Automatizacija postupka određivanja redoslijeda razvoja podsustava informacijskog sustava

Kudelić, Robert

Sveučilište u Zagrebu

Fakultet organizacije i informatike

Robert Kudelić

AUTOMATIZACIJA POSTUPKA ODREĐIVANJA REDOSLIJEDA RAZVOJA PODSUSTAVA INFORMACIJSKOG SUSTAVA

DOKTORSKI RAD

Varaždin, 2015.

PODACI O DOKTORSKOM RADU

I. AUTOR

Ime i prezime	Robert Kudelić
Datum i mjesto rođenja	10.7.1985., Zagreb
Naziv fakulteta i datum diplomiranja na VII/I stupnju	Fakultet organizacije i informatike, 15.7.2008.
Naziv fakulteta i datum diplomiranja na VII/II stupnju	
Sadašnje zaposlenje	Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin, znanstveni novak/asistent

II. DOKTORSKI RAD

Naslov	Automatizacija postupka određivanja redoslijeda razvoja podsustava informacijskog sustava
Broj stranica, slika, tabela, priloga, bibliografskih podataka	102 stranice, 18 slika, 9 tabela, 1 prilog, 57 bibliografskih jedinica
Znanstveno područje i polje iz kojeg je postignut doktorat znanosti	Društvene znanosti, informacijske znanosti
Mentori ili voditelji rada	Prof. dr. sc. Alen Lovrenčić
Fakultet na kojem je obranjen doktorski rad	Fakultet organizacije i informatike, Varaždin
Oznaka i redni broj rada	125

III. OCJENA I OBRANA

Datum sjednice Fakultetskog vijeća na kojoj je prihvaćena tema	25.9.2014.
Datum predaje rada	7.4.2015.
Datum sjednice Fakultetskog vijeća na kojoj je prihvaćena pozitivna ocjena rada	2.6.2015.
Sastav povjerenstva koje je rad ocijenilo	Prof. dr. sc. Neven Vrček Prof. dr. sc. Robert Manger Prof. dr. sc. Zdravko Dovedan Han Prof. dr. sc. Mirko Čubrilo Prof. dr. sc. Alen Lovrenčić
Datum obrane doktorskog rada	6.7.2015.
Sastav povjerenstva pred kojim je rad obranjen	Prof. dr. sc. Neven Vrček Prof. dr. sc. Robert Manger Prof. dr. sc. Zdravko Dovedan Han Prof. dr. sc. Mirko Čubrilo Prof. dr. sc. Alen Lovrenčić
Datum promocije	

Sveučilište u Zagrebu

Fakultet organizacije i informatike

Robert Kudelić

AUTOMATIZACIJA POSTUPKA ODREĐIVANJA REDOSLIJEDA RAZVOJA PODSUSTAVA INFORMACIJSKOG SUSTAVA

DOKTORSKI RAD

Mentor: Prof. dr. sc. Alen Lovrenčić

Varaždin, 2015.

University of Zagreb

Faculty of Organization and Informatics

Robert Kudelić

AUTOMATIC DETERMINATION OF INFORMATION SYSTEM SUBSYSTEMS DEVELOPMENT ORDER

DOCTORAL THESIS

Varaždin, 2015.

Z A H V A L A

"Mnoge sam stvari u svojim rukama držao, i sve sam ih izgubio; ali što god sam u Božje ruke stavio, to još uvijek posjedujem."

- Martin Luther, profesor teologije

S. D. G.

Sažetak

Prilikom razvoja informacijskog sustava potrebno je odrediti slijed razvoja podsustava informacijskog sustava. Ovaj problem trenutno nije formalno riješen. Stoga predlažemo rješenje koje će kao kriterij, određivanja slijeda razvoja podsustava informacijskog sustava, imati sumu težina povratnih lukova u slijedu podsustava informacijskog sustava. Nadalje, dokazali smo kako je ovaj problem NP-potpun, NP-težak, i APX-težak. Isto tako, kako bismo riješili ovaj problem osmislili smo: algoritam temeljen na metodi Grananja i ograničenja, Monte Carlo randomizirani algoritam, i heuristički algoritam. Za sva tri algoritma smo procijenili složenost. Sva tri algoritma su implementirana i empirijski testirana. Na kraju smo pokazali na koji način se u praksi, po potrebi, mogu uvrštavati dodatna ograničenja, i gdje se još osmišljeni algoritmi potencijalno mogu koristiti.

Ključne riječi: slijed čvorova, informacijski sustav, suma težina povratnih lukova, usmjereni ciklički graf, metoda grananja i ograničenja, monte carlo randomizacija, heuristika, np-težak, np-potpun, apx-težak, algoritmi, složenost, uvrštavanje dodatnih ograničenja, primjena na ostale probleme.

Abstract

When we are developing Information System we must determine development order of its subsystems. Currently, this problem is not formally solved. Therefore, we have proposed a solution which takes sum of weights of feedback arcs as a criteria for determining development order of Information System subsystems. Furthermore, we have proved that the problem of Information System Subsystems Development Order is NP-complete, NP-hard, and APX-hard. Also, in order to solve this problem we have created: Branch and Bound algorithm, Monte Carlo randomized algorithm, and heuristic algorithm. Complexity has been calculated for all three algorithms. All three algorithms have been implemented and empirically analysed. Lastly, we have showed how one can apply additional constraints upon the problem of Information System Subsystems Development Order, and where can one potentially use developed algorithms outside of Information System Subsystems Development Order problem.

Key words: sequence of nodes, Information System, sum of weights of feedback arcs, Directed Cyclic Graph, Branch and Bound method, Monte Carlo randomization, heuristics, NP-hard, NP-complete, APX-hard, algorithms, complexity, applying additional constraints, application on other problems.

Prošireni sažetak

Prilikom razvoja informacijskog sustava potrebno je odrediti slijed razvoja podsustava informacijskog sustava. Opišemo li ovaj problem preko usmjerenog grafa primjećujemo kako graf sadrži cikluse. Ti ciklusi očito predstavljaju problem, ako želimo odrediti slijed razvoja podsustava informacijskog sustava. Stoga predlažemo rješenje koje će kao kriterij, određivanja slijeda razvoja podsustava informacijskog sustava, imati sumu težina povratnih lukova. Razlog predlaganja ovog kriterija leži u sljedećoj činjenici. Suma težina povratnih lukova proizlazi direktno iz opisa podsustava informacijskog sustava. U pred istraživanju smo primijetili kako je problem Određivanja redoslijeda razvoja podsustava informacijskog sustava vrlo vjerojatno težak za riješiti. Stoga smo, u ovom radu, dokazali kako je ovaj problem NP-potpun, NP-težak, i APX-težak. Za prethodne dokaze koristili smo Povratni skup lukova (eng. Feedback Arcs Set). Povratni skup lukova predstavlja poznati problem u teoriji grafova za kojeg se zna kako je težak za riješiti. Kako bismo riješili ovaj problem (Određivanje redoslijeda razvoja podsustava informacijskog sustava) osmislili smo: algoritam temeljen na metodi Grananja i ograničenja, Monte Carlo randomizirani algoritam, i heuristički algoritam. U vezi algoritma koji je temeljen na metodi Grananja i ograničenja. Pretpostavimo li kao ulaz u algoritam potpuni graf čije su težine lukova jednake, usprkos implementaciji dopustivih heuristika, složenost algoritma je $O(n!)$, pri čemu je n broj čvorova grafa. S druge strane, pretpostavimo li optimističan ulaz u algoritam, složenost algoritma je $\Omega\left(\frac{3(n^2-n)+2}{2}\right)$. Empirijska analiza je pokazala kako je, na pretpostavljenom prosječnom grafu, složenost algoritma eksponencijalna. U vezi Monte Carlo randomiziranog algoritma. Složenost algoritma u jednom izvršavanju je $O(|V|^3)$. Što znači kako je složenost za neki broj izvršavanja k jednaka $O(k|V|^3)$. Vjerojatnost da smo nakon pi izvršavanja algoritma dosegli optimum jednaka je $\left(1-\left(1-\left(\frac{n-2}{n}\right)^{\frac{n}{2}}\right)^{pi}\right)$. U vezi heurističkog algoritma. Složenost ovog algoritma je $O(n^3)$. Radi preciznijeg utvrđivanja performansi algoritama i kvalitete rješenja sva tri algoritma su implementirana i empirijski testirana. Isto tako, pokazali smo na koji način se u praksi, s obzirom da je nekada potrebno narušiti algoritamski pronađeni sljed, mogu uvrštavati dodatna ograničenja. Na kraju, dali smo kratak pregled nekoliko srodnih problema na koje se osmišljeni algoritmi potencijalno mogu primijeniti. Srodni problemi na koje smo se osvrnuli su: Određivanje kritičnog puta (eng. Critical Path Method, je poznata metoda koja se koristi u tu svrhu), Raspoređivanje zadataka na procesoru, i Trgovački putnik.

Ključne riječi: slijed čvorova, informacijski sustav, suma težina povratnih lukova, usmjereni ciklički graf, metoda grananja i ograničenja, monte carlo randomizacija, vjerojatnost, heuristika, np-težak, np-potpun, apx-težak, algoritmi, složenost, uvrštavanje dodatnih ograničenja, primjena na ostale probleme, povratni skup lukova, ciklusi, eksperiment, procjena performansi.

Extended abstract

When we are developing Information System we must determine development order of its subsystems. If we describe this problem through Directed Graph it can be observed that this graph has cycles. These cycles obviously represent a problem, if we want to determine development order of Information System subsystems. Therefore, we have proposed a solution which takes sum of weights of feedback arcs as a criteria for determining development order of Information System subsystems. Reason for this lies in the following fact. Sum of weights of feedback arcs comes directly from Information System subsystems description. In previous research we have noticed that a problem of Information System Subsystems Development Order is probably hard to solve. Therefore in this research, we have proved that this problem is NP-complete, NP-hard, and APX-hard. For previous proofs we have used Feedback Arcs Set. Feedback Arcs Set represents a very known problem in graph theory for which it is known that it is hard to solve. In order to solve this problem (Information System Subsystems Development Order) we have created: Branch and Bound algorithm, Monte Carlo randomized algorithm, and heuristic algorithm. Concerning Branch and Bound algorithm. If algorithm input is complete graph with all arc weights equal, despite implemented admissible heuristics, algorithm complexity is $O(n!)$, where n represents number of graph nodes. On the other hand, if algorithm input is optimistic, complexity is $\Omega\left(\frac{3(n^2-n)+2}{2}\right)$. Empirical analysis has shown that, on an assumed common Information System graph, algorithm complexity is exponential. Concerning Monte Carlo randomized algorithm. If we run this algorithm once, then complexity is $O(|V|^3)$. Which means that for some number of executions k, complexity is $O(k|V|^3)$. Probability of finding optimum after pi algorithm executions is $\left(1-\left(1-\left(\frac{n-2}{n}\right)^{\frac{n}{2}}\right)^{pi}\right)$. Concerning heuristic algorithm. Complexity of this algorithm is $O(n^3)$. For the purpose of more precisely determining performance and quality of solutions, all three algorithms have been implemented and empirically analysed. Also, we have showed how one can apply additional constraints upon the problem of Information System Subsystems Development Order, since sometimes in practical situations this is necessary. Lastly, we have briefly analysed a few similar problems on which application of previously mentioned created algorithms is potentially possible. These briefly analysed and similar problems are: Finding critical path (for which a well known Critical Path Method is used), Scheduling of tasks on a processor, and Travelling salesman.

Key words: sequence of nodes, Information System, sum of weights of feedback arcs, Directed Cyclic Graph, Branch and Bound method, Monte Carlo randomization, probability, heuristics, NP-hard, NP-complete, APX-hard, algorithms, complexity, applying additional constraints, application on other problems, Feedback Arcs Set, cycles, experiment, performance assessment.

Kazalo

Popis slika

Popis tablica

Popis algoritama

Popis implementacija algoritama

Popis kratica

IS	Informacijski sustav
BP	Baza podataka
P/K	Procesi/klase podataka
LO	Linearan redoslijed
ISs	Podsustav informacijskog sustava
NP	Nedeterminističko polinomno vrijeme
MPP	Modeliranje poslovnih procesa
UPP	Upravljanje poslovnim procesima
BPMS	Sustav za upravljanje poslovnim procesima
DFS	Dubinsko pretraživanje
MST	Minimalno razapinjuće stablo
BB	Metoda grananja i ograničenja
APX	Aproksimativan
OSRPIS	Određivanje redoslijeda razvoja podsustava informacijskog sustava
PSL	Povratni skup lukova
MPSL	Minimalni povratni skup lukova
SPL	Suma težina povratnih lukova

Poglavlje 1

Uvod

Kazalo poglavlja

"Dizajn informacijskih sustava obavlja se uz pomoć alata za računalno potpomognuto softversko inženjerstvo koji jesu napravili napredak međutim taj napredak može biti i veći. "Naime, veliki dio postupaka je zapravo manualan rad projektanta, a alat služi samo za praćenje rezultata." (Lovrenčić, 2008) Stoga bi bilo dobro kada bi se barem određene aktivnosti kod dizajna informacijskog sustava mogle automatizirati. "Jedan dio posla koji bi se mogao automatizirati je traženje slijeda razvoja podsustava informacijskog sustava." (Lovrenčić, 2008)" (Kudelić, 2014) Stoga ćemo se u prvom podpoglavlju ovog poglavlja baviti opisom problema. U drugom podpoglavlju ćemo definirati ciljeve i hipoteze a u trećem podpoglavlju ćemo definirati doseg istraživanja i metodologiju.

1.1 Opis i okolina problema

Prilikom dizajna IS-a postoje određeni postupci koji su od značaja i koji bi se mogli automatizirati. Neki od tih postupaka su: "dekompozicija IS-a, određivanje redoslijeda izrade podsustava IS-a, automatizacija postupaka izrade relacijskog modela i normalizacija BP, ujednačavanje postojećih BP korištenjem sintaksnih obrazaca, automatsko punjenje baze podataka iz postojećih baza podataka uz provjeru novodefiniranih tabličnih i referencijalnih ograničenja, korištenje generativnog programiranja pri izradi obrazaca za izradu pojedinih funkcija IS-a" (Lovrenčić, 2008). U ovom radu ćemo se bazirati na postupak određivanja redoslijeda razvoja podsustava informacijskog sustava i njegovu automatizaciju. Stoga u nastavku dajemo opis problema. Na početku dizajna IS-a, u svrhu razu-

mijevanja interakcija između procesa i klasa podataka (proces predstavlja niz aktivnosti a klasa podataka skup podataka koji je od poslovnog značaja (Robinson College of Business, n.g.)) radi se p/k matrica (matrica gdje su u zaglavljima popisani procesi i klase podataka a u tijelu matrice je zabilježena interakcija (Davenport and Short, 1990)).

Tablica 1.1: Primjer p/k matrice sa proizvoljno određenim podsustavima.

p/k	k1	k2	k3	k4	k5	k6	k7	k8	k9	k10
p1	CRU	C	CRU			R				R
p2	R	RU	R						R	
p3				CRU	C					R
p4	R			R	RU		R	RU		RU
p5				RU	R					
p6		RU	R			CRU	C		RU	
p7						R	RU			R
p8				RU				CRU	C	
p9		R				R		R	RU	R
p10			CRU					RU	R	

Ta matrica predstavlja interakcije cijelog sustava koji se dizajnira i polazna je točka za određivanje podsustava informacijskog sustava.

Definicija 1.1.1. *Skup procesa, grupiran po određenom kriteriju, predstavlja podsustav IS-a.*

Prije nego krenemo na opis našeg problema objasnimo ukratko na koji način bismo mogli odrediti podsustave.

Definicija 1.1.2 (Lovrenčić, 1997)**.** *Neka su p_i i p_j dva procesa i neka su $c_1,\ldots,c_k$ klase podataka koje se stvorene ili korištene od strane oba procesa. Neka konekcije koje tvore te klase podataka imaju težine $w_1,\ldots,w_k$, $w_l \geq 0$, $l = 1,\ldots,k$. Tada definiramo adheziju između procesa p_i i p_j kao*

$$Ad(p_i,p_j) = \sum_{l=1}^{k} w_l \tag{1.1}$$

Definicija 1.1.3 (Lovrenčić, 1997)**.** *Neka su k_i i k_j dva podsustava i neka je podsustav k_i sačinjen od procesa $p_1^{(i)},\ldots,p_m^{(i)}$, a podsustav k_j od procesa $p_1^{(j)},\ldots,p_n^{(j)}$. Tada je adhezija između podsustava k_i i k_j jednaka*

$$Ad(k_i,k_j) = \sum_{q=1}^{m}\sum_{r=1}^{n} Ad(p_q^{(i)},p_r^{(j)}) \tag{1.2}$$

Definicija 1.1.4 (Lovrenčić, 1997)**.** *Neka je S sustav dekomponiran u podsustave $k_1,\ldots,k_m$. Njegova ukupna adhezija sustava je*

$$Ad(S) = \sum_{i=1}^{m}\sum_{\substack{j=1\\ j\neq i}}^{m} Ad(k_i,k_j) \tag{1.3}$$

"S obzirom da će povećanje adhezije uzrokovati smanjenje kohezije i obratno, kvaliteta dekompozicije sustava obrnuto je proporcionalna adheziji između podsustava. Stoga je cilj ovakvog načina određivanja podsustava IS-a maksimalno smanjenje adhezije s obzirom na definirani broj podsustava." (Lovrenčić, 1997) Nakon što smo grupirali procese u podsustave dolazimo do našeg problema Određivanja redoslijeda razvoja podsustava IS-a. Problem Određivanja redoslijeda razvoja podsustava IS-a nalazi se u području teorije grafova i opisuje se preko usmjerenog cikličkog grafa (u daljnjem tekstu graf) $G = (V, E)$, gdje E predstavlja skup lukova odnosno lukove između čvorova (skup klasa podataka između podsustava), a V predstavlja podsustave IS-a odnosno čvorove. (Kudelić et al., 2011)

Definicija 1.1.5 (Lovrenčić, 1997)**.** *Procesi p_i i p_j su spojeni klasom podataka c_m ako jedan od njih stvara klasu a drugi ju koristi.*

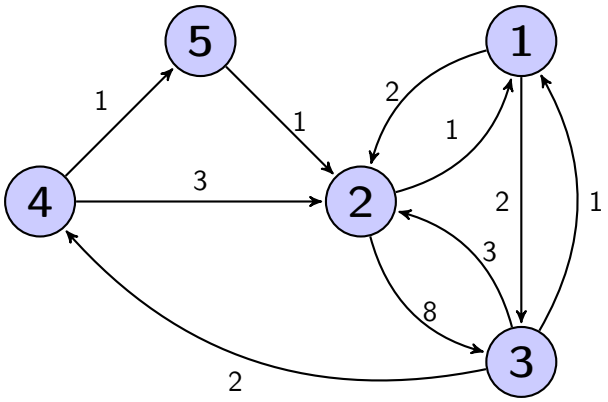
Graf na svakom luku ima težinu pri čemu je $T(E) \geq 1$. Ako imamo luk težine t_{ij}, kojemu je izvorište podsustav ISs_i a odredište podsustav ISs_j, to znači da iz podsustava ISs_i u podsustav ISs_j putuje t_{ij} klasa podataka. "Kao što je vidljivo iz tablice 1.1 normalna je pojava da podsustavi IS-a razmjenjuju relativno mnoštvo klasa podataka. Barem jedna klasa podataka mora biti razmijenjena između podsustava IS-a i IS-a, u suprotnom podsustav nije dio IS-a. Graf G sadrži cikluse što znači kako postoji čvor V u grafu G za koji vrijedi $v_i \to v_{i+1} \to v_{i+2} \to \cdots v_i$. Ovakvi ciklusi predstavljaju problem kod određivanja slijeda razvoja podsustava IS-a s obzirom kako to znači da je podsustav A ovisan o nekom podsustavu B te se postavlja pitanje koji bi podsustav trebali prije razviti. Očito je problem razvijati podsustav koji je snažno ovisan o ostatku IS-a, potrebne su mu klase podataka koje trenutno nema, i stoga bi takve situacije bilo poželjno izbjeći" (Kudelić et al., 2011) s obzirom da je u takvim slučajevima potrebno raditi privremena sučelja koja moraju simulirati rad podsustava koji još nisu implementirani. "Radi lakše predodžbe pogledajmo sljedeći primjer uz zadani graf i linearan redoslijed (LO). Definirajmo graf $G = (V, E)$ sa sljedećom matricom; matrica sadrži težine odgovarajućih lukova.

$$G = \begin{pmatrix} 0 & 2 & 2 & 0 & 0 \\ 1 & 0 & 8 & 0 & 0 \\ 1 & 3 & 0 & 2 & 0 \\ 0 & 3 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{1.4}$$

Pretpostavimo linearan redoslijed od G. $LO(G) := [ISs_1, ISs_2, \ldots, ISs_{i-1}, ISs_i]$ je u našem slučaju" (Kudelić et al., 2011)

$$LO[ISs_3, ISs_2, ISs_1, ISs_4, ISs_5]. \tag{1.5}$$

Što znači kako se pretpostavljeni IS izvršava odnosno razvija od podsustava 3 do podsus-



Slika 1.1: Graf prema matrici 1.4.

tava 5. (Kudelić et al., 2011) I sada se problem Određivanja redoslijeda razvoja podsustava IS-a vrlo jasno vidi. (Kudelić et al., 2011) Podsustav ISs_3 sveukupno zahtjeva 10 klasa podataka od ostalih podsustava. Što znači, ako želimo prvo razviti podsustav ISs_3 moramo imati na raspolaganju 10 skupova podataka ili drugačije rečeno trebamo imati na raspolaganju n "dijelova" ostalih podsustava kako bismo mogli u potpunosti razviti, testirati, i pustiti u pogon ISs_3. Stoga se postavlja logično pitanje: Kojim slijedom bi podsustave trebalo razvijati? Očito bi bilo dobro prvo razviti podsustav koji je slabo ovisan o ostatku IS-a i na taj način slijedno graditi podsustave. Ako se podsustavi razvijaju paralelno, što je u pravilu slučaj, slijed razvoja označava prioritet podsustava u razvoju cijelog IS-a. Dakle, davanjem većeg prioriteta podsustavu koji je slabo ovisan o ostatku IS-a osiguravamo potpuniji razvoj ostalih podsustava.

Definicija 1.1.6. *Svaki luk* (u,v), $1 \leq v < u \leq |V|$, *predstavlja povratni luk u linearnom slijedu čvorova grafa.*

Stoga, riješiti problem Određivanja redoslijeda razvoja podsustava IS-a zapravo znači pronaći linearan slijed razvoja sa minimalnom sumom težina povratnih lukova.

1.2 Kritički osvrt i znanstveni doprinos

Razmišljajući znanstveno iz svih kutova mogli bismo rezonirati i na sljedeći način. Naime, istina je kako je uputno razvijati podsustave na prethodno opisani način, međutim podaci ("dijelovi") odnosno moduli ostatka IS-a se mogu simulirati pa je u tom pogledu prioritet razvoja IS-a nebitan. I realno gledano, istina je kako je to moguće. Za jednostavnije sustave ovakav način rada može funkcionirati. Međutim, što je IS koji treba razviti složeniji vrlo je logično za pretpostaviti kako će biti i teže predvidjeti sva realna stanja u kojima će se IS u radu naći.

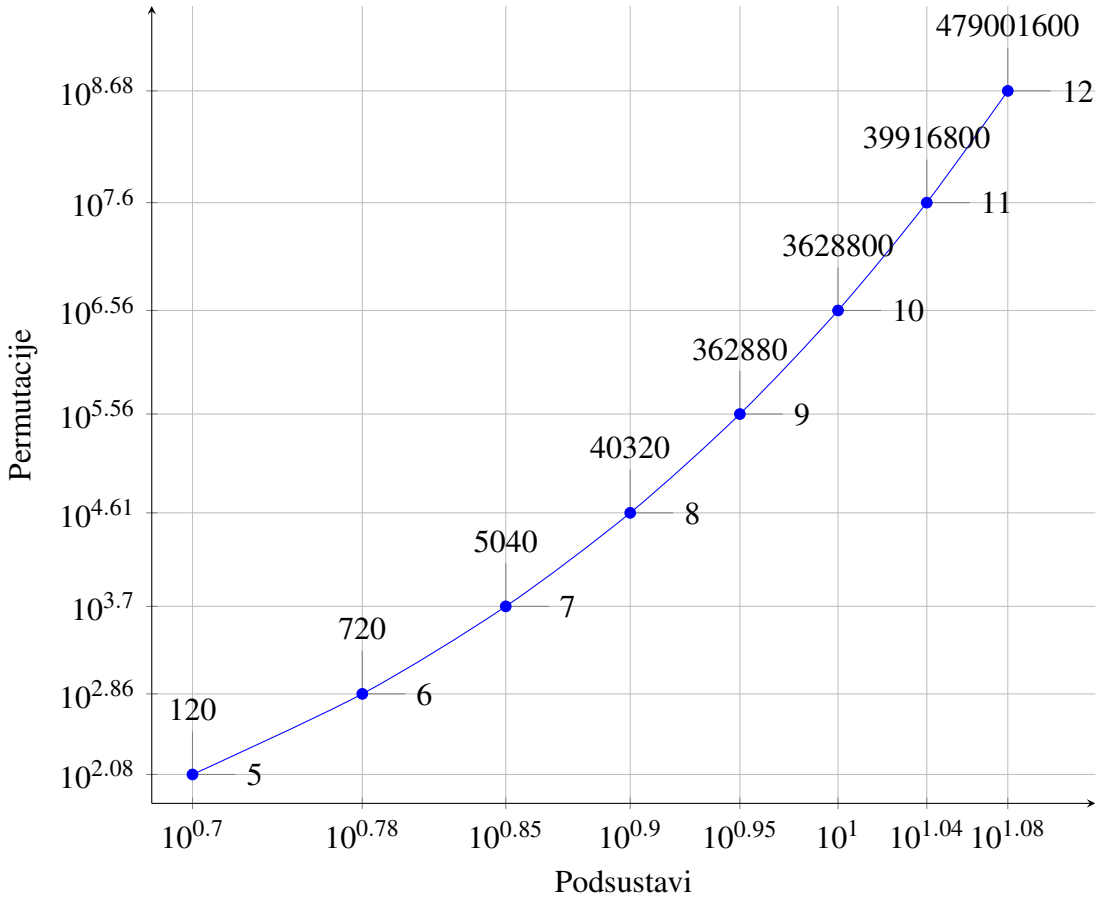
Isto tako, što je skup podataka i modul koji treba simulirati složeniji logično je za pretpostaviti kako će biti teže izraditi simulaciju koja bi bila dovoljno dobra za potrebe razvoja podsustava koji se trenutno razvija (Stonedahl and Rand, 2012) te na taj način postepeno dolazimo do točke gdje je sustav toliko složen da je jedina prava simulacija dijela IS-a zapravo sam taj dio koji treba simulirati u potpunosti implementiran. Nadalje, svaka greška koja nije otkrivena na početku kasnije otežava fazu debagiranja (Zeller, 2009) i testiranja (Myers et al., 2012) i potencijalno povećava vjerojatnost da ne otkrivena greška završi u produkcijskom sustavu. Objavljeno je kako greške koje se pronađu kasnije u razvoju koštaju više od grešaka koje se pronađu u ranijim fazama razvoja (Boehm and Basili, 2001). Ovaj način razmišljanja dodatno potvrđuju podaci koji govore o mnoštvu IS-a koji unatoč najvećim naporima uz mnogo novca ne uspijevaju biti završeni (Computer World, n.g.; Dorsey, 2005; Geambasu et al., 2011).

S obzirom da bi pronalazak slijeda razvoja podsustava sa minimalnom sumom težina povratnih veza osigurao prioritetni razvoj podsustava koji su manje ovisni o ostatku IS-a, logično je zaključiti kako bi takav razvoj, ako ne u potpunosti eliminirao, smanjio probleme koji se javljaju u kasnijim fazama kao što su debagiranje i testiranje. Razvoj IS-a prema takvom slijedu potencijalno bi povećao vjerojatnost ispravljanja grešaka i u ranijim fazama kao što su definiranje zahtjeva i dizajn IS-a s obzirom da ovakav razvoj podrazumijeva raniji razvoj većeg djela IS-a i u tom smislu i ranije vraćanje na prijašnje faze.

Stoga zaključujemo kako je rješavanje ovog problema važno ne samo u teoretskom smislu kroz teoriju grafova i dodirna područja, nego i praktično kao dodatak metodologijama razvoja IS-a. Ova potreba metodologija razvoja IS-a može se vidjeti u (Russo, 1995; Wynekoop and Russo, 2003).

1.3 Hipoteze i ciljevi

Ako bismo htjeli naći optimalno rješenje za problem opisan u podpoglavlju Opis problema trebali bismo sa iscrpnim pretraživanjem proći kroz sve permutacije kako bismo pronašli minimalnu sumu težina povratnih veza u linearnom slijedu razvoju. Za naših 5 podsustava je to 120 što nije puno, međutim sa povećanjem broja podsustava ovakva složenost algoritma vrlo brzo raste. S obzirom na to i na napravljenu analizu u podpoglavlju koje



Slika 1.2: Permutacije kod iscrpnog pretraživanja na logaritamskoj skali.

se bavi sa opisom ovog problema vrlo je vjerojatno kako je ovaj problem NP-težak. Iz ove analize dolazimo do prve hipoteze koja glasi kako sljedi.

HIPOTEZA 1 Problem Određivanja redoslijeda razvoja podsustava u informacijskom sustavu je NP-težak i APX-težak.

S druge strane kao računalne znanstvenike zanima nas koji je algoritam rješavanja odnosno koji je efikasan način rješavanja problema. Iz tog razloga te uz pretpostavku da će prva hipoteza biti dokazana hipoteza druga glasi kako sljedi.

HIPOTEZA 2 Algoritam temeljen na metodi Monte Carlo rješava problem Određivanja redoslijeda razvoja podsustava u informacijskom sustavu s proizvoljnom vjerojatnošću dosizanja optimuma u polinomnom vremenu.

S obzirom na dane hipoteze cilj istraživanja glasi kako sljedi.

CILJ Pronaći efikasno algoritamsko rješenje za problem Određivanja redoslijeda razvoja podsustava informacijskog sustava.

1.4 Doseg istraživanja i metodologija

U ovom radu napraviti ćemo pregled područja s obzirom na prethodno opisani problem. S obzirom da bismo htjeli znati o koliko teškom problemu se radi dokazati ćemo da je problem NP-težak i APX-težak. U suprotnom nismo sigurni što napraviti ako ne uspijemo pronaći efikasan algoritam koji pronalazi optimalno rješenje.

Nakon toga, preko metode Grananja i ograničenja, pronaći ćemo algoritam koji uvijek pronalazi optimalno rješenje. Uz pomoć tog algoritma pronaći ćemo skup optimalnih rješenja za grafove u svrhu daljnjeg testiranja efikasnijeg algoritma i procjene pogreške. Nakon toga ćemo razviti Monte Carlo randomizirani algoritam i usporediti rezultate sa rezultatima algoritma koji se temelji na metodi Grananja i ograničenja. Odabir Monte Carlo randomiziranog algoritma biti će jasniji kada krenemo s osmišljavanjem dotičnog algoritma.

Nakon toga ćemo pokazati na koji način se razvijeno rješenje uklapa u postojeće metodologije razvoja te na koji način je moguće ugrađivati dodatna ograničenja koja su ponekad u praksi potrebna. Radi potpunosti rješavanja problema pokazati ćemo i prethodno pronađenu heuristiku. Isto tako, dotaknuti ćemo se i nekoliko srodnih problema na koje se osmišljeni algoritmi potencijalno mogu primijeniti. S obzirom na hipoteze, ciljeve, i doseg istraživanja, metodologiju istraživanja definirati ćemo na sljedeći način.

Pregled literature. Pregled literature kroz područja vezana za problem Određivanja redoslijeda razvoja podsustava u informacijskom sustavu.

Analiza poznatih metoda i algoritama. Analiza pronađenih metoda i algoritama te komparacija sa problemom istraživanja. Radi bolje povezanosti materije ovaj korak biti će napravljen zajedno sa pregledom literature.

Dokazivanje težine problema. Dokazivanje da je problem Određivanja redoslijeda razvoja podsustava u informacijskom sustavu NP-težak i APX-težak.

Traženje optimalnog rješenja. Osmišljavanje algoritma, koji pronalazi optimalno rješenje, temeljenog na metodi Grananja i ograničenja.

Traženje efikasnog algoritma. Osmišljavanje Monte Carlo randomiziranog algoritma koji će rješavati problem Određivanja redoslijeda razvoja podsustava u informacijskom sustavu s proizvoljnom vjerojatnošću dosizanja optimuma u polinomnom vremenu.

Načini uvrštavanja dodatnih ograničenja i primjena na ostale probleme. Provođenje jednog primjera za svaki algoritam sa uvođenjem dodatnih ograničenja. Razmatranje potencijalne primjene osmišljenih algoritama na ostale slične probleme.

Poglavlje 2

Pregled literature

Kazalo poglavlja

Problem Određivanja redoslijeda razvoja IS-a svojim opisom pripada teoriji grafova. Svoje izvorište ima u razvoju informacijskih sustava a indirektno preko razvoja IS-a dodiruje i modeliranje poslovnih procesa (MPP). Stoga je pripadajući pregled literature moguće napraviti isključivo spajanjem ta tri područja. U sljedeća tri poglavlja proći ćemo kroz dodirne točke našeg problema APOSRPIS i područja: modeliranja poslovnih procesa, modeliranja IS-a, i teorije grafova. Kroz takav pregled pokazati ćemo: vezu između svakog područja i našeg problema, na koji način se prema našim saznanjima slijed razvoja po područjima radi sada, i kakav utjecaj takav način rada ima na naš problem.

2.1 Modeliranje poslovnih procesa

Prvo definirajmo što je to MPP. Kako bismo dali potpuniju i razumljiviju definiciju definirati ćemo dva pojma, općenitiji, upravljanje poslovnim procesima (UPP), i sažetiji koji je nama potreban, MPP. Prema (Ko et al., 2008; van der Aalst et al., 2003) pojam UPP je definiran kao "potpora poslovnim procesima uz korištenje metoda, tehnika i softvera radi dizajniranja, propisivanja, kontroliranja i analiziranja operacijskih procesa koji se tiču ljudi, organizacija, softvera, dokumenata i ostalih izvora informacija".

Prema (Ko et al., 2008; van der Aalst et al., 2003) UPP je najbolje shvatiti preko životnog ciklusa. Prema (Ko et al., 2008; van der Aalst et al., 2003) životni ciklus UPP-a "sastoji se od sljedeće četiri faze:

1. dizajniranje procesa,

2. namještanje sustava,

3. propisivanje procesa,

4. dijagnostika."

Iz ove dvije definicije vidimo kako je MPP dio UPP-a. Dakle, MPP predstavlja prvu i eventualno drugu fazu životnog ciklusa UPP-a za koju bismo mogli reći da upotpunjuje prvu fazu. Prva faza je prema (Ko et al., 2008; van der Aalst et al., 2003) definirana kao "elektroničko modeliranje trenutnog stanja poslovnih procesa u sustav za UPP (BPMS)". A druga faza je prema (Ko et al., 2008; van der Aalst et al., 2003) definirana kao "namještanje BPMS-a i temeljne infrastrukture sustava (npr. sinkronizacija uloga, ...)".

Modeliranje poslovnih procesa se u pravilu obavlja preko skupa tehnika. Najčešće korišteni skup tehnika je prema (Giaglis, 2001):

1. "dijagram toka,

2. IDEF tehnike (IDEF0, IDEF3),

3. petrijeve mreže,

4. simulacija,

5. tehnike bazirane na znanju (eng. Knowledge-based techniques),

6. dijagramiranje aktivnosti uloga (eng. Role activity diagramming)."

Detaljan opis načina rada sa tehnikama vidi se u (Giaglis, 2001). Globalno gledano, MPP pokušava preslikati trenutno stanje organizacije u računalne modele a UPP pokušava to stanje poboljšati s obzirom na definirani kriterij. (Ko et al., 2008) S obzirom da je MPP fokusiran prvenstveno na organizaciju, određivanje slijeda razvoja budućeg IS-a nije u glavnom fokusu.

Međutim, svaka organizacije ima određenu djelatnost s kojom se bavi. S obzirom na tu djelatnost organizacija je postavljena u određenu organizacijsku strukturu sa određenim poslovnim funkcijama, što se može vidjeti u (Žugaj et al., 2004). Ta organizacijska struktura s obzirom na djelatnost i poslovne funkcije ima svoj logičan slijed, npr. od proizvodnje pa sve do marketinga i prodaje. Što znači da kada radimo UPP organizacija se modelira i optimizira upravo s obzirom na tu djelatnost i poslovne funkcije jer se tamo nalaze poslovni procesi. Stoga se kao logičan slijed razvoja IS-a koji proizlazi iz MPP-a odnosno UPP-a nameće onaj slijed koji proizlazi iz poslovnih funkcija i djelatnosti organizacije.

Na primjer, jedan slijed bi mogao biti (sljedeći slijed razvoja IS-a predstavlja djelomičan popis poslovnih funkcija prema (Žugaj et al., 2004)):

1. "nabava,
2. proizvodnja,
3. prodaja,
4. računovodstvo,
5. menadžment."

Pokušamo li ovakav slijed razvoja sagledati općenito uviđamo sljedeće. Ovakav slijed razvoja podsustava IS-a je relativno statičan. Nabava, proizvodnja, kao i sve ostale funkcije u pravilu će zauzimati iste ili vrlo bliske pozicije u slijedu razvoja podsustava IS-a za različite organizacije. Ovaj slijed ne mora biti univerzalan i kao što se vidi iz (Žugaj et al., 2004) on se razlikuje s obzirom na branšu organizacije, međutim unutar branše nije logično očekivati prevelike promjene. Razlog tome je vrlo jednostavan i kao ilustraciju uzmimo sljedeći primjer.

Ako se nalazimo u poslu proizvodnje namještaja prethodno dani primjer slijeda razvoja podsustava IS-a može poslužiti kao osnovan popis funkcija i kao osnovan slijed rada organizacije s obzirom na djelatnost. Uzmemo li sada kao primjer funkciju nabave i proizvodnje, i pokušamo li ih preslikati na neku drugu organizaciju postavlja se sljedeće pitanje: Bismo li morali promijeniti redoslijed tih funkcija i ako bismo morali mijenjati slijed za koliko mjesta bi to bilo? S obzirom da su organizacije u istoj branši bilo bi iznimno nelogično očekivati drastične promjene ili recimo promjenu koja bi zahtijevala prvo proizvodnu funkciju pa onda nabavnu. Možda bismo i mogli između proizvodnje i nabavljanja postaviti npr. kontroliranje kvalitete materijala ili nešto treće, međutim odnos pozicija funkcije nabave i proizvodnje bi ostao isti uz manje promjene koje bi se događale oko tih funkcija.

Iz ovog primjera je očito kako su u ovakvom načinu određivanja slijeda razvoja IS-a pomaci funkcija odnosno podsustava budućeg IS-a vrlo mali. Dakle, ovakav način određivanja slijeda je u pravilu vrlo statičan. Ako to usporedimo sa našim opisom problema uviđamo da je općenito gledano vrlo teško za očekivati kako će ovakav način određivanja slijeda funkcionirati, naime naš opis problema uzima u obzir dinamičnost podsustava IS-a preko klasa podataka dok ovakvo određivanje slijeda uzima u obzir statičnu strukturu funkcija organizacije i slijed proizvoda ili usluge koju nudimo, što se rijetko i u pravilu vrlo malo mijenja.

Stoga zaključujemo kako je s obzirom na opis problema koji je dan u prijašnjem poglavlju ovakav način određivanja slijeda razvoja IS-a neadekvatan. Potrebno je naći rješenje koje će biti u stanju s obzirom na dinamiku svakog IS-a pojedinačno odrediti slijed koji najbolje odgovara toj specifičnoj organizaciji umjesto cijeloj grupi organizacija. Slijed koji bi trebao odgovarati cijeloj grupi organizacija, s obzirom da je rađen generički za svakoga, najvjerojatnije odgovara nikome ili maloj grupi organizacija.

2.2 Razvoj informacijskih sustava

S obzirom na problem koji želimo riješiti, razvoj IS-a promatrati ćemo kroz metodologije razvoja IS-a. Metodologija je definirana kao "općenita strategija istraživanja koja okvirno određuje način na koji treba provesti istraživanje. Metodologija identificira skup metoda koje će se u metodologiji koristiti. Te metode opisuju način prikupljanja podataka ili način računanja određenog rezultata. ... " (Howell, 2012)

S obzirom na to prema (Centers for Medicare & Medicaid Services, 2008) metodologija razvoja softvera definirana je kao "okvir koji se koristi u svrhu strukturiranja, planiranja i kontroliranja procesa razvoja IS-a." Metodologije koje su ostale u opticaju i koje se spominju kao najčešći izbor kod razvoja IS-a su:

1. "tradicionalne metodologije:
   (a) Slap (Laplante and Neill, 2004),
   (b) Transformacija (Winter et al., n.g.);
2. evolucijske metodologije:
   (a) Pobjeda-Pobjeda (Boehm et al., 1998),
   (b) Spirala (Boehm, 2000; Boehm et al., 1998);
3. Brzi razvoj softvera (eng. Rapid Application Development) (Geambasu et al., 2011),
4. agilne metodologije:
   (a) Scrum (Sutherland et al., 2007),
   (b) Ekstremno programiranje (Geambasu et al., 2011)," (Simão, 2009)
   (c) Razvojna metoda za dinamične sustave (eng. Dynamic systems development method) (Coleman and Verbruggen, 1998; Stapleton, 1999),
   (d) Vitki razvoj softvera (eng. Lean software development) (Poppendieck and Poppendieck, 2003),
   (e) Agilno modeliranje (Boehm, 2002; Scott Ambler Associates, n.g.);
5. Rational unificirani proces (eng. Rational Unified Process) (Geambasu et al., 2011).

Proučavajući metodologije razvoja IS-a nismo naišli na formalno definiran način određivanja slijeda razvoja podsustava IS-a koji bi riješio problem kako je definiran u uvodu ovog rada. Ipak, tijekom razvoja IS-a slijed razvoja se mora odrediti i on se određuje usprkos tome što ga metodologije ne propisuju.

Mogli bismo argumentirati kako je to zato što se slijed razvoja IS-a trenutno određuje preko poslovnih funkcija, odnosno slijeda poslovanja kako je opisano u prethodnom poglavlju, ili nekog drugog poslovnog ili razvojnog kriterija. Prema našem iskustvu neki od kriterija za određivanje slijeda razvoja podsustava IS-a su:

1. poslovne funkcije (životni ciklus proizvoda/usluge),
2. ograničenja tehnologije,
3. prioritetno rješavanje izvirućih problema ili "težina" razvoja podsustava,
4. količina rada,
5. naređenje naručitelja,
6. dovršenost postojećeg rješenja,
7. najbolja praksa.

Niti jedan kriterij ovog tipa nije u mogućnosti riješiti problem Određivanja redoslijeda razvoja podsustava IS-a kako je opisano u podpoglavlju Opis problema. Razlog, niti jedan kriterij nema za cilj smanjenje sume težina povratnih veza u linearnom slijedu razvoja podsustava IS-a pa je u tom smislu nerealno za očekivati da će bilo koji od tih kriterija ili skup kriterija dati takav linearan slijed razvoja gdje će suma težina povratnih veza u slijedu razvoja IS-a biti konzistentno minimalna.

Stoga zaključujemo kako problem Određivanja redoslijeda razvoja podsustava IS-a nije adekvatno riješen. Potragu za mogućim rješenjem nastavljamo u sljedećem poglavlju kroz teoriju grafova. Teorija grafova sadrži veliki skup algoritama od kojih bi barem jedan trebao približno biti primjenjiv na naš problem. Kada nađemo takav algoritam on će biti temelj za daljnja razmišljanja i moguće rješenje našeg problema.

2.3 Teorija grafova

U ovom poglavlju dati ćemo pregled algoritama i metoda iz područja teorije grafova ali i računalnih znanosti općenito koji bi nam mogli pomoći u rješavanju našeg problema. Poglavlje je strukturirano na način da prvo dajemo opis algoritma ili metode, nakon toga dajemo kritički osvrt metode ili algoritma u vidu rješavanja našeg problema, i na kraju razmatramo primjenu algoritma ili metode na naš problem.

Jedan od algoritama koji bi nam mogao biti od koristi je **Topološko sortiranje** (eng. Topological Sort) (Pearce and Kelly, 2006). "Topološko sortiranje je algoritam koji služi za sortiranje vrhova u usmjerenom acikličkom grafu. Topološko sortirani graf $G = (V,E)$ predstavlja linearan redoslijed svih svojih vrhova ako za svaki luk (u,v) vrijedi da u prethodi v u linearnom redoslijedu. Ako graf nije aciklički, tada ne postoji topološko sortirani linearan redoslijed. Topološko sortiranje ima linearnu složenost $O(|V|+|E|)$." (Cormen et al., 2009; Pearce and Kelly, 2006) Pseudokod algoritma Topološkog sortiranje prema (Kahn, 1962) dan je algoritmom 2.1. "Kao što se može vidjeti iz algoritma ako ne pos-

Algoritam 2.1 Pseudokod algoritma Topološkog sortiranja. (Kahn, 1962)

Ulaz: Aciklički graf.
Izlaz: Topološko sortirani niz.
{$L \leftarrow$ Prazna lista koja će sadržavati topološko sortirani niz.}
{$S \leftarrow$ Skup vrhova ulaznog stupnja 0.}

```
1:  while S ne prazan do
2:      makni vrh n iz S
3:      umetni n u L
4:      for all vrh m sa lukom e = (n,m) do
5:          makni luk e iz grafa
6:          if vrh m nema drugih ulaznih lukova then
7:              umetni m u S
8:          end if
9:      end for
10: end while
11: if graf sadrži lukove then
12:     return greška (graf sadrži ciklus)
13: else
14:     return L (topološko sortirani niz)
15: end if
```

toji barem jedan vrh koji nema ulazne veze tada nije moguće odrediti topološko sortirani slijed. Sortirani graf dan je u L, u suprotnom topološko sortirani slijed ne postoji jer tada imamo barem jedan ciklus u grafu koji želimo topološki sortirati." (Kahn, 1962) Postoji druga implementacija algoritma Topološkog sortiranja. Kod ove implementacije radi se topološko sortiranje vrhova acikličkog usmjerenog grafa pri čemu pronalazak čvora v dva puta dokazuje pojavnost ciklusa u grafu, što znači da graf ne može biti topološki sortiran.

(Cormen et al., 2009) Na ovaj način topološko sortirani slijed pronalazi se pomoću algoritma Dubinskog pretraživanja (eng. Depth First Search). Topološko sortiranje najčešće se koristi za rješavanje problema gdje je potrebno odrediti linearan slijed vrhova s obzirom na neko ograničenje, kao što je sortiranje usmjerenih acikličkih grafova (Arge et al., 2003; Poras et al., 2011) i drugih mreža koje se mogu svesti na problem ovog tipa. Topološko sortiranje vrlo se često koristi za sortiranje zadataka s obzirom na preduvjete (Saltz et al., 1991) što zapravo predstavlja problematiku određivanja slijeda razvoja podsustava IS-a. Postoje napori koji primjenjuju ovaj algoritam na probleme koji se javljaju kod optimalizacije poslovnih procesa kroz binarnu matricu incidencije (Yuan and Wang, 2012). Radi se i na ostalim istraživanjima kao što su upravljanje rasporedom u višeprocesorskom radu (Satish et al., 2008) i evolucijski pristup izradi rasporeda osjetljivih zadataka (Sutherland et al., 2007). Algoritam za Topološko sortiranje očito je dobro poznat i često korišten. Isto tako radi se o algoritmu koji je efikasan. Međutim radi se i o algoritmu koji nije sposoban razriješiti cikluse. S druge strane naš problem podrazumijeva pojavu ciklusa i nije ih moguće izbjeći s obzirom da u grafu koji predstavlja naš problem ne postoji pravilo povezivanja čvorova i lukovi se u graf dodjeljuju mogli bismo reći arbitrarno. Kada bismo postavili određena ograničenja nad povezivanje čvorova sa lukovima došli bismo u situaciju gdje ne bi bilo moguće opisati svaki IS. S obzirom na to ovaj algoritam nije direktno primjenjiv na naš problem, ali ako bismo uspjeli pronaći neki način razrješavanja ciklusa koji bi garantirao minimalnu sumu težina povratnih veza ovaj algoritam bio bi u stanju riješiti problem Određivanja redoslijeda razvoja podsustava IS-a.

Sljedeći algoritam koji bi nam mogao biti od koristi u rješavanju našeg problema, a već smo ga i spomenuli, je **Dubinsko pretraživanje** (eng. Depth First Search). "DFS algoritam služi za pretraživanje stabla/grafa. Radi na takav način da za određeni vrh v pretražuje graf u dubinu i pri tome se naravno lukovi E istražuju od vrha na kojem trenutno jesmo. Nakon što više nije moguće ići u dubinu moramo se vratiti do pozicije koja sadrži lukove kojima još nismo putovali i tada pretražujemo dodatne vrhove koje još nismo posjetili. Ovaj postupak se ponavlja sve dok nismo posjetili sve vrhove grafa $G = (V, E)$ krećući od vrha kojeg smo odabrali za početni (početni se vrh u pravilu odabire arbitrarno). U slučaju da DFS ne uspije otkriti sve vrhove moramo cijeli postupak ponoviti, ali ovaj put sa drugim vrhom kao početnim." (Cormen et al., 2009) Složenost DFS-a je $O(|V| + |E|)$ (Tarjan, 1972). Pseudokod rekurzivnog DFS algoritma prema (Cormen et al., 2009; Goodrich and Tamassia, 2001) dan je algoritmom 2.2. DFS algoritam ima široku primjenu. Često se koristi za topološko sortiranje (Cormen et al., 2009; Tarjan, 1976), detekciju ciklusa (Tarjan, 1972) i testiranje planarnosti (de Fraysseix et al., 2006). Algoritam je efikasan i pogodan za rješavanje problema koji su u svojoj naravi, nazovimo ih tako, slijedni. Izvršimo li DFS algoritam na grafu koji opisuje naš problem dobiti ćemo u linearnom vremenu određeni linearan slijed no ne znamo koliko je to rješenje dobro. Pretvorimo li pak graf koji opisuje naš problem u potpuni graf mogli bismo iskoristiti

Algoritam 2.2 Pseudokod rekurzivnog DFS algoritma. (Cormen et al., 2009; Goodrich and Tamassia, 2001)

Ulaz: Graf G i početni čvor $v \in G$.
Izlaz: Pronađeni i označeni skup čvorova koji su u dosegu početnog čvora $v \in G$.

```
1: funkcija DFS(G, v) :
2: označi v (čvor je otkriven)
3: for all lukove (v, w) ∈ G.susjedniLukovi(v) do
4:     if čvor w nije označen (čvor nije otkriven) then
5:         pozovi DFS(G, w)
6:     end if
7: end for
```

DFS algoritam za pronalazak optimalnog rješenja za naš problem. Naime, ukoliko izvršimo DFS algoritam na ovakvom grafu za svaki čvor kao početni tražeći sve putove i pri tome režući grane koje nisu perspektivne moguće je deterministički odrediti slijed čvorova koji bi imao minimalnu sumu težina povratnih veza. Mana ovakvog algoritma je njegova eksponencijalna ili gora složenost (Levitin, 2011), tako da nam efikasno rješenje još uvijek izmiče.

Rješavanje problema Dubinskim pretraživanjem općenito je u informatici poznato kao **metoda Traganja s vraćanjem** (eng. Backtracking). Metoda Traganja s vraćanjem često se koristi kod zahtjevnih problema gdje je potrebno znati optimalno rješenje kako bi se izbjeglo korištenje metode Iscrpnog pretraživanja koja pretražuje cijelo područje pretrage (eng. Brute force). "Metoda Traganja s vraćanjem se bazira na inteligentnijem pristupu metodi Iscrpnog pretraživanja. Princip ove metode je graditi rješenje postepeno kroz komponente i pri tome evaluirati takva rješenja na sljedeći način. Ako se komponenta može unaprijediti bez kršenja ograničenja problema, tada se u komponentu dodaje prvi sljedeći obećavajući čvor, u protivnom se algoritam vraća natrag na prvu komponentu sa kojom može raditi. Ustaljena implementacija ove metode je preko algoritma koji koristi stablo stanja kao način dolaska do konačnog rješenja. Vrh takvog stabla zove se obećavajući u slučaju da može dovesti do konačnog rješenja, inače je naziv ne obećavajući. U slučaju obećavajućeg čvora gradi se sljedeća dublja komponenta, u suprotnom se ne obećavajuća grana odbacuje i vraćamo se natrag po stablu kroz čvorove sve dok ne dođemo do komponente sa kojom možemo dalje raditi. Kada stignemo do lista stabla komponente koja ne krši ograničenja problema, ta komponenta predstavlja rješenje problema ili jedno rješenje od više njih. Kada algoritam dostigne rješenje proces pretrage se u potpunosti zaustavlja ili se nastavlja sve dok se ne nađu sva rješenja, ako se radi o problemu gdje rješenje nije jedinstveno." (Knuth, 2011; Levitin, 2011)

Sljedeća metoda koja bi nam mogla pomoći u rješavanju našeg problema je **metoda Pohlepe** (eng. Greedy). Metoda Pohlepe je vrlo jednostavna i izravna. "Metoda Pohlepe nalaže sljedeće. Uzimajući u obzir ulaz n te funkciju cilja, pohlepno stvaraj lokalni optimum u nadi pronalaska globalnog optimuma. Rješenje koje zadovoljava funkciju cilja

naziva se moguće rješenje. Iz ovoga proizlazi da moramo pronaći moguće rješenje koje maksimizira ili minimizira funkciju cilja te se moguće rješenje, koje ovo radi, naziva najbolje rješenje." (Levitin, 2011) Metoda je iznimno brza iako kratkovidna. Pseudokod prema (Horowitz and Sahni, 1984) za općenitu funkciju koja koristi metodu Pohlepe dan je algoritmom 2.3. "Funkcija *Izaberi* uzima za argument A i vraća potencijalno rješenje te ga stavlja u x, *Moguće* je funkcija koja vraća istinitost tvrdnje s obzirom na moguće rješenje $rješenje$ i x, funkcija $\cup$ u slučaju da x poboljšava moguće rješenje stvara novo rješenje i ažurira stanje funkcije cilja." (Levitin, 2011) Ova metoda najčešće se koristi kada nije

Algoritam 2.3 Pseudokod općenite funkcije metode Pohlepe. (Horowitz and Sahni, 1984)

Ulaz: Skup kandidata A i broj izvršavanja algoritma n.
Izlaz: Rješenje stvoreno iz skupa kandidata A.

```
1: funkcija Greedy(A,n) :
2: rješenje ← ∅
3: for i ← 0 to n do
4:     x ← Izaberi(A)
5:     if Moguće(rješenje,x) then
6:         rješenje ← ∪(rješenje,x)
7:     end if
8: end for
9: return rješenje
```

nužno znati optimalno rješenje ali nam je potreban odgovor brzo. Iako, postoje algoritmi kao što je na primjer Primov algoritam (Prim, 1957) za pronalazak minimalnog razapinjućeg stabla (MST) koji pomoću metode Pohlepe dokazano pronalazi optimum. Ovu metodu je naravno moguće iskoristiti za rješavanje našeg problema. Primjer mogućeg heurističkog rješenja vidi se u našem pred istraživanju u (Kudelić et al., 2011). Međutim, problem takvog heurističkog rješenja je to što ne znamo koliko algoritam griješi i stoga nismo sigurni u rezultat koji algoritam daje kao izlaz. Stoga će svrha ovog rada biti pronalazak algoritma za koji možemo izračunati mjeru sigurnosti za izlaz koji daje.

Sljedeća i zadnja metoda koju ćemo spomenuti, s obzirom da rješavamo optimizacijski problem, je **metoda Grananja i ograničenja** (eng. Branch and Bound). Ovo je metoda koja dokazano daje optimalno rješenje, iako u najgorem slučaju složenost algoritma može otići u eksponencijalno vrijeme ili gore. (Levitin, 2011; Winston, 1992) Uz slojeve koji se dodaju na ovu metodu kroz dopustive heuristike metoda u određenim slučajevima može biti iznimno brza. (Winston, 1992) Ova metoda za razliku od DFS-a, odnosno metode Traganja s vraćanjem, istovremeno proširuje sve grane čvora koji je perspektivan. (Winston, 1992) Perspektivnost čvora određuje se preko funkcije pomoću koje utvrđujemo vodi li čvor potencijalno do rješenja koje je bolje od onog kojeg trenutno imamo. (Winston, 1992) Grane koje nisu perspektivne, one koje ne vode do rješenja i one koje ne vode do boljeg rješenja, se režu. (Winston, 1992) Na ovaj način smanjujemo područje pretrage.

(Winston, 1992) Nakon proširenja odabire se novi perspektivan čvor koji se dalje proširuje. (Winston, 1992) Ovaj postupak se ponavlja sve dok ne dođemo do lista koji je izabran kao perspektivan. (Winston, 1992) To znači da je za daljnje proširenje odabran čvor koji se ne može više proširiti i stoga je niz čvorova koji je doveo do lista stabla trenutno najbolje rješenje. (Levitin, 2011; Winston, 1992) Najbolje rješenje pamtimo i cijeli postupak ponavljamo sve dok nismo sve ispitali. (Winston, 1992) Neki od primjera korištenja ove metode jesu problem Trgovačkog putnika (Conway et al., 2003), optimizacija sa više ciljeva (Žilinskas and Zhigljavsky, 2004). Iz opisa prethodne metode i podpoglavlja Opis problema očito je kako ova metoda predstavlja odličan izbor za rješavanje našeg problema, optimalno. Stoga ćemo u sljedećem poglavlju, koje se bavi rezultatima našeg istraživanja, osmisliti algoritam koji pomoću ove metode pronalazi optimalno rješenje za naš problem.

Poglavlje 3

Rezultati

Kazalo poglavlja

Sada kada smo se detaljnije upoznali sa problemom možemo krenuti na njegovo rješavanje. Stoga ćemo ovo poglavlje započeti sa problematikom težine nakon čega ćemo krenuti u osmišljavanje, analizu, i izradu algoritama te eksperimentiranje i procjenu kvalitete rezultata. Poglavlje ćemo zaključiti sa opisom načina korištenja algoritama te sa načinima uvrštavanja dodatnih ograničenja koja su ponekada u praksi nužna. Isto tako, ukratko ćemo se dotaknuti primjene algoritama, koji rješavaju problem Određivanja redoslijeda razvoja podsustava IS-a, na ostale probleme.

3.1 Problem Određivanja redoslijeda razvoja podsustava IS-a je NP-težak

Nakon matematičkog opisa problema, koji je dan u podpoglavlju Opis problema, Određivanja redoslijeda razvoja podsustava informacijskog sustava (OSRPIS), kroz teoriju grafova, uvidjeli smo kako je dotični optimizacijski problem vrlo vjerojatno NP-težak. Pronalazak efikasnog algoritma koji daje optimalno rješenje tog problema uvelike ovisi o toj mogućnosti. Stoga bi bilo dobro saznati jeli to zaista tako kako smo to mi naslutili. Problem Određivanja redoslijeda razvoja podsustava informacijskog sustava u optimizacijskoj verziji glasi kako sljedi.

Instanca: Usmjereni graf $G = (V, E)$.

Pitanje: Koji linearan slijed vrhova v_i $(i = 1 \ldots |V|)$ grafa G daje minimalnu sumu težina povratnih veza (lukova) $\sum_{\substack{(u,v)\in E \\ u>v}} T(u,v)$?

Problem Određivanja redoslijeda razvoja podsustava IS-a u verziji odluke glasi kako sljedi.

Instanca: Usmjereni graf $G = (V, E)$, prirodan broj $K \leq \sum E$.

Pitanje: Postoji li linearan slijed vrhova v_i $(i = 1 \ldots |V|)$ grafa G pri čemu je suma težina povratnih veza (lukova) $\sum_{\substack{(u,v)\in E \\ u>v}} T(u,v) \leq K$?

Pretražujući znanstvenu literaturu u potrazi za problemom koji bismo mogli svesti na naš problem u (Crescenzi and Kann, 1998) nailazimo na dobro poznati problem iz teorije grafova, Minimalni povratni skup lukova (MPSL; eng. Minimum Feedback Arcs Set). Ovaj problem u verziji odluke je NP-potpun (Karp, 1972) i prema (Crescenzi and Kann, 1998) glasi kako sljedi.

"Instanca: Usmjereni graf $G = (V, A)$, prirodan broj $K \leq |A|$.

Pitanje: Postoji li podskup $A' \subseteq A$ pri čemu je $|A'| \leq K$ i A' sadrži barem jedan luk iz svakog usmjerenog ciklusa u G?"

S obzirom da je ovaj problem u verziji odluke NP-potpun, optimizacijska verzija u kojoj je potrebno pronaći minimalan broj lukova koji svojim uklanjanjem čine graf slobodnim od usmjerenih ciklusa je NP-teška (Lawler, 1964), čak i u jednostavnijoj verziji turnira (Kenyon-Mathieu and Schudy, 2007). Prema (Kann, 1992) problem pronalaska Minimalnog povratnog skupa lukova je APX-težak. Sada kada znamo kakav je problem koji želimo svesti na naš problem pogledajmo jeli naš problem zaista onakav kakav smo pretpostavili da bi mogao biti, odnosno dokažimo da je problem Određivanja redoslijeda razvoja podsustava IS-a NP-težak.

Teorem 3.1.1. *Problem Određivanja redoslijeda razvoja podsustava informacijskog sustava, u verziji odluke, je NP-potpun.*

Dokaz. Jasno je kako OSRPIS pripada klasi *NP*. Naime, OSRPIS ima oblik problema odluke. Također, ako za zadani primjerak OSRPIS-a netko pogodi ispravan linearni redoslijed čvorova, tada se u polinomnom vremenu mogu pronaći povratni lukovi, odrediti suma njihovih težina, te provjeriti je li ta suma zaista $\leq$ od zadanog K.

Odaberimo proizvoljni primjerak problema PSL. Neka se taj primjerak zove Primjerak 1, neka je on zadan grafom $G = (V, E)$ i brojem K. Konstruirajmo odgovarajući primjerak problema

OSRPIS, nazovimo ga Primjerak 2. Taj Primjerak 2 zadan je istim grafom $G = (V, E)$ i istim brojem K. Jedina mala razlika je to što su u Primjerku 2 svim lukovima pridružene jedinične težine.

Tvrdimo kako vrijedi: Primjerak 2 ima rješenje (odgovor na njegovo pitanje je "da") ako i samo ako Primjerak 1 ima rješenje (odgovor na njegovo pitanje je "da"). Sljede detalji dokaza ove tvrdnje.

- Pretpostavimo da odgovor na pitanje iz Primjerka 2 glasi "da". To znači da postoji linearni slijed vrhova iz G takav da je suma težina odgovarajućih povratnih lukova $\leq K$. Neka je S skup svih tih povratnih lukova. Tvrdimo da je S rješenje za Primjerak 1. Zaista, zbog jediničnih težina, očito je da $|S|$ mora biti $\leq K$. Također, ne može postojati usmjereni ciklus u G koji ne sadrži ni jedan luk iz S, jer bi promatranjem tog ciklusa našli još jedan povratni luk kojeg nema u S (što bi bila kontradikcija sa činjenicom da je S skup svih povratnih lukova).

- Pretpostavimo da odgovor na pitanje iz Primjerka 2 glasi "ne". Tvrdimo da odgovor na pitanje iz Primjerka 1 tada također mora biti "ne". Da bi to pokazali, pretpostavimo suprotno, da postoji skup lukova S, takav da je $|S| \leq K$ i takav da svaki ciklus iz G sadrži bar jedan luk iz S. Izbacimo lukove iz S iz G, tada preostali dio od G (nazovimo ga G') mora biti acikličan jer smo prekinuli sve cikluse. Pronađimo topološko sortirani slijed vrhova za G' (ona postoji jer je G' acikličan). Tvrdimo kako je dobiveni slijed vrhova rješenje za Primjerak 2. Zaista, povratni lukovi iz G ne postoje u G' pa je jasno da su svi oni uključeni u S, pa je zbroj njihovih jediničnih težina $\leq K$. Dakle našli smo rješenje za Primjerak 2, što je kontradikcija s pretpostavkom da odgovor na pitanje iz Primjerka 2 glasi "ne".

Jasno je da se sve opisane konstrukcije mogu izvesti u polinomnom vremenu. Time je dokaz završen. *Q.E.D.*

Korolar 3.1.1. *Iz Teorema 3.1.1 direktno proizlazi da je optimizacijska verzija problema OSRPIS NP-teška.*

Korolar 3.1.2. *Iz Teorema 3.1.1 direktno proizlazi, s obzirom da je prema (Kann, 1992) problem MPSL APX-težak, da je OSRPIS problem APX-težak.*

Sada kada smo odredili kojim klasama računalne teorije složenosti problem OSRPIS pripada možemo adekvatno izabrati način rješavanja problema imajući u vidu sljedeće.

1. Dosizanje optimalnog rješenja.
2. Efikasnost algoritma.

Razlozi odabira konkretnih metoda i algoritama za rješavanje problema OSRPIS detaljnije su opisani u narednim podpoglavljima.

3.2 Algoritam temeljen na metodi Grananja i ograničenja

Sada znamo da je problem OSRPIS NP-težak. Stoga ćemo iskoristiti metodu Grananja i ograničenja kako bismo razvili algoritam koji pronalazi minimalnu sumu

$$\sum_{\substack{(u,v)\in E \\ u>v}} T(u,v) \tag{3.1}$$

u linearnom slijedu vrhova v_i $(i = 1 \dots |V|)$. S obzirom da je opis metode Grananja i ograničenja dan kod pregleda literature krenuti ćemo direktno sa stvaranjem algoritma koji će riješiti problem OSRPIS. Ulaz u algoritam predstavlja graf G dan kroz popis lukova. Primjer popisa lukova dan je jednadžbom 3.2.

$$G = \left\{ \begin{array}{ll} \big[(1,3),2\big], & \big[(1,2),2\big], \\ \big[(2,3),8\big], & \big[(2,1),1\big], \\ \big[(3,1),1\big], & \big[(3,2),3\big], \\ \big[(3,4),2\big], & \big[(4,2),3\big], \\ \big[(4,5),1\big], & \big[(5,2),1\big], \end{array} \right\} \tag{3.2}$$

Izlaz iz algoritma je minimalna suma težina povratnih veza i linearan slijed čvorova koji daje tu sumu. S obzirom da razvijamo algoritam temeljen na metodi Grananja i ograničenja moramo moći stvarati stablo permutacija i s obzirom na trenutni minimum određene grane proširivati stablo sve dok ne dođemo do optimuma. U tu svrhu algoritam ćemo podijeliti na tri funkcije.

Algoritam 3.1 Pseudokod funkcije *zapocni_stablo*(...) : algoritma koji optimalno rješava problem OSRPIS.

Ulaz: Usmjereni ciklički graf $G = [(u,v), T(u,v)]$.
Izlaz: Minimalna suma težina povratnih veza i slijed vrhova v_i $(i = 1 \dots |V|)$.

1: Pronađi skup svih vrhova V
2: Pomoću metode Pohlepe pronađi slijed koji ima vrhove sa većim izlaznim stupnjem na početku, suma $\sum_{(u,v)\in E,\, u>v} T(u,v)$ tog slijeda predstavlja početni globalni optimum
3: Prolazeći kroz skup svih vrhova V započni stvaranje stabla i pozovi funkciju *zavrsi_stablo*(*korijen*, V, *početni_globalni_optimum*) :
4: Ukoliko je funkcija *zavrsi_stablo*(...) : vratila bolje rješenje spremi podatke i nastavi prolaziti kroz skup vrhova V sve dok se ne iscrpe svi vrhovi skupa
5: Ukoliko funkcija *zavrsi_stablo*(...) : nije vratila bolje rješenje nastavi prolaziti kroz skup vrhova V sve dok se ne iscrpe svi vrhovi skupa
6: Nakon što su iscrpljeni svi vrhovi skupa V vrati minimalnu sumu težina povratnih veza i slijed vrhova v_i $(i = 1 \dots |V|)$

Prva funkcija, *zapocni_stablo*(...) :, bit će zadužena za pronalazak početne maksimalne sume težina povratnih lukova, pronalazak svih čvorova iz ulaznog grafa radi kasnijeg

lakšeg stvaranja permutacija stabla, te za započinjanje podstabala i praćenje trenutnog optimuma. Maksimalnu početnu sumu težina povratnih lukova odnosno početni globalni optimum odrediti ćemo preko metode Pohlepe. Prvo ćemo poredati čvorove silazno prema broju izlaznih lukova a zatim ćemo za takav slijed izračunati sumu

$$\sum_{\substack{(u,v)\in E \\ u>v}} T(u,v).$$

Ova suma je blizu optimuma (pretpostavljamo) i predstavlja vrijednost koja ne narušava pronalazak optimuma s obzirom da je dobivena iz jednog realnog slijeda ulaznog grafa. Pomoću ove sume ćemo odrezati grane stabla koje nikako ne mogu biti optimalno rješenje.

Druga funkcija, *zavrsi_stablo*(...) :, bit će zadužena za stvaranje podstabala, odnosno podskupa permutacija $p \subset P$. Ova funkcija može prekinuti svoje izvršavanje jedino u dva slučaja.

Algoritam 3.2 Pseudokod funkcije *zavrsi_stablo*(...) : algoritma koji optimalno rješava problem OSRPIS.

Ulaz: Korijen stabla, popis vrhova grafa $G = [(u,v), T(u,v)]$, trenutni optimum.

Izlaz: Novi optimum $\sum_{(u,v)\in E,\, u>v} T(u,v)$ i slijed vrhova v_i $(i = 1 \ldots |V|)$.

1: Za čvor k odnosno granu koja potencijalno vodi do boljeg rješenja pronađi skup vrhova $v \subset V$
2: Za svaki čvor pronađenog podskupa v izračunaj sumu težina povratnih veza (svaki čvor sadrži ukupnu sumu do sebe uključujući i sebe)
3: Proširi čvor koji potencijalno vodi do boljeg rješenja
4: Ukoliko čvor nije moguće proširiti to znači da smo došli do novog optimuma te vračamo $\sum_{(u,v)\in E,\, u>v} T(u,v)$ i slijed vrhova v_i $(i = 1 \ldots |V|)$
5: Pronađi granu stabla sa najmanjom sumom $\sum_{(u,v)\in E,\, u>v} T(u,v)$ pri čemu se kod grana sa istom sumom odabire ona koja je na višoj razini
6: Ukoliko je $\sum_{(u,v)\in E,\, u>v} T_{ptnc}(u,v) \geq \sum_{(u,v)\in E,\, u>v} T_{trn}(u,v)$ prekini izvršavanje funkcije i vrati praznu listu
7: Vrati se na početak

1. Ukoliko moramo proširiti čvor koji je list s obzirom da to znači da smo naišli na novi optimum.

2. Ukoliko je suma težina povratnih veza potencijalno optimalne grane

$$\sum_{\substack{(u,v)\in E \\ u>v}} T_{ptnc}(u,v) \geq D_{opt}. \tag{3.3}$$

Što znači kako već imamo linearan slijed čvorova koji je jednako dobar ili bolji od

trenutno najboljeg u novom podstablu i daljnjim proširivanjem novog podstabla ne možemo poboljšati dosadašnji optimum D_{opt}.

Funkcija će uz kriterij najmanje sume težina povratnih veza za odabir potencijalno optimalne grane koristiti i kriterij razine čvora. Naime, ukoliko je potencijalno optimalna vrijednost pronađena na čvoru više razine ta grana će potencijalno brže dovesti do novog optimuma. Ovaj brži dolazak do novog optimuma ovisi o ulaznom grafu. Ukoliko se to ne dogodi ovaj kriterij neće dovesti do bespotrebnog proširivanja čvorova s obzirom da uniforman odabir potencijalne grane zahtijeva proširivanje potencijalno svih grana trenutno najmanje vrijednosti.

Algoritam 3.3 Pseudokod funkcije *tezina_veze*(...) : algoritma koji optimalno rješava problem OSRPIS.

Ulaz: Čvor k i njegovi preci.
Izlaz: $\sum_{1\leq u\leq(k-1)} T(k,u)$.
1: Vrati $\sum_{1\leq u\leq(k-1)} T(k,u)$

Treća funkcija, *tezina_veze*(...) :, bit će zadužena za računanje ukupne sume težina povratnih veza određenog čvora u grani koja se proširuje tim čvorom. S obzirom da će se za svaki čvor računati suma težina povratnih veza, za ukupnu sumu koja se računa u funkciji *zavrsi_stablo*(...) : dovoljno je izračunati sumu težina povratnih veza čvora koji se dodaje u podstablo i oca. Što znači da će svaki čvor određene grane sadržavati podatak o ukupnoj sumi težina povratnih veza do tog čvor u toj grani uključujući i taj čvor. Pseudokod ovih funkcija vidi se u algoritmima 3.1, 3.2, 3.3.

Implementacija algoritma 3.1: Implementacija algoritma, temeljenog na metodi Grananja i ograničenja, za optimalno rješavanje problema OSRPIS.

```
__author__ = 'Robert'

from treelib import Tree # izvor: https://github.com/caesar0301/pyTree, ucitano: 29.08.2014.

# ulazni graf
g = [[[1, 3], 2], [[1, 2], 2], [[2, 3], 8],
     [[2, 1], 1], [[3, 1], 1], [[3, 2], 3],
     [[3, 4], 2], [[4, 2], 3], [[4, 5], 1], [[5, 2], 1]]

# dovrsi stablo i pronadi optimum podstabla
def zavrsi_stablo(korijen, vrhovi_grafa,
        minimalna_suma_slijeda):

 podstablo = Tree()
 podstablo.create_node(str(korijen) + ";0", korijen) # stvaranje korijena podstabla
 instanca_rod_tren_cvora = podstablo.parent(korijen) # instanca cvora podstabla preko koje dolazimo do oca
 trenutni_cvor = podstablo.get_node(korijen) # trenutni potencijalno optimalni odabir
 id_cvora = len(vrhovi_grafa) + 1 # jedinstveni identifikator cvorova u podstablu

 popis_cvorova_perm = [] # popis svih predaka odredenog cvora podstabla
 cvor_prosiren = False # indikacija moze li se trenutni cvor dalje prosirivati

 while True:
     while True:
         if instanca_rod_tren_cvora is not None: # trazenje cvorova permutacije radi sirenja stabla
             popis_cvorova_perm.append(int
                 (instanca_rod_tren_cvora.tag
                 [:instanca_rod_tren_cvora.tag.rfind(';')]))
             instanca_rod_tren_cvora = podstablo.parent
                 (instanca_rod_tren_cvora.identifier)
         else:
             break

     # suma povratnih lukova do trenutnog cvora
     suma_trenutni_cvor = int(trenutni_cvor.tag[trenutni_cvor.tag.rfind(';') + 1:])

```

```
37      for novi_cvor in vrhovi_grafa:
38          # filtriranje cvorova permutacije
39          if novi_cvor not in popis_cvorova_perm \
40                  and novi_cvor != int(trenutni_cvor.tag
41                          [:trenutni_cvor.tag.rfind(';')]):
42
43              suma_novi_cvor = \
44                  tezina_veze(novi_cvor, int(trenutni_cvor.tag
45                  [:trenutni_cvor.tag.rfind(';')]),
46                  popis_cvorova_perm) + suma_trenutni_cvor
47              # dodavanje sinova na trenutni potencijalno optimalni odabir
48              id_cvora += 1
49              podstablo.create_node(str(novi_cvor) + ";"
50                 + str(suma_novi_cvor),
51                 id_cvora, parent=trenutni_cvor.identifier)
52              cvor_prosiren = True
53
54      if not cvor_prosiren: # pronaden novi optimum
55          popis_cvorova_perm.reverse()
56          popis_cvorova_perm.append(int(trenutni_cvor.tag
57                  [:trenutni_cvor.tag.rfind(';')])) # linearan slijed
58
59          suma = int(trenutni_cvor.tag
60                  [trenutni_cvor.tag.rfind(';') + 1:])
61
62          podstablo.show()
63          return suma, popis_cvorova_perm
64
65      listovi_stabla = podstablo.leaves() # dohvacanje instanci listova
           podstabla
66      najmanja_suma_list = listovi_stabla[0]
67
68      # pronalazak lista sa najmanjom sumom povratnih lukova
69      for list_ in listovi_stabla:
70          if int(list_.tag[list_.tag.rfind(';') + 1:]) \
71                  < int(najmanja_suma_list.tag[najmanja_suma_list.tag.rfind(
                       ';') + 1:]):
72
73              najmanja_suma_list = list_
74
75          elif int(list_.tag[list_.tag.rfind(';') + 1:]) \
76                  == int(najmanja_suma_list.tag[najmanja_suma_list.tag.rfind
                       (';') + 1:]):
77                  # u slucaju da je list sa jednakom vrijednosti povratnih
                       lukova na visoj razini podstabla
78                  if podstablo.level(list_.identifier) > podstablo.level(
                       najmanja_suma_list.identifier):
```

```
79
80                          najmanja_suma_list = list_
81          # ako je suma trenutnog slijeda veca od do sada optimalnog slijeda
               prekini daljnju potragu
82          if int(najmanja_suma_list.tag
83             [najmanja_suma_list.tag.rfind(';') + 1:]) >= minimalna_suma_slijeda
                  :
84
85              podstablo.show()
86              return []
87
88          # ponovno postavljanje privremenih varijabli
89          trenutni_cvor = najmanja_suma_list
90          instanca_rod_tren_cvora = podstablo.parent(trenutni_cvor.identifier)
91          popis_cvorova_perm = []
92          cvor_prosiren = False
93
94  def zapocni_stablo(graf): # zapocni sva stabla
95
96      suma_lukova = 0 # maksimalna globalna suma na pocetku i minimalna suma
           kasnije
97      popis_vrhova = [] # popis svih vrhova grafa
98      slij_vrhova_min_pov_sume_luk = [] # slijed cvorova sa minimalnom sumom
           povratnih lukova
99
100     for tezina_luka in graf:
101         # pronalazak vrhova grafa
102         if tezina_luka[0][0] not in popis_vrhova:
103             popis_vrhova.append(tezina_luka[0][0])
104
105         # u slucaju da cvor nema izlaznih lukova
106         if tezina_luka[0][1] not in popis_vrhova:
107             popis_vrhova.append(tezina_luka[0][1])
108
109     # popis svih vrhova sa izlaznim lukovima
110     popis_vrh_temp = []
111     # pronalazak broja izlaznih lukova za svaki vrh
112     for vrh_ in popis_vrhova:
113         suma_vrh = 0
114         for tezina_luka in graf:
115             if tezina_luka[0][0] == vrh_:
116                 suma_vrh += tezina_luka[1]
117
118         popis_vrh_temp.append([vrh_, suma_vrh])
119
120     # sortiranje prema broju izlaznih lukova
121     popis_vrh_temp = sorted(popis_vrh_temp,
```

```
122         key=lambda popis: popis[1])
123
124  brojac_vrh = 0
125  broj_vrhova = len(popis_vrh_temp)
126  popis_pov_luk = [] # popis cvorova i povratnih lukova
127
128  # racunanje povratnih veza za sortirani niz lukova
129  while brojac_vrh < broj_vrhova:
130      brojac_vrhovi = brojac_vrh + 1
131      suma_pov_luk = 0
132      while brojac_vrhovi < broj_vrhova:
133          for luk in graf:
134              if popis_vrh_temp[brojac_vrh][0] == luk[0][0] and
                    popis_vrh_temp[brojac_vrhovi][0] == luk[0][1]:
135                  suma_pov_luk += luk[1]
136                  break
137          brojac_vrhovi += 1
138
139      # pohranjivanje sume povratnih lukova za svaki cvor
140      popis_pov_luk.append([popis_vrh_temp[brojac_vrh][0], suma_pov_luk])
141      brojac_vrh += 1
142
143  for luk_i in popis_pov_luk: # sumiranje povratnih veza i postavljanje
        pocetnog optimuma
144      suma_lukova += luk_i[1]
145
146  # postavljanje sume lukova na jedan vise radi pronalazenja heuristicki
        otkrivenog optimuma
147  suma_lukova += 1
148
149  for vrh in popis_vrhova: # prolazak kroz podstabla
150      temp = zavrsi_stablo(vrh, popis_vrhova, suma_lukova)
151      if len(temp) == 2: # pronaden novi optimum
152          suma_lukova = temp[0]
153          slij_vrhova_min_pov_sume_luk = temp[1]
154  return "Suma->", suma_lukova, "Slijed->", slij_vrhova_min_pov_sume_luk #
        vracanje globalnog optimuma
155
156 def tezina_veze(u, v, preci_v):
157
158  suma_tezina = 0 # suma tezina izmedu cvora u i svih prijasnjih cvorova
159  popis_ocuvanog_slijeda = [v] # popis cvorova ocuvanog slijeda
160
161  popis_ocuvanog_slijeda.extend(preci_v)
162
163  for vrh_ in popis_ocuvanog_slijeda:
164      for luk in g:
```

```
165             # sumiranje povratnih lukova
166             if luk[0][0] == u and luk[0][1] == vrh_:
167                 suma_tezina += luk[1]
168                 break
169 
170     return suma_tezina
171 
172 # Ulaz: graf g{(u,v),T(u,v)}; Izlaz: minimalna suma povratnih veza i slijed
        vrhova koji daje tu sumu
173 print(zapocni_stablo(g))
```

Sada kada znamo na koji način algoritam radi i pronalazi optimalno rješenje problema OSRPIS, vraćajući kao izlaz minimalnu sumu težina povratnih veza i slijed vrhova v_i $(i = 1 \ldots |V|)$, pogledajmo kako bi izgledala implementacija algoritma. Algoritam je implementiran u programskom jeziku *Python v*3.4 i dan je implementacijom algoritma 3.1. Uz algoritam je dan i testni graf $G = [(u,v), T(u,v)]$. Nakon izvršavanja algoritma na zaslon su ispisana sva stabla, optimalno rješenje

$$\sum_{\substack{(u,v)\in E \\ u>v}} T(u,v)$$

i slijed vrhova v_i $(i = 1 \ldots |V|)$ koji daje taj optimum. Ukoliko ne želimo ispis podstabala potrebno je iz algoritma maknuti linije koda *podstablo.show*(). Izvršimo li implementaciju algoritma 3.1 sa danim probnim grafom, kao izlaz dobiti ćemo skup stabala na slikama 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8. Pri čemu je optimalno rješenje

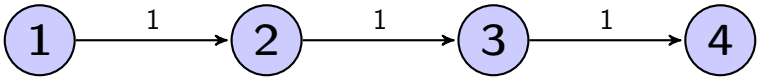
$$\sum_{\substack{(u,v)\in E \\ u>v}} T(u,v) = 7$$

za slijed vrhova

$$v_i \ (i = 1 \ldots |V|) = [1,4,5,2,3].$$

Pogledamo li izlazna podstabla vidimo kako nam je određivanje početnog globalnog optimuma preko metode Pohlepe pomoglo u rezanju određenog broja grana. Isto tako, u par slučajeva nam je uzimanje potencijalne grane sa više razine stabla omogućilo brži završetak proširivanja čvorova.

Pretpostavimo li kao ulaz u algoritam potpuni graf čije su težine lukova jednake, usprkos implementaciji dopustivih heuristika, složenost algoritma je $O(n!)$, pri čemu je n broj čvorova grafa.



Slika 3.9: Mogući ulaz u algoritam, temeljen na metodi Grananja i ograničenja, koji rješava problem OSRPIS.

S druge strane, neka kao ulaz u algoritam bude graf kao na slici 3.9. Složenost algoritma u tom slučaju je $\Omega\left(\frac{3(n^2-n)+2}{2}\right)$. Kako bismo procijenili rad algoritma u realnim uvjetima, na pretpostavljenom prosječnom grafu, i dobili optimalna rješenja za grafove koji će nam trebati kasnije kod testiranja efikasnijeg algoritma napraviti ćemo eksperiment.

Prilikom eksperimenta broj čvorova grafa određivati ćemo mi, na temelju tog broja

```
1;0
|-- 2;1
|    |-- 3;5
|    |    |-- 4;8
|    |    +-- 5;6
|    |         +-- 4;10
|    |-- 4;4
|    |    |-- 3;10
|    |    +-- 5;5
|    |         +-- 3;11
|    +-- 5;2
|         |-- 3;6
|         |    +-- 4;10
|         +-- 4;6
|              +-- 3;12
|-- 3;1
|    |-- 2;10
|    |-- 4;1
|    |    |-- 2;10
|    |    +-- 5;1
|    |         +-- 2;10
|    +-- 5;1
|         |-- 2;10
|         +-- 4;2
|              +-- 2;11
|-- 4;0
|    |-- 2;1
|    |    |-- 3;7
|    |    +-- 5;2
|    |         +-- 3;8
|    |-- 3;3
|    |    |-- 2;12
|    |    +-- 5;3
|    |         +-- 2;12
|    +-- 5;0
|         |-- 2;1
|         |    +-- 3;7
|         +-- 3;3
|              +-- 2;12
```

Slika 3.1: Izlazno stablo permutacija implementacije algoritma 3.1 sa zadanim grafom $G = [(u,v), T(u,v)]$ za vrh 1 (1/2).

```
+-- 5;0
    |-- 2;1
    |   |-- 3;5
    |   |   +-- 4;9
    |   +-- 4;5
    |       +-- 3;11
    |-- 3;1
    |   |-- 2;10
    |   +-- 4;2
    |       +-- 2;11
    +-- 4;1
        |-- 2;2
        |   +-- 3;8
        +-- 3;4
            +-- 2;13
```

Slika 3.2: Izlazno stablo permutacija implementacije algoritma 3.1 sa zadanim grafom $G = [(u,v), T(u,v)]$ za vrh 1 (2/2).

čvorova generirati ćemo lukove grafa $G = [(u,v), T(u,v)]$. Za svaki graf generirati ćemo

$$\frac{n(n-1)}{2} + 1 \tag{3.4}$$

lukova kako bismo bili sigurni da graf ima barem jedan ciklus pri čemu će distribucija za odabir luka biti uniformna a težine lukova biti će $T(E) \in [1, |V|]$ kako bismo osigurali adekvatnu raspršenost težina s obzirom da bi odstupanje od ovog skupa težina u bilo koju stranu gurnulo generirane grafove u ekstreme koji najvjerojatnije nebi predstavljali uobičajeni graf podsustava IS-a. Isto tako, kako ne bismo imali previše čvorova sa velikim izlaznim stupnjem, koji u realnim uvjetima predstavljaju određeno skladište podataka, odrediti ćemo da je vjerojatnost generiranja težine luka manja što je težina veća te ćemo stoga za generiranje težine luka koristiti eksponencijalnu distribuciju pri čemu je $lambda = 0.15$. Eksperimentiranjem na nizu slučajeva otkrili smo kako ova vrijednost *lambde* odgovara prethodno opisanoj raspršenosti težina lukova. Tijekom eksperimenta pratiti ćemo sljedeće.

1. Vrijeme izvršavanja.
2. Početni globalni optimum.
3. Broj prekidanja stvaranja stabla zbog pronalaska novog optimuma.
4. Broj prekidanja stvaranja stabla zbog boljeg starog optimuma.
5. Skup generiranih čvorova grafa.
6. Broj proširenja čvorova stabla.
7. Broj posječenih čvorova stabla.

```
2;0
|-- 1;2
|    |-- 3;6
|    |    |-- 4;9
|    |    +-- 5;7
|    |-- 4;5
|    |    |-- 3;11
|    |    +-- 5;6
|    |         +-- 3;12
|    +-- 5;3
|         |-- 3;7
|         +-- 4;7
|-- 3;3
|    |-- 1;7
|    |-- 4;6
|    |    |-- 1;10
|    |    +-- 5;7
|    +-- 5;4
|         |-- 1;8
|         +-- 4;8
|-- 4;3
|    |-- 1;5
|    |    |-- 3;11
|    |    +-- 5;6
|    |         +-- 3;12
|    |-- 3;8
|    +-- 5;4
|         |-- 1;6
|         |    +-- 3;12
|         +-- 3;9
+-- 5;1
     |-- 1;3
     |    |-- 3;7
     |    +-- 4;7
     |-- 3;4
     |    |-- 1;8
     |    +-- 4;8
     +-- 4;5
          |-- 1;7
          +-- 3;10
```

Slika 3.3: Izlazno stablo permutacija implementacije algoritma 3.1 sa zadanim grafom $G = [(u,v), T(u,v)]$ za vrh 2.

```
3;0
|-- 1;2
|    |-- 2;11
|    |-- 4;2
|    |    |-- 2;11
|    |    +-- 5;2
|    |         +-- 2;11
|    +-- 5;2
|         |-- 2;11
|         +-- 4;3
|              +-- 2;12
|-- 2;8
|-- 4;0
|    |-- 1;2
|    |    |-- 2;11
|    |    +-- 5;2
|    |         +-- 2;11
|    |-- 2;8
|    +-- 5;0
|         |-- 1;2
|         |    +-- 2;11
|         +-- 2;8
+-- 5;0
     |-- 1;2
     |    |-- 2;11
     |    +-- 4;3
     |         +-- 2;12
     |-- 2;8
     +-- 4;1
          |-- 1;3
          |    +-- 2;12
          +-- 2;9
```

Slika 3.4: Izlazno stablo permutacija implementacije algoritma 3.1 sa zadanim grafom $G = [(u,v), T(u,v)]$ za vrh 3.

```
4;0
|-- 1;0
|    |-- 2;1
|    |    |-- 3;7
|    |    +-- 5;2
|    |         +-- 3;8
|    |-- 3;3
|    |    |-- 2;12
|    |    +-- 5;3
|    |         +-- 2;12
|    +-- 5;0
|         |-- 2;1
|         |    +-- 3;7
|         +-- 3;3
|              +-- 2;12
|-- 2;0
|    |-- 1;2
|    |    |-- 3;8
|    |    +-- 5;3
|    |         +-- 3;9
|    |-- 3;5
|    |    |-- 1;9
|    |    +-- 5;6
|    |         +-- 1;10
|    +-- 5;1
|         |-- 1;3
|         |    +-- 3;9
|         +-- 3;6
|              +-- 1;10
|-- 3;2
|    |-- 1;4
|    |    |-- 2;13
|    |    +-- 5;4
|    |         +-- 2;13
|    |-- 2;10
|    +-- 5;2
|         |-- 1;4
|         |    +-- 2;13
|         +-- 2;10
```

Slika 3.5: Izlazno stablo permutacija implementacije algoritma 3.1 sa zadanim grafom $G = [(u,v), T(u,v)]$ za vrh 4 (1/2).

```
+-- 5;0
    |-- 1;0
    |   |-- 2;1
    |   |   +-- 3;7
    |   +-- 3;3
    |       +-- 2;12
    |-- 2;0
    |   |-- 1;2
    |   |   +-- 3;8
    |   +-- 3;5
    |       +-- 1;9
    +-- 3;2
        |-- 1;4
        |   +-- 2;13
        +-- 2;10
```

Slika 3.6: Izlazno stablo permutacija implementacije algoritma 3.1 sa zadanim grafom $G = [(u,v), T(u,v)]$ za vrh 4 (2/2).

8. Broj izbora grane sa minimum na višoj razini stabla.

9. Minimalnu sumu težina povratnih veza.

10. Linearan slijed vrhova koji daje minimalnu sumu.

Algoritam nad kojim je rađen eksperiment, uključujući i funkciju koja stvara grafove za testiranje, može se vidjeti u poglavlju Prilozi pod implementacijom algoritma 5.1.

Počevši od broja čvorova 3 pa sve do broja čvorova 9 eksperiment smo ponavljali deset puta za svaki broja čvorova kako bismo dobili realniju sliku efikasnosti algoritma nad generiranim grafovima. Nakon broja čvorova 9 kombinatorna ekspanzija postala je prevelika a računanje sume težina povratnih lukova počelo je značajno usporavati rad algoritma. Eksperiment je rađen na procesoru radnog takta od 2.4 GHz i 8 GB radne memorije. Obrađeni rezultati eksperimenta vide se u tablici 3.1. S obzirom da se radi o nizu eksperimenata, ako nije drugačije naznačeno, svaka vrijednost u stupcu tablice 3.1 predstavlja očekivanu vrijednost, osim broja čvorova grafa i ukupnog broja svih čvorova svih permutacija. Vremena izvršavanja unutar jedne sekunde tretirana su kao nula.

Pogledamo li rezultate eksperimenta u tablici 3.1 vidimo sljedeće. Vremena izvršavanja za prva četiri eksperimenta koji se kreću od 3 do 6 čvorova su sva unutar jedne sekunde bez obzira na generirani graf. Nakon toga krećemo sa eksperimentom nad 7 čvorova gdje zamjećujemo prvo veće vrijeme izvršavanja. Ista situacija ponovila se je i za sljedeća dva eksperimenta ali sa još većim vremenima izvršavanja. Pogledamo li standardnu devijaciju vidimo kako je raspon normalnih vrijednosti s obzirom na mjere centraliteta neuobičajeno velik. Pregledom generiranih grafova i vremena izvršavanja uvidjeli smo kako vremena izvršavanja ne predstavljaju homogeni skup. Posebno velika vremena izvršavanja primjećuju se za grafove koji imaju linearnu raspodjelu težina lukova. Zbog ove raspodjele algoritmu je teško u ranoj fazi stvaranja stabla prekinuti proširivanje čvorova i krenuti na

```
5;0
|-- 1;0
|    |-- 2;1
|    |    |-- 3;5
|    |    |    +-- 4;9
|    |    +-- 4;5
|    |         +-- 3;11
|    |-- 3;1
|    |    |-- 2;10
|    |    +-- 4;2
|    |         +-- 2;11
|    +-- 4;1
|         |-- 2;2
|         |    +-- 3;8
|         +-- 3;4
|              +-- 2;13
|-- 2;0
|    |-- 1;2
|    |    |-- 3;6
|    |    |    +-- 4;10
|    |    +-- 4;6
|    |         +-- 3;12
|    |-- 3;3
|    |    |-- 1;7
|    |    +-- 4;7
|    +-- 4;4
|         |-- 1;6
|         |    +-- 3;12
|         +-- 3;9
|-- 3;0
|    |-- 1;2
|    |    |-- 2;11
|    |    +-- 4;3
|    |         +-- 2;12
|    |-- 2;8
|    +-- 4;1
|         |-- 1;3
|         |    +-- 2;12
|         +-- 2;9
```

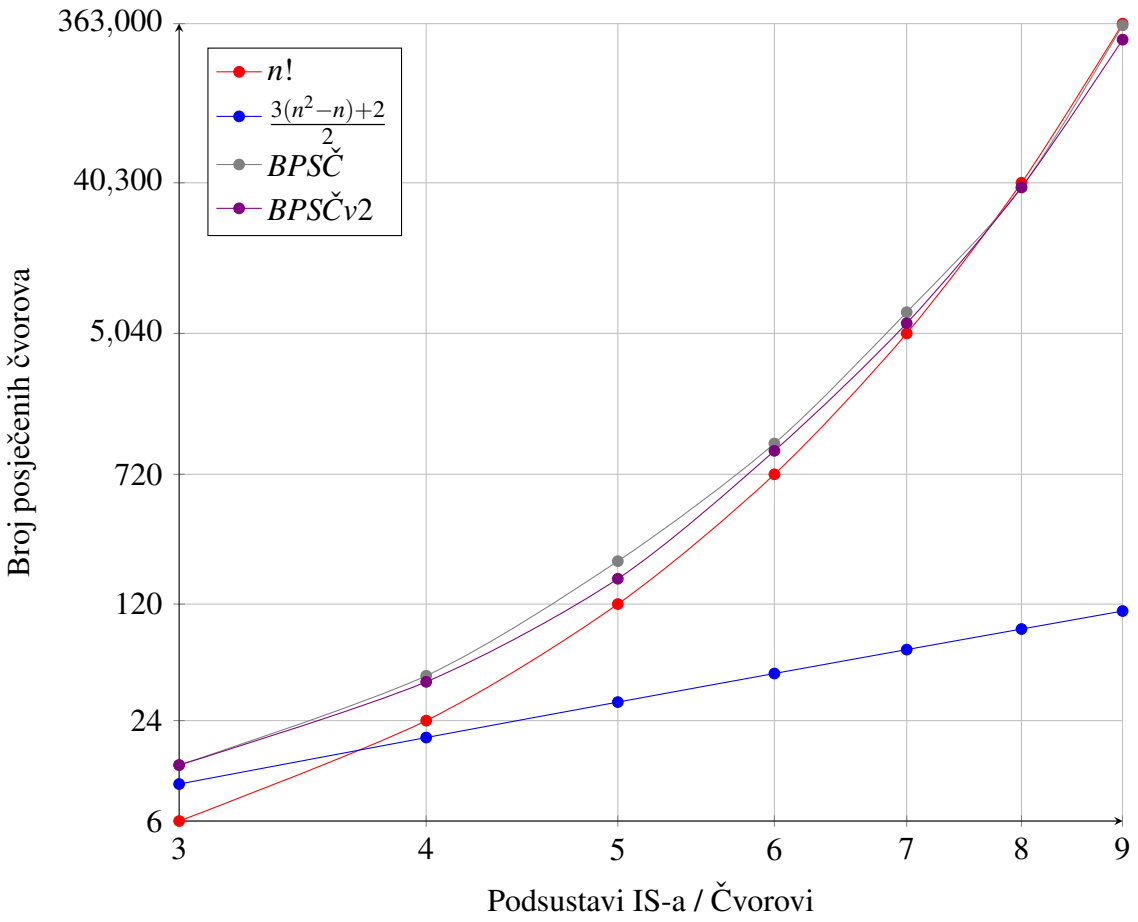
Slika 3.7: Izlazno stablo permutacija implementacije algoritma 3.1 sa zadanim grafom $G = [(u,v), T(u,v)]$ za vrh 5 (1/2).

```
+-- 4;1
    |-- 1;1
    |   |-- 2;2
    |   |   +-- 3;8
    |   +-- 3;4
    |       +-- 2;13
    |-- 2;1
    |   |-- 1;3
    |   |   +-- 3;9
    |   +-- 3;6
    |       +-- 1;10
    +-- 3;3
        |-- 1;5
        |   +-- 2;14
        +-- 2;11
```

Slika 3.8: Izlazno stablo permutacija implementacije algoritma 3.1 sa zadanim grafom $G = [(u,v), T(u,v)]$ za vrh 5 (2/2).

sljedeće stablo s obzirom da postoji puno lukova sa istim ili bliskim težinama lukova. Ova raspodjela očito ne predstavlja naš pretpostavljeni prosječni graf. Druga stvar koja utječe na vrijeme izvršavanja je broj generiranih ciklusa u grafu pri čemu je očito teže riješiti graf sa većim brojem ciklusa. Početni optimum koji se dobiva pomoću metode Pohlepe značajno je manji od ukupne sume težina lukova. Vrlo je blizu ili u najgorem slučaju dva puta veći od optimuma. S obzirom na niske vrijednosti NO i visoke vrijednosti SO vidimo da ova dopustiva heuristika radi dobro. Usporedimo li broj proširenja čvorova sa brojem izabranih grana sa minimumom na višoj razini (što u našem slučaju predstavlja broj čvorova s obzirom da svaki čvor ima sumu težina povratnih lukova u linearnom slijedu do sebe uključujući i sebe) vidimo kako su njihove vrijednosti vrlo blizu što bi dalo naslutiti da ovaj mehanizam guranja grana stabla u dubinu radi dobro. Međutim, ovaj mehanizam uvelike ovisi o generiranom grafu i bliske vrijednosti *BPČ* i *BVR* ne moraju značiti da smo uspjeli sa našom nakanom odbacivanja određenog broja proširenja radi guranja stabla u dubinu i ranijeg pronalaska optimuma. Što su težine lukova manje raspršene to nam ovaj mehanizam manje koristi ali bitno je napomenuti da nam u najgorem slučaju neće odmoći. Na kraju, usporedimo li broj posječenih čvorova grafa i ukupan broj čvorova svih permutacija vidimo kako algoritam značajno smanjuje područje pretrage što je područje pretrage veće i na slici 3.10 vidimo da krivulja posjeta čvorova za naš algoritam predstavlja rast značajno manji od $n!$.

Izuzevši iz skupa generiranih grafova nad kojima je algoritma testiran samo podskup koji se sastoji od grafova koji su identični ili vrlo blizu pretpostavljenog prosječnog grafa dobili smo podatke čija se obrada vidi u tablici 3.2. Kao i u prvom eksperimentu za broj čvorova od 3 do 6 vrijeme izvršavanja je unutar jedne sekunde i prikazano je kao nula. Sa brojem čvorova grafa 7 nadalje algoritam počinje sa vremenima izvršavanja koja su blizu ili ispod vremena koja su dobivena eksperimentem čiji rezultati se vide u



Slika 3.10: Prikaz vremena izvršavanja algoritma, temeljenog na metodi Grananja i ograničenja koji rješava problem OSRPIS, i krivulja složenosti na logaritamskoj skali.

tablici 3.1. S obzirom da je ovaj skup grafova homogeniji od prijašnjeg vidimo da je standardna devijacija u ovom eksperimentu manja nego u prijašnjem eksperimentu te prikazuje realniju procjenu intervala vremena izvršavanja algoritma nad pretpostavljenim prosječnim grafom. Pogledamo li broj posječenih čvorova vidimo da je on u pravilu manji ili u nekim slučajevima dosta manji od broja posječenih čvorova prijašnjeg eksperimenta što znači da je pretpostavljeni prosječni graf ipak nešto jednostavniji za riješiti, što je bilo za očekivati. S obzirom na velike raspone broja posječenih čvorova standardna devijacija kreće se u razumnim okvirima i trebala bi predstavljati realniju procjenu. Na kraju, usporedimo li odnos broja posječenih čvorova i ukupnog broj čvorova ovog eksperimenta i prijašnjeg eksperimenta vidimo kako algoritam na pretpostavljenom prosječnom grafu dodatno smanjuje područje pretrage. Na slici 3.10 krivulja ovog eksperimenta označena je sa *BPSČv2*.

Postoji još jedan način sa kojim možemo poboljšati vrijeme izvršavanja našeg algoritma, pri čemu će broj posječenih čvorova ostati isti, a to je vremensko prostorni kompromis. Kako bismo trenutni algoritam temeljen na metodi Grananja i ograničenja modificirali da koristi vremensko prostorni kompromis moramo napraviti sljedeće. Potrebno

je dodati polje u koje ćemo spremati sve već izračunate sume težina povratnih lukova za parove čvorova. To polje će nam omogućiti čitanje i zapisivanje vrijednosti u $O(1)$.

```
pamtim = [] # polje prethodno izračunatih povratnih lukova
```

Nakon toga je potrebno modificirati metodu $zapocni\_stablo(graf)$: kao bismo za svaki graf instancirali prethodno polje na pravu veličinu sa početnim vrijednostima koje će nam omogućavati provjeru izračuna.

```
def zapocni_stablo(graf):  # zapocni sva stabla
...
global pamtim
...
# stvaranje polja za pamčenje izračunatih povratnih lukova
pamtim = [[-1 for u in range(len(popis_vrhova))]
              for v in range(len(popis_vrhova))]
...
```

I na kraju je potrebno modificirati metodu $tezina\_veze(u, v, preci\_v, grf)$: na sljedeći način.

```
def tezina_veze(u, v, preci_v, grf):

    global pamtim   # polje prethodno izračunatih povratnih lukova

    # suma težina između čvora u i svih prijašnjih čvorova
    suma_tezina = 0
    popis_ocuvanog_slijeda = [v]   # popis čvorova očuvanog slijeda

    popis_ocuvanog_slijeda.extend(preci_v)

    for vrh_ in popis_ocuvanog_slijeda:
        # ako imamo već izračunati skup povratnih lukova za par u,v
        if pamtim[u - 1][vrh_ - 1] >= 0:
            suma_tezina += pamtim[u - 1][vrh_ - 1]
            continue

        # u slučaju da ne postoji luk između ova dva čvora
        pamtim[u - 1][vrh_ - 1] = 0
        for luk in grf:
            # sumiranje povratnih lukova
```

```
            if luk[0][0] == u and luk[0][1] == vrh_:
                suma_tezina += luk[1]

                # pamčenje povratnih lukova
                pamtim[u - 1][vrh_ - 1] = luk[1]
                break

    return suma_tezina
```

Ako metodu *eksperiment*(*br_cvorova*,*br_pon_eks*) : modificiramo na sljedeći način moguće je nad modificiranim algoritmom izvršavati eksperimente.

```
...
global pamtim
...
pamtim = []
...
```

Nakon što smo ovaj algoritam opet izvršili nad grafovima za devet čvorova uvidjeli smo kako algoritam radi brže. Na primjer, eksperiment 1 je prije trajao 53 minute dok je sada na novom algoritmu sa vremensko prostornim kompromisom trajao 25 minuta. Isto tako, eksperiment 8 je prije trajao, na vrlo "teškom" grafu, 10 sati dok je sada na novom algoritmu trajao 8 sati. Što znači kako je vremensko prostorni kompromis značajno ubrzao rad algoritma s obzirom da je sada vrijeme izvođenja eksperimenta 1 za 53% brže a vrijeme izvođenja eksperimenta 8 za 20% brže. Slična ubrzanja primjećuju se i na ostalim grafovima što znači da bismo ovaj algoritam eventualno mogli primijeniti i za 10 ili možda i 11 čvorova na pretpostavljenom prosječnom grafu sa vremenima izvršavanja unutar nekoliko dana do nekoliko tjedana.

Iz razloga osmišljavanja algoritma koji će biti efikasan i moći raditi na većim grafovima i potencijalno imati vrijeme odziva vrlo blizu realnom vremenu u sljedećem poglavlju baviti ćemo se razvojem Monte Carlo randomiziranog algoritma s proizvoljnom vjerojatnošću dosizanja optimuma.

Tablica 3.1: Rezultati eksperimenta nad algoritmom koji optimalno rješava problem OSRPIS i temeljen je na metodi Grananja i ograničenja.

	T													
Eksperiment	Med	$\bar{x}$	σ	PO	$\sum E$	NO	SO	BČ	BPČ	BPSČ	UBČ	%	BVR	Optimum
1	0,00	0,00	0,00	1,6	6,9	1,1	1,9	3	7,0	13,0	18	72,22	0,3	1,4
2	0,00	0,00	0,00	4,2	15,4	1,1	2,9	4	22,4	44,6	96	46,45	6,4	2,9
3	0,00	0,00	0,00	8,0	28,0	1,6	3,4	5	113,5	217,0	600	36,16	53,2	6,1
4	0,00	0,00	0,00	14,6	45,9	1,9	4,1	6	516,1	1101,8	4320	25,50	368,8	9,8
5	6,95	7,44	5,04	30,1	77,8	1,5	5,5	7	3094,7	6752,8	35280	19,14	2554,8	19,1
6	249,74	292,36	194,32	37,8	112,1	1,7	6,3	8	15001,4	37748,7	322560	11,70	14189,1	24,5
7	12344,46	15602,84	9758,31	59,4	147,0	2,9	6,1	9	145212,9	354840,4	3265920	10,86	141241,3	36,4

T procesorsko vrijeme izvršavanja u sekundama ne uključujući vrijeme prekida

PO početni optimum u broju povratnih lukova

NO broj prekidanja stvaranja stabla radi pronalaska novog optimuma

SO broj prekidanja stvaranja stabla radi boljeg starog optimuma

BČ broj čvorova grafa

BPČ broj proširenja čvorova stabla

BPSČ broj posječenih čvorova stabla

UBČ ukupan broj svih čvorova svih permutacija

% $\frac{BPSČ}{UBČ}$

BVR broj izabranih grana sa minimum na višoj razini stabla

Optimum minimalna suma težina povratnih veza

Tablica 3.2: Rezultati eksperimenta (bez ekstrema koji ne predstavljaju pretpostavljeni prosječni graf) nad algoritmom koji optimalno rješava problem OSRPIS i temeljen je na metodi Grananja i ograničenja.

	T			BPSČ					
Eksperiment	Med	$\bar{x}$	σ	Med	$\bar{x}$	σ	BČ	UBČ	%
1 (9)	0,00	0,00	0,00	13,00	13,00	1,11	3	18	72,22
2 (5)	0,00	0,00	0,00	40,00	41,00	4,00	4	96	42,70
3 (5)	0,00	0,00	0,00	159,00	170,20	30,62	5	600	28,36
4 (5)	0,00	0,00	0,00	1011,00	995,80	182,02	6	4320	23,05
5 (4)	5,42	5,08	2,25	6106,50	5799,00	1365,35	7	35280	16,43
6 (5)	253,06	290,44	161,49	36016,00	37865,40	11338,71	8	322560	11,73
7 (6)	10846,91	10411,71	5644,77	317809,00	290968,00	93571,42	9	3265920	8,90

Eksperiment broj eksperimenta zajedno sa brojem mjerenja u zagradi

T procesorsko vrijeme izvršavanja u sekundama ne uključujući vrijeme prekida

BPSČ broj posječenih čvorova stabla

BČ broj čvorova grafa

UBČ ukupan broj svih čvorova svih permutacija

% $\frac{BPSČ(\bar{x})}{UBČ}$

3.3 Monte Carlo randomizirani algoritam

Prethodno smo dokazali kako je problem OSRPIS APX-težak. Stoga smo za efikasno rješavanje problema odabrali Monte Carlo randomizaciju koja bi nam trebala omogućiti rješavanje problema OSRPIS sa proizvoljnom vjerojatnošću dosizanja optimuma. Tako ćemo u ovom poglavlje biti posvečeni osmišljavanju, razvoju i testiranju Monte Carlo randomiziranog algoritma.

U knjizi (Motwani and Raghavan, 1995) pronalazimo poznati Monte Carlo Min-Cut randomizirani algoritam koji bi se uz određene modifikacije mogao prilagoditi tako da rješava problem OSRPIS. Prije nego krenemo na osmišljavanje Monte Carlo randomiziranog algoritma definirajmo što je to multigraf i na koji način se radi pretvorba u multigraf. S obzirom da u literaturi termin multigraf nema jedinstveno značenje definicija u nastavku predstavlja naše poimanje multigrafa. Definicija koje najbliže odgovara našoj definiciji može se pronaći u (Zwillinger Daniel et al., 2003).

Definicija 3.3.1. *Graf koji: dozvoljava mnoštvo lukova između parova čvorova, nema petlje, i nema težine na lukovima; predstavlja multigraf.*

Stoga, imajući u vidu definiciju 3.3.1, pretvoriti graf u multigraf znači: maknuti petlje iz grafa, maknuti postojeće lukove a težine na lukovima dodati u graf kao lukove (vrijednost težine označava broj lukova koje je potrebno dodati između para čvorova), i maknuti težine sa lukova. Vjerojatnost izračunata u lemi 3.3.1 vrijedi za općeniti slučaj; nije izračunata uz određena ograničenja.

Lema 3.3.1. *Pretvorimo li graf problema OSRPIS u multigraf, uzastopnim prekidanjem lukova po uniformnoj distribuciji vjerojatnost da se prekinu svi ciklusi u grafu, između parova čvorova sa najmanjim brojem međusobnih lukova, obrnuto je proporcionalna vjerojatnosti odabira lukova između tih istih parova čvorova.*

Dokaz. Minimalan broj lukova grafa, na grafu koji ima barem jedan povratni luk, je kn pri čemu je k broj lukova između para čvorova sa najmanjim brojem međusobnih lukova a n broj čvorova. Prema ideji za maksimalni aciklički podgraf u (Skiena, 2008) možemo generirati nasumičan slijed vrhova. Očito je kako je podgraf sa lukovima koji idu od desna na lijevo acikličan, kao i onaj sa lukovima koji idu sa lijeva na desno. (Skiena, 2008) Što znači kako manji skup lukova od ova dva skupa predstavlja gornju granicu broja lukova koje treba prekinuti da bi graf postao acikličan. Stoga je broj lukova koji treba prekinuti da bi graf postao acikličan

$$Br.\,luk. \leq \frac{|E|}{2}. \tag{3.5}$$

Uvrstimo li sada u ovu formulu prethodni broj lukova dobijemo

$$Br.\,luk. \leq \frac{kn}{2}. \tag{3.6}$$

Što znači kako je broj skupova lukova koje je potrebno prekinuti

$$Br.\ skup.\ luk. \leq \frac{\frac{kn}{2}}{k} \leq \frac{n}{2}. \tag{3.7}$$

Dakle, nakon $\frac{n}{2}$ skupova prekinutih lukova vjerojatnost da smo prekinuli minimalan broj lukova koji graf čini acikličnim, pri čemu suma težina tih lukova u topološki sortiranom slijedu predstavlja minimalnu sumu težina povratnih lukova, jednaka je nuli. Ovo je gornja granica do koje algoritam treba prekidati skupove lukova na grafu. Vjerojatnost da prvi put prekinemo optimalan skup lukova (najmanji) je sljedeća.

$$Vj(prvi\ skup\ luk.) \geq 1 - \frac{k}{|E|} \geq 1 - \frac{k}{kn} \geq 1 - \frac{1}{n} \geq \frac{n-1}{n} \tag{3.8}$$

Iz čega proizlazi kako je vjerojatnost optimalnog prekidanja ostalih cikličnih skupova lukova

$$\begin{aligned} Vj(ost.\ skup.\ luk.) &\geq 1 - \frac{k}{preostali\ lukovi} \geq 1 - \frac{k}{k(n-i)} \\ &\geq 1 - \frac{1}{n-i} \geq \frac{n-i-1}{n-i} \end{aligned} \tag{3.9}$$

pri čemu je *i* broj prekinutih skupova lukova do sada. Što znači kako je vjerojatnost uspješnog prekidanja optimalnog broja skupova lukova jednaka kako sljedi.

$$\begin{aligned} Vj(opt) &\geq \left(\frac{n-1}{n}\right)\left(\frac{n-2}{n-1}\right)\left(\frac{n-3}{n-2}\right)\cdots\left(\frac{n-\frac{n}{2}-1}{n-\frac{n}{2}}\right) \\ &\geq \left(\frac{2n-n-2}{2n-n}\right)^{\frac{n}{2}} \geq \left(\frac{n-2}{n}\right)^{\frac{n}{2}} \end{aligned} \tag{3.10}$$

Uzastopnim izvršavanjem algoritma ova vjerojatnost može se povećati. Ponovimo li ovaj algoritam za neki broj ponavljanja *pi* vjerojatnost da smo svaki put dobili rješenje koje nije optimalno je

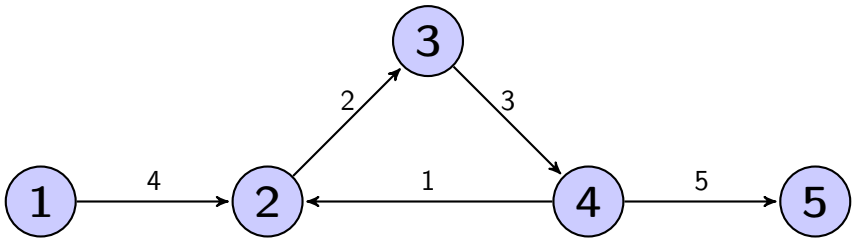
$$Vj(ne\ opt.\ skup\ luk.) = \left(1 - \left(\frac{n-2}{n}\right)^{\frac{n}{2}}\right)^{pi}. \tag{3.11}$$

Iz čega proizlazi kako je vjerojatnost da smo nakon *pi* izvršavanja algoritma dosegli optimum jednaka kako sljedi.

$$Vj(opt.\ skup\ luk.\ pi) = 1 - \left(1 - \left(\frac{n-2}{n}\right)^{\frac{n}{2}}\right)^{pi} \tag{3.12}$$

Q.E.D.

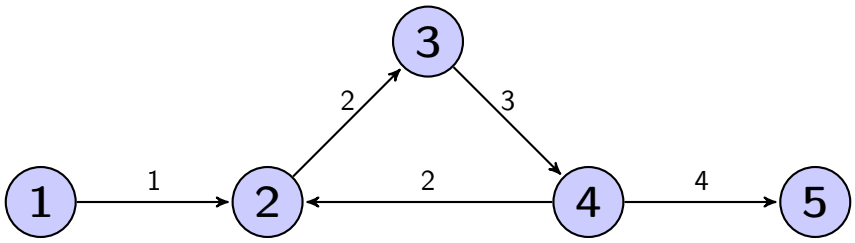
Dakle, vjerojatnost da na multigrafu uniformno prekinemo minimalan broj lukova koji taj graf čini acikličnim pri čemu taj broj lukova predstavlja minimalnu sumu težina povratnih



Slika 3.11: Multigraf zamišljene instance problema OSRPIS (težina luka predstavlja broj lukova).

lukova topološki sortiranog slijeda čvorova grafa jednaka je rezultatu izračuna leme 3.3.1.

Prije nego krenemo na implementaciju algoritma za koji vrijedi izračunata vjerojatnost prvo ćemo implementirati algoritam za koji ne vrijedi izračunata vjerojatnost, osim ako je algoritmu za ulaz dan graf nad kojim algoritam prekida samo one lukove koji su dio nekog ciklusa, ali je efikasniji. Moguće je kao ulaz u algoritam dobiti graf kao na slici 3.12. Naime, osim što u ciklusu $(2,3,4,2)$ imamo tri skupa lukova od kojih su dva identična po



Slika 3.12: Multigraf težeg slučaja zamišljene instance problema OSRPIS (težina luka predstavlja broj lukova).

broju lukova imamo i skup lukova između čvorova 1 i 2 koji je najmanji u cijelom grafu i još k tome nije ni u jednom ciklusu. Efikasniji algoritam na ovakvim grafovima neće biti u stanju postići optimalno rješenje ali će zato biti u stanju doći do optimalnog rješenja na grafovima tipa slika 3.11. Koliko dobro radi efikasniji algoritam na grafovima koji su teži za riješiti morati ćemo detaljnije empirijski testirati.

Ulaz u algoritam je multigraf reprezentacija interakcija podsustava IS-a. Nakon što dobije ulaz algoritam treba određeni broj puta po uniformnoj distribuciji prekinuti skup lukova. Nakon čega je potrebno na grafu na kojem smo prekinuli skupove lukova izvršiti algoritam Topološkog sortiranja kako bismo dobili slijed čvorova. Na kraju, algoritam kao izlaz treba dati slijed čvorova, sumu težina povratnih lukova i izračun vjerojatnosti (za efikasniji algoritam vjerojatnost ne vrijedi, osim ako je algoritmu za ulaz dan graf nad kojim algoritam prekida samo one lukove koji su dio nekog ciklusa; izračun vjerojatnosti u ovom algoritmu služi samo kao priprema za algoritam koji radi općenito). Kod izračuna vjerojatnosti smo imali situaciju $\frac{n}{2}$ za broj skupova lukova koje treba prekinuti. Prolazeći kroz popis susjedstva možemo saznati realan broj skupova lukova ($TBSL$) ulaznog grafa. Te je prema $\frac{n}{2}$ broj skupova lukova u realnom slučaju $\frac{TBSL}{2}$.

Prethodno opisani Monte Carlo randomizirani algoritam implementirati ćemo u programskom jeziku *Python v*3.4. Algoritam je podjeljen na dvije funkcije. Jedna funkcija

Algoritam 3.4 Pseudokod Monte Carlo randomiziranog algoritma za problem OSRPIS (efikasnija verzija).

Ulaz: Graf opisan kroz popis susjedstva $(V_{\overline{n}}, V_k), V_k \in V; V_k \neq V_{\overline{n}}$, broj ponavljanja i pomak. Moraju biti dani svi čvorovi bez obzira na ulazne lukove a ulazni lukovi moraju biti grupirani.

Izlaz: $Vj(opt), \sum_{(u,v)\in E,\, u>v} T(u,v)$ i v_i $(i = 1 \ldots |V|)$.

1: **funkcija** $ponavljanje(graf, ponovi, pomak)$:
2: postavi sjeme generatora pseudoslučajnih brojeva
3: **while** *ponovi* **do**
4: $temp = montecarlo(graf, pomak)$
5: **if** pronađeno bolje rješenje **then**
6: $najbolje\_rjesenje = temp$
7: **end if**
8: smanji *ponovi* za 1
9: **end while**
10: izračunaj $1 - \left(1 - \left(\frac{n-2}{n}\right)^{TBSL}\right)^{pi}$
11: **return** $najbolje\_rjesenje$
12:
13: **funkcija** $montecarlo(graf, pomak)$:
14: **while** postoje neposječeni lukovi grafa $(V_{\overline{n}}, V_k), V_k \in V; V_k \neq V_{\overline{n}}$ **do**
15: detekcijom grupa lukova odredi ukupan skup lukova $TBSL$
16: **end while**
17: broj prekidanja skupova lukova je $br\_prek = \frac{TBSL}{2} - pomak$
18: **while** $br\_prek > 0$ **do**
19: generiraj dva čvora po uniformnoj distribuciji
20: prekini luk
21: **if** prekinut zadnji luk skupa **then**
22: umanji br_prek za 1
23: spremi prekinuti skup lukova
24: **end if**
25: **end while**
26: izračunaj sumu težina povratnih lukova
27: topološki sortiraj dobiveni graf
28: **return** $\frac{n-2}{n}^{TBSL}, \sum_{(u,v)\in E,\, u>v} T(u,v)$ i v_i $(i = 1 \ldots |V|)$

zadužena je za pronalazak rješenja a druga za uzastopno ponavljanje algoritma. Pseudokod se može vidjeti pod algoritmom 3.4. Implementacija algoritma može se vidjeti pod implementacijom algoritma 3.2. Izvršimo li algoritam pod implementacijom algoritma 3.2 za dani graf, 100 puta, sa pomakom nula, kao izlaz dobiti ćemo

$$\sum_{\substack{(u,v)\in E \\ u>v}} T(u,v) = 8$$

nad slijedom

$$v_i\ (i = 1 \ldots |V|) = [5,4,1,2,3].$$

S obzirom da se radi o grafu, kao na slici 1.1, koji je problematičan, kako je opisano ranije u ovom poglavlju, možemo iskoristiti varijablu *pomak* kako bismo smanjili broj skupova lukova koje algoritam treba prekinuti. Postavimo li ovu varijablu na 1 dobivamo sljedeći rezultat.

$$\sum_{\substack{(u,v)\in E \\ u>v}} T(u,v) = 7$$

nad slijedom

$$v_i\ (i = 1 \ldots |V|) = [4,5,1,2,3].$$

Ovaj rezultat (suma težina povratnih lukova) predstavlja optimalno rješenje danog grafa u vidu našeg kriterija optimuma. Što se tiče same implementacije napomenimo, kako ne bi bilo zabune, da dio koda

```
if vj_opt == 1:
    najbolji_rez[0] = 0.9999999999999999
else:
    najbolji_rez[0] = vj_opt
```

koristimo kako bismo spriječili pogrešku izračuna vjerojatnosti od 1 (ne koristi se kod efikasnijeg algoritma i priprema je za algoritam koji će raditi na svim grafovima). Sve ostalo trebalo bi biti jasno iz komentara u implementaciji te iz materije poglavlja.

Implementacija algoritma 3.2: Implementacija Monte Carlo randomiziranog algoritma za problem OSRPIS (efikasnija verzija).

```
__author__ = 'Robert'

from math import floor
import random
import copy

# ulazni graf
g = {
    1: [2, 3],
    2: [1, 1, 3, 3, 3, 4, 4, 4, 5],
    3: [1, 1, 2, 2, 2, 2, 2, 2, 2, 2],
    4: [3, 3],
    5: [4],
}


def osrpis_mc_rand(ul_graf, pomak):     # (graf, mijenjanje broja prekidanja skupova lukova (0 je bez promjene, >= 0))
    cikl_graf = copy.deepcopy(ul_graf)  # kopiranje ulaznog ciklickog grafa

    br_skup_luk = 0          # ukupan broj skupova lukova
    popis_prek_luk = []      # popis nasumicno prekinutih lukova

    sortirani = []           # skup sortiranih cvorova
    bez_ulaza = []           # skup cvorova bez ulaznih lukova
    graf_tsrt = []           # graf za topolosku numeraciju
    sum_pov_luk = 0          # suma povratnih lukova

    for cvor, lukovi in cikl_graf.items():  # trazenje ukupnog broja skupova lukova
        if len(lukovi) is 0:    # u slucaju da smo nasli prazan skup
            continue

        trenutni = lukovi[0]

        for luk in lukovi:
            if trenutni is not luk:     # gledanje dali su dva susjedna skupa razlicita
                br_skup_luk += 1
                trenutni = luk

        br_skup_luk += 1    # svaki skup lukova ima jedan granicnik manje

    skup_prekid = floor(br_skup_luk / 2) - pomak
```

```
42      vj_skup_prekid = skup_prekid    # pamcenje broj skupova lukova koji ce
            se prekinuti radi izracuna vjerojatnosti
43      # uniformno prekidanje lukova ulaznog grafa
44      raspon_rnd_br = len(cikl_graf)      # raspon brojeva za generator
            pseudoslucajnih brojeva
45
46      while skup_prekid:
47          c1 = random.randrange(1, raspon_rnd_br + 1)  # uniforman izbor
                cvora jedan za luk
48          lukovi = cikl_graf[c1]  # dohvacanje lukova radi eliminacije
49
50          if len(lukovi) is 0:
51              continue  # ponovan izbor cvora za luk u slucaju da je izabran
                    cvor bez lukova
52
53          c2 = random.sample(lukovi, 1)        # uniforman izbor cvora dva za
                 luk
54
55          lukovi.remove(c2[0])    # eliminacija odabranog luka
56
57          if lukovi.count(c2[0]) is 0:    # skup lukova je prekinut
58              skp_lk = ul_graf[c1]
59              popis_prek_luk.append([c1, skp_lk.count(c2[0])])
60              skup_prekid -= 1
61
62      for broj in popis_prek_luk:
63          sum_pov_luk += broj[1]    # suma povratnih lukova
64
65      # topolosko sortiranje
66      for cvor, lukovi in cikl_graf.items():  # pronalazak pocetnog skupa
            cvorova bez ulaznih lukova
67          if len(lukovi) is 0:
68              bez_ulaza.append(cvor)  # ako smo nacli cvor bez ulaza dodajemo
                     ga u popis
69
70          else:
71              graf_tsrt.append([cvor, lukovi])  # graf bez cvorova bez
                    ulaznih lukova
72      while bez_ulaza:    # dok god postoji barem jedan cvor bez ulaza
73          zadnji = bez_ulaza.pop()    # makni cvor iz bez_ulaza
74          sortirani.append(zadnji)    # dodaj cvor u sortirani popis
75
76          for lukovi in graf_tsrt:  # micanje izlaznih lukova
77              if len(lukovi[1]) is 0:    # ako smo taj cvor vec dodali
                    preskacemo ga
78                  continue
79
```

```
80              lukovi[1] = [luk for luk in lukovi[1] if luk is not zadnji]
81
82              if len(lukovi[1]) is 0:    # ako smo maknuli sve ulazne lukove
                    cvora
83                  bez_ulaza.append(lukovi[0])
84
85      if len(ul_graf) > len(sortirani):
86          return []  # Graf ima barem jedan ciklus
87      else:
88          v = len(ul_graf)
89          razlomak = (v - 2)/v
90          vjerojatnost = pow(razlomak, vj_skup_prekid)  # vjerojatnost
91
92          return [vjerojatnost, sum_pov_luk, sortirani]
93
94
95  def ponav_mon_car(graf, br_pon, pomak):     # ponavljanje randomiziranog
        algoritma proizvoljan broj puta
96      random.seed(a=random.SystemRandom())    # postavljanje sjemena
            generirajuce funkcije
97
98      povij_svih_rez = []         # pamcenje svih rezultata monte carlo
            algoritma
99      najbolji_rez = [0, 0, []]   # najbolje pronadeno rjesenje
100     vj_br_pon = br_pon          # pamcenje radi izracuna vjerojatnosti
101
102     while br_pon:
103         graf_salji = copy.deepcopy(graf)   # kopija ulaznog grafa, inace
                radi sa referencama
104
105         temp = osrpis_mc_rand(graf_salji, pomak)
106
107         if len(temp) is 3:  # ako je pronaden slijed cvorova
108             povij_svih_rez.append(temp)
109
110             if najbolji_rez[0] is 0:    # prvo rjesenje
111                 najbolji_rez = temp
112
113             elif temp[1] < najbolji_rez[1]:     # pronadeno bolje rjesenje
114                 najbolji_rez = temp
115
116         br_pon -= 1
117
118     # vjerojatnost greske
119     vj_gr = pow(1 - najbolji_rez[0], vj_br_pon)
120
121     # vjerojatnost dosega optimuma
```

```
122     vj_opt = 1 - vj_gr
123 
124     if vj_opt == 1:      # vjerojatnost je premala
125         najbolji_rez[0] = 0.9999999999999999
126     else:
127         najbolji_rez[0] = vj_opt
128 
129     return najbolji_rez
130 
131 
132 print(ponav_mon_car(g, 100, 0))     # (graf, broj ponavljanja, smanjenje
    broja prekidanja skupova lukova (0 je bez promjene)
```

Složenost algoritma u jednom izvršavanju je $O(|E|^2)$. Što znači kako je složenost za neki broj izvršavanja k jednaka $O(k|E|^2)$. S obzirom da smo u velikom djelu algoritma radili sa asocijativnom memorijom algoritam bi trebao raditi brže od ove složenosti ali ovo je gornja granica. Sada ćemo, koristeći grafove sa kojima smo radili prilikom izvršavanja algoritma temeljenog na metodi Grananja i ograničenja, napraviti eksperiment nad Monte Carlo algoritmom. Eksperiment je rađen na procesoru radnog takta od 2.4 GHz i 8 GB radne memorije. Rezultati eksperimenta vide se u tablici 3.3.

Tablica 3.3: Rezultati eksperimenta nad Monte Carlo randomiziranim algoritmom, na grafovima koji imaju manje brojne skupove lukova izvan ciklusa (vidi sliku 3.12; efikasnija verzija).

	Monte Carlo			GiO	
Oznaka grafa	Broj ponavljanja	Pomak	Rješenje	Optimum	Pohlepa
1	10^6	6	25	24	31
2	10^6	3	39	34	54
3	10^6	3	38	36	58
4	10^6	4	38	35	50
5	10^6	0	47	37	57
6	10^6	2	57	47	81
7	10^6	4	38	32	55
8	10^6	1	56	52	93
9	10^6	2	43	40	49
10	10^6	6	30	27	66

Oznaka grafa broj eksperimenta (grafa) za devet čvorova
Pomak promjena broja skupova lukova koje je potrebno prekinuti
Rješenje suma težina povratnih lukova
GiO rezultati algoritma temeljenog na metodi Grananja i ograničenja
Optimum optimalna suma težina povratnih lukova za dani graf
Pohlepa suma težina povratnih lukova dobivena pomoću metode Pohlepe

Grafovi nad kojima je rađen eksperiment očito ne predstavljaju grafove koji su tipa slika 3.11. Stoga je bilo i za očekivati da algoritam neće biti u stanju pronaći optimalna rješenja. Ipak, eksperiment je pokazao kako su rješenja Monte Carlo algoritma blizu optimuma i značajno bolja od sume težina povratnih lukova koju daje algoritam temeljen na metodi Pohlepe. Eksperiment je proveden na sljedeći način.

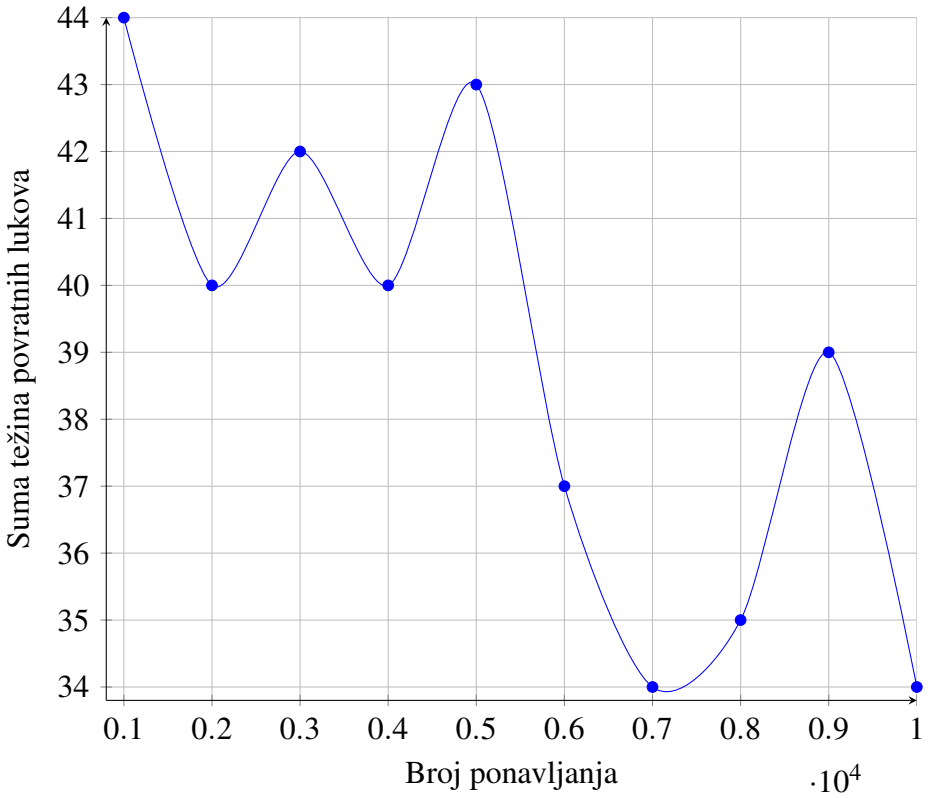
Prvo smo povećavali broj izvršavanja algoritma dok nismo došli do točke gdje algoritam sporo poboljšava izlaz. Nakon toga smo smanjivali broj skupova lukova koje algoritam prekida pomoću varijable *pomak* dok nismo našli pomak za koji algoritam ne može naći rješenje, neka ovaj pomak bude *pom*. Algoritam smo izvršavali 10^6 puta, što je trajalo u prosjeku 25 minuta. Algoritam smo pokrenuli paralelno za $pom-1, pom, pom+1, pom+2$

Tablica 3.4: Procjena pogreške Monte Carlo randomiziranog algoritma, na grafovima koji imaju manje brojne skupove lukova izvan ciklusa (vidi sliku 3.12; efikasnija verzija).

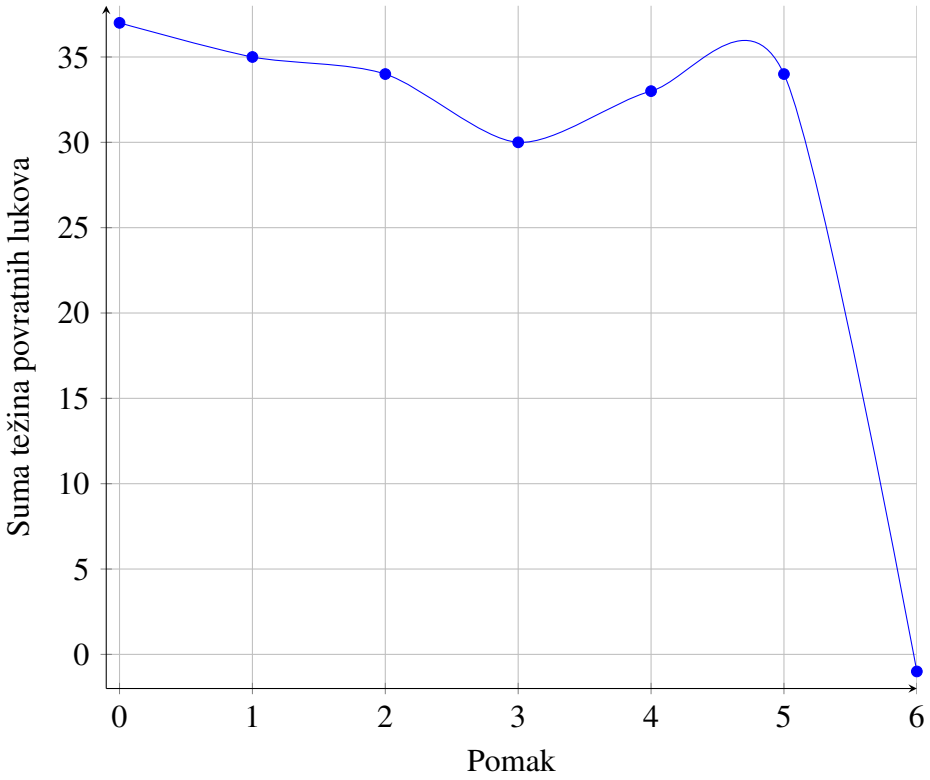
Oznaka grafa	Rješenje	Optimum	Razlika	Relativno
1	25	24	1	4,1666
2	39	34	5	14,7058
3	38	36	2	5,5555
4	38	35	3	8,5714
5	47	37	10	27,0270
6	57	47	10	21,2766
7	38	32	6	18,7500
8	56	52	4	7,6923
9	43	40	3	7,5000
10	30	27	3	11,1111
	Očekivana vrijednost		4,7000	12,6356
		σ	3,1287	7,5641

Oznaka grafa broj eksperimenta (grafa) za devet čvorova
Rješenje Monte Carlo suma težina povratnih lukova
Optimum optimalna suma težina povratnih lukova za dani graf
Razlika razlika rješenja i optimuma
Relativno relativan odnos rješenja i optimuma

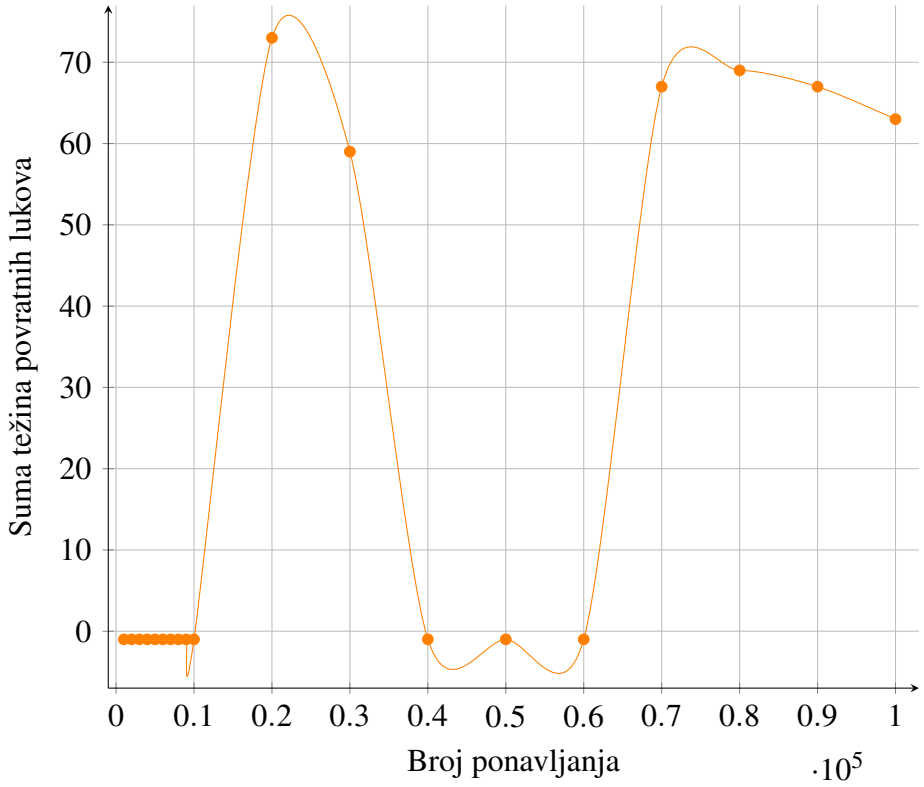
(računalni procesor na kojemu je algoritam testiran ima 4 jezgre). Na ovakav način smo smanjili broj izvršavanja algoritma za različite pomake. Ukoliko je algoritam pronašao rješenje i za $pomak + 2$ povećavali smo pomak sve dok nismo naišli na pomak za koji algoritam ne pronalazi rješenje. Ukoliko su bolja rješenja gravitirala ka suprotnoj strani dodatno smo pokretali algoritam za pomake koji idu u tom smjeru. Na kraju smo od svih dobivenih izlaza odabrali najbolji. Eksperimentiranjem nad grafovima pokazalo se je kako sa 10^6 izvršavanja algoritam pronalazi dobra rješenja u kratkom vremenu. Isto tako važno je napomenuti kako Monte Carlo algoritam dosljedno pronalazi bolja rješenja od rješenja dobivenih sa metodom Pohlepe i sa pomakom 0 nad 1000 izvršavanja sa vremenima izvršavanja unutar jedne sekunde. Na kraju, pogledamo li obrađene rezultate eksperimenta u tablici 3.4 vidimo kako bi algoritam, na grafovima koji nisu teži od grafova korištenih u našim eksperimentima, trebao moći pronalaziti rešenja unutar $4,7 \pm 3,1287$ težina povratnih lukova od optimuma. Ovo predstavlja vrijednosti unutar prvog kvartila što je, barem bismo tako mi argumentirali, solidan rezultat. Pogledamo li sliku 3.13 vidimo kako algoritam na danom grafu sa lakoćom pronalazi rješenja i to sa relativno malim brojem ponavljanja. Vremena izvršavanja za sva rješenja kretala su se između nekoliko desetaka sekundi do nekoliko minuta. Isto tako, vidimo da su najbolja rješenja na ovoj slici blizu optimalnog rješenja za dotični graf. Pogledamo li sliku 3.14 vidimo kako je algoritam pronašao rješenje koje je doslovno nadomak optimalnog rješenja. Algoritam je za svaki pomak izvršavan nekoliko minuta. Nadalje primjećujemo, na ovoj slici ali i



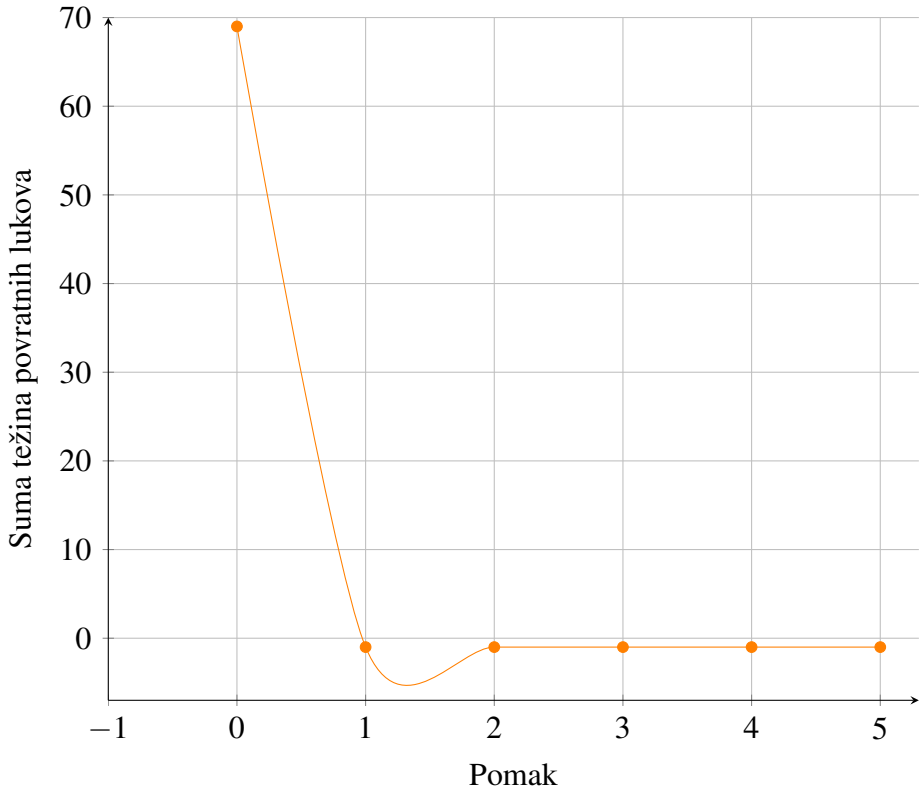
Slika 3.13: Rješenja Monte Carlo algoritma sa fiksnim pomakom (0) i varijabilnim vremenom izvršavanja za graf 1 u tablici 3.3.



Slika 3.14: Rješenja Monte Carlo algoritma sa varijabilnim pomakom i fiksnim vremenom izvršavanja (10^4) za graf 1 u tablici 3.3.



Slika 3.15: Rješenja Monte Carlo algoritma sa fiksnim pomakom (0) i varijabilnim vremenom izvršavanja za graf 6 u tablici 3.3.



Slika 3.16: Rješenja Monte Carlo algoritma sa varijabilnim pomakom i fiksnim vremenom izvršavanja (10^5) za graf 6 u tablici 3.3.

iz tablice 3.3, formula $Br.\,luk. \leq \frac{|E|}{2}$ dobro aproksimira broj lukova, u multigrafu broj skupova lukova, koje je potrebno prekinuti. Usporedimo li slike 3.13 i 3.14 vidimo kako naizmjeničnim mijenjanjem parametara algoritam postupno dolazi do sve boljih rješenja. Referentne točke slike 3.14 ispod nule na y osi predstavljaju slučaj kada algoritam nije uspio pronaći niti jedno rješenje. Primijetimo da pomak od 3 za ovaj graf ne daje najbolje rješenje do kojeg algoritam može doći, kao što se i vidi iz tablice 3.3. Dakle, naizmjenično mijenjanje parametara ponavljamo sve dok nismo zadovoljni sa rješenjem i dok vrijeme izvršavanja algoritma nije predugačko. S druge strane, pogledamo li sliku 3.15 vidimo kako je algoritam za ovaj graf bio relativno nepouzdan i tek za 70.000 i više izvršavanja je počeo dosljedno pronalaziti rješenja. Ipak, algoritam je uspio za 30.000 izvršavanja u samo nekoliko minuta pronaći rješenje gdje je suma težina povratnih lukova 59 što je samo dva više od rješenja koje je algoritam pronašao u tablici 3.3. Vremena izvršavanja za sve pomake na ovoj slici kretala su se unutar nekoliko minuta. Na kraju, pogledamo li sliku 3.16 vidimo kako algoritam ne pronalazi rješenja sa mijenjanjem pomaka. Analizom grafa primjećujemo da se radi o grafu koji ima relativno veliki broj skupova lukova, sa malim brojem lukova izvan ciklusa, te stoga ovakav rasplet događaja nije iznenađujući. Ipak, povećanjem broj izvršavanja algoritma i smanjenjem pomaka, kao što se vidi u tablici 3.3, pronalazimo rješenje koje je blizu optimuma. Dakle, algoritam konvergira sporije što je broj skupova lukova sa malim brojem lukova izvan ciklusa veći. Vremena izvršavanja za sve pomake na ovoj slici bila su unutar nekoliko minuta. Referentne točke na prethodnim slikama ispod nule predstavljaju slučaj kada algoritam nije pronašao niti jedno rješenje. Prethodna četiri eksperimenta predstavljaju nezavisne slučajeve te iz tog razloga primjećujemo različite rezultate za ista stanja algoritma. Stoga zaključujemo, razvijeni Monte Carlo algoritam (efikasnija verzija) pronalazi rješenja koja su unutar $Q1$ nakon određenog broja izvršavanja pi sa određenim pomakom pom.

Ipak, bilo bi dobro kada bismo razvili algoritam koji bi dosezao optimum s proizvoljnom vjerojatnošću na svim grafovima. Problem trenutnog algoritam je taj što prekida potencijalno sve lukove a ne samo one koji su u ciklusu. Međutim, ako za svaki luk koji algoritam nasumično generira provjerimo nalazi li se u ciklusu ovaj problem se može izbjeći. Ova provjera se efikasno može napraviti sa algoritmom Dubinskog pretraživanja (DFS). Pseudokod modificiranog Monte Carlo randomiziranog algoritma, koji radi na svim grafovima, dan je algoritmom 3.5. Modifikacija se nalazi u funkciji $montecarlo(graf, pomak)$:. Dodan je DFS algoritam za detekciju ciklusa, selekcija koja preusmjerava izvršavanje algoritma u slučaju da je generiran luk koji nije dio ciklusa što znači da taj luk ne želimo prekinuti i potrebno je generirati dva nova čvora, i dio koji osigurava da algoritam ne završi u beskonačnoj petlji. Složenost ovog novog algoritma, zbog dodatnog računanja, je $O(k|V|^3)$. S obzirom da smo u velikom djelu algoritma radili sa asocijativnom memorijom algoritam bi trebao raditi brže od ove složenosti ali ovo je gornja granica.

Nakon što smo algoritam implementirali ponovili smo prethodni eksperiment, rezul-

Algoritam 3.5 Pseudokod Monte Carlo randomiziranog algoritma za problem OSRPIS (općenita verzija).

Ulaz: Graf opisan kroz popis susjedstva $(V_{\overline{n}}, V_k), V_k \in V; V_k \neq V_{\overline{n}}$, broj ponavljanja i pomak. Moraju biti dani svi čvorovi bez obzira na ulazne lukove a ulazni lukovi moraju biti grupirani.

Izlaz: $Vj(opt), \sum_{(u,v)\in E,\, u>v} T(u,v)$ i v_i $(i = 1 \ldots |V|)$.

1: **funkcija** $ponavljanje(graf, ponovi, pomak)$:
2: postavi sjeme generatora pseudoslučajnih brojeva
3: **while** *ponovi* **do**
4: $temp = montecarlo(graf, pomak)$
5: **if** pronađeno bolje rješenje **then**
6: $najbolje\_rjesenje = temp$
7: **end if**
8: smanji *ponovi* za 1
9: **end while**
10: izračunaj $1 - \left(1 - \left(\frac{n-2}{n}\right)^{TBSL}\right)^{pi}$
11: **return** $najbolje\_rjesenje$
12:
13:
14: **funkcija** $montecarlo(graf, pomak)$:
15: **while** postoje neposječeni lukovi grafa $(V_{\overline{n}}, V_k), V_k \in V; V_k \neq V_{\overline{n}}$ **do**
16: detekcijom grupa lukova odredi ukupan skup lukova $TBSL$
17: **end while**
18: broj prekidanja skupova lukova je $br\_prek = \frac{TBSL}{2} - pomak$
19: **while** $br\_prek > 0$ **do**
20: generiraj dva čvora po uniformnoj distribuciji
21: Dubinskim pretraživanjem provjeri da li je generirani luk dio ciklusa
22: **if** generirani luk (u,v) nije dio ciklusa **then**
23: ukoliko smo, za redom, generirali veći broj lukova od ukupnog broja lukova grafa prekini generiranje lukova
24: vrati se na generiranje dva čvora po uniformnoj distribuciji
25: **end if**
26: prekini luk
27: **if** prekinut zadnji luk skupa **then**
28: umanji *br_prek* za 1
29: spremi prekinuti skup lukova
30: **end if**
31: **end while**
32: izračunaj sumu težina povratnih lukova
33: topološki sortiraj dobiveni graf
34: **return** $\frac{n-2}{n}^{TBSL}, \sum_{(u,v)\in E,\, u>v} T(u,v)$ i v_i $(i = 1 \ldots |V|)$

tati prethodnog eksperimenta vide se u tablicama 3.3 i 3.4. Rezultati novog eksperimenta vide se u tablicama 3.5 i 3.6. Broj ponavljanja algoritma je 10^6, kao i u prethodnom eksperimentu. Pomak smo mijenjali s obzirom na rješenja koja smo dobivali. Ipak, primjećujemo kako je novi algoritam, za razliku od starog, vrlo često pronalazio rješenja sa pomakom 0. Novi eksperiment pokazuje kako ovaj, modificirani, algoritam vrlo često pronalazi optimalna rješenja, što je bilo za očekivati. Pogledamo li tablicu 3.6 vidimo kako algoritam pronalazi rješenja čija udaljenost od optimuma je unutar $0,5 \pm 0,8498$ težina povratnih lukova. Odnosno, gledano relativno, algoritam pronalazi rješenja čija udaljenost od optimuma je unutar $1,0973\% \pm 1,8937\%$ težina povratnih lukova. Što znači kako ovaj novi, modificirani, algoritam kod vjerojatnosti $\geq 0,9999$, uz adekvatan pomak i broj izvršavanja, pronalazi rješenja koja su optimalna ili u prosjeku udaljena od optimuma $\leq 3\%$.

Tablica 3.5: Rezultati eksperimenta nad Monte Carlo randomiziranim algoritmom (općenita verzija).

	Monte Carlo				GiO	
Oznaka grafa	Broj ponavljanja	Pomak	Vjerojatnost	Rješenje	Optimum	Pohlepa
1	10^6	7	0,9999	24	24	31
2	10^6	4	0,9999	34	34	54
3	10^6	0	0,9999	36	36	58
4	10^6	4	0,9999	35	35	50
5	10^6	0	0,9999	37	37	57
6	10^6	0	0,9999	48	47	81
7	10^6	0	0,9999	32	32	55
8	10^6	0	0,9999	54	52	93
9	10^6	0	0,9999	42	40	49
10	10^6	0	0,9999	27	27	66

Oznaka grafa broj eksperimenta (grafa) za devet čvorova

Pomak promjena broja skupova lukova koje je potrebno prekinuti

Rješenje suma težina povratnih lukova

GiO rezultati algoritma temeljenog na metodi Grananja i ograničenja

Optimum optimalna suma težina povratnih lukova za dani graf

Pohlepa suma težina povratnih lukova dobivena pomoću metode Pohlepe

Tablica 3.6: Procjena pogreške Monte Carlo randomiziranog algoritma (općenita verzija).

Oznaka grafa	Rješenje	Optimum	Razlika	Relativno
1	24	24	0	0,0000
2	34	34	0	0,0000
3	36	36	0	0,0000
4	35	35	0	0,0000
5	37	37	0	0,0000
6	48	47	1	2,1276
7	32	32	0	0,0000
8	54	52	2	3,8461
9	42	40	2	5,0000
10	27	27	0	0,0000
	Očekivana vrijednost		0,5000	1,0973
		σ	0,8498	1,8937

Oznaka grafa broj eksperimenta (grafa) za devet čvorova

Rješenje Monte Carlo suma težina povratnih lukova

Optimum optimalna suma težina povratnih lukova za dani graf

Razlika razlika rješenja i optimuma

Relativno relativan odnos rješenja i optimuma

3.4 Heuristika

Radi potpunosti istraživanja u ovom poglavlju ćemo dati kratak pregled heuristike koja je razvijena tijekom pred istraživanja. "Algoritam prvo pronalazi početno rješenje nakon čega to rješenje pokušava kroz niz koraka poboljšati. Početno rješenje se traži na način da se za svaki čvor računa ulazni i izlazni stupanj kao u tablici 3.7, koja je preuzeta iz (Kudelić et al., 2011). Nakon tog koraka algoritam radi silazno sortiranje čvorova grafa s

Tablica 3.7: Ulazni i izlazni stupanj čvorova grafa. (Kudelić et al., 2011)

Podsustav	Ulazni stupanj	Izlazni stupanj
1	2	4
2	9	9
3	10	6
4	2	4
5	1	1

obzirom na broj izlaznih veza. Stoga je potrebno napraviti n prolaza kroz skup podataka tablice 3.7 i svaki put pronaći

$$max[v_i \xrightarrow{izlaz} D(E)_i] \tag{3.13}$$

te izdvojiti pronađeni čvor na kraj popisa linearnog slijeda. Drugim riječima, pomoću metode Pohlepe izabiremo čvorove grafa koji će biti pomaknuti na početak slijeda razvoja podsustava IS-a. Ukoliko imamo slučaj jednakog broja klasa podataka kod više čvorova, odabir čvora vrši se s obzirom na

$$min[v_j \xrightarrow{ulaz} D(E)_j]. \tag{3.14}$$

Ako i ulazni stupanj nije jedinstven čvor se odabire proizvoljno. Primjenimo li rečeno na tablicu 3.7 dobiti ćemo početni linearan slijed vrhova kao u tablici 3.8 koja je preuzeta iz (Kudelić et al., 2011). Stoga je početni slijed vrhova $SO = [2,3,1,4,5]$. Nakon početnog

Tablica 3.8: Matrica početnog slijeda. (Kudelić et al., 2011)

Podsustav	Ulazni stupanj	Izlazni stupanj
2	9	9
3	10	6
1	2	4
4	2	4
5	1	1

slijeda vrhova za svaki čvor računaju se povratne

$$\sum_{j=1}^{i-1} D(E)_{\bar{i}j} \tag{3.15}$$

i slijedne

$$\sum_{j=1}^{i-1} D(E)_{j\bar{i}} \tag{3.16}$$

veze. Stoga, ako je

$$ISs_i(Bc_{i-k}) > ISs_{i-k}(Fc_i), \tag{3.17}$$

pri čemu $k = 1, \ldots, i-1$, pronašli smo bolje mjesto za promatrani čvor u vidu broja povratnih lukova linearnog slijeda i stoga možemo napraviti zamjenu čvorova. Dakle, ako je za neku instancu ISs_p

$$\left(ISs_p(Bc_{p-k}) \leq ISs_{p-k}(Fc_p)\right) = false \tag{3.18}$$

sljedi kako je

$$\left(ISs_p(Bc_{p-k}) > ISs_{p-k}(Fc_p)\right) = true \tag{3.19}$$

i stoga je potrebno napraviti zamjenu čvorova kako bismo poboljšali ukupnu sumu težina povratnih veza u linearnom slijedu razvoja podsustava IS-a. Međutim, prethodni uvjet nije dovoljan s obzirom da moramo uzeti u obzir i sve čvorove između. Stoga je potrebno odrediti sumu

$$\sum_{k=1}^{p-1} [ISs_p(Bc_{p-k}) - ISs_{p-k}(Fc_p)] = SPL \tag{3.20}$$

(Suma težina Povratnih Lukova) pri čemu je potrebno napraviti zamjenu čvorova jedino ako je $SPL > 0$. Nadalje, s obzirom da želimo pronaći najbolju poziciju moramo za svaki čvor pronaći

$$max\left(\sum_{k=1}^{p-1} \left[ISs_p(Bc_{p-k}) - ISs_{p-k}(Fc_p)\right] > 0\right) \tag{3.21}$$

kako bismo mogli dodatno optimizirati početni slijed vrhova. Ukoliko je ta suma jednaka nuli to znači kako ne postoji poboljšanje i nije isplativo raditi pomak a ako je manja od nule to znači da bi broj stvorenih povratnih veza bio veći od broja elimininiranih povratnih veza." (Kudelić et al., 2011) Algoritam je implementiran u programskom jeziku *C#* i može se vidjet u radu (Kudelić et al., 2012). "Algoritam je podjeljen na četiri funkcije:

- funkcija za pronalazak ulaznog stupnja,
- funkcija za pronalazak izlaznog stupnja,
- funkcija za pronalazak početnog slijeda,

- funkcija za inkrementalno poboljšanje početnog slijeda i pronalazak konačnog rješenja.

Složenost funkcije koja računa izlazni stupanj je $O(n^2)$. Složenost je proporcionalna veličini strukture koja sadrži podatke i stoga je funkcija efikasna. Složenost funkcija koje računaju ulazni stupanj i početni slijed je isto tako $O(n^2)$ i funkcije su efikasne. Međutim, zadnja funkcija koja radi inkrementalno poboljšanje slijeda i pronalazi konačno rješenje ima složenost $O(n^3)$ što nije proporcionalno veličini strukture koja sadrži podatke te stoga ova funkcija u tom smislu nije efikasna. Ipak, složenost je polinom trećeg stupnja što je, imajući u vidu težinu problema koji se rješava, zadovoljavajuće s obzirom da ova funkcija za razliku od prethodne dvije mora odraditi dodatne korake kako bi došla do konačnog rješenja." (Kudelić et al., 2012)

Empirijska analiza dana je u radu (Kudelić et al., 2013). "Algoritam radi unutar jedne sekunde za do 200 čvorova i 500 lukova. Kada je algoritam testiran za 500 čvorova i 100 lukova vrijeme izvršavanja doseglo je $11,91s$ a za 600 čvorova i 1200 lukova vrijeme izvršavanja bilo je $20,27s$." (Kudelić et al., 2013) Ova heuristika očito je praktično primjenjiva s obzirom da je dovoljno brza za pronalazak linearnog slijeda razvoja bilo kojeg IS-a. Teško je zamisliti IS koji bi bio dekomponiran do razine detaljnosti koja bi od prethodno opisane heuristike zahtijevala vrijeme izvršavanja u mjesecima ili godinama. Ipak, algoritam ima jednu manu a to je ne mogućnost procjene udaljenosti izlaza algoritma od optimuma.

3.5 Korištenje razvijenih algoritama i načini uvrštavanja dodatnih ograničenja

Kako bismo riješili problem OSRPIS razvili smo dva nova algoritma. Jedan temeljen na metodi Grananja i ograničenja koji uvijek kada se izvrši do kraja pronalazi optimalno rješenje. Drugi temeljen na metodi Monte Carlo koji doseže optimum sa proizvoljnom vjerojatnošću u polinomnom vremenu. Algoritmi su razvijeni sa ciljem dopunjavanja postojećih metodologija razvoja IS-a.

Ukoliko se odlučimo za određivanje slijeda razvoja podsustava IS-a pomoću algoritma temeljenog na metodi Grananja i ograničenja prvo trebamo opisati IS preko p/k matrice. Nakon toga je p/k matricu potrebno pretvoriti u graf. Struktura podataka za graf je popis lukova.

```
# primjer ulaznog grafa za algoritam
# temeljen na metodi grananja i ograničenja
g = [
    [[1, 3], 2],
    [[1, 2], 2],
    [[2, 3], 8],
    [[2, 1], 1],
    [[3, 1], 1],
    [[3, 2], 3],
    [[3, 4], 2],
    [[4, 2], 3],
    [[4, 5], 1],
    [[5, 2], 1]
]
```

Svaki luk sačinjen je od dva čvora i težine luka. Prvi čvor je početni a drugi završni, što znači kako je svaki luk usmjeren od lijeva na desno. Nakon što takav graf damo algoritmu kao ulaz algoritam če pronaći optimalno rješenje u obliku sume

$$\sum_{\substack{(u,v)\in E \\ u>v}} T(u,v)$$

i slijeda podsustava IS-a

$$v_i \ (i = 1 \ldots |V|).$$

Jedan primjer izlaza algoritma izgleda kako sljedi.

```
('Suma->', 7, 'Slijed->', [1, 4, 5, 2, 3])
```

Nakon čega grupa ljudi koja je odgovorna za određivanje slijeda razvoj podsustava informacijskog sustava postupa prema dobivenom izlazu iz algoritma.

Ukoliko se pak iz određenih razloga odlučimo za Monte Carlo randomizirani algoritam postupak dobivanja grafa za ulaz u algoritam je isti kao kod algoritma temeljenog na metodi Grananja i ograničenja. Međutim, ulazni graf za Monte Carlo algoritam je zbog efikasnije izvedbe algoritma promijenjen u popis susjedstva. Isto tako, graf koji predstavlja ulaz u ovaj algoritam je multigraf. Ulazna struktura podataka izgleda kako sljedi.

```
# primjer ulaznog grafa
# za Monte Carlo algoritam
g = {
    1: [2, 2, 2, 2, 2, 2, 2, 4, 4, 4, 7, 7, 6, 6, 6, 6, 6, 3, 3,
        3, 3, 3],
    2: [9, 9, 5, 5, 5, 6, 6, 6, 6, 6, 6, 8, 8, 8, 8, 8, 8, 8, 3,
        3, 3, 3, 3, 3, 3, 3, 3],
    3: [5, 6, 6],
    4: [2, 2, 2, 2, 2, 2, 2, 2, 2, 7, 5, 5, 5, 5, 5, 5, 8, 8, 9,
        9, 9, 9, 1, 1],
    5: [6, 6, 6, 6, 1, 1, 1, 1, 1, 4, 4, 4, 4, 4, 4, 4, 4, 7, 7,
        2, 2, 2, 2, 2, 2, 8, 3, 3],
    6: [1, 1, 9, 9, 9, 9, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8, 8, 8],
    7: [5, 5, 5, 5, 8],
    8: [7, 7, 7, 7, 7, 7, 7, 7, 7, 4, 2],
    9: [8, 8, 8, 5, 5, 5, 5, 5, 5, 7, 7, 7, 7],
}
```

Ključevi asocijativne memorije predstavljaju ulazne čvorove dok vrijednosti svakog ključa predstavljaju izlazne čvorove multigrafa. Nakon što algoritmu damo ulazni graf, broj izvršavanja, te pomak, algoritam će kroz niz koraka opisan u prijašnjem poglavlju pronaći vjerojatnost

$$\left(1-\left(1-\left(\frac{n-2}{n}\right)^{\frac{n}{2}}\right)^{pi}\right),$$

sumu

$$\sum_{\substack{(u,v)\in E\\ u>v}} T(u,v),$$

i topološko sortirani slijed podsustava IS-a v_i $(i = 1 \ldots |V|)$. Jedan primjer izlaza algoritma izgleda kako sljedi.

```
[0.9999816985099514, 52, [9, 8, 7, 6, 3, 2, 4, 1, 5]]
```

Postupak ponavljamo sve dok nismo zadovoljni sa rješenjem. Nakon čega grupa ljudi

koja je odgovorna za određivanje slijeda razvoj podsustava informacijskog sustava postupa prema dobivenom izlazu iz algoritma. Ukoliko iz nekog razloga postoji potreba narušavanja slijeda razvoja podsustava IS-a pronađenog od strane bilo kojeg od ova dva algoritma to možemo napraviti na sljedeći način.

Prvi način narušavanja slijeda razvoja moguće je ostvariti sa brisanjem čvora iz grafa što će rezultirati i brisanjem svih lukova tog čvora. Drugi način je brisanje jednog od lukova zbog kojih je taj čvor u ciklusu ili više lukova ukoliko je čvor sadržan u više ciklusa. U slučaju da grupa ljudi zadužena za određivanje slijeda razvoja podsustava IS-a odluči kako je ovisnost čvora s kojim se želi narušiti algoritamski pronađeni slijed o ostatku čvorova nevažna, bolje rješenje je brisanje čvora i svih popratnih lukova iz grafa s obzirom da će ovaj način narušavanja očito značajno ubrzati rad oba algoritma. Ako bi se ipak htjela zadržati ovisnost čvora, koja proizlazi iz opisa poslovnog sustava, sa kojim se želi narušiti slijed koji daje algoritam onda je bolje rješenje brisanje jednog ili više lukova koji će izbaciti taj čvor iz svih ciklusa. Brisanjem različitih lukova čvora kojeg se želi izbaciti iz ciklusa moguće je pomicati mjesto tog čvora u slijedu čvorova, naime tada taj čvor svaki put ovisi o drugim čvorovima. Na ovaj način možemo upravljati pozicijom čvora u slijedu čvorova s obzirom na djelomičnu ovisnost čvora o ostatku grafa a da ne prekinemo u potpunosti svaku vezu čvora i ostatka grafa.

3.6 Primjena OSRPIS algoritama na ostale probleme

S obzirom da je problem OSRPIS vrlo općenit, algoritmi koji se koriste za rješavanje tog problema se potencijalno mogu koristiti i za rješavanje drugih problema koji su suštinski isti ili slični. Općenito gledano ovi algoritmi se potencijalno mogu koristiti za svaki problem koji u sebi sadrži neku vrstu određivanja rasporeda. Neki od poznatiji problema gdje bi se OSRPIS algoritmi mogli koristiti jesu: Određivanje kritičnog puta (poznata metoda s kojom se rješava ovaj problem zove se Metoda kritičnog puta, eng. Critical Path Method), Raspoređivanje zadataka na procesoru, i Trgovački putnik.

Metoda kritičnog puta "računa najduži put od početne i završne točke za planirane aktivnosti s obzirom na vremena početka i završetka planiranih aktivnosti. Vremena početka i završetka kreću se u određenom rasponu. Što znači da kritični put predstavlja onaj niz aktivnosti koji svojim produljenjem direktno produljuje trajanje projekta." (Armstrong-Wright, 1969) Imajući u vidu rečeno, prilikom promatranja algoritama koji rješavaju problem OSRPIS vidimo kako je moguće modificirati dotične algoritme na način da rješavaju problem Određivanja kritičnog puta. Naime, svaki algoritam koji se bazira na računanju sume težina povratnih lukova moguće je modificirati tako da računa vrijeme trajanja, kako je definirano u Metodi kritičnog puta. Isto tako, moguće je da bi OSRPIS algoritmi davali slijed aktivnosti koji je blizu izlazu koji daje Metoda kritičnog puta, međutim ovo bi trebalo dodatno istražiti. Nadalje, algoritme koji su bazirani na prekidanju lukova moguće je iskoristiti kako bismo ciklički graf pretvorili u aciklički graf nad kojim je moguće odrediti niz aktivnosti koje predstavljaju kritični put.

Raspoređivanje zadataka na procesoru predstavlja problem gdje skup procesa u određenom trenutku mora ostvariti međusobnu komunikaciju kako bi se izveo određeni zadatak. (Tanenbaum, 2009) Usporedimo li ovaj problem sa problemom OSRPIS vidimo kako su problemi vrlo slični. Naime, ukoliko bi graf $G = (V,E)$ koji predstavlja ulaz u OSRPIS algoritam Grananja i ograničenja predstavljao skup procesa i njihove međusobne veze, linearan slijed tih procesa, gdje bi suma

$$\sum_{\substack{(u,v)\in E \\ u>v}} T(u,v)$$

predstavljala broj poziva procesa na procese, predstavljao bi slijed sa minimalnim brojem zamjena konteksta.

Naposljetku, promotrimo li problem Trgovačkog putnika i naš problem zapažamo sljedeće. Problem Trgovačkog putnika (minimum) glasi kako sljedi.

"Instanca: Skup C od m gradova, udaljenost $d(c_i,c_j) \in N$ za svaki par gradova $c_i,c_j \in C$.

Rješenje: Tura od C, odnosno, permutacija $\pi : [1..m] \longrightarrow [1..m]$.

Mjera: Dužina ture, odnosno, $d\left(\{c_{\pi(m)}, c_{\pi(1)}\}\right) + \sum_{i=1}^{m-1} d\left(\{c_{\pi(i)}, c_{\pi(i+1)}\}\right)$." (Crescenzi and Kann, 1998)

Bez modifikacije, OSRPIS algoritmi se ne mogu koristiti za ovaj problem, to je očito. Međutim, OSRPIS algoritmi bi nam mogli dati odgovor na jedno drugo zanimljivo pitanje, koje problem Trgovačkog putnika ne postavlja, koje glasi: Koja tura, minimalne duljine, gradova C, prema (Crescenzi and Kann, 1998)

$$d\left(\{c_{\pi(m)}, c_{\pi(1)}\}\right) + \sum_{i=1}^{m-1} d\left(\{c_{\pi(i)}, c_{\pi(i+1)}\}\right),$$

nam daje minimalne povratne udaljenosti

$$\sum_{\substack{(u,v)\in E \\ u>v}} T(u,v)?$$

Kako bismo dobili odgovor na ovo pitanje potrebno je stvoriti novi općenitiji problem koji se sastoji od problema Trgovačkog putnika i problema OSRPIS. Zaključno napomenimo kako je OSRPIS algoritme moguće koristiti ili vrlo brzo modificirati za rješavanje PSL, MPSL, i pronalaženje topološki sortiranog slijeda.

Poglavlje 4

Zaključak

Tijekom istraživanja, nakon opisa problema i kritičkog osvrta, postavili smo dvije hipoteze i jedan cilj. Prva hipoteza glasila je kako sljedi.

HIPOTEZA 1 Problem Određivanja redoslijeda razvoja podsustava u informacijskom sustavu je NP-težak i APX-težak.

Preko teorema 3.1.1 dokazali smo kako je problem OSRPIS NP-težak (vidi korolar 3.1.1) i APX-težak (vidi korolar 3.1.2). Stoga zaključujemo, prva hipoteza je dokazana. Dodatno smo dokazali kako je problem OSRPIS NP-potpun, teoremom 3.1.1. Druga hipoteza glasila je kako sljedi.

HIPOTEZA 2 Algoritam temeljen na metodi Monte Carlo rješava problem Određivanja redoslijeda razvoja podsustava u informacijskom sustavu s proizvoljnom vjerojatnošću dosizanja optimuma u polinomnom vremenu.

S obzirom da smo dokazali lemu 3.3.1 i razvili pripadajući algoritam čija složenost je polinomno ograničena, zaključujemo, druga hipoteza je dokazana. Isto tako, s obzirom da su obje hipoteze dokazane, zaključujemo, ostvarili smo zadani cilj. Zadani cilj glasio je kako sljedi.

CILJ Pronaći efikasno algoritamsko rješenje za problem Određivanja redoslijeda razvoja podsustava informacijskog sustava.

U svrhu rješavanja problema OSRPIS osmisli smo: algoritam temeljen na metodi Grananja i ograničenja, Monte Carlo randomizirani algoritam, i heuristički algoritam. Algoritmi su analizirani, implementirani, i empirijski testirani. Empirijskim testiranjem utvrdili smo praktična ograničenja algoritama. Za Monte Carlo randomizirani algoritam dodatno je izračunata vjerojatnost. Na kraju smo objasnili: na koji način smo zamislili korištenje algoritama, na koji način se mogu uvrštavati dodatna ograničenja, i koji su to ostali problemi gdje se OSRPIS algoritmi potencijalno mogu koristiti.

_____

(vlastoručni potpis)

Poglavlje 5

Prilozi

Implementacija algoritma 5.1: Eksperiment nad algoritmom, temeljenom na metodi Grananja i ograničenja, za optimalno rješavanje problema OSRPIS.

```
1  __author__ = 'Robert'
2
3
4  from treelib import Tree  # izvor: https://github.com/caesar0301/pyTree,
       ucitano: 29.08.2014.
5  import random
6  import time
7
8  prekid_novi_opt = 0          # prekidanje pretrage stabla zbog pronalaska
       novog optimuma
9  prekid_stari_opt_bolji = 0   # prekidanje pretrage stabla zbog boljeg starog
        optimuma
10 broj_pros_cvor = 0           # broj prosirenja cvorova stabla
11 broj_pos_cvor = 0            # broj posjecenih cvorova pri izgradnji stabla
12 min_visa_raz = 0             # broj izbora grane sa minimumom na visoj
       razini
13
14
15 def zavrsi_stablo(korijen, vrhovi_grafa, minimalna_suma_slijeda, grf):    #
        dovrsi stablo i pronadji optimum podstabla
16
17     podstablo = Tree()
18     podstablo.create_node(str(korijen) + ";0", korijen)      # stvaranje
           korijena podstabla
19     instanca_rod_tren_cvora = podstablo.parent(korijen)      # instanca
           cvora podstabla preko koje dolazimo do oca
20     trenutni_cvor = podstablo.get_node(korijen)      # trenutni potencijalno
            optimalni odabir
21     id_cvora = len(vrhovi_grafa) + 1                 # jedinstveni
           identifikator cvorova u podstablu
22     global prekid_novi_opt       # globalne varijable za eksperiment
```

```
23      global prekid_stari_opt_bolji
24      global broj_pros_cvor
25      global broj_pos_cvor
26      global min_visa_raz
27      min_visa_raz_detekcija = False
28
29      popis_cvorova_perm = []      # popis svih predaka odredjenog cvora
            podstabla
30      cvor_prosiren = False        # indikacija moze li se trenutni cvor dalje
            prosirivati
31
32      print(korijen)  # ispis korijena stabla koje se trenutno prosiruje
33
34      while True:
35          while True:
36              if instanca_rod_tren_cvora is not None:  # trazenje cvorova
                    permutacije radi sirenja stabla
37                  popis_cvorova_perm.append(int(instanca_rod_tren_cvora.tag[:
                        instanca_rod_tren_cvora.tag.rfind(';')]))
38                  instanca_rod_tren_cvora = podstablo.parent(
                        instanca_rod_tren_cvora.identifier)
39              else:
40                  break
41
42          # suma povratnih lukova do trenutnog cvora
43          suma_trenutni_cvor = int(trenutni_cvor.tag[trenutni_cvor.tag.rfind(
                ';') + 1:])
44
45          for novi_cvor in vrhovi_grafa:
46              # filtriranje cvorova permutacije
47              if novi_cvor not in popis_cvorova_perm \
48                      and novi_cvor != int(trenutni_cvor.tag[:trenutni_cvor.
                            tag.rfind(';')]):
49
50                  suma_novi_cvor = \
51                      tezina_veze(novi_cvor, int(trenutni_cvor.tag[:
                            trenutni_cvor.tag.rfind(';')]),
52                                  popis_cvorova_perm, grf) +
                                        suma_trenutni_cvor
53                  # dodavanje sinova na trenutni potencijalno optimalni
                        odabir
54                  id_cvora += 1
55                  podstablo.create_node(str(novi_cvor) + ";" + str(
                        suma_novi_cvor), id_cvora,
56                                        parent=trenutni_cvor.identifier)
57                  cvor_prosiren = True
58                  broj_pos_cvor += 1      # eksperiment
```

```
59
60          if cvor_prosiren:
61              broj_pros_cvor += 1      # eksperiment
62
63          if not cvor_prosiren:  # pronadjen novi optimum
64              prekid_novi_opt += 1     # eksperiment
65
66              popis_cvorova_perm.reverse()
67              popis_cvorova_perm.append(int(trenutni_cvor.tag[:trenutni_cvor.
                    tag.rfind(';')]))      # linearan slijed
68
69              suma = int(trenutni_cvor.tag[trenutni_cvor.tag.rfind(';') + 1
                    :])
70
71              #podstablo.show(line_type='ascii')
72              return suma, popis_cvorova_perm
73
74          listovi_stabla = podstablo.leaves()    # dohvacanje instanci listova
                podstabla
75          najmanja_suma_list = listovi_stabla[0]
76
77          for list_ in listovi_stabla:     # pronalazak lista sa najmanjom
                sumom povratnih lukova
78              if int(list_.tag[list_.tag.rfind(';') + 1:]) \
79                      < int(najmanja_suma_list.tag[najmanja_suma_list.tag.
                            rfind(';') + 1:]):
80                  najmanja_suma_list = list_
81
82              elif int(list_.tag[list_.tag.rfind(';') + 1:]) \
83                      == int(najmanja_suma_list.tag[najmanja_suma_list.tag.
                            rfind(';') + 1:]):
84                      # u slucaju da je list sa jednakom vrijednosti
                            povratnih lukova na visoj razini podstabla
85                      if podstablo.level(list_.identifier) > podstablo.level(
                            najmanja_suma_list.identifier):
86                          najmanja_suma_list = list_
87                          min_visa_raz_detekcija = True    # eksperiment
88
89          if min_visa_raz_detekcija:
90              min_visa_raz += 1    # eksperiment
91
92          # ako je suma trenutnog slijeda veca od do sada optimalnog slijeda
                prekini daljnju potragu
93          if int(najmanja_suma_list.tag[najmanja_suma_list.tag.rfind(';') + 1
                :]) >= minimalna_suma_slijeda:
94              prekid_stari_opt_bolji += 1      # eksperiment
95
```

```
96              #podstablo.show(line_type='ascii')
97              return []
98
99          # ponovno postavljanje privremenih varijabli
100         trenutni_cvor = najmanja_suma_list
101         instanca_rod_tren_cvora = podstablo.parent(trenutni_cvor.identifier
                )
102         popis_cvorova_perm = []
103         cvor_prosiren = False
104         min_visa_raz_detekcija = False
105
106
107 def zapocni_stablo(graf):  # zapocni sva stabla
108
109     suma_lukova = 0     # maksimalna globalna suma na pocetku i minimalna
            suma kasnije
110     popis_vrhova = []   # popis svih vrhova grafa
111     slij_vrhova_min_pov_sume_luk = []   # slijed cvorova sa minimalnom
            sumom povratnih lukova
112     global broj_pos_cvor    # varijabla za eksperiment
113
114     for tezina_luka in graf:
115         if tezina_luka[0][0] not in popis_vrhova:   # pronalazak vrhova
                grafa
116             popis_vrhova.append(tezina_luka[0][0])
117         if tezina_luka[0][1] not in popis_vrhova:   # u slucaju da cvor
                nema izlaznih lukova
118             popis_vrhova.append(tezina_luka[0][1])
119
120     print("Skup vrhova grafa: " + str(popis_vrhova))    # ispis svih vrhova
            grafa
121
122     popis_vrh_temp = []         # popis svih vrhova sa izlaznim lukovima
123
124     # pronalazak broja izlaznih lukova za svaki vrh
125     for vrh_ in popis_vrhova:
126         suma_vrh = 0
127
128         for tezina_luka in graf:
129             if tezina_luka[0][0] == vrh_:
130                 suma_vrh += tezina_luka[1]
131
132         popis_vrh_temp.append([vrh_, suma_vrh])
133     # sortiranje prema broju izlaznih lukova
134     popis_vrh_temp = sorted(popis_vrh_temp, key=lambda popis: popis[1])
135
136     brojac_vrh = 0
```

```
137     broj_vrhova = len(popis_vrh_temp)
138     popis_pov_luk = []      # popis cvorova i povratnih lukova
139
140     # racunanje povratnih veza za sortirani niz lukova
141     while brojac_vrh < broj_vrhova:
142         brojac_vrhovi = brojac_vrh + 1
143         suma_pov_luk = 0
144
145         while brojac_vrhovi < broj_vrhova:
146             for luk in graf:
147                 if popis_vrh_temp[brojac_vrh][0] == luk[0][0] and
                        popis_vrh_temp[brojac_vrhovi][0] == luk[0][1]:
148                     suma_pov_luk += luk[1]
149                     break
150
151             brojac_vrhovi += 1
152
153         # pohranjivanje sume povratnih lukova za svaki cvor
154         popis_pov_luk.append([popis_vrh_temp[brojac_vrh][0], suma_pov_luk])
155
156         brojac_vrh += 1
157
158     for luk_i in popis_pov_luk:     # sumiranje povratnih veza i
            postavljanje pocetnog optimuma
159         suma_lukova += luk_i[1]
160
161     print("Pocetni optimum: " + str(suma_lukova))    # ispis heuristicki
            pronadjenog optimuma
162     suma_lukova += 1    # postavljanje sume lukova na jedan vise radi
            pronalazenja heuristicki otkrivenog optimuma
163     print("Pocetni uvecani optimum: " + str(suma_lukova))    # ispis
            uvecanog heuristicki pronadjenog optimuma u slucaju pronadjenog
            optimuma
164     print("\n::::: Pocetak hodanja po stablu :::::")
165
166     for vrh in popis_vrhova:    # prolazak kroz podstabla
167         broj_pos_cvor += 1  # eksperiment
168
169         temp = zavrsi_stablo(vrh, popis_vrhova, suma_lukova, graf)
170         if len(temp) == 2:      # pronadjen novi optimum
171             suma_lukova = temp[0]
172             slij_vrhova_min_pov_sume_luk = temp[1]
173
174     print("::::: Zavrsetak hodanja po stablu :::::\n")
175     print("Pronasli smo novi optimum: " + str(prekid_novi_opt))     # broj
            prekidanja stvaranja stabla zbog pronalaska novog optimuma
```

```
176     print("Prekid␣jer␣je␣stari␣optimum␣bio␣bolji:␣" + str(
            prekid_stari_opt_bolji))  # broj prekidanja zbog boljeg starog
            optimuma
177     print("Broj␣prosirenja␣cvorova:␣" + str(broj_pros_cvor))        # broj
            prosirenja cvorova stabla
178     print("Broj␣posjecenih␣cvorova␣stabla:␣" + str(broj_pos_cvor))  #
            ukupan broj posjecenih cvorova stabla
179     print("Broj␣grana␣sa␣minimumom␣na␣visoj␣razini:␣" + str(min_visa_raz))
             # grana sa minimumom na visoj razini
180
181     return "Suma->", suma_lukova, "Slijed->", slij_vrhova_min_pov_sume_luk
              # vracanje globalnog optimuma
182
183
184 def tezina_veze(u, v, preci_v, grf):
185
186     suma_tezina = 0                 # suma tezina izmedju cvora u i svih
            prijasnjih cvorova
187     popis_ocuvanog_slijeda = [v]    # popis cvorova ocuvanog slijeda
188
189     popis_ocuvanog_slijeda.extend(preci_v)
190
191     for vrh_ in popis_ocuvanog_slijeda:
192         for luk in grf:
193             if luk[0][0] == u and luk[0][1] == vrh_:   # sumiranje
                    povratnih lukova
194                 suma_tezina += luk[1]
195                 break
196
197     return suma_tezina
198
199
200 def eksperiment(br_cvorova, br_pon_eks):    # zapocni eksperiment i
        ponavljaj ga br_pon_eks puta
201
202     # u slucaju da je dan trivijalan graf ili
203     # ako je broj ponavljanja eksperimenta manji od jedan izadji van iz
            funkcije
204     if br_cvorova ≤ 2 or br_pon_eks < 1:
205         return "Broj␣cvorova␣mora␣biti␣veci␣od␣2\nBroj␣ponavljanja␣
                eksperimenta␣mora␣biti␣veci␣od␣0"
206
207     br_lukova = ((br_cvorova * (br_cvorova - 1))/2) + 1     # broj lukova
            koje je potrebno generirati
208     random.seed(a=random.SystemRandom())
209
210     for br_pon in range(1, br_pon_eks + 1):
```

```
211         print("Eksperiment " + str(br_pon) + ":-------")
212
213         popis_cvorova = []  # skup cvorova koje je potrebno povezati sa
                lukovima
214
215         # generiranje skupa cvorova
216         for cvor in range(1, br_cvorova + 1):
217             popis_cvorova.append(cvor)
218
219         graf = []   # popis lukova i tezina na temelju kojega ce se zvati
                funkcija zapocni_stablo():
220         temp = []   # privremeni popis za spremanje luka i tezine
221         skup_lukova = []    # skup svih generiranih lukova
222         # stvaranje grafa g((u,v),T(u,v))
223         for broj_luka in range(1, int(br_lukova) + 1):
224
225             while True:     # uniforman odabir jedinstvenog luka
226                 luk = random.sample(popis_cvorova, 2)   # uniforman odabir
                        luka
227
228                 if luk not in skup_lukova:
229                     skup_lukova.append(luk)
230                     temp.append(luk)    # stvaranje strukture luk-tezina
231                     break
232
233             tmp_tezina_luk = 0  # tezina generiranog luka
234
235             if broj_luka == 1:
236                 print("-----Distribucija tezina-----")  # pocetak ispisa
                        distribucije tezina
237
238             # generiranje tezine luka u rasponu [1,br_cvorova] prema
                    eksponencijalnoj distribuciji
239             while tmp_tezina_luk < 1:
240                 tmp_tezina_luk = int(random.expovariate(0.15)) % (
                        br_cvorova + 1)
241
242             print(tmp_tezina_luk)   # ispis tezina lukova
243             if broj_luka == br_lukova:
244                 print("-----Kraj ispisa distribucije tezina-----")  #
                        zavrsetak ispisa tezina lukova
245
246             # stvaranje strukture luk-tezina
247             temp.append(tmp_tezina_luk)
248
249             # stvaranje grafa
250             graf.append(temp)
```

```
251
252             # ponovno postavljanje privremenih varijabli
253             temp = []
254
255         print(graf)
256
257         t_poc = time.process_time()     # mjerenje vremena pocetka
                eksperimenta (bez vremena pauze)
258
259         # pozivanje funkcije za trazenje optimalnog rjesenja
260         print(zapocni_stablo(graf))
261
262         t_zav = time.process_time() - t_poc
263         print("Vrijeme izvrsavanja algoritma: " + str(t_zav))
264
265         # postavljanje globalnih varijabli na pocetne vrijednosti
266         global prekid_novi_opt
267         global prekid_stari_opt_bolji
268         global broj_pros_cvor
269         global broj_pos_cvor
270         global min_visa_raz
271
272         prekid_novi_opt = 0
273         prekid_stari_opt_bolji = 0
274         broj_pros_cvor = 0
275         broj_pos_cvor = 0
276         min_visa_raz = 0
277
278         print("----------------------")
279         print("----------------------")
280
281     return "\n----|Kraj eksperimenata|--"
282
283
284 print(eksperiment(5, 1))
```

Literatura

Arge, L., L. Toma, and N. Zeh
Lipanj 2003. I/O-efficient topological sorting of planar DAGs. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, A. Rosenberg?, ed.

Armstrong-Wright, A.
Siječanj 1969. *Critical Path Method*, 1 edition. Prentice Hall Press.

Boehm, B.
Veljača 2000. Spiral Development: Experience, Principles, and Refinements. http://www.dtic.mil/dtic/tr/fulltext/u2/a382590.pdf, učitano 11.8.2014.

Boehm, B. and V. Basili
Siječanj 2001. Software Defect Reduction Top 10 List. *Computer*.

Boehm, B., A. Egyed, J. Kwan, D. Port, A. Shah, and R. Madachy
Srpanj 1998. Using the WinWin Spiral Model: A Case Study. *Computer*.

Centers for Medicare & Medicaid Services
Ožujak 2008. Selecting a development approach. http://www.cms.gov/Research-Statistics-Data-and-Systems/CMS-Information-Technology/XLC/Downloads/SelectingDevelopmentApproach.pdf, učitano 8.8.2014.

Coleman, G. and R. Verbruggen
Srpanj 1998. A Quality Software Process for Rapid Application Development. *Software Quality Journal*.

Conway, R. W., W. L. Maxwell, and L. W. Miller
Lipanj 2003. *Theory of Scheduling*, 1 edition. Dover Publications.

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein
Srpanj 2009. *Introduction to Algorithms*, 3 edition. MIT Press.

Crescenzi, P. and V. Kann
Kolovoz 1998. A compendium of NP optimization problems. http://www.csc.byblos.lau.edu.lb/classes/csc888-Optimization/spring2004/NP-Problems.pdf, učitano 20.8.2014.

Davenport, T. H. and J. E. Short

Srpanj 1990. The New Industrial Engineering: Information Technology and Business Process Redesign. *MIT Sloan Management Review*.

de Fraysseix, H., P. O. de Mendez, and P. Rosenstiehl

Lipanj 2006. Trémaux Trees and Planarity. *International Journal of Foundations of Computer Science*.

Geambasu, C. V., I. Jianu, I. Jianu, and A. Gavrila

n.m. 2011. Influence Factors for the Choice of a Software Development Methodology. *Accounting and Management Information Systems*.

Giaglis, G. M.

Travanj 2001. A Taxonomy of Business Process Modeling and Information Systems Modeling Techniques. *The International Journal of Flexible Manufacturing Systems*.

Goodrich, M. T. and R. Tamassia

Listopad 2001. *Algorithm Design: Foundations, Analysis, and Internet Examples*, 1 edition. Wiley.

Horowitz, E. and S. Sahni

Rujan 1984. *Fundamentals of computer algorithms*, 2 edition. Computer Science Press.

Howell, K. E.

Studeni 2012. *An Introduction to the Philosophy of Methodology*, 1 edition. SAGE Publications Ltd.

Žilinskas, A. and A. Zhigljavsky

Kolovoz 2004. Branch and probability bound methods in multi-objective optimization. *Optimization Letters*.

Kahn, A. B.

Studeni 1962. Topological sorting of large networks. *Communications of the ACM*.

Kann, V.

Svibanj 1992. *On the Approximability of NP-complete Optimization Problems*. PhD thesis, Royal Institute of Technology, Stockholm.

Karp, R. M.

Ožujak 1972. Reducibility among Combinatorial Problems. In *Proceedings of a symposium on the Complexity of Computer Computations*, R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, eds.

Kenyon-Mathieu, C. and W. Schudy

Lipanj 2007. How to rank with few errors. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, D. Johnson?, ed.

Knuth, D. E.

Ožujak 2011. *The Art of Computer Programming, Volumes 1-4A Boxed Set*, 1 edition. Addison-Wesley Professional.

Ko, R. K., S. S. Lee, and E. W. Lee

Prosinac 2008. Business process management (BPM) standards: a survey. *Business Process Management Journal*.

Kudelić, R.

Srpanj 2014. Javni razgovor: Automatizacija postupka određivanja redoslijeda razvoja podsustava informacijskog sustava. Službeni dokument Fakulteta organizacije i informatike, Varaždin.

Kudelić, R., M. Konecki, and A. Lovrenčić

Srpanj 2013. Multi-agent information system design and implementation: empirical analysis of IS subsystems execution and development order algorithm. *Journal of software*.

Kudelić, R., A. Lovrenčić, and M. Konecki

Ožujak 2012. Information system subsystems execution and development order algorithm implementation and analysis. *International Journal of Computer Science Issues*.

Kudelić, R., A. Lovrenčić, and M. Konecki

Srpanj 2011. Automatic Determination of Information System Subsystems Execution and Development Order. *International Review on Computers and Software*.

Laplante, P. A. and C. J. Neill

Veljača 2004. "The Demise of the Waterfall Model Is Imminent" and Other Urban Myths. *ACM Queue*.

Lawler, E.

Lipanj 1964. A Comment on Minimum Feedback Arc Sets. *Circuit Theory, IEEE Transactions on*.

Levitin, A.

Listopad 2011. *Introduction to the Design and Analysis of Algorithms*, 3 edition. Addison-Wesley.

Lovrenčić, A.
Lipanj 1997. The Problem of Optimization of the Process of Decomposition of an Information System. *Journal of Information and Organizational Sciences.*

Lovrenčić, A.
Ožujak 2008. Automatizacija postupaka u projektiranju informacijskih sustava. http://zprojekti.mzos.hr/page.aspx?pid=96&lid=1, učitano 4.8.2014.

Motwani, R. and P. Raghavan
Kolovoz 1995. *Randomized Algorithms*, 1 edition. Cambridge University Press.

Myers, G. J., T. Badgett, and C. Sandler
Studeni 2012. *The Art of Software Testing*, 3 edition. Wiley.

Pearce, D. J. and P. H. J. Kelly
Srpanj 2006. A Dynamic Topological Sort Algorithm for Directed Acyclic Graphs. *ACM Journal of Experimental Algorithmics.*

Poppendieck, M. and T. Poppendieck
Svibanj 2003. *Lean Software Development: An Agile Toolkit*, 1 edition. Addison-Wesley Professional.

Poras, A., R. K. Sharma, S. K. Gaur, and A. K. Bajpai
Prosinac 2011. Dynamic Modeling for Strategic Planning of a Small Foundry Using System Dynamics. In *Proceedings of the International Conference on Soft Computing for Problem Solving (SocProS 2011)*, K. Deep, A. Nagar, M. Pant, and J. C. Bansal, eds.

Prim, R. C.
Studeni 1957. Shortest connection networks and some generalizations. *Bell System Technical Journal.*

Russo, N. L.
Svibanj 1995. The use and adaptation of system development methodologies. http://www.andrews.edu/ vyhmeisr/papers/sdm.html, učitano 5.8.2014.

Saltz, J. H., R. Mirchandaney, and D. Baxter
Svibanj 1991. Run-time parallelization and scheduling of loops. *Computers, IEEE Transactions on.*

Satish, N. R., K. Ravindran, and K. Keutzer
Listopad 2008. Scheduling task dependence graphs with variable task execution times onto heterogeneous multiprocessors. In *Proceedings of the 8th ACM international conference on Embedded software*, L. de Alfaro? and J. Palsberg?, eds.

Simão, E. M.
Listopad 2009. Software Development Methodologies: Analysis and choice aid. Master's thesis, Universidade do Minho: Escola de Engenharia.

Skiena, S. S.
n.m. 2008. *The Algorithm Design Manual*, 2 edition. Springer.

Stapleton, J.
Srpanj 1999. DSDM: Dynamic Systems Development Method. In *Technology of Object-Oriented Languages and Systems, 1999. Proceedings of*, n.u., ed.

Stonedahl, F. and W. M. Rand
Studeni 2012. When Does Simulated Data Match Real Data? Comparing Model Calibration Functions Using Genetic Algorithms. *Robert H. Smith School Research Paper No. RHS-06-151.*

Sutherland, J., A. Viktorov, J. Blount, and N. Puntikov
Siječanj 2007. Distributed Scrum: Agile Project Management with Outsourced Development Teams. In *Proceedings of the 40th Hawaii International Conference on System Sciences*, J. Ralph H. Sprague, ed.

Tanenbaum, A. S.
n.m. 2009. *Modern Operating Systems*, 3 edition. Pearson Prentice Hall.

Tarjan, R.
Lipanj 1972. Depth-First Search and Linear Graph Algorithms. *SIAM J. COMPUT.*

Tarjan, R.
Lipanj 1976. Edge-disjoint spanning trees and depth-first search. *Acta Informatica.*

Žugaj, M., J. Šehanović, and M. Cingula
n.m. 2004. *Organizacija*, 2 edition. TIVA Tiskara Varaždin.

van der Aalst, W. M. P., A. H. M. ter Hofstede, and M. Weske
Lipanj 2003. Business Process Management: A Survey. In *Proceedings of the International Conference on Business Process Management*, W. M. P. van der Aalst and M. Weske, eds.

Winston, P. H.
Svibanj 1992. *Artificial Intelligence*, 3 edition. Addison-Wesley.

Wynekoop, J. L. and N. L. Russo
Listopad 2003. Studying system development methodologies: an examination of research methods. *Information Systems Journal.*

Yuan, H. L. and D. W. Wang

Siječanj 2012. Application of DSM Topological Sort Method in Business Process. In *Frontiers of Advanced Materials and Engineering Technology*, R. Chen, D. Sun, and W.-P. Sung, eds.

Zeller, A.

Lipanj 2009. *Why Programs Fail: A Guide to Systematic Debugging*, 2 edition. Morgan Kaufmann.

Zwillinger Daniel et al.

n.m. 2003. *CRC Standard Mathematical Tables and Formulae*, 31 edition. CHAPMAN & HALL/CRC.

Životopis

Robert Kudelić rođen je 10.7.1985., u Zagrebu. Osnovnu školu započeo je u Dugom Selu a završio u Puli u Osnovnoj školi Šijana. Srednjoškolsko obrazovanje smjera Elektrotehničar završio je u Obrtničkoj školi u Koprivnici. Diplomirao je na Fakultetu organizacije i informatike u Varaždinu sa temom: Primjena i analiza Windows Presentation Foundation tehnologije u desktop aplikacijama. Na istom Fakultetu zapošljava se 2009. godine kao znanstveni novak / asistent na Katedri za teorijske i primijenjene osnove informacijskih znanosti gdje sudjeluje u znanstvenom radu i izvođenju nastave na predmetima vezanim za programiranje. Znanstveni i stručni interes vezan mu je uz informacijske sustave, algoritme i programiranje.

Popis objavljenih djela

Radovi u časopisima

1. Robert Kudelić; Mladen Konecki; Alen Lovrenčić. Multi-agent information system design and implementation: empirical analysis of IS subsystems execution and development order algorithm. Journal of software, srpanj 2013.

2. Robert Kudelić; Alen Lovrenčić; Mladen Konecki. Information system subsystems execution and development order algorithm implementation and analysis. IJCSI International Journal of Computer Science Issues, ožujak 2012.

3. Kudelić, Robert; Lovrenčić, Alen; Konecki, Mladen. Automatic determination of information system subsystems execution and development order. International Review on Computers and Software, srpanj 2011.

Radovi na konferencijama

1. Schatten, Markus; Grd, Petra; Konecki, Mladen; Kudelić, Robert. Towards a Formal Conceptualization of Organizational Design Techniques for Large Scale Multi Agent Systems, Conference Proceedings of the 2nd International Conference on System-Integrated Intelligence: Challenges for Product and Production Engineering.

Thoben, Klaus-Dieter; Busse, Matthias; Denkena, Berend; Gausemeier, Jürgen. Srpanj 2014.

2. Robert Kudelić; Mirko Maleković; Alen Lovrenčić. Mind Map Generator Software, Proceedings 2012 IEEE International Conference on Computer Science and Automation Engineering. Shaozi Li; Yun Cheng; Ying Dai. Svibanj 2012.

3. Konecki, Mario; Lovrenčić, Alen; Kudelić, Robert. Making Programming Accessible to the Blinds, Proceedings of the 34th MIPRO International Convention on Computers in Technical Systems. Bogunović, Nikola; Ribarić, Slobodan. Svibanj 2011.

4. Konecki, Mladen; Cingula, Marijan; Kudelić, Robert. Integracija sigurnosti informacijskog i komunikacijskog sustava u korporativno upravljanje, VI. znanstvenostručna konferencija s međunarodnim sudjelovanjem "Menadžment i sigurnost" (M&S 2011). Taradi, Josip. Lipanj 2011.

5. Konecki, Mladen; Kudelić, Robert; Lovrenčić, Alen. Efficiency of lossless data compression, Proceedings of the 34th MIPRO International Convention on Computers in Technical Systems. Bogunović, Nikola; Ribarić, Slobodan. Svibanj 2011.

6. Kudelić, Robert; Konecki, Mladen; Maleković, Mirko. Mind Map Generator Software Model with Text Mining Algorithm, Proceedings of the 33rd International Conference on Information Technology Interfaces. Lužar-Stiffler, Vesna; Jarec, Iva; Bekić, Zoran. Lipanj 2011.

7. Konecki, Mario; Kudelić, Robert; Radošević, Danijel. Challenges of the blind programmers, Proceedings of the 21st Central European Conference on Information and Intelligent Systems. Boris Auer; Miroslav Bača; Markus Schatten. Rujan 2010.

8. Konecki, Mladen; Kudelić, Robert. Justifiability of open source software development, Proceedings of the 21st Central European Conference on Information and Intelligent Systems. Boris Auer; Miroslav Bača; Markus Schatten. Rujan 2010.

Stranica namjerno ostavljena prazna