F. C. Chow and J. L. Hennessy. The priority-based coloring approach to register allocation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(4):501–536, 1990.

[11] A. T. Clementson and C. H. Elphick. Approximate colouring algorithms for composite graphs. *Journal of the Operational Research Society*, 34(6):503–509, 1983.

[12] A. Colorni, M. Dorigo, V. Maniezzo, et al. Distributed optimization by ant colonies. In *Proceedings of the first European conference on artificial life*, volume 142, pages 134–142. Paris, France, 1991.

[13] D. Cvetković, M. Čangalović, Đ. Dugošija, V. Kovačević-Vujčić, S. Simić, and J. Vuleta. *Kombinatorna optimizacija*. Društvo operacionih istraživača Jugoslavije, 1996.

[14] D. de Werra. An introduction to timetabling. *European journal of operational research*, 19(2):151–162, 1985.

[15] G. Di Caro and M. Dorigo. *Ant colony optimization and its application to adaptive routing in telecommunication networks*. PhD thesis, PhD thesis, Faculté des Sciences Appliquées, Université Libre de Bruxelles, Brussels, Belgium, 2004.

[16] T. Dinski and X. Zhu. A bound for the game chromatic number of graphs. *Discrete Mathematics*, 196(1-3):109–115, 1999.

[17] S. M. Douiri and S. Elbernoussi. New algorithm for the sum coloring problem. *International Journal of Contemporary Mathematical Sciences*, 6(10):453–463, 2011.

[18] K. A. Dowsland and J. Thompson. Simulated annealing. *Handbook of natural computing*, pages 1623–1655, 2012.

[19] M. Drazic. Influence of a neighborhood shape on the efficiency of continuous variable neighborhood search. *Yugoslav Journal of Operations Research*, 30(1):3–17, 2019.

[20] M. Drazić, V. Kovačević-Vujčić, M. Čangalović, and N. Mladenović. Glob—a new vns-based software for global optimization. In *Global optimization*, pages 135–154. Springer, 2006.

[21] P. Erdős and A. Hajnal. On chromatic number of graphs and set-systems. *Acta Math. Acad. Sci. Hungar*, 17(61-99):1, 1966.

[22] K. Erfani, S. Rahimi, et al. New variable neighborhood search method for minimum sum coloring problem on simple graphs. *Iranian Journal of Numerical Analysis and Optimization*, 8(2):39–54, 2018.

[23] G. Fishman. *Monte Carlo: concepts, algorithms, and applications*. Springer Science & Business Media, 2013.

[24] F. Furini, E. Malaguti, S. Martin, and I. C. Ternier. Ilp models and column generation for the minimum sum coloring problem. *Electronic Notes in Discrete Mathematics*, 64:215–224, 2018.

[25] A. Gamst. Some lower bounds for a class of frequency assignment problems. *IEEE transactions on vehicular technology*, 35(1):8–14, 1986.

[26] M. R. Garey and D. S. Johnson. A guide to the theory of np-completeness. 1978.

[27] M. R. Garey and D. S. Johnson. Computers and intractability. *A Guide to the*, 1979.

[28] F. Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.

[29] F. W. Glover and G. A. Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006.

[30] G. Hahn, J. Kratochvıl, J. Širáň, and D. Sotteau. On the injective chromatic number of graphs. *Discrete mathematics*, 256(1-2):179–192, 2002.

[31] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European journal of operational research*, 130(3):449–467, 2001.

[32] P. Hansen and N. Mladenović. A tutorial on variable neighborhood search. *Les Cahiers du GERAD ISSN*, 711:2440, 2003.

[33] P. Hansen, N. Mladenović, and J. A. Moreno Perez. Variable neighbourhood search: methods and applications. *4OR*, 6(4):319–360, 2008.

[34] P. Hansen, N. Mladenović, R. Todosijević, and S. Hanafi. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, 5(3):423–454, 2017.

[35] A. Helmar and M. Chiarandini. A local search heuristic for chromatic sum. In *Proceedings of the 9th metaheuristics international conference*, volume 1101, pages 161–170, 2011.

[36] H. Holland John. Adaptation in natural and artificial systems. *Ann Arbor: University of Michigan Press*, 1975.

[37] I. Holyer. The np-completeness of edge-coloring. *SIAM Journal on computing*, 10(4):718–720, 1981.

[38] Y. W. Jeong, J. B. Park, S. H. Jang, and K. Y. Lee. A new quantum-inspired binary pso: application to unit commitment problems for power systems. *IEEE Transactions on Power Systems*, 25(3):1486–1495, 2010.

[39] Y. Jin, J. P. Hamiez, and J. K. Hao. Algorithms for the minimum sum coloring problem: a review. *Artificial Intelligence Review*, 47(3):367–394, 2017.

[40] Y. Jin and J. K. Hao. Hybrid evolutionary search for the minimum sum coloring problem of graphs. *Information Sciences*, 352:15–34, 2016.

[41] Y. Jin, J. K. Hao, and J. P. Hamiez. A memetic algorithm for the minimum sum coloring problem. *Computers & Operations Research*, 43:318–327, 2014.

[42] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[43] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[44] Z. Kokosiński and K. Kwarciany. On sum coloring of graphs with parallel genetic algorithms. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 211–219. Springer, 2007.

[45] D. Koschier, J. Bender, B. Solenthaler, and M. Teschner. Smoothed particle hydrodynamics techniques for the physics based simulation of fluids and solids. *arXiv preprint arXiv:2009.06944*, 2020.

[46] V. Kovačević-Vujčić, M. Čangalovic, M. Dražić, N. Mladenović, L. T. H. An, and P. D. Tao. Vns-based heuristics for continuous global optimization. In *Modelling, Computation and Optimization in Information Systems and Managemant Sciences*. 2004.

[47] J. Krarup and D. De Werra. Chromatic optimisation: limitations, objectives, uses, references. *European Journal of Operational Research*, 11(1):1–19, 1982.

[48] L. G. Kroon, A. Sen, H. Deng, and A. Roy. The optimal cost chromatic partition problem for trees and interval graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 279–292. Springer, 1996.

[49] E. M. Kubicka. *The chromatic sum and efficient tree algorithms*. Western Michigan University, 1989.

[50] F. T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards*, 84(6):489, 1979.

[51] Y. Li, C. Lucet, A. Moukrim, and K. Sghiouer. Greedy algorithms for the minimum sum coloring problem. In *Logistique et transports*, pages LT–027, 2009.

[52] L. Liberti and M. Drazic. Variable neighbourhood search for the global optimization of constrained nlps. In *Proceedings of GO Workshop, Almeria, Spain*, volume 2005, 2005.

[53] P. Lučić and D. Teodorović. Bee system: modeling combinatorial optimization transportation engineering problems by swarm intelligence. In *Preprints of the TRISTAN IV triennial symposium on transportation analysis*, pages 441–445, 2001.

[54] P. Lučić and D. Teodorović. Computing with bees: attacking complex transportation engineering problems. *International Journal on Artificial Intelligence Tools*, 12(03):375–394, 2003.

[55] N. Mladenović, M. Dražić, V. Kovačević-Vujčić, and M. Čangalović. General variable neighborhood search for the continuous optimization. *European Journal of Operational Research*, 191(3):753–770, 2008.

[56] N. Mladenović, M. Dražić, M. Čangalović, and V. Kovačević-Vujčić. Variable neighborhood search in global optimization. In *Proceedings of XXX Symposium on Operations Research, Herceg Novi*, pages 327–330, 2003.

[57] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[58] N. Mladenović, J. Petrović, V. Kovačević-Vujčić, and M. Čangalović. Solving spread spectrum radar polyphase code design problem by tabu search and variable neighbourhood search. *European Journal of Operational Research*, 151(2):389–399, 2003.

[59] A. Moukrim, K. Sghiouer, C. Lucet, and Y. Li. Lower bounds for the minimal sum coloring problem. *Electronic Notes in Discrete Mathematics*, 36:663–670, 2010.

[60] I. H. Osman and J. P. Kelly. Meta-heuristics: an overview. *Meta-heuristics*, pages 1–21, 1996.

[61] L. Pinedo Michael. Scheduling theory, algorithms, and systems. new york university. isbn: 978-0-387-78934-7, e-isbn: 978-0-387-78935, 2008.

[62] R. Poli. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, 2008, 2008.

[63] T. Stützle, M. Dorigo, et al. Aco algorithms for the traveling salesman problem. *Evolutionary algorithms in engineering and computer science*, 4:163–183, 1999.

[64] K. J. Supowit. Finding a maximum planar subset of a set of nets in a channel. *IEEE transactions on computer-aided design of integrated circuits and systems*, 6(1):93–94, 1987.

[65] T. Szkaliczki. Routing with minimum wire length in the dogleg-free manhattan model is \calnp-complete. *SIAM Journal on Computing*, 29(1):274–287, 1999.

[66] E.-G. Talbi. *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009.

[67] D. Teodorović, M. Šelmić, and T. Davidović. Bee colony optimization-part ii: The application survey. *Yugoslav Journal of Operations Research*, 25(2):185–219, 2015.

[68] C. Thomassen, P. Erdös, Y. Alavi, P. J. Malde, and A. J. Schwenk. Tight bounds on the chromatic sum of a connected graph. *Journal of Graph Theory*, 13(3):353–357, 1989.

[69] V. Voloshin. Graph coloring: History, results and open problems. *Alabama Journal of Mathematics, Spring/Fall*, 2009.

[70] Y. Wang, J. K. Hao, F. Glover, and Z. Lü. Solving the minimum sum coloring problem via binary quadratic programming. *arXiv preprint arXiv:1304.5876*, 2013.

[71] Q. Wu and J. K. Hao. An effective heuristic algorithm for sum coloring of graphs. *Computers & Operations Research*, 39(7):1593–1600, 2012.

[72] Q. Wu and J. K. Hao. Improved lower bounds for sum coloring via clique decomposition. *arXiv preprint arXiv:1303.6761*, 2013.

[73] M. Čangalović and J. A. M. Schreuder. Exact colouring algorithm for weighted graphs applied to timetabling problems with lectures of different lengths. *European Journal of Operational Research*, 51(2):248–258, 1991.