UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Aleš Černivec

Porazdeljen algoritem za problem najmanjšega k-centra

MAGISTRSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Boštjan Slivnik

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št.: 135-MAG-RI/2010
Datum: 24. 11. 2010

Aleš ČERNIVEC, univ. dipl. inž. rač. in inf.

Ljubljana

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Porazdeljen algoritem za problem najmanjšega *k*-centra**

A distributed algorithm for the minimum *k*-center problem

Tematika naloge:
Zasnujte porazdeljeni algoritem za časovno in komunikacijsko kar se da učinkovito reševanje problema najmanjšega k-centra. Pri tem predpostavite, da mora biti algoritem prilagojen omrežjem enakovrednih subjektov. To pomeni, da mora algoritem omogočati dinamično vključevanje in izključevanje vozlišč v porazdeljeni sistem, poleg tega pa naj ob lokalni spremembi omrežja izračun nove rešitve na osnovi obstoječe zahteva le izračun na vozliščih v okolici spremembe.

Algoritem najprej preizkusite v simuliranem okolju in se prepričajte o učinkovitosti delovanja. Kakovost rešitev dobljenih z novim algoritmom primerjajte z rešitvami, ki jih dobimo z uporabo znanih klasičnih aproksimacijskih algoritmov. Aplikacijo preizkusite tudi v realnem okolju in določite zahtevnost realizacije algoritma v realnem okolju. Opišite razlike v simuliranem in realnem okolju ter predstavite glavne težave, ki lahko nastopijo v realnih porazdeljenih okoljih. Opišite osnovne prednosti in slabosti razvitega prototipa in podajte možnosti izboljšav.

Mentor:
B. Slivnik
doc. dr. Boštjan Slivnik

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme mag. dela s podpisom mentorja in dekana ter žigom fakultete, ki ga mag.kandidat dvigne v študentskem referatu, preden odda izdelek v vezavo!

Namesto te strani **vstavite** izpolnjeno izjavo o avtorstvu in soglasje, ki omogoča...

Zahvala

Za pozorno mentorstvo in pomoč se zahvaljujem mentorju doc. dr. Boštjanu Slivniku, ki me je usmerjal in vodil na področju porazdeljenih algoritmov skozi podiplomski študij. Za izredno pomoč in potrpežljivost se zahvaljujem tudi dr. Mateju Artaču, ki je spremljal moje magistrsko delo od vsega začetka in popravil nemalo napak ter prispeval marsikatero idejo. Za pomoč, svetovanje in motivacijo se zahvaljujem tudi dr. Gregorju Pipanu, dr. Danielu Vladušiču, mag. Jaku Močniku, Maticu Cankarju in preostalim v skupini podjetja XLAB d.o.o., ki so mi omogočili nemoteno raziskovanje ter izredno zanimivo in vzpodbudno delovno okolje.

Nenazadnje se zahvaljujem dekletu Lidiji Križnar za vso podporo, potrpežljivost in tople besede pri nastajanju tega dela ter seveda družini in prijateljem, ki so bili vedno tam takrat, ko sem jih potreboval.

Magistrsko delo delno financira Evropska unija, in sicer iz Evropskega socialnega sklada.

To delo posvečam družini in prijateljem.

Kazalo

Seznam uporabljenih kratic in simbolov

kratica	angleško	slovensko
API	application programming interface	aplikacijski programski vmesnik
AS	autonomous system	avtonomni sistem
BA	Barabasi-Albert	tip omrežja Barabasi-Albert
DHT	distributed hash table	porazdeljena zgoščevalna tabela
DNS	domain name server	strežnik domenskih imen
FIFO	first in first out	razvrščanje po principu prvi noter prvi ven
IP	internet protocol	internetni protokol
IPC	inter process communication	medprocesna komunikacija
MANET	mobile ad-hoc networks	mobilna ad-hoc omrežja
MPI, MPICH	message passing interface	protokol za paralelno računanje
MST	minimum spanning tree	minimalno vpeto drevo
P2P	peer to peer network	omrežje enakovrednih subjektov
QoS	quality of service	kvaliteta storitve
RPC	remote procedure call	klic oddaljenih rutin
RTT	round trip time	čas, potreben za povratek paketa
TCP	transfer control protocol	protokol kontrole prenosa
VM	virtual machine	navidezni računalnik
VNS	variable neighbourhood search	iskanje s spremenljivo soseščino
WAN	wide area network	omrežje velikega dosega

Povzetek

V pričujočem magistrskem delu predstavimo novi porazdeljeni algoritem nad omrežji enakovrednih subjektov, ki zna učinkovito rešiti problem najmanjši k-center. Poleg razvoja algoritma za reševanje omenjenega problema razmeščanja ponudnikov ter s tem **izboljšanja kvalitete storitev** (QoS), je bil naš namen tudi njegova realizacija na obstoječem porazdeljenem sistemu **PlanetLab**. Algoritem smo zasnovali tako, da z učinkovitim razmeščanjem iterativno izboljšuje kvaliteto streženja in se prilagaja spremembam v omrežju.

Pri razvoju algoritma smo uporabili simulator porazdeljenega omrežja **PeerSim** in lastno komunikacijsko ogrodje **DCenter** za porazdeljeno računanje. V simuliranem okolju smo analizirali rešitve novega algoritma nad omrežji v velikosti 1500 vozlišč in ugotovili, da pri določenih omejitvah kvaliteta rešitev algoritma sledi trendom obstoječih zaporednih 2-aproksimacijskih rešitev iz literature. Algoritem smo preizkusili nad statičnimi omrežji standardne knjižnice **OR-LIB** pri različnih vrednostih k. Izkazalo se je, da daje algoritem pri majhnih vrednostih parametra k celo boljše rezultate kot 2-aproksimativni **Gonzalezov algoritem**. Algoritem smo preizkusili nad dinamičnimi omrežji, kjer se je izkazalo, da je bolj stabilen in komunikacijsko manj zahteven tako pri odstranjevanju (od 20 do 50 odstotkov več prometa sporočil v omrežju) kot pri dodajanju vozlišč (do 2-krat več prometa sporočil v omrežju). Delovanje algoritma smo preizkusili v raziskovalnem omrežju PlanetLab, kjer smo merili čas izvajanja algoritma nad generičnimi omrežji do velikosti 40 vozlišč. Generična omrežja smo ustvarili z univerzalnim generatorjem omrežij **BRITE**, s katerim smo si pomagali tudi v primeru testiranja dinamičnih omrežij. Ugotovili smo, da vzdrževanje dobre razmestitve ponudnikov v realnem omrežju ni bilo časovno potratno, saj je časovno najbolj zahtevna operacija izgradnje prve rešitve za omrežja s 40 vozlišči potrebovala največ 160 sekund.

Z novim porazdeljenim algoritmom za izračun problema najmanjši k-center smo hevristično rešili omenjeni NP-poln problem. V delu pokažemo, da z novim algoritmom uporabnikom omogočimo učinkovitejši dostop do dostopnih točk storitev, s čimer izboljšamo uporabnikovo izkušnjo v omrežjih enakovrednih subjektov.

Ključne besede: porazdeljeni algoritem, najmanjši k-center, porazdeljeni sistem, ogrodje porazdeljenih algoritmov, omrežje enakovrednih subjektov

Abstract

In this thesis we present a distributed algorithm designed for peer-to-peer (P2P) networks that can efficiently solve the problem of minimum k-center. The algorithm iteratively improves the deployment of service access points and by that the quality of service (QoS) for the service clients. Algorithm autonomously adapts to changes in the network and converges towards better service deployment. By doing so, it improves the QoS for the clients in P2P networks. The main goals of our work was development of a distributed algorithm solving the facility location problem of minimum k-centers. The algorithm was applied to the problem of service deployment location in the PlanetLab globally distributed research network.

In the development of the algorithm, we used a simulator for distributed algorithms **PeerSim** and our own communication framework **DCenter**. We analyzed the solutions of the new distributed algorithm by simulating the algorithm in the networks of up to 1500 nodes. We found out that the quality of solutions to these problems follow the trends of existing serial 2-approximation algorithms solving the problem of the minimum k-center. We tested the algorithm on static networks provided by the standard OR-LIB library on different values of the parameter k. It was observed that in cases of small value of k the new distributed algorithm yielded better results than the serial 2-approximative **Gonzalez algorithm**. We tested the algorithm on dynamic networks where we noticed that removing the nodes from the network was not the most time consuming operation (20 to 50 percent more messages). Adding the nodes introduced up to 2-times more network traffic than removing the nodes. The implementation was tested in PlanetLab research network running the algorithm in generic networks of up to 40 nodes. Networks were created using the universal network generator **BRITE** which was also used in the case of dynamic networks. However, we observed that running the algorithm on Planetlab in networks of up to 40 nodes is not time expensive since the algorithm took at most 160 seconds to end.

Using the proposed distributed algorithm we have approximately solved the NP-complete minimum k-center problem. Moreover, we have shown that the usage of this algorithm would provide better user experience in the distributed environment in P2P networks by improving on the QoS in the network.

Keywords: distributed algorithm, minimum k-center, distributed system, distributed algorithms framework, peer-to-peer networks

Poglavje 1

Uvod

1.1 Motivacija

Danes lahko vsak uporabnik interneta postane del večjih zasebnih ali javnih omrežij (Grid5000, PlanetLab), kjer mu dostop do skoraj neomejenih računskih virov onemogoča le cena oz. pripadnost neki organizaciji. Računski viri so s svojo povezljivostjo tako na lokalni ravni kot na ravni svetovnega spleta omogočili razmah aplikacij in storitev, ki delujejo porazdeljeno in iz vnaprej neznanih fizičnih lokacij. Poleg tega nastajajo nove tehnologije, ki povezujejo že znane rešitve v nove dimenzije računstva in povezujejo mobilna omrežja, delovne postaje in računska središča v homogeno, uporabniku transparentno, celoto s praktično neomejenimi računskimi viri (porazdeljen operacijski sistem XtreemOS, http://xtreemos.eu/)[41].

Poleg novih tehnologij, ki z neverjetno hitrostjo prodirajo in povezujejo namizno računstvo v t.i. računanje v oblakih, so še vedno popularna omrežja enakovrednih subjektov za izmenjavo datotek (nestrukturirano omrežje Gnutella, BitTorrent, nestrukturirani super-vozliščni sistem Kazaa). V sistemih za izmenjavo datotek uporabniki preko določenih indeksnih strežnikov dobijo sezname uporabnikov z želeno vsebino. V vseh omenjenih novejših sistemih pa velikokrat pozabljamo na kakovost storitve (angl. *quality of service*). V splošnem je to sicer ohlapen pojem, vendar ga je moč v nekaterih pogledih številsko oceniti [30]. Za doseganje ustrezne stopnje kakovosti storitve lahko sistem prilagajamo med njegovim delovanjem ali poskrbimo za rabo postopkov, ki nam vsaj v teoretičnem smislu omogočajo optimalno izrabo sistema. Eden od zelo pomembnih vidikov kakovosti storitve je odzivnost storitve v porazdeljenem sistemu [13]. Ta je med drugim odvisna od hitrosti dostopa odjemalca do strežnika, na katero vpliva oddaljenost povezave med odjemal-

cem in storitvijo, pasovna širina povezav in obremenitev omrežja. S tem je povezana tudi splošna izraba omrežja, kjer želimo, da bodo hitrejše povezave bolje izkoriščene.

V zadnjih letih se je pojavilo kar nekaj člankov, kjer so se avtorji lotili reševanja problema lokacije ponudnikov v dinamičnem porazdeljenem okolju in mobilnih omrežjih ter problema zagotavljanja kakovosti storitve v omrežjih [9, 32, 33, 44, 48, 52]. V omenjenih delih so avtorji ali reševali problem lokacije ponudnikov v bolj specifičnih okoljih ali pa so reševali spremenjen problem. V svojem delu smo se lotili osnovnega problema NAJMANJŠI k-CENTER nad splošnimi grafi, za katerega smo razvili porazdeljen algoritem.

1.2 Cilji in namen dela

V svojem delu smo raziskali glavne pristope k reševanju problemov razmeščanja s porazdeljenimi algoritmi in razvili nov pristop k reševanju specifičnega problema razmeščanja. Algoritem smo tudi predstavili in analizirali. Zadali smo si naslednje cilje:

- Pregled področja problema razmeščanja v porazdeljenih in mobilnih omrežjih.
- Predstavitev problema razmeščanja.
- Predstavitev različnih pristopov reševanja problema razmeščanja s porazdeljenimi algoritmi.
- Predstavitev reševanja problema razmeščanja z novim porazdeljenim algoritmom.
- Primerjava pristopov reševanja problemov razmeščanja na porazdeljen način.
- Predstavitev rezultatov našega pristopa in primerjava rezultatov reševanja s pristopi iz literature.
- Testiranje porazdeljenega algoritma v realnem porazdeljenem okolju.

1.3 Arhitektura računalniških sistemov in omrežno računanje

Med leti 1960 in 1994 je veljal samo en model povezljivosti v internetu [55]. Računalniki, ki so bili del interneta, so imeli dodeljene naslove IP. Šele s hitrim razvojem namiznih računalnikov je prišlo do potrebe po povezljivosti osebnih računalnikov v internet in tako je nastal drug model povezljivosti v omrežje — klicni dostop. S tem je internet postalo dinamično in nepredvidljivo okolje, kjer viri dinamično vstopajo in izstopajo. Z bojaznijo internetnih ponudnikov, da bo zmanjkalo naslovnega prostora za vse naprave, ki želijo biti del interneta, so uporabili model dinamičnega dodeljevanja naslovov. Ker pa so domače naprave postajale iz dneva v dan močnejše, so se uporabniki začeli zavedati, da so njihovi viri brez enostavne povezljivosti in brez možnosti deljenja virov v veliki meri neizkoriščene. Krepila se je ideja o deljenju uporabniških virov in tako je postalo področje raziskav omrežij enakovrednih subjektov v raziskovalni srenji zelo pomembno. Razvilo se je več tipov arhitektur porazdeljenih sistemov, ki določajo relacijo med komponentami (aplikacije, strežniki, odjemalci) takega sistema. Izbor tipa arhitekture vpliva na storilnost (angl. *performance*), zanesljivost in varnost mrežnega sistema. V računalniških sistemih poznamo naslednje tipe arhitektur:

- **strogo zaporedne sisteme** — sistemi SISD (angl. *single instruction single data*),
- **paralelni sistemi** — sistemi SIMD (angl. *single instruction multiple data*) in MIMD (angl. *multiple instruction multiple data*) [31].
- **Porazdeljene sisteme** — omrežja odjemalec-strežnik (angl. *client-server*), omrežja enakovrednih subjektov (angl. *peer-to-peer networks*), ...

Strogo zaporedni sistemi so sistemi z enim procesorjem in pomnilnikom, ki tok ukazov izvaja strogo zaporedno. **Paralelni sistemi** so kompleksnejši, ki imajo lahko eno ali več procesorskih enot s skupnim (tesno sklopljeni sistemi) in deljenim pomnilnikom. Sistemi z deljenim pomnilnikom (skupki računalnikov) se najpogosteje uporabljajo za procesorsko intenzivne aplikacije. Taki sistemi so lahko povezani z namenskimi povezovalnimi mrežami in uporabljajo namenske komunikacijske protokole za prenašanje sporočil (npr. MPI, MPICH). **Porazdeljeni sistemi** so bolj rahlo sklopljeni in navadno povezani preko lokalnega omrežja (angl. *local area network*) ali interneta, kjer si izmenjujejo sporočila. Porazdeljene sisteme naprej delimo glede na **način komunikacije**

med viri, ki se lahko vrši kot **asimetrična**, odjemalec-strežnik (angl. *client-server*) komunikacija ali kot **simetrična** komunikacija. Slednji tip omrežij imenujemo tudi **omrežja enakovrednih subjektov** (angl. *peer-to-peer networks*, P2P). V omrežjih enakovrednih subjektov so subjekti enakovredni in lahko nastopajo v vlogi strežnika in odjemalca hkrati. Vloge se v takih sistemih lahko s časom menjajo in so odvisne od vrste sporočila med subjektoma. Podajmo natančno definicijo obeh tipov subjektov — odjemalca in strežnika:

Definicija 1.3.1. Odjemalec *je subjekt (vozlišče, program, modul), ki lahko izdaja povpraševanje, ne more pa streči povpraševanjem.*

Definicija 1.3.2. Strežnik *je subjekt (vozlišče, program, modul), ki streže poizvedbam odjemalcev.*

Kljub temu, da so se raziskave na področju porazdeljenih sistemov in omrežij enakovrednih subjektov zelo razmahnile, še vedno ne poznamo natančne definicije sistema enakovrednih subjektov [55]. Za potrebe postopkov, ki so opisani v tem delu, bomo zato uporabljali naslednjo definicijo:

Definicija 1.3.3. *Omrežje enakovrednih subjektov (angl. peer to peer, P2P) je omrežje decentraliziranih subjektov, ki so popolnoma avtonomni in enakopravni.*

V definiciji 1.3.3 smo uporabili pojma **avtonomni** in **enakopravni**. Pod pojmom avtonomni imamo v mislih samostojno delovanje mimo drugih strežnikov, npr. strežnikov DNS (angl. *domain name server*). Sistem mora biti sposoben zagotoviti zanesljiv naslov subjekta in pri tem upoštevati dinamičnost. Enakopravnost v takih sistemih zagotavlja enakost vseh subjektov v smislu, da vsak lahko nastopi kot strežnik ali odjemalec in da je komunikacija med vozlišči simetrična.

V **mrežnem računstvu** (angl. *grid computing*) gre za koordinirano deljenje virov in reševanje problemov v omrežju s podporo velikih **virtualnih organizacij**, katerih meje se lahko raztezajo preko več organizacij. Virtualna organizacija je dinamična skupina posameznikov ali organizacij, ki lahko izkoriščajo skupne deljene vire s pomočjo pravil in pogojev za deljenje. Virtualne organizacije so lahko različnih velikosti, domen in časovnih okvirov. Z mrežnim računstvom omogočimo deljenje visoko zmogljivih virov (superračunalnikov, računskih skupkov).

Začasna omrežja (angl. *ad-hoc networks*) so omrežja, v katerih nastopajo subjekti s sposobnostjo komunikacije brez obstoječe komunikacijske infrastrukture (razen že obstoječih subjektov, npr. računalnikov z brezžični komunikacijskimi vmesniki). Tako omrežje zagotavlja komunikacijo, poimenovanje

(angl. *naming*) in varnost s pomočjo subjektov v omrežju. Taka omrežja so npr. mobilna začasna omrežja (angl. *mobile ad-hoc networks*) in omrežja tipal (angl. *sensor networks*).

V svojem modelu bomo predpostavili model porazdeljenega omrežja, kjer subjekti lahko igrajo vlogo odjemalca in strežnika hkrati — tako omrežje imenujemo **omrežje enakovrednih subjektov**. Preostali načini komunikacije in tipi virov v omrežju namreč predstavljajo posplošitev sestavin osnovnega modela, ki pa presega začrtani okvir magistrskega dela. V nadaljevanju bomo zato natančneje razdelali omrežja enakovrednih subjektov in si podrobneje pogledali različne arhitekture omrežij enakovrednih subjektov.

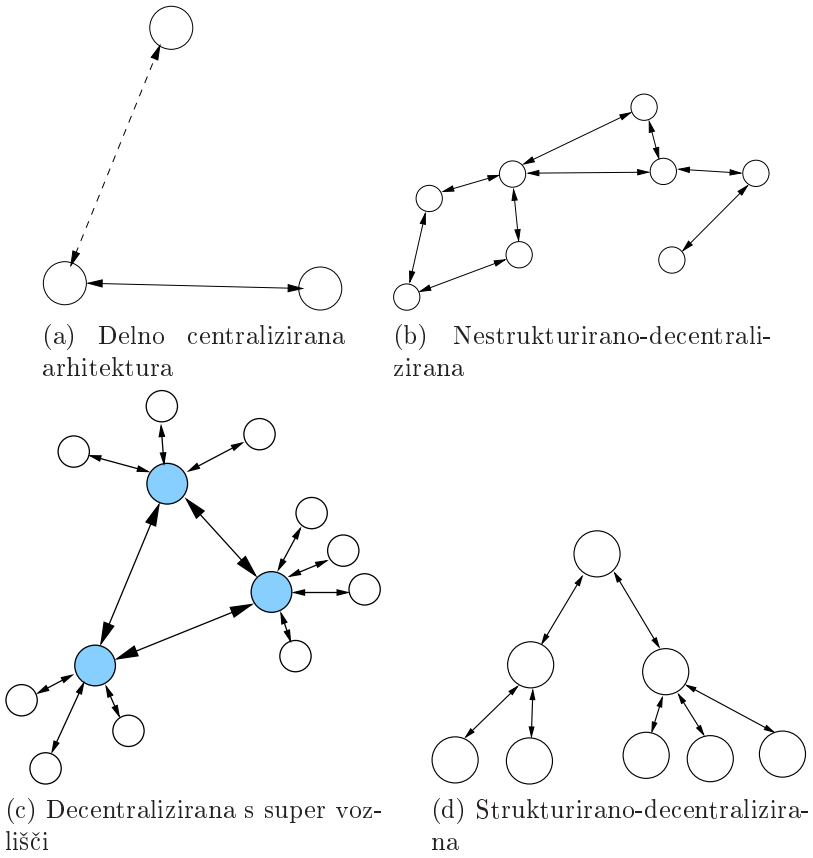
1.3.1 Arhitektura omrežij enakovrednih subjektov

V omrežjih enakovrednih subjektov poznamo več tipov arhitektur [3]. Razlikujemo jih glede na:

- način hranjenja informacije, ki je potrebna za dosego želenega cilja uporabnika (npr. najti datoteko v omrežju, najti pravo število ustreznih virov, idr.): **centralizirana** in **decentralizirana** arhitektura,
- način povezave med vozlišči (vozlišča v omrežju povezujemo in vzdržujemo po nekem pravilu): **strukturirana** in **nestrukturirana** arhitektura.

Na sliki 1.1 so prikazani štirje glavni tipi arhitektur omrežij enakovrednih subjektov: delno centralizirana, nestrukturirano-decentralizirana, decentralizirana omrežja s super vozlišči in strukturirano-decentralizirana omrežja .

Delno centralizirana arhitektura (slika 1.1a) ima majhno število namenskih strežnikov, ki so posredniki komunikacije med posameznimi odjemalci. Ti namenski strežniki prevzamejo nalogo indeksiranja virov in razpoložljivih vozlišč. Tovrstna omrežja imenujemo tudi **hibridna**, saj vsebujejo značilnosti tako arhitekture odjemalec-strežnik, kakor tudi arhitekture omrežij enakovrednih subjektov. Podobnost z arhitekturo odjemalec-strežnik je očitna, saj se v omrežju nahaja majhen delež vozlišč, ki niso odjemalci in imajo specifičen namen streženja zahtev iskanja in beleženja. Po drugi strani pa v omrežju omogočajo sodelovanje in izkoriščanje virov ostalih odjemalcev, kar delno centralizirano arhitekturo šteje tudi med omrežja enakovrednih vozlišč. Pozitivna plat teh arhitektur je intuitivnost uporabe (primerni so za strežbo storitev) in preprostost izvedbe. Seveda to prinaša negativno plat, ki je odločilna, da se ta arhitektura umika sodobnejšim konceptom: neodpornost na napake in



(a) Delno centralizirana arhitektura

(b) Nestrukturirano-decentralizirana

(c) Decentralizirana s super vozlišči

(d) Strukturirano-decentralizirana

Slika 1.1: Tipi arhitektur omrežij enakovrednih subjektov

napade ter (ne)skalabilnost. Skalabilni sistemi delujejo zanesljivo in učinkovito kljub spreminjanju velikosti omrežja. Nekaj primerov izvedb tega tipa omrežij: SETI@Home, BOINC, Napster in OpenNap.

Nestrukturirano-decentralizirana omrežja (slika 1.1b) so omrežja enakovrednih subjektov, kjer so posamezni subjekti omrežja povezani naključno brez kakega reda. Vsako vozlišče v omrežju je odgovorno za zagotavljanje virov, usmerjanje mrežnega prometa in vzdrževanje omrežnih povezav. Za razliko od delno centralizirane arhitekture je zaradi enakovrednosti vozlišč komunikacija simetrična, kar pomeni, da so v povprečju povezave enakomerno izkoriščene. Ta omrežja imajo zmožnost gostovanja skalabilnih in na napake odpornih sistemov, vendar ti lastnosti nista samo po sebi umevni, ker moramo zanje poskrbeti s pametno rabo virov in načrtovanjem. Največja slabost sistemov takih omrežij je želja po **hkratnem** zanesljivem zagotavljanju **točk dostopa do storitev** in **skalabilnosti** sistema. Skalabilnost se namreč zago-

tovi z omejevanjem radija komunikacije med vozlišči (t.i. iskalni horizont), kar pa je obratno sorazmerno z iskanjem in odkrivanjem objektov v omrežju. Iskalni horizont se določi z uvedbo vrednosti **časa življenja** (angl. *time to live*) sporočila. Vsako vozlišče, ki posreduje sporočilo svojim sosedom, zmanjša to vrednost. Ko vrednost pade na 0, se sporočilo neha širiti. Zato lahko vozlišča komunicirajo le z vozlišči svoje bližnje okolice, kar onemogoča zanesljivo zagotavljanje oz. odkrivanje točk dostopa. Poleg tega lahko zaradi enakovredne obravnave vseh vozlišč pride do slabe izkoriščenosti močnejših in zasičenosti šibkejših vozlišč. Nekaj primerov izvedb tega tipa: Gnutella (protokol verzije 0.4), GPU in Freenet.

V **decentraliziranih omrežjih s super vozlišči** (slika 1.1c) niso vsa vozlišča enakovredna. V teh omrežjih delimo vozlišča na **super vozlišča** in **liste**. Super vozlišča premorejo večjo kapaciteto prejetih in posredovanih sporočil v porazdeljenem sistemu kot listi. Poleg tega super vozlišča gostijo storitve za indeksiranje virov v omrežju in s tem razbremenijo vozlišča-liste za iskanje virov. Kljub temu, da omogočajo zanesljivejše točke dostopa kot decentraliziran pristop, še vedno ne premorejo zanesljivega iskanja objektov v omrežju. Kljub izboljšavam sporočila še vedno vsebujejo vrednost časa življenja in imajo večji doseg kot v nestrukturirano-decentraliziranih omrežjih. Poleg tega se v decentraliziranih omrežjih s super vozlišči izboljša skalabilnost, a le na račun asimetričnosti komunikacije med vozlišči. Asimetričnost povzroči neodpornost na napade in zmanjšanje odpornosti na napake. Primeri izvedb tega tipa omrežij so: Gnutella (različica 0.6), Kazaa, SG-1 in SUPS.

Strukturirano-decentralizirana omrežja so prikazana na sliki 1.1d. Ta tip omrežij je primeren za iskanje virov in za usmerjanje mrežnih sporočil. Za ta omrežja je značilno prekrivno omrežje, ki določa strukturo povezanosti med vozlišči. Struktura je v veliko primerih teh omrežij izvedena s pomočjo **porazdeljenih zgoščevalnih tabel** (angl. *distributed hash table*, DHT). Podatkovna struktura DHT ima enako funkcionalnost kot navadna zgoščevalna tabela, le da je tu edinstven ključ prirejen virom v omrežju. Vozlišča se v takem omrežju povezujejo z vozlišči na podlagi ključ-vrednosti in s tem ohranjajo strukturo omrežja. Poleg tega so popolnoma enakovredna, podobno kot v decentraliziranih omrežjih. Strukturirana omrežja so primerna za izvedbo iskanja virov v porazdeljenem omrežju v primeru iskanja vrednosti po ključu (že poznamo naslov IP iskanega računalnika). Iskanje ključa, ki vsebuje želeno vrednost, pa je nekoliko težje in za to potrebujemo dodatne mehanizme (npr. porazdeljene strežnike, ki označujejo imena datotek z vrednostmi naslovov IP virov). Primeri izvedb decentraliziranih strukturiranih omrežij so: Chord, CAN, Pastry in Tapestry.

Vsaka od predstavljenih arhitektur omrežij enakovrednih subjektov ima pozitivno in negativno plat in nobena ne predstavlja celostne rešitve, ki je potrebna za učinkovito in zanesljivo delovanje storitve in iskanje storitve s strani odjemalcev v omrežju. Nestrukturirana omrežja in omrežja super-vozlišč ponujajo učinkovito iskanje storitve, medtem ko strukturirana omrežja ponujajo zanesljivo podporo objavljanja in hranjenja vrednosti v porazdeljenem sistemu. Zato so potrebni večslojni pristopi, kjer se uporabljajo določeni elementi iz vsake arhitekture kot samostojna plast prekrivnega omrežja.

1.3.2 Problemi v omrežjih enakovrednih subjektov

Znotraj vsakega tipa omrežij enakovrednih subjektov se pojavljajo podobni problemi [42], ki se jih je potrebno lotiti izbranemu okolju primerno:

- usmerjanje (angl. *routing*),
- objavljanje (angl. *publishing*),
- iskanje virov (angl. *discovery*),
- povpraševanje (angl. *querying*).

Usmerjanje je izbor najboljše poti, po kateri paket potuje od izvornega do ponornega vozlišča. Usmerjanje je primarna naloga mrežne plasti. Osnovno usmerjanje paketov mrežne plasti v omrežjih P2P nadgradimo s prekrivnim omrežjem in iskalnim algoritmom, ki je zadolžen za usmerjanje paketov med vozlišči (vsako vozlišče vzdržuje usmerjevalno tabelo). Tako se mrežni paketi usmerjajo le med sosednjimi vozlišči v prekrivnem omrežju. **Objavljanje** in **iskanje virov** se rešuje s pomočjo prekrivnega omrežja. Tako so realizirane npr. porazdeljene zgoščevalne tabele (Apache Cassandra, CAN, Chord, Pastry ...), ki s pomočjo algoritmov nad prekrivnimi omrežji razdelijo prostor ključev virov med prisotna vozlišča. Ta vozlišča se povezujejo le z delom (npr. $O(\log(n))$) drugih vozlišč omrežja. Vsa vozlišča tvorijo strukturo tabele in kljub odpovedi dela vozlišč struktura še vedno ostane. Težava je pri ohranjanju informacije na odstranjenih vozliščih, česar brez podvajanja ne moremo zagotoviti. Omenjene naloge v porazdeljenem okolju opravljajo namenske storitve za razširjanje (angl. *dissemination*) in združevanje (angl. *aggregation*) podatkov.

Raziskave na področju omrežij enakovrednih subjektov vključujejo iskanje rešitev naslednjih zelo pomembnih problemov: **skalabilnost** (angl. *scalability*), **zanesljivost** in **povezljivost** (angl. *interoperability*). **Skalabilnost** je

lastnost sistema, ki nam pove, do kakšne mere se bo sistem obnašal predvidljivo in zanesljivo ob upoštevanju vključevanja novih subjektov v sistem. Sistemi, ki so centralno usmerjeni, npr. sistem odjemalec-strežnik, se pod veliki bremeni (veliko uporabnikov želi dostopati do določene internetne strani) običajno ne obnesejo najbolje brez dodatnih mehanizmov za preusmerjanje prometa ali podvajanja strežnikov — centralni sistemi imajo slabo skalabilnost. Po drugi strani imajo nestrukturirana omrežja najboljšo skalabilnost, saj so zaradi enakovrednosti vozlišč medsebojno neodvisna. Kljub izpadu ali prihodu novih vozlišč so zaradi omejenega horizonta taki sistemi precej robustni (kar neposredno vodi k zanesljivosti). Pojem **zanesljivost** je namreč tesno povezan s pojmom skalabilnosti, saj so skalabilni sistemi zanesljivejši od neskalabilnih. Pojem **povezljivosti** pa nam pove, na kakšen način obstoječi sistem izpostavlja svoj vmesnik storitvam oz. drugim sistemom **navzven**. Pomembno je namreč, da sistem omogoča zunanjim uporabnikom, ki niso del prekrivnega omrežja, povpraševanje oz. objavljanje. Problem se rešuje s pomočjo posrednikov (angl. *proxy*) [7].

Znano je, da se v omrežjih enakovrednih subjektov najbolje obnesejo delno centralizirana omrežja s super vozlišči [3]. Pojem super vozlišč nas napeljuje na izbor vozlišč, ki bi bila najprimernejša za postavitev centrov in gradnjo strukturiranega omrežja iz teh centrov. Vsak center bi imel pripadajoči segment omrežja in s tem določen horizont iskanja virov v omrežju. Posamezne centre lahko povežemo z novim prekrivnim omrežjem in s tem ustvarimo hierarhični sistem iskanja, nad katerim lahko realiziramo poljubno iskalno metodo v strukturiranih omrežjih enakovrednih subjektov. Za problem iskanja centrov izberemo model problema NAJMANJŠI k-CENTER, ki spada med probleme razmeščanja. V nadaljevanju predstavimo področje razmeščanja ponudnikov v obstoječi literaturi.

1.4 Opis splošnih problemov razmeščanja ponudnikov

Pri razmeščanju v problemu POKRITJE MNOŽICE gre za **pokrivanje** porabnikov tako, da so lahko servisirani v vnaprej podanem času. Problem NAJVEČJE POKRITJE je v splošnem definiran z utežmi na vozliščih: cilj je postaviti omejeno število ponudnikov in maksimizirati števila pokritih porabnikov (njihovih zahtev). Druga dva problema (k-MEDIANA in NAJMANJŠI k-CENTER) pa se od problemov pokrivanja razlikujeta v ciljni funkciji. Pri prvem gre za minimizacijo povprečne razdalje porabnikov do razmeščenih ponudnikov, pri drugem

pa za minimizacijo največje razdalje od kateregakoli porabnika do njemu prirejenega ponudnika.

Slika 1.2: Opredelitev problemov razmeščanja

Ciljna funkcija problema k-RAZPRŠITEV je v razdalji med lociranimi ponudniki v omrežju. Poiskati je potrebno tâko postavitev ponudnikov, da je razdalja med vsakim parom ponudnikov največja možna.

Cilj optimizacije problema STALNA OBREMENITEV [43] je cena postavitve ponudnikov na neki podmnožici vozlišč v omrežju (natančna definicija se nahaja v [39]). V tem problemu za razliko od problemov pokritja iščemo podmnožico lokacij za postavitev strežnikov, tako da pokrijemo množico odjemalcev kar najceneje.

Za razumevanje problema ZVEZDIŠČE [45] moramo najprej spoznati pomen **zvezdišča** (angl. *hub*). To je centralna naprava, ki preklaplja povezave med vozlišči v omrežju. Omrežja z zvezdišči si lahko predstavljamo kot dvonivojska omrežja: komunikacija med odjemalcem in strežnikom poteka preko zvezdišč (najmanj eno zvezdišče je posrednik), zvezdišča pa lahko komunicirajo med drugimi zvezdišči ali navadnimi vozlišči. Cilj je poiskati podmnožico lokacij zvezdišč tako, da minimizirajo pretok v omrežju.

Za razliko od zgornjih modelov, ki težijo k postavitvi ponudnikov kar najbližje porabnikom, je rešitev problema NAJVEČJA VSOTA taka postavitev strežnikov, da je vsota uteženih razdalj med ponudniki in njim prirejenimi porabniki največja. Primer takega problema je razmeščanje nezaželenih ponudnikov (zapori, tovarne, zbirališča odpadkov).

1.5 Modeli problemov razmeščanja ponudnikov

Glede na okolje problema pokrivanja, na katerega se nanaša razmestitveni model, poznamo **zvezne**, **diskretne** in **omrežne** modele. Pri prvem modelu

predstavlja problemski prostor d-dimenzionalen prostor $\mathbb{R}^d$, kjer se v praksi največ uporablja evklidska ravnina (torej $d = 2$). Pri diskretnih modelih gre za določeno število točk in diskretno okolje, ki je podano z matriko razdalj med točkami, medtem ko omrežni model modeliramo s povezanim grafom in s cenami povezav med točkami grafa (podrobnejše opise modelov najdemo v [36]).

V tem delu se osredotočimo na omrežni model. Status centra dodelimo vozliščem, ki imajo največje cene povezav do vseh ostalih vozlišč v neki sprejemljivi toleranci. Za problem 1-CENTER je to najmanjša vrednost največjih cen povezav centra do vseh ostalih vozlišč v grafu, ki jih vrne algoritem najcenejših poti med vsemi pari vozlišč. Če za ceno med vozlišči izberemo kar razdaljo v zveznem prostoru, moramo poiskati k gruč v podanem omrežju tako, da je največji pokrivni radij gruče najmanjši.

Primer problema NAJMANJŠI k-CENTER iz vsakdanjega življenja je razmestitev gasilskih domov ali bolnišnic po mestih nekega področja, da bodo kolikor se da najbolje pokrivali celotno področje. Pomenov besede **najbolje** je lahko več, saj kot mero lahko uporabimo npr. čas trajanja intervencije, razdaljo med razmeščenimi domovi ali pa ceno take razmestitve gasilskih domov. Tako, kakor za vsak osnovni problem razmeščanja, poznamo tudi za problem NAJMANJŠI k-CENTER več različic problema. Naštejmo nekaj možnih različic:

- **vozliščni**, kjer so potencialna mesta za centre na vozliščih,
- **absolutni**, kjer centrom lahko priredimo mesta poljubno na povezavah med vozlišči (s tem bi povezavo, na katero postavimo center, prekinili in na mesto prekinjene povezave dodali nov center),
- **problem NAJMANJŠI k-CENTER s trikotniško neenakostjo med vozlišči**, kjer pri razdaljah med vozlišči upoštevamo trikotniško neenakost.

Povezave v omrežju G so lahko utežene, zato poznamo tudi problem UTEŽENI k-CENTER. V uteženem problemu v nasprotju od neuteženega obravnavamo posamezna vozlišča drugače (npr. povezava med dvema vozliščema je slaba, zato je primerno utežena).

Za formalnejšo predstavitev problemov razmeščanja sta avtorja v delu [23] predlagala **klasifikacijsko shemo** za modele problemov razmeščanja. Predlagala sta strukturo peterk, kjer vsak element peterke predstavlja določeno sestavino v modelu. Splošna struktura je tako $P_1/P_2/P_3/P_4/P_5$. Na kratko predstavimo še pomen posamezne sestavine:

- P_1 število in tip novih ponudnikov,
- P_2 tip modela lokacije (prostor možnih rešitev), npr. razlikovanje med zveznim, omrežnim in diskretnim modelom lokacije,
- P_3 opis dodatnih posebnosti modela lokacije, npr. posebne omejitve (kapacitete), možne rešitve idr.,
- P_4 relacije med obstoječimi in novimi ponudniki (funkcije razdalje ali cene),
- P_5 opis ciljne funkcije.

S to shemo formalno opišemo problem, ki smo se ga lotili reševati v svojem delu kot

$$k/\mathscr{G}/w_m/d(\mathscr{V}, \mathscr{V})/\max$$

Ta opis pomeni, da problem rešujemo z iskanjem k centrov v splošnem neusmerjenem grafu kot neutežen problem lokacije ponudnikov, kjer je funkcija razdalje kar razdalja med vozlišči v grafu. Ciljna funkcija (mera) je največja razdalja med vozliščem in njegovem najbližjem centru.

1.6 Matematično predznanje

V tem poglavju seznanimo bralca z osnovnimi matematičnimi konstrukti, ki smo jih uporabili v svojem delu.

1.6.1 Približno reševanje problemov

Problem NAJMANJŠI k-CENTER spada med NP-težke optimizacijske probleme, kar nam da zelo dober razlog, da se usmerimo na iskanje približnih rešitev. Za reševanje NP-težkih problemov ne moremo računati na uporabo optimalnega algoritma, ker po vsej verjetnosti le-ta ne obstaja. Obstajajo pa algoritmi za približno reševanje NP-težkih problemov, ki zagotavljajo rešitev do neke točnosti natančno. Taki algoritmi so **aproksimacijski**. Pravimo, da je algoritem A aproksimacijski za problem P, če za vsak izbor vhodnih parametrov problema (naloga problema) $x \in I$, kjer je I množica vseh nalog problema P, vrne neko dopustno rešitev $A(x)$ [50]. Z $m(x, A(x))$ označimo vrednost približne rešitve

in z $m^*(x)$ vrednost optimalne rešitve. Primerjajmo ti dve vrednosti s **performančnim kvocientom**. Performančni kvocient rešitve $y = A(x)$ naloge x je

$$R(x,\, y) = \max\{\ \tfrac{m(x,y)}{m^*(x)},\ \tfrac{m^*(x)}{m(x,y)}\ \} \tag{1.1}$$

V našem primeru reševanja problema gre za minimizacijski problem, zato velja $R(x,\, y) = m(x,\, y)/m^*(x)$. Velja tudi $1 \leq R(x,\, y)$. Pravimo, da je algoritem A **r-aproksimacijski** algoritem za problem P, če obstaja konstanta $r \geq 1$, da je $R(x,\, A(x)) \leq r$ za vsako nalogo $x \in I$. Ta mera nam določa **stopnjo aproksimativnosti**.

1.6.2 Drevo najkrajših poti

Usmerjevalni algoritmi morajo pogosto poznati najkrajše poti med pari vozlišč (najkrajše poti med vsemi pari vozlišč ali najkrajše poti med vozliščem do vseh preostalim vozlišči). To podatkovno strukturo bomo poimenovali kar **drevo najkrajših poti** [10, 56] (ponorno drevo, angl. *sink tree*).

Izrek 1.6.1. *[56] Za vsako vozlišče r v M obstaja drevo $T_r = (V, E_r)$, ki je sestavljeno iz optimalnih poti $p \in P$ od r do kateregakoli vozlišča $v \in V$. Vsako pot iz P sestavljajo povezave $e \in E_r$, kjer $E_r \subset E$.*

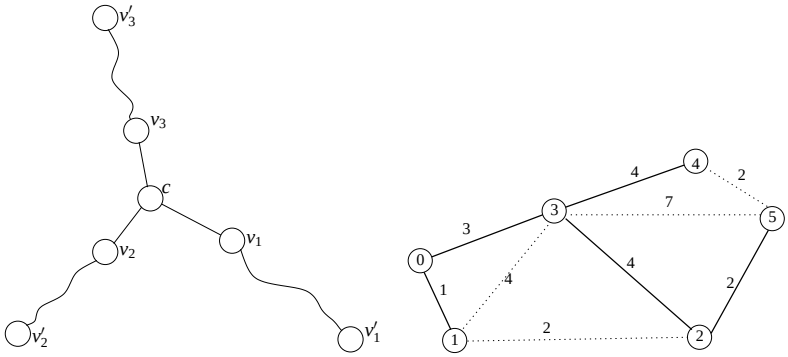
Dokaz: Induktivno moramo konstruirati zaporedje dreves $T_i = (V_i, E_i)$, $i = 0, \ldots, N$, pri čemer mora imeti vsako drevo T_i naslednje lastnosti:

1. Vsako drevo T_i je poddrevo omrežja G.
2. Vsako drevo T_i je za vsak $i < N$ poddrevo drevesa T_{i+1}.
3. Za vsak $i > 0$ velja $v_i \in V_i$ in $r \in V_i$.
4. Za vsako vozlišče $v \in V_i$ je enostavna pot od v do r v T_i optimalna pot v omrežju G.

Konstrukcija se prične s korenskim vozliščem $V_0 = r$ in s prazno množico povezav $E_0 = \varnothing$. Za vozlišče $v_{i+1} \notin V_i$ poiščemo optimalno pot od v_{i+1} do r: $P = \langle u_0,\, \ldots,\, u_k \rangle$, tako da je $u_l \in V_i$ in l najmanjši možni. Sledi

$$V_{i+1} = V_i \cup \{u_j : j < l\}\ \wedge\ E_{i+1} = E_i \cup \{(u_j,\, u_{j+1}) : j < l\}$$

Da je T_i poddrevo novonastalega drevesa T_{i+1}, je preprosto preveriti, saj je drevesu T_{i+1} dodano samo eno novo vozlišče in s tem nova povezava. Z



(a) Primer gruče (b) Primer ponornega drevesa

Slika 1.3: Leva slika prikazuje gručo s perspektive centra. Desna slika prikazuje omrežje z optimalnim ponornim drevesom vozlišča 3.

matematično indukcijo lahko dokažemo, da je T_{i+1} drevo (število povezav je za 1 manj kot število vozlišč).

Sedaj je potrebno še pokazati, da za vse $w \in V_{i+1}$ obstaja enolična optimalna pot od w do r v G. Za vozlišča $w \in V_i$ to velja, saj $V_i \subset V_{i+1}$ in je pot od w do r optimalna. Če je $w = u_j \in V_{i+1} \setminus V_i$ in $j < l$, pa razdelimo pot $\langle u_j, \ldots, r\rangle$ na dva dela: $P_1 = \langle u_0, \ldots, u_l\rangle$ in $P_2 = \langle u_l, \ldots, u_k\rangle$. Ne pozabimo, da je optimalna pot enaka $P = \langle u_0 = u_j, \ldots, u_k = r\rangle$. Recimo, da poznamo pot $Q = \langle u_l, \ldots, r\rangle$ in trdimo, da je optimalna. Cena $C(Q)$ mora biti enaka $C(P_2)$. Recimo, da ni, torej $C(Q) < C(P_2)$. Ta trditev je kontradiktorna, saj bi cena združene poti P_1 in Q bila manjša od P, torej je P_2 optimalna. Podobno sklepamo za pot P_1. Recimo, da poznamo pot $R = \langle u_j, \ldots, r\rangle$, ki ima manjšo ceno od poti $\langle u_j, \ldots, u_l\rangle$ združeno z P_2. Iz tega sledi, da mora biti $C(R)$ manjša od cene poti $\langle u_j, \ldots, u_k\rangle$, ki pa je del P. Tu pa zopet pridemo do nasprotja, saj je v tem primeru $\langle u_0, \ldots, u_j\rangle$ združeno z R optimalna. Iz tega sledi, da je P optimalna. □

Lema 1.6.2. *[56] Vpeto drevo, ki ima koren v vozlišču r, se imenuje ponorno drevo za r. Optimalno ponorno drevo je ponorno drevo z lastnostmi iz teorema 1.6.1.*

Naj bo N_c množica sosednjih vozlišč korenskega vozlišča c in naj velja $|N_c| = g$. Iz korenskega vozlišča c v drevesu najkrajših poti lahko pridemo preko sosednjega vozlišča $v \in N_c$ v končno korakih do lista v'. Listi v drevesu najkrajših poti so najbolj oddaljena vozlišča od vozlišča c (slika 3.1).

Označimo $v_\alpha \rightarrow v_\beta$, kjer $\rightarrow$ označuje, da je v_β (takojšnji) naslednik v_α. Naj bo $\xrightarrow{*}$ refleksivna in tranzitivna relacija. Relacija $v_\alpha \xrightarrow{*} v_\omega$ pomeni, da

je v_ω naslednik v_α. Na sliki 1.3a lahko vidimo primer $g = 3$ in $v_1 \xrightarrow{*} v_{1'}$, $v_2 \xrightarrow{*} v_{2'}$, $v_3 \xrightarrow{*} v_{3'}$. S širjenjem informacije v_1, v_2 in v_3 navzgor proti vozlišču c lahko omogočimo izračun najkrajših poti med pari vozlišč iz množice $G = \{v_1, v_2, v_3, c\} \bigcup G_{others}$ v korenskem vozlišču c. V G_{others} so lahko tudi drugi nasledniki v_i, vendar niso pomembni za premik centra. Premike centra bomo spoznali v poglavju 3.

Poglavje 2

Definicija problema NAJMANJŠI k-CENTER

V tem poglavju predstavimo nekaj osnovnih problemov razmeščanja. Ker je za naše delo zanimiv problem NAJMANJŠI k-CENTER, si ga bomo natančneje ogledali in podali njegovo matematično definicijo. Spoznali bomo tudi nekaj pristopov k njegovemu reševanju.

Probleme razmeščanja v splošnem razdelimo na osem osnovnih problemov [15]: POKRITJE MNOŽICE, NAJVEČJE POKRITJE, NAJMANJŠI k-CENTER, k-MEDIANA, k-RAZPRŠITEV (angl. *p-dispersion*), STALNA OBREMENITEV (angl. *fixed charge*), ZVEZDIŠČE (angl. *hub*) in NAJVEČJA VSOTA (angl. *maksi-sum*)

V splošnem je problem razmeščanja problem prostorske dodelitve virov [6], kjer razmeščeni strežniki servisirajo zahteve odjemalcem, razpršenim po nekem prostoru. V praksi najsplošneje delimo probleme lokacije glede na ciljni gospodarski sektor, kjer rešujemo probleme: področja v **zasebnem** (lokacija tovarn, trgovin, načrtovanje komunikacijskih omrežij) in **javnem sektorju** (iskanje primernih lokacij za postavitev oz. razmestitev vozil za nujne vožnje, razmestitev javnih centrov, načrtovanje javnih omrežij). Drugi zanimivi problemi razmeščanja, ki jih rešujemo v praksi, vključujejo še načrtovanje omrežij, iskanje najprimernejših postavitev skladišč, razmestitev gasilskih domov po nekem širšem področju, določitev primernih lokacij za postavitev konkurenčne trgovine po nekem območju, kjer že obstaja konkurenca.

2.1 Problem NAJMANJŠI k-CENTER

Odločitvena inačica optimizacijskega problema NAJMANJŠI k-CENTER je podan v [18] kot problem ND50. V literaturi se zanj pojavlja tudi ime p-center. Cilj pri reševanju problema NAJMANJŠI k-CENTER je v določitvi k vozlišč v omrežju kot centre tako, da je razdalja med navadnimi vozlišči in vozlišču najbližjem centru v omrežju najmanjša. Obstaja več inačic osnovnega problema. Problem je lahko **vozliščni**, če za centre lahko izbiramo vozlišča v omrežju. Druga možnost je **absolutni** problem, kjer se center lahko pojavi tudi na povezavah med vozlišči v omrežju tako, da povezavo razdelimo na dva dela in vsako izmed robnih vozlišč prvotne povezave povežemo z novim centrom. Delitev lahko nadaljujemo na podlagi uteži na vozliščih omrežja: **utežena** inačica problema nastopi, ko moramo pri izračunu vsako povezavo med centrom in vozliščem pomnižiti še z utežjo vozlišča omrežja. V literaturi se pogosteje obravnava **neutežena** inačica problema, kjer upoštevamo le razdalje povezav. Praktični primer absolutnega uteženega problema k-center je lokacija postavitve strežnika, ki bi zadostil vsem odjemalcem v omrežju. Vsak odjemalec v takem omrežju ima lahko različno težo v smislu pomembnosti oz. veličine (stopnje) odjemanja storitve.

Lažje si bomo pomagali s formalno definicijo problema. Problem lahko definiramo s pomočjo linearnega programiranja [15] ali pa s krajšim matematičnim zapisom [18]. Slednjega smo izbrali tudi v pričujočem delu.

2.1.1 Definicija

Formalno problem opišemo tako:

Definicija 2.1.1. *[11] Problem* k-CENTER.
VHOD*: poln graf* $G = \langle V, E\rangle$*, totalna funkcija* $d\colon E \longrightarrow \mathbb{Z}_0^+$*, in celo število* $k \in \{1 \ldots n\}$.
CILJ*: poiskati množico* $C \subseteq V$*, kjer* $|C| = k$*, tako da*

$$\max_{v\in V} \min_{c\in C} d(v,c) \leq \max_{v\in V} \min_{c\in C'} d(v,c)$$

za vsako podmnožico $C' \subseteq V$*, kjer velja* $|C'| = k$.
MERA*:* $\max_{v\in V} \min_{c\in C} d(v,c)$.

Funkcija $d(x, y)$ v zgornji definiciji problema označuje razdaljo med vozliščema x in y (oz. utež na povezavi). Definicija 2.1.1 pravi, da moramo za

natančno rešitev problema NAJMANJŠI k-CENTER poiskati tako razmestitev k centrov C, da za vsako podmnožico C' moči k množice V velja naslednje: največja najkrajša razdalja med kateremkoli odjemalcem in centrom v C mora biti manjša ali enaka od največje najkrajše razdalje med kateremkoli odjemalcem in centrom v C'. Opazimo, da gre za optimizacijski problem, saj iščemo tako razmestitev centrskih vozlišč, da je cenilna funkcija $\max_{v \in V} \min_{c \in C} d(v, c)$ najmanjša.

2.1.2 Težavnost reševanja problema NAJMANJŠI k-CENTER

Obstaja tudi malce spremenjena definicija splošnega, vozliščnega problema k-CENTER, NAJMANJŠI k-CENTER. Razlika med tem problemom in problemom z definicijo 2.1.1 je v številu k. Pri problemu NAJMANJŠI k-CENTER moramo namreč poiskati k ali manj lokacij za centre, medtem ko pri našem problemu iščemo natanko k centrov. Problem k-CENTER je NP-poln [1, 18]. Če razdalje med vozlišči ne zadostijo trikotniški neenakosti, potem problem niti ni aproksimabilen [24]. Za probleme, kjer je velja trikotniška neenakost, poznamo kar nekaj 2-aproksimativnih rešitev [19, 25, 38].

Za fiksno vrednost parametra k je problem k-CENTER rešljiv v času $O(n^k)$, ker moramo za optimalno rešitev pregledati vse možne množice razmestitev centrov, teh pa je $\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k!} \leq O(n^k)$. Če k nastopa kot spremenljivka in želimo optimizirati tudi po tem parametru (iščemo najmanjši k), potem je problem NP-težek [29].

2.2 Problemi razmeščanja

Pregled več kot 50 problemov razmeščanja je opisanih v delu [6], vključno z najbolj znanimi: problemom median, najmanjšim k-centrom in s standardnim problemom razmeščanja ponudnikov. Delo [6] vsebuje tudi natančno taksonomijo problemov razmeščanja in opis nekaj pristopov k reševanju opisanih problemov.

Za problem NAJMANJŠEGA k-CENTRA obstajajo zaporedni 2-aproksimacijski algoritmi [19, 25]. Gonzalez [19] je prvi objavil 2-aproksimativni požrešni algoritem za izračun rešitev tega problema. Njegova metoda temelji na iterativni izbiri centrov. Algoritem prične s prazno množico centrov. Za tem dodaja od trenutne množice centrov najbolj oddaljena vozlišča v množico cen-

trov. Končno rešitev predstavlja vseh $k-1$ izbranih centrov, medtem ko prvega lahko izberemo na več različnih pristopov [36]. Izbira prvega vozlišča kot centra ne vpliva veliko na kakovost dobljene rešitve (prihaja do majhnih variacij) [37].

Hochbaum and Shmoys [25] sta reševala problem NAJMANJŠI k-CENTER na drugačen način, in sicer s pragovno metodo , kjer sta rešila problem v času $O(|E| \log |E|)$.

Ravninsko (evklidsko) različico problema NAJMANJŠI k-CENTER je moč rešiti v času natančno $O(n^{O(\sqrt{k})})$ [26]. Ta problem je tudi eden redkih, kjer za NP-poln problem obstaja sub-eksponentna odvisnost od glavnega vhodnega podatka. Veliko dela je bilo posvečenega raziskovanju problema NAJMANJŠI k-CENTER za probleme z majhnim parametrom k. Za problem ravninskega problema 2-center obstajajo skoraj linearne rešitve v času. Tako je v delu [51] avtor pokazal, da za ravninski problem 2-center obstaja skorajšnja časovno linearna rešitev v ravnini, in sicer s časovno zahtevnostjo $O(n \log^9 n)$. S tem je izboljšal do takrat znano najboljšo rešitev $O(n^2 \log n)$. V delu [16] je ta rezultat celo izboljšan na $O(n \log^2 n)$. Za diskretni problem 2-center je do sedaj najhitrejši algoritem opisan v delu [2], in sicer s časovno zahtevnost $O(n^{4/3} \log^5 n)$.

Problem NAJMANJŠI k-CENTER je možno rešiti tudi s pomočjo triangulacije oz. z dualnim problemom triangulacije, tj. z Voronoijevimi diagrami [53], kjer vlogo centrov privzamejo kar centri Voronoijevih diagramov.

Zanimiv je tudi pristop z metahevristikami , t.i. z iskanjem s tabuji in tehniko iskanja s spremenljivo soseščino (angl. *variable neighbourhood search*, VNS) [40].

Reševanja problema NAJMANJŠI k-CENTER se lahko lotimo tudi z algoritmom za reševanje problema DOMINANTNE MNOŽICE [36]. Slednji je eden izmed boljših hevrističnih požrešnih pristopov reševanja problema NAJMANJŠI k-CENTER in daje zelo dobre rezultate [49]. Pristop k reševanju tega problema je uporaba rešitve problema POKRITJA MNOŽIC (oz. serije takih problemov): nastavimo mejo za radij in preverimo, ali pokrijemo vse uporabnike, ki so na dosegu meje z vsemi k centri. Če jih dosežemo, zmanjšamo mejo, sicer pa povečamo mejo in ponovimo postopek. V [49] najdemo tudi pregled nekaterih hevrističnih zaporednih pristopov k reševanju problema NAJMANJŠI k-CENTER.

V [21] smo zasledili reševanje metričnega problema NAJMANJŠI k-CENTER z lokalnim iskalnim algoritmom. V tem delu so avtorji vpeljali posplošitev problemov k-MEDIANA, k-MEANS in k-CENTER s pomočjo l_p norme in predstavili $O(p)$ aproksimacijski algoritem za reševanje omenjenih problemov. Problem

k-MEANS je zanimiv na področju statistike in strojnega učenja. Gre za iskanje k gruč med n elementi, kjer vsak element pripada gruči z najbližjim povprečjem. V tem delu so avtorji podali tudi dokaze, da obstajajo aproksimacijski algoritmi, ki temeljijo na izbrani tehniki, za splošni problem LOKACIJE PONUDNIKOV.

V literaturi s področja porazdeljenih algoritmov najdemo veliko pristopov k reševanju problemov razmeščanja, vendar gre večinoma za nekoliko spremenjene modele prvotne rešitve. V delu [32] tako najdemo opis porazdeljenega algoritma (v delu je imenovan kot lokalni algoritem) nad senzornimi omrežji. V omenjenem delu je avtor reševal spremenjen problem razmeščanja ponudnikov, imenovan problem k-PONUDNIKOV. V tem delu ni eksplicitne ocene zahtevnosti reševanja (niti stopnje aproksimacije), je pa pokazana neodvisnost velikosti omrežja in števila sporočil za izračun lokacije ponudnikov.

Obstaja porazdeljena inačica Gonzalezovega algoritma [9], ki pa ima eno veliko slabost — v dinamičnem omrežju se ta rešitev lahko izrodi v poljubno slabo rešitev, poleg tega pa ne zagotavlja odpornosti na napake.

Tudi v delu [17] obstaja podoben opis algoritma za reševanje problema lokacije ponudnikov v senzornih omrežjih na porazdeljen način. V tem delu je opisana predelava zaporednega algoritma v porazdeljen algoritem. Delo opisuje preprost zaporeden 1,61-aproksimacijski algoritem, ki reši metrični problem lokacije ponudnikov in preoblikovanje zaporednega algoritma v porazdeljen, zaporednemu algoritmu ekvivalenten 1,61-aproksimacijski algoritem problema lokacije. Predstavljene so tudi spremembe, ki so potrebne za izvajanje algoritma v **omrežjih z več koraki usmerjanja** (angl. *multi-hop networks*). V takšnih omrežjih vedno uporabimo najmanjše razdalje med posameznimi vozlišči, enako kot bi jih dobili z algoritmom računanja najkrajših poti med vsemi pari vozlišč. Opisani algoritem tudi ne potrebuje eksplicitne sinhronizacije med posameznimi procesi — vsebuje namreč implicitno sinhronizacijo med posameznimi sosednjimi vozlišči, medtem ko čakajo na dohodna sporočila.

Poznamo tudi pristop reševanja problema s pomočjo mobilnih agentov v omrežju enakovrednih subjektov [52]. Ta pristop uporablja pogostost obiskanosti vozlišč mobilnih agentov kot hevristiko za izbor primernih lokacij za postavitev centrov. Vozlišča z večjo stopnjo imajo tako večjo verjetnost, da bodo izvoljene za postavitev centra. Pomanjkljivost tega dela je predvsem v pomanjkanju testiranja nad večjimi instancami problema — metodo smo realizirali v simulatorju porazdeljenih algoritmov, izkazala pa se je kot zelo slaba rešitev.

S pomočjo porazdeljene sheme za združevanje vozlišč v gruče [48] lahko samodejno združujemo vozlišča v skupke, kjer si pomagamo s simulacije pro-

meta v porazdeljenem sistemu.

Zanimiv pristop je uporabljen v [33], kjer so avtorji realizirali porazdeljen algoritem za razvrščanje ponudnikov brez omejitev kapacitet problema k-MEDIANA. Algoritem temelji na poznavanju trenutnega stanja okolice centrov, s pomočjo katerega trenutni center naredi premik na lokacijo znotraj tega področja. Slednja je bila izračunana z že znanim centraliziranim algoritmom za reševanje problema 1-CENTER.

V delu [57] lahko najdemo pregled smernic raziskav na področju razmeščanja storitev v mobilnih ad-hoc omrežjih (t.i. omrežjih MANET). V tem delu je opisanih 10 pristopov k reševanju osnovnega problema razmeščanja ponudnikov. Porazdeljen pristop k računanju primernih lokacij problema k-MEDIAN je podan tudi v [44], kjer so predstavljene različne politike premikov centrov na primerne lokacije. Politike ohranjajo monotono padajočo vrednost ciljne funkcije centrov, ki pod določenimi pogoji lahko doseže celo optimalno vrednost.

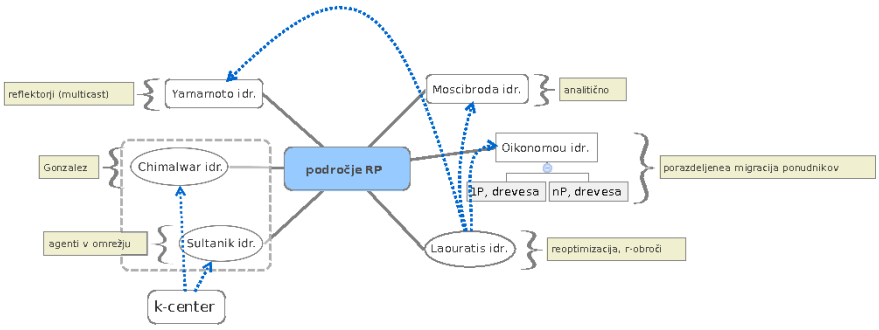
Zanimivo je, da je tematika porazdeljenih algoritmov na področju razmeščanja ponudnikov v zadnjih letih postala zelo aktivna, vendar ni zaslediti porazdeljenega algoritma, ki bi reševal problem NAJMANJŠI k-CENTER. Malo člankov je, ki raziskujejo izključno ta problem [9, 52].

2.3 Obstoječi pristopi reševanja problemov lokacije ponudnikov s porazdeljenimi algoritmi

V tem poglavju predstavimo tri pristope reševanja problema razmeščanja ponudnikov v omrežjih enakovrednih subjektov. V prvem pristopu opišemo porazdeljeno različico Gonzalezovega algoritma [19]. V nadaljevanju opišemo tudi pristop reoptimizacije z uporabo centraliziranega algoritma za reševanje problemov razmeščanja brez omejitev kapacitet. Predstavimo tudi shemo za tvorbo gruč s pomočjo simulacije prometa v omrežju in nenazadnje tudi pristop z uporabo mobilnih agentov v omrežju enakovrednih subjektov. Slika 2.1 prikazuje povezave med posameznimi pristopi (elipse na sliki predstavljajo v nadaljevanju opisane rešitve).

2.3.1 Porazdeljeni Gonzalezov algoritem

Avtorji so v delu [9] opisali pristop približnega reševanja problema NAJMANJŠI k-CENTER s pomočjo porazdeljenega algoritma. Asinhroni porazdeljeni algoritem so poimenovali ALGORITEM KC. Algoritem uporablja enako hevristiko



Slika 2.1: Literatura na temo porazdeljenih algoritmov za reševanje problemov razmeščanja.

za izbiro novih centrov kot Gonzalezov algoritem — izbor najbolj oddaljenega vozlišča od trenutne množice centrov za novi center.

Algoritem sestavljata dve fazi: faza iskanja prvega centra in faza iskanja preostalih $k-1$ centrov. Prva faza po Gonzalezu naključno izbere prvi center, medtem ko v algoritmu ALGORITEM KC v prvi fazi za center izberemo vozlišče z največjo stopnjo povezav v omrežju. V tem algoritmu so avtorji privzeli, da ima vsako vozlišče usmerjevalno tabelo, ki vozlišču poda informacijo o številu korakov prenosa sporočila do kateregakoli vozlišča v omrežju. Vsaka komunikacija med odjemalci se vrši preko centrskih vozlišč: če odjemalec pošilja sporočilo odjemalcu v isti gruči, se sporočilo usmeri preko skupnega centra. V primeru, ko poteka komunikacija med odjemalci v različnih gručah, se sporočilo usmerja preko centrov.

Po izboru vozlišča c_1 za center vozlišče c_1 izbere novo vozlišče c_2, ki bo predstavljalo naslednji center. S tem algoritem preide v drugo fazo delovanja, fazo izbora preostalih centrov. Konec druge faze je z izborom k-tega centra. Ob vsakem izboru centra nov center razpošlje sporočila vsem preostalim vozliščem na način **razpršenega oddajanja** (angl. *broadcasting*). Ob sprejemu sporočila centra odjemalec izračuna najbližji center in informacijo o njem razpošlje vsem preostalim centrom.

Prednost opisanega pristopa je v enostavnosti algoritma. Slabosti algoritma so predvsem v obravnavi napak, kjer algoritem obravnava dva primera:

- dodajanje, napake in premiki vozlišč-odjemalcev,
- napake in premiki centrov.

Odjemalci se namreč lahko dodajajo šele po tem, ko so centrska vozlišča

že izbrana. Ob vstopu novega vozlišča le-ta poplavi celotno omrežje s sporočili centrom, nakar se po prejemu k sporočil priključi najbližji gruči in s tem izbere center. Težava je tudi pri napaki na vozlišču — napako tu enačimo s prenehanjem delovanja vozlišča. Ko center ugotovi, da je izgubil eno vozlišče, razpošlje iskalna sporočila vsem preostalim centrom. Če ugotovi, da obstaja iskano vozlišče drugje (vozlišče se je premaknilo), posodobi usmerjevalne tabele.

Napake na centrih se v tem algoritmu odkrivajo s strani odjemalcev, ko odjemalec α_s pošilja sporočilo odjemalcu v gruči vozlišča α_s ali pa vozlišču, ki se nahaja v drugi gruči. V vsakem primeru se sporočilo usmerja preko centra z napako in komunikacijski protokol obvesti α_s (v primeru napake na centru znotraj iste gruče) oz. njegov center (v primeru napake na centru izven gruče vozlišča α_s) o napaki. V primeru napake na centru znotraj iste gruče se s pomočjo poplavljanja sporočil in izborom najbližjega centra α_s poveže na nov center, kjer se zopet izvrši druga faza — iskanje k-tega centra. Če je pokvarjen center (preko katerega bi se moralo usmeriti poslano sporočilo) v drugi gruči, center odjemalca α_s sproži drugi korak.

2.3.2 Porazdeljeni algoritem za razvrščanje ponudnikov brez omejitev kapacitet problema k-MEDIANA

V delu [33] avtorji predstavijo problem za reševanje problema k-MEDIANA (število centrov je konstantno) in osnovnega problema razmeščanja ponudnikov (optimizacija števila centrov v omrežju). Njihov pristop omejuje informacijo o topologiji in porabi odjemalcev na določen radij okrog centrov (ponudnika). S postopno reoptimizacijo se centri premikajo po omrežju in tako se najde razmestitev centrov, ki karseda dobro reši problem. Ta pristop je zelo podoben pristopu, ki ga opisujemo v pričujočem delu.

Poimenujmo algoritem iz dela [33] kar DFL (angl. *distributed facility location*). Algoritem ob pričetku naključno izbere začetno množico centrov $F^{(0)}$ moči k. V iteraciji m se izbere center $c_i \in F$, ki izvede naslednje korake:

1. s pomočjo protokola odkrivanja pridobi informacijo o topologiji r-okolice (gruča z radijem r),
2. zlivanje sosednjih gruč,
3. reoptimizacija gruče s pomočjo centraliziranega algoritma (reševanje problema z algoritmom za problem k-MEDIANA oz. osnovnega problema razmeščanja,

4. brisanje centrov iz prejšnje množice $F^{(m-1)}$ in dodajanje centrov v novonastalo množico centrov $F^{(m)}$,

5. test konvergence (če je nova množica centrov $F^{(m)}$ enaka $F^{(m-1)}$, algoritem zaključi z izvajanjem.

V delu manjka natančni opis algoritma DFL in je predstavljena le ideja, zato je realizacija algoritma DFL in primerjava z našim algoritmom težavna. Avtorji so opravili majhno število testov na omrežjih ER (Erdos-Renyi) in BA (Barabasi-Albert) [35] stopnje 2. Prednost opisanega algoritma DFL je v optimizaciji števila ponudnikov in v nekaterih primerih upoštevanje bremena odjemalcev v omrežju.

2.3.3 Reševanje problema razmeščanja s pomočjo mobilnih agentov v omrežju enakovrednih subjektov

Pristop je opisan v delu [52]. Algoritem poimenujmo KCAGENT. Kot hevristiko za izbor primernih lokacij KCAGENT uporablja frekvenco obiskanosti vozlišč mobilnih agentov. Vozlišča z večjo stopnjo imajo tako večjo verjetnost, da bodo izbrane za postavitev centrov. Postopek iskanja centrov je naslednji:

1. naključna razmestitev naključno premakljivih sprehajalcev - agentov po omrežju,

2. vsako vozlišče (odjemalec) hrani frekvenco obiskov agentov,

3. vozlišče s statusom **center** za hevristiko premikov uporabi zabeleženo frekvenco na vozliščih (premika se, dokler ne najde največjo vrednost),

4. v primeru prihoda na lokalni maksimum center opravi naključni sprehod v iskanju novega lokalnega ekstrema.

Pomanjkljivost tega dela je predvsem v pomanjkanju testiranja nad večjimi instancami problema — metodo smo realizirali v simulatorju porazdeljenih algoritmov, izkazala pa se je kot zelo nestabilna rešitev za reševanje problema NAJMANJŠI k-CENTER.

Poglavje 3

Algoritem za porazdeljeno reševanje problema NAJMANJŠI k-CENTER

Za reševanje problema NAJMANJŠI k-CENTER smo implementirali porazdeljeni algoritem, ki smo ga poimenovali DKCENTER. V tem poglavju podrobno opišemo zasnovani algoritem in ga analiziramo. Najprej opišemo pojem porazdeljenega reševanja problema in predstavimo **plast vzdrževanja**, ki je kot sestavina našega porazdeljenega algoritma nujna pri obveščanju o napakah tipa **odpoved-ustavitev** (angl. *fail-stop*). Predstavimo tudi pojem **konteksta centra**, ki se uporablja pri razširanju informacije po drevesu najkrajših poti in zagotavlja pravilni izračun in dodelitev odjemalcev centrom znotraj gruč centrov. Za lažje razumevanje algoritma bomo najprej predstavili algoritem za reševanje problema 1-CENTER s pristopom, ki smo ga kasneje uporabili pri algoritmu za izračun problema NAJMANJŠI k-CENTER - s pristopom **premika centrov**.

3.1 Porazdeljeno reševanje problemov

Pri uporabi pojma **porazdeljeni sistem** si lahko predstavljamo sistem kot domeno avtonomnih enot, nad katerimi porazdelimo določeno količino informacije. Vsaka taka enota je sposobna komunicirati z ostalimi samostojnimi enotami te domene (lahko le s podmnožico) in se avtonomno odločati na podlagi prejete informacije sosednjih enot v domeni. Enote imajo nadzor nad lastnim (lokalnim) tokom izvajanja, nahajajo pa se na vozliščih usmerjenega omrežnega grafa. Pojem enote v vozlišču omrežnega grafa lahko utemeljeno

zamenjamo s pojmom procesa, saj je proces najmanjša samostojna instanca računalniškega programa, ki je sposobna izvajanja ukazov in komunikacije z ostalimi procesi (na istem ali oddaljenem sistemu v omrežju). Tako komunikacijo imenujemo tudi **medprocesna komunikacija** (angl. *inter-process communication*, IPC).

Vrste problemov, ki se pojavijo na porazdeljenih sistemih, so različnega značaja, njihove rešitve pa so zanimive za širok spekter strokovnjakov na različnih področjih. Vrsta porazdeljenega algoritma, ki ga obravnavamo, je odvisna predvsem od parametrov izbranega modela porazdeljenega okolja:

- način medprocesne komunikacije,
- časovni model,
- model napak,
- vrsta problema nad porazdeljenim sistemom.

Način medprocesne komunikacije Porazdeljeni algoritmi tečejo na skupini procesorjev, ki morajo med seboj komunicirati. Splošne metode komunikacije so: (eno-vozliščno) oddajanje (angl. *single node broadcast*) ali prenos iz točke v točko (angl. *point-to-point*), dostop do skupnega pomnilnika in klic oddaljenih procedur (angl. *remote procedure call*). Kot bomo spoznali kasneje, so koraki procesorja zgrajeni iz osnovnih korakov, zato na tem mestu lahko omenimo atomičnost koraka *prejmi/pošlji* ali razdeljenost koraka *prejmi/pošlji* pri medprocesni komunikaciji. V prvem primeru lahko procesor oddaja v enem koraku **enaka** sporočila vsem ali izbranim procesorjem (oddajanje). V drugem primeru lahko v osnovnem koraku procesor oddaja sporočilo le enemu procesorju. Temu rečemo prenos točka v točko. V našem modelu bomo v enem koraku prenašali enako sporočilo izbrani množici sosedom po načinu oddajanja.

Komunikacija med procesi v omrežju danes najpogosteje poteka preko protokola TCP/IP [47]. Protokol TCP (angl. *Transfer Control Protocol*) omogoča zanesljiv prenos paketov in zagotavlja njihovo prejemanje v pravilnem vrstnem redu - prvi noter prvi ven (FIFO). Protokol TCP omogoča upravljanje z več povezavami TCP istočasno, omogoča mehanizem za preverjanje podatkov (odprava napak ali zahteve po ponovnem pošiljanju poškodovanih paketov), omogoča pa tudi proženje dogodkov TCP (**Connect**, **Send**, **Receive**, **Disconnect**). Na tem mestu se ne bomo spuščali v podrobnosti protokola TCP, kar presega začrtani obseg tega dela.

Časovni model Časovni model označuje način delovanja procesa, komunikacijskega kanala in način komunikacije po njem. Splošno lahko določimo tri parametre, ki nam določajo lastnosti časovnega modela porazdeljenega sistema [34]:

1. sinhronost ali asinhronost procesa,
2. sinhronost ali asinhronost komunikacijskega kanala,
3. sinhronost ali asinhronost vrstnega reda sporočil v komunikacijskem kanalu.

Procese imenujemo **sinhroni**, ko se stanje vsakega izmed njih v sistemu spremeni v istem trenutku. V sistemu s sinhronimi procesi obstaja globalni čas, ki je za vse procese enak. Večja stopnja sinhronosti procesov je **sinhronost komunikacije**, kjer je vsak proces p_0 po pošiljanju sporočila prejemniku p_1 v stanju čakanja dokler p_0 ne dobi odgovora od p_1. Pri asinhroni komunikaciji gre za prenos podatkov brez zunanje ure. V takem prenosu so hitrosti prenosov med vozlišči lahko različne. Poleg tega sinhronizacija ur oddajnika in prejemnika sporočila ni nujna. **Komunikacijski kanal** je **asinhron** v primeru, ko se po njem vrši asinhrona komunikacija. V primeru, ko se vrstni red sporočila v komunikacijskem kanalu ohranja (vrstni red FIFO), pravimo, da gre za **sinhronost vrstnega reda sporočil** v komunikacijskem kanalu.

Model napak Strojno opremo, na kateri teče porazdeljeni algoritem, lahko privzamemo kot popolnoma zanesljivo ali pa dopustimo napake (napačno delovanje procesorja ali celo njegova ustavitev). Poleg napak procesorja poznamo tudi napake na komunikacijskem kanalu. Napake delimo na **napake izpuščanja** (angl. *omission failures*), **naključne napake** (angl. *arbitrary failures*) in **časovne napake** (angl. *timing failures*). Pri **napakah izpuščanja** proces ali komunikacijski kanal preprosto ne izvrši akcije, ki bi jo moral izvršiti. Napaka izpuščanja procesorja se imenuje tudi **odpoved-ustavitev** (angl. *fail-stop*) procesorja. Zaznavanje teh napak je v sinhronem sistemu enostavno (časovna omejitev poteče), medtem ko je za asinhrone procese zelo težavno oz. nemogoče, saj v splošnem v takem sistemu časovni potek ni mogoče nastaviti na neko zgornjo mejo [34]. Napake izpuščanja na komunikacijskem kanalu lahko še naprej delimo na napake izpuščanja pri pošiljanju in napake izuščanja pri prejemu sporočil (dodatna razlaga teh ni potrebna, saj imena dovolj jasno razložijo delitev). **Naključne napake** ali **bizantinske napake** (angl. *byzantine failures*) so vrsta napak, ki z neko verjetnostjo generirajo poljubna sporočila

ali celo poljubna stanja procesorja. Do napak pa lahko pride ne samo na procesorjih, ampak tudi na komunikacijskem kanalu (izguba sporočil ali podvojena sporočila). Detekcija naključnih napak na procesorjih je nemogoča, ker je nemogoče ugotoviti, ali je proces naredil akcijo na podlagi prejetega sporočila ali ne. Naključne napake na komunikacijskem kanalu so redke, ker za podvajanje sporočil in ugotavljanje izgube sporočila obstajajo protokoli (npr. TCP), za detekcijo pokvarjenih sporočil pa se uporablja **kontrolna vsota** (angl. *checksum*). Tretji tip napak so **časovne napake**, kjer lahko prihaja do napak zaradi preobremenitve procesorja ali komunikacijskega kanala. Ura procesa lahko preseže mero zaostajanja za realnim časom (angl. *drift of time*). Poleg problemov ure procesa lahko pride do napak pri hitrosti streženja zahtev na procesu ali časa prenosa mrežnega sporočila na komunikacijskem kanalu. Podrobnejšo delitev napak v porazdeljenih algoritmih si bralec lahko ogleda v [22].

Vrsta problema V poglavju 1.3.2 smo omenili probleme, s katerimi se soočamo pri omrežjih enakovrednih subjektov. Različna raziskovalna področja v porazdeljenem računanju odpirajo različne tipe problemov, ki jih rešujemo s porazdeljenimi algoritmi. Omenimo nekaj vrst problemov, ki so pogosti pri snovanju porazdeljenih algoritmov. Najpogostejši so problemi dodeljevanja virov (angl. *resource allocation*), problem doseganja soglasnosti porazdeljenih procesorjev (angl. *consensus problem*), nadzor nad sočasno uporabo podatkovne baze (angl. *database concurrency control*), sinhronizacije (angl. *synchronization*), detekcije smrtnega objema (angl. *deadlock detection*), posnetki globalnega stanja sistema (angl. *global snapshot*) idr. Pri problemih dodeljevanja virov porazdeljeni sistem išče podmnožico virov, ki zadostijo uporabnikovi poizvedbi in je izražena z opisnikom virov. Pri problemu doseganja soglasnosti mora porazdeljeni sistem doseči pravilno stanje oz. odločitev na podlagi različne informacije, ki jo je pridobil na posameznih delih sistema. Primer problema soglasnosti je sistem v letalu, kjer je množica tipal na podlagi katerih mora sistem ugotoviti konsistentno stanje in se odločiti za neko akcijo. Za nadzor nad sočasno rabo dostopa do npr. podatkovne baze se uporabljajo algoritmi, ki zagotovijo konsistentno stanje sistema kljub sočasnim spreminjanjem stanj posameznih delov sistema. Za reševanje problema sinhronizacije poznamo kar nekaj pristopov, npr. alfa, beta in gama sinhronizatorje [34], ki delujejo na različnih nivojih porazdeljenega algoritma (alfa na nivoju vozlišč, beta na nivoju drevesa, gama pa v kombinaciji obeh in omogoča združevanje vozlišč, angl. *clustering*). S pomočjo sinhronizacije lahko vsak asinhron algoritem spremenimo v sinhron. Posebno zanimivi so problemi posnetkov glo-

balnega stanja sistema, saj z njihovo pomočjo lahko naredimo pregled stanja posameznih procesov hkrati s stanjem na komunikacijskih kanalih. S posnetkom globalnega stanja lahko ugotavljamo ustavljivost asinhronega algoritma (kar je v porazdeljenih algoritmih zelo pomembna lastnost). S posnetki globalnega stanja se lahko izognemo smrtnim objemom, saj jih je tako lažje zaznati. Več o posnetkih globalnega stanja in detekcije smrtnih objemov si lahko bralec ogleda v delu [34].

3.2 Lastnosti našega modela

- Vsako vozlišče pozna svoje sosede in lahko komunicira le z njimi.

  V splošni definiciji problema NAJMANJŠI k-CENTER velja, da je vhodni graf poln. Tega seveda ne moremo privzeti v realnem okolju, saj ne moremo pričakovati, da bo vsako vozlišče poznalo vse preostale vozlišče. Še več, temu se skušamo izogniti, saj bi širjenje informacije in vzdrževanje vseh sosedov v tem primeru vzela občutno preveč časa [20]. Zahtevamo le, da je celoten graf povezan.

- V komunikacijskem kanalu ne prihaja do napak.

  Dopuščamo napake na vozliščih, ki se jih da zaznati (npr. z navzgor omejenim časom).

- Komunikacijski kanal ohranja vrstni red sporočil.

  Če ne prihaja do napak nad komunikacijsko infrastrukturo, zadnji dve zahtevi zagotavlja že protokol TCP/IP.

- Vsako vozlišče ima enolično določeno identiteto.

  Zahteva je trivialna, saj lahko že samo na podlagi naslova TCP/IP sklepamo o identiteti vozlišča. Privzamemo, da ima vsako računsko vozlišče prirejen svoj naslov IP in da se vozlišča ne skrivajo v podomrežjih.

- Plast vzdrževanja proži sporočila za odstranjevanje sosednjih vozlišč v primeru detektirane napake ali odjave vozlišča.

  Ko med vozliščema izgine logična povezava, omenjena plast o tem obvesti obe vozlišči. V primeru, da se je eno vozlišče ustavilo ali je odpovedalo, plast obvesti soseda na drugem koncu dotične povezave (seje) o domnevni odpovedi. Lahko si jo predstavljamo kot mehanizem, ki stalno preverja stanje povezave TCP med vozliščema. Plast vzdrževanja zazna napake

oz. odhode vozlišč iz omrežja. Ko vozlišče v_0 zaradi napake tipa odpoved-ustavitev izgine iz omrežja, plast vzdrževanja sporoči vsem sosednjim vozliščem informacijo o izginotju vozlišča v_0.

3.3 Reševanje problema NAJMANJŠI k-CENTER

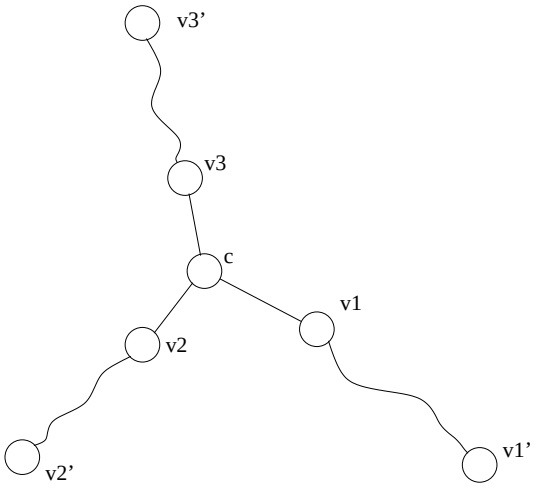
Najprej opišemo algoritem za reševanje problema 1-CENTER. V nadaljevanju predstavimo koncept premika centra v omrežju in se prepričamo, katere lastnosti ohranja premik centrov. Omenimo tudi koncept **konteksta centra**, s katerim omogočimo gradnjo in vzdrževanje drevesa najkrajših poti v gruči enega centra. V nadaljevanju natančno definiramo različne tipe sporočil, ki so uporabljeni v algoritmu. Med lastnostmi našega modela v poglavju 3.2 smo omenili pojem **plast vzdrževanja**, ki igra pomembno vlogo pri izvedbi rešitve problema NAJMANJŠI k-CENTER in ga bomo razložili prvega.

3.3.1 Problem 1-CENTER

Poseben primer problema NAJMANJŠI k-CENTER, ki ga rešujemo, je pri vrednosti $k = 1$. Razumevanje splošnejšega problema bo lažje, če začnemo pri tem posebnem primeru, hkrati pa nam slednji predstavlja dobro osnovo za analizo izvedbe našega algoritma. V nadaljevanju bomo predstavili asinhroni algoritem DKCENTER.

Algoritem temelji na (1) izračunu dreves najkrajših poti in za tem na (2) izbiri vozlišča, ki minimizira radij izračunanega drevesa najkrajših poti. Na grobo bi lahko algoritem razdelili na ta dva dela. Odločitvi o pravi smeri premika centra algoritmu botruje hevristika, ki se odloči s pomočjo informacije z robov drevesa izračunanih najkrajših poti v sistemu enakovrednih subjektov. Ko določimo smer premika centra na primernejše lokacije, se lahko premaknemo z večkratnim korakom – premik lahko nadaljujemo, če ugotovimo priložnost izboljšanja ob že obstoječi ugodni informaciji o robovih brez dodatnega računanja. S popravljanjem povezav po enkratnem premiku algoritem zna oceniti, kdaj naj ustavi premikanje in ponovno požene fazo izgradnje drevesne strukture (1), preko katere izve stanje na robovih trenutnega drevesa najkrajših poti (oz. gruče odjemalcev). Premikanje centra torej poteka po drevesni strukturi. Koren drevesa je vedno center, označimo ga s c. Vsako vozlišče, ki ni koren drevesa, pozna svojega predhodnika in svoje naslednike. Predhodnik je bliže centru c kot nasledniki. Informacijo o meji drevesa (gruče) dobimo preko **propagacije nazaj** (angl. *convergecast*).

Vozlišče **list** izve svoje stanje tako, da spremlja število prijavljenih naslednikov v drevesu najkrajših poti. Takoj, ko to pade na 0, obvesti svojega predhodnika v drevesu. Vozlišča, ki niso listi, propagirajo informacijo o oddaljenosti listov nazaj. Naj bo vozlišče v vozlišče, ki ni list in ni koren drevesa. Ko v zbere vsa sporočila o oddaljenosti listov poddreves s koreni njegovih naslednikov, tudi v propagira največjo razdaljo lista iz poddreves, ki so dosegljivi preko v. Tako nazadnje center c zbere vse največje razdalje listov, teh je enako številu stopnje $deg(c)$. Na sliki 3.1 lahko vidimo zgrajeno sliko centra c o gruči svojih odjemalcev po propagaciji nazaj.



Slika 3.1: Slika centra s pripadajočimi listi: $deg(c) = 3$, center c je s pomočjo propagacije nazaj spoznal oddaljenosti listov.

Ker je algoritem popolnoma asinhron, se lahko zgodi, da list l postane vozlišče z naslednikom. To se lahko zgodi v primeru, ko algoritem najde pot do sosednjega vozlišča, ki je dosegljiv iz l, po drugi, krajši poti. V tem primeru imajo predhodniki l napačno vrednost največje oddaljenosti dosegljivega lista. To dejstvo vzamemo v zakup, saj bo l po določenem času prejel oddaljenost od svojega naslednika in s tem popravil informacijo o dosegljivosti listov.

Premik centra Predpostavimo, da je vozlišče l_0 najbolj oddaljeno vozlišče poddrevesa T_0 izmed trenutno znanih listov centra c (slika 3.2). Naj bo l_1 list drugega najvišjega poddrevesa T_1 (drugi najbolj oddaljen list). Center se premakne, ko je izpolnjen naslednji pogoj:

$$d(c, l_0) > d(c, l_1) + d(c, P(c, l_0))\,. \tag{3.1}$$

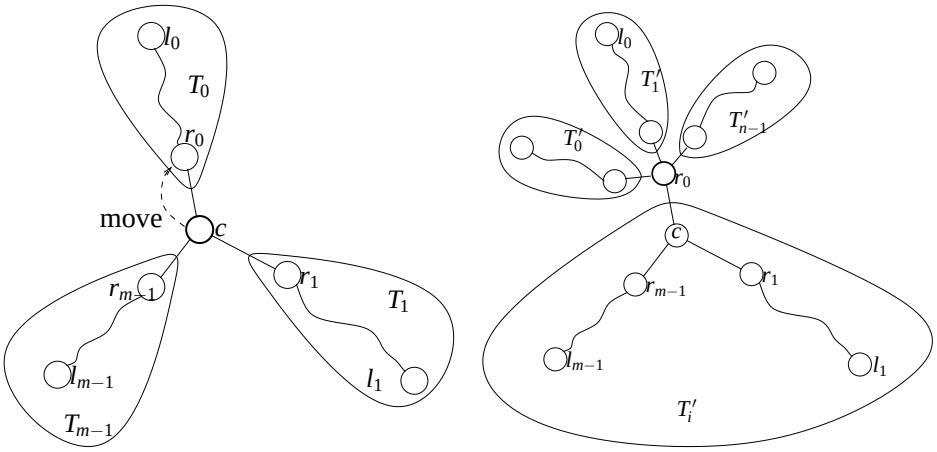
Zapis $d(v_1, v_2)$ označuje razdaljo med vozliščema v_1 in v_2. Relacija $P(x, y)$ je formalno definirana kot

$$P(x,y) = \begin{cases} \emptyset & \text{, če } y = x \\ z \wedge (x,z) \in E \wedge P(z,y) & \text{, sicer.} \end{cases}$$

Vozlišče z je torej prvi naslednik na poti med vozliščema x in y. Če upoštevamo stanje s slike 3.2a, lahko zapišemo $r_0 = P(c, l_0)$ in pogoj 3.1 se poenostavi v pogoj 3.2:

$$\Delta d > d(c, r_0)\,, \tag{3.2}$$

kjer Δd označuje razliko višin najvišjega poddrevesa T_0 in T_1. Z drugimi besedami, ko je razdalja med centrom c in najbolj oddaljenim vozliščem l_0 večja od razdalje med c in drugim najbolj oddaljenim listom l_1, povečane za dolžino povezave (c, r_0), je pogoj 3.1 izpolnjen. Vozlišče r_0 je prvo vozlišče na poti med c in l_0.



(a) Pred premikom centra;$m = 3$ (b) Po premiku centra; $m = 3,\ n = 3$

Slika 3.2: Leva slika predstavlja situacijo pred premikom centra. Desna slika predstavja drevo najkrajših poti po premiku. Vozlišče r_0 postane center po premiku.

Po premiku centra c na vozlišče r_0 (slika 3.2b) je struktura drevesa najkrajših poti spremenjena in dobljeno drevo ni več drevo najkrajših poti. Kljub temu lahko z manjšimi popravki spremenimo logične povezave med vozlišči tako, da ima novo dobljeno drevo določene lastnosti. Pod pogojem, da je topologija omrežja nespremenjena, so razdalje med vozlišči pravilno izračunane

tudi po premiku. Edini problem predstavlja zadnji premik na vozlišče r_0, ki je lahko vpeljal bližnjice do poddreves prejšnjega centra c. Ne želimo pa ponovno računati celotnega drevesa najkrajših poti, ker je to najpotratnejša faza algoritma. Poglejmo, kako nam do sedaj že znana informacija pripomore k nadaljnji odločitvi o premiku centra.

Relacijo med r_0 in c popravimo tako, da c dobi predhodnika r_0 (slika 3.2b). Takoj za tem r_0 dobi informacijo o najbolj oddaljenem listu vozlišča c in popravi do sedaj zbrane razdalje o listih poddreves r_0. S tem dobimo **oceno** drevesa najkrajših poti. Če povzamemo stanje drevesa najkrajših poti T s korenom c (slika 3.2a) z izpolnjenim pogojem 3.2, potem velja $H(T_0) - H(T_1) > d(c, r_0)$, kjer je $H(T)$ višina drevesa T. Označimo množico poddreves $\Theta_c = \{T_0, T_1, \ldots, T_{m-1}\}$, t.j. množica poddreves korena c. Ko se center premakne na vozlišče r_0, dobimo novo množico poddreves Θ_{r_0}. Prejšnja poddrevesa $\Theta_{r_0} \setminus \{T_0\} = \{T_1, T_2, \ldots, T_{m-1}\}$ se sedaj zlijejo v novo poddrevo $\{T_i'\} \in \Theta_{r_0}$ višine $H(T_i') = H(T_1) + d(c, r_0)$. Višine preostalih poddreves pa preprosto dobimo tako, da odštejemo od že znanih višin poddreves $\Theta_{r_0} \setminus \{T_i'\}$ razdaljo $d(c, r_0)$. Tako smo dobili **oceno** višin poddreves Θ_{r_0} in zopet lahko preverimo pogoj 3.1 za premik centra v katero izmed poddreves Θ_{r_0}.

Lema 3.3.1. *Za poddrevesa s korenom r_0 velja zgornja ocena: $\forall T \in \Theta_{r_0} \setminus T_i' :$ $H(T) := H(T) - d(c, r_0)$ kjer je $H(T)$ nova višina poddrevesa T.*

Dokaz: Dokaz je trivialen. Vsako višino poddrevesa v Θ_{r_0} moramo zmanjšati za $d(c, r_0)$, saj smo premaknili center bližje poddrevesu po povezavi (c, r_0). Če bi s premikom poslabšali višino kateremukoli izmed teh dreves, bi pomenilo, da je bilo drevo najkrajših poti že v osnovi izračunano narobe. □

Lema 3.3.2. *Zgornja meja α_0 višine drevesa T_i', ki je del drevesa najkrajših poti s korenom r_0 in je sestavljeno iz dreves $\{T_1, \ldots, T_{m-1}\}$ s korenom c, je enaka $\alpha_0 = H(T_1) + d(c, r_0)$.*

Dokaz: Dokažemo s protislovjem. Recimo, da obstaja vrednost $a > \alpha_0$ za višino drevesa s korenom r_0. V tem primeru je neko drevo $T_k \in \{T_1, \ldots, T_{m-1}\}$ dobilo naslednika s povezavo, ki je v skrajnem primeru daljša od $H(T_1) + d(c, r_0)$, kar pa se ne more zgoditi, saj bi krajša pot vodila preko c. □

Postopek premikov centra lahko nadaljujemo toliko časa, dokler velja pogoj 3.1. Ocena α_0 nam tako zagotavlja oceno, kdaj se moramo ustaviti in znova poračunati drevo najkrajših poti. S tem zopet dobimo natančno informacijo o oddaljenosti listov na meji gruče in s tem pokrivno razdaljo 1-centra. Ko center preide na vozlišče, kjer je pokritje dobro, ne obstaja nobena povezava,

s katero bi lahko izboljšali pokrivno razdaljo.

3.3.2 Kontekst centra

Raba enega samega centra v omrežju ima to prednost, da izračun drevesa najkrajših poti ni težavna. V tem primeru je namreč le eno drevo najkrajših poti in zato smo lahko prepričani, da se podatki o oddaljenosti nanašajo na koren tega drevesa. Največja težava enega centra pa je seveda centraliziranost sistema, čemur pa se želimo izogniti (neskalabilnost, ranljiva točka sistema – dovzetna za napad). V sistemih z več centri, $k > 1$, se gradnja množice dreves najkrajših poti začne asinhrono v več centrih takoj po fazi izbire centrov v omrežju. Večina postopkov začne s centri, postavljenimi v množici med seboj sosednjih vozlišč, nato pa jih v procesu premika razmaknejo in premaknejo na ugodnejše pozicije. Ob teh premikih centrov in izračunih dreves mora algoritem paziti, da se sporočila pri koordinaciji gradnje dreves za posamezen center ne mešajo preko meja gruč. V ta namen smo vpeljali koncept **konteksta centra**, ki mora biti v vsakem trenutku za vsakega izmed dreves najkrajših poti enoličen.

Definicija 3.3.1. *Kontekst centra je par števil $\langle c, g \rangle$. Parameter $c \in \mathbb{Z}$ predstavlja identiteto izvora sporočila (izvor je center, ki je hkrati koren drevesa). Parameter $g \in \mathbb{Z}$ predstavlja število premikov centra (oz. generacijo centra).*

Parametra c in g v definiciji 3.3.1 sta v nadaljevanju zaradi lažjega razumevanja algoritmov označena kot **center** in **generation** po vrsti. Vsako sporočilo algoritmov v nadaljevanju je označeno s kontekstom centra. Vozlišče ob sprejemu sporočila lahko hitro preveri, ali je sporočilo veljavno ali pa ga lahko zavrže brez posledic. V primeru, da vozlišče v prejme sporočilo iz sosednje gruče vozlišč, v lahko to ugotovi preko parametra c. Pri izračunu drevesa najkrajših poti pa se lahko zgodi, da je center migriral na drugo lokacijo in s tem povečal parameter g. Vozlišča s pomočjo tega parametra konteksta centra ugotovijo, ali hranijo trenutno podatek starejšega centra. V primeru, da podatek g označuje novejšo generacijo sporočil enakega centra, morajo vozlišča ta podatek posodobiti.

3.3.3 Izračun in vzdrževanje drevesa najkrajših poti

Drevo najkrajših poti je v algoritmu uporabljen za razširjanje podatka o oddaljenosti vozlišč z roba gruče centra. Center ob gradnji drevesa poleg svojih

najbližjih sosedov spozna tudi najbolj oddaljene liste. Ko po gradnji preko propagacije nazaj spozna njihove oddaljenosti, lahko na osnovi teh razdalj sklepa o premiku, ki bi lahko zmanjšal višino drevesa in s tem radij gruče centra. V razdelku 3.3.1 smo pokazali, da ob predpostavki statičnega omrežja lahko na ta način kvečjemu izboljšamo **pokrivno razdaljo centra**, česar pa v primeru dinamičnega omrežja ne moremo trditi. Z dinamičnostjo omrežja se namreč spreminja struktura drevesa, ki lahko že ob prvem dogodku postane le navadno drevo. Zato moramo občasno ponoviti izgradnjo drevesa najkrajših poti — to se zgodi po premiku centra — da vzdržujemo lastnost drevesa najkrajših poti. Težave pri pokrivanju zaradi dinamičnosti omrežja rešimo z večkratnimi premiki centra oz. z iterativno gradnjo drevesa najkrajših poti. Po začetni izgradnji drevesa center lahko migrira v več korakih, dokler je izpolnjen pogoj 3.2. Ko pogoj ni več izpolnjen, ponovno izračunamo drevo najkrajših poti.

Drevo najkrajših poti s korenom c je drevo, v katerem so poti med vozliščem c in kateremkoli drugim vozliščem v drevesu vozlišča c najkrajše. Definicija drevesa najkrajših poti je podana z definicijo 1.6.2. Vsako vozlišče, ki ni koren, mora poznati svojega predhodnika in skupino naslednikov v drevesu, če jih ima. Poleg tega vsako vozlišče v hrani razdaljo do najbolj oddaljenega vozlišča, ki je dosegljivo po vsakem izmed poddreves vozlišča v (vsako poddrevo s predhodnikom v ima za koren naslednika vozlišča v). Izračun in vzdrževanje drevesa najkrajših poti je sestavljeno iz dveh operacij: **dodajanja vozlišča** in **odstranjevanja vozlišča**.

Naš algoritem uporablja pristop s pomočjo osnovnih porazdeljenih algoritmov, ki jih najdemo v literaturi [34, 54]. Gradnja je zelo podobna algoritmu **AsyncBellmanFord** [34]. V slednjem se najkrajše poti izračunajo s propagacijo doslej poznane najkrajše poti od korena drevesa. Vsakokrat, ko vozlišče spozna krajšo povezavo od trenutno znane najkrajše povezave, propagira krajšo razdaljo izmed teh dveh med svoje sosede. Vzdrževanje je zelo podobno algoritmu **Netchange** [54], kjer je opisan princip obravnavanja novonastalih in izbrisanih povezav.

Začetna izgradnja drevesa. Začetna izgradnja drevesa najkrajših poti se prične s širjenjem sporočil, ki propagirajo najkrajšo razdaljo od centra c do vozlišč $v \in V$. Ko v sprejme sporočilo z razdaljo od sosednjega vozlišča v', se odloči o morebitni novi relaciji z v' — ali postane direktni naslednik ali pa se propagacija razdalje ustavi. Če je razdalja do centra c v prejetem sporočilu krajša od trenutne razdalje vozlišča v do centra c, potem v postane direktni naslednik v'. V tem primeru se mora v odklopiti od prejšnjega predhodnika. Ko se v priključi na novega predhodnika v', prične s širjenjem najkrajše razdalje

od korena drevesa do v. V primeru, da je razdalja, prejeta od v', večja od trenutne znane, se propagacija ustavi, vozlišče v pa o tej odločitvi sporoči vozlišču v'.

Ključnega pomena pri porazdeljeni izvedbi opisanih postopkov so pravilno oblikovana sporočila, ki si jih izmenjujejo vozlišča. Tabela 3.1 prikazuje vse definirane tipe sporočil in njihov opis v fazi gradnje in vzdrževanja drevesa najkrajših poti.

Tip sporočila	Opis
connect	propagiranje informacije o oddaljenosti od centra
connect_as_child	dodajanje novega vozlišča v množico *Children*
connect_new	dodajanje logične povezave med dvema sosednjima vozliščema
disconnect_as_child	brisanje vozlišča iz množice *Children*
disconnect	brisanje logične povezave med dvema sosednjima vozliščema
move	premikanje statusa centra med vozlišči
refuse	sporočanje o ustavitvi izračuna drevesa
update_leaf	propagiranje razdalje z lista proti centru

Tabela 3.1: Tipi sporočil, ki so v uporabi pri reševanju problema NAJMANJŠI k-CENTER v asinhronem omrežju.

Definicija 3.3.2. *Sporočilo tipa* **connect** *ima obliko*

$$\langle \textbf{connect}, \textit{distance}, \textit{center}, \textit{generation} \rangle .$$

Parameter **distance** *označuje oddaljenost od centra* **center**, *par* **center** *in* **generation** *predstavlja kontekst centra.*

Algoritem 1 prikazuje prejem sporočila **connect**, čigar tip je definiran z definicijo 3.3.2. Seznam globalnih spremenljivk, s katerimi je določeno stanje vozlišča v, se nahaja v tabeli 3.2. Te spremenljivke so skupne vsem algoritmom iz tega poglavja.

V primeru, ko je izvor sporočila (parameter **center**) enak centru c vozlišča v, je potrebno pregledati vrednost parametra **generation**. Če je v prejetem sporočilu algoritma 1 (vrstice 3–4) ta starejša, se sporočilo zavrže. V primeru, ko **generation** prejetega sporočila označuje, da je bilo poslano z novejšega

centra c' znotraj iste gruče odjemalcev, se upošteva razdalja prejetega sporočila.

V primeru, ko pa gre za sporočilo s sosednje gruče računalnikov (razvidno iz konteksta sporočila), vozlišče v primerja prejeto razdaljo s svojo razdaljo. Ko je prejeta razdalja večja od trenutne razdalje vozlišča v do c, v zavrne nadaljnjo gradnjo drevesa najkrajših poti s sporočilom **refuse** v vrstici 8 algoritma 1. Sporočila tipa **refuse** je definirano z definicijo 3.3.3.

Definicija 3.3.3. *Sporočilo tipa* **refuse** *ima obliko*

$$\langle \textbf{refuse}, \textit{center}, \textit{generation} \rangle .$$

Par **center** *in* **generation** *predstavlja kontekst centra.*

V primeru, ko je razdalja prejetega sporočila manjša od razdalje do c (vrstice 10–16), vozlišče v zamenja pripadnost gruči in širi novo, krajšo razdaljo vsem svojim sosedom. Vozlišče v najprej sporoči v', da bo v postal direktni naslednik v' (vrstica 10). Za tem se s odveže od prejšnjega predhodnika (vrstica 11). V nadaljevanju se nastavijo potrebne spremenljivke in nato vozlišče v sporoči svojim sosedom novo najkrajšo razdaljo do centra (vrstica 16). Enakosti lahko razrešimo tako, da obvelja razdalja centra z manjšim *id*, vendar tega v opisu našega algoritma zaradi preglednosti in dolžine opisa algoritma eksplicitno ne omenjamo.

V algoritmu 1 sta uporabljena še dva tipa sporočil, ki sta namenjena sporočanju sosedom za dodajanje logične povezave oz. brisanje logične povezave v strukturi drevesa najkrajših poti v primeru prehajanja vozlišč med sosednjimi grupami. Podajmo še definiciji teh dveh tipov sporočil za dodajanje (definicija 3.3.4) in odstranjevanje (definicija 3.3.5) logičnih povezav.

Definicija 3.3.4. *Sporočilo tipa* **connect_as_child** *ima obliko*

$$\langle \textbf{connect\_as\_child}, \textit{center}, \textit{generation} \rangle .$$

Par **center** *in* **generation** *predstavlja kontekst centra gruče, del katere vozlišče postaja.*

Definicija 3.3.5. *Sporočilo tipa* **disconnect_as_child** *ima obliko*

$$\langle \textbf{disconnect\_as\_child}, \textit{center}, \textit{generation} \rangle .$$

Par **center** *in* **generation** *predstavlja kontekst centra od katerega se vozlišče odklaplja.*

Algoritem 1 Širjenje informacije: vozlišče v prejme sporočilo $\langle$**connect**, $distance'$, $center'$, $generation'\rangle$ od vozlišča v'

```
1: if center = center' then
2:   if generation > generation' then
3:     ignore the message
4:     return
5:   end if
6: end if
7: if distance ≤ distance' then
8:   SEND ⟨refuse, center', generation'⟩ TO v'
9: else
10:  SEND ⟨connect_as_child, center', generation'⟩ TO v'
11:  SEND ⟨disconnect_as_child, center', generation'⟩ TO parent
12:  parent ← v'
13:  distance ← distance'
14:  center ← center'
15:  generation ← generation'
16:  ∀v'' ∈ Neighborhood \ {parent}:
     SEND ⟨connect, distance + d(v, v''), center', generation'⟩ TO v''
17: end if
```

Pri realizaciji dela algoritma DKCENTER na vozlišču, ki sprejema ta dva tipa sporočil, moramo paziti na nekaj podrobnosti. Ni dovolj, da samo dodamo oz. odstranimo logično povezavo. Preveriti moramo tudi lastnost vozlišča, če je list oz., če se je spremenila razdalja do najbolj oddaljenega lista v poddrevesu vozlišča, ki prejema sporočilo. V primeru, če vozlišče postane list, moramo propagirati razdaljo vozlišča svojemu predhodniku. V drugem primeru (spremenila se je razdalja do najbolj oddaljenega lista) moramo poiskati novo največjo oddaljenost lista in slednjo propagirati predhodniku trenutnega vozlišča. Zaradi preglednosti na tem mestu ne bomo dodali opisa algoritma teh dveh primerov — potek razviden iz zgornjega opisa.

Dodajanje povezave. Dodajanje nove povezave se vrši preko sporočila **connect_new**, ki je definirano z definicijo 3.3.6. Ta tip se razlikuje od tipa definicije 3.3.4 v tem, da sporoča dodajanje novega vozlišča med sosede vozlišča in ne dodajanja nove logične povezave na nivoju drevesa najkrajših poti.

Spremenljivka	Opis
center	identiteta centra
parent	predhodnik v drevesu najkrajših poti
generation	določa generacijo drevesa najkrajših poti, v katerem je vozlišče
distance	označuje razdaljo med centrom in vozliščem
Neighborhood	množica vseh sosednjih vozlišč
M	množica parov $(p, d) \in V \times \mathbb{R}$, kjer je p koren poddrevesa, v katerem se nahaja najbolj oddaljeno vozlišče (list), in d označuje razdaljo do tega lista
Children	množica naslednikov vozlišča v drevesu najkrajših poti

Tabela 3.2: Spremenljivke in njihovi opisi, ki nastopajo v algoritmih 1, 2, 3, 4, 5 in 6.

Definicija 3.3.6. *Sporočilo tipa* **connect_new** *ima obliko*

$$\langle \textbf{connect\_new} \rangle .$$

Sporočilo tega tipa ne potrebuje informacije o kontekstu centra, saj gre za novo vozlišče v omrežju. Algoritem 2 prikazuje postopek pri sprejemu sporočila tipa **connect_new**. Algoritem 2 razdelimo na dva pogoja. Prvi pogoj je posledica algoritma 3 in obravnava primer izgube direktnega predhodnika v drevesu najkrajših poti. V tem primeru se odzove z enako reakcijo, kot se je odzval pošiljatelj sporočila **connect_new**. S tem mora v ponovno poiskati direktnega predhodnika. To stori s pošiljanjem sporočilom enakega tipa vsem svojim sosedom (vrstici 2–3) algoritma 2.

Algoritem 2 Povezovanje: vozlišče v prejme sporočilo $\langle$**connect_new**$\rangle$ od vozlišča v'

1: **if** $v' = parent$ **then**
2: $\quad parent \leftarrow undefined$
3: $\quad \forall v'' \in Neighborhood$: SEND $\langle$**connect_new**$\rangle$ TO v''
4: **else**
5: $\quad$ SEND $\langle$**connect**, *distance*, *center*, *generation*$\rangle$ TO v'
6: **end if**

Drugi pogoj je aktualen v primeru, ko vozlišče v' ni direktni predhodnik v drevesu. Takrat mu vozlišče v sporoči zahtevo za povezavo preko sporočila tipa **connect**. S tem bo v dobil povratno sporočilo s podatkom o oddaljenosti sosednjega vozlišča, ki je del druge gruče centra. Na ta način odjemalci prehajajo med gručami sosednjih centrov.

Odstranjevanje povezave Plast vzdrževanja, ki je opisana v poglavju 3.2, je zadolžena za proženje zahtev odstranjevanja logičnih povezav med sosednjimi vozlišči. Ker sta vozlišča lahko v dveh različnih relacijah, moramo obravnavati oba primera odstranjevanja povezave med sosednjima vozliščema v in v'. Algoritem 3 predstavlja postopek vozlišča v primeru izbrisa logične povezave.

Na vozlišču v, ki je direktni predhodnik v drevesni strukturi, se sosednje vozlišče v' odstrani iz množice znanih naslednikov (vrstice 2–4). Vozlišče mora zopet pregledati vse svoje naslednike in popraviti podatek o najbolj oddaljenem vozlišču in oddaljenost posredovati svojemu predhodniku v drevesu. To stori s sporočilom **update_leaf**, ki hrani podatek o najbolj oddaljenem vozlišču v poddrevesu s korenom v.

Algoritem 3 Brisanje: vozlišče v prejme sporočilo ⟨**disconnect**⟩ od sosednjega vozlišča v'

```
1: if v' is a child of v in the sink tree then
2:     M ← M \ {(v', x)}
3:     max_distance ← max_dist(M)
4:     SEND ⟨update_leaf, max_distance, center, generation⟩ TO parent
5: end if
6: if v' = parent then
7:     center_dir ← undefined
8:     distance ← ∞
9:     ∀v'' ∈ Neighborhood: SEND ⟨connect_new⟩ TO v''
10: end if
```

V primeru, da je vozlišče v' predhodnik v drevesu najkrajših poti (vrstice 7–9), pa mora vozlišče v o tem obvestiti vse svoje naslednike. To stori hkrati z zahtevkom po novem povezovanju z doslej znanimi sosednjimi vozlišči brez v'. Informacija, da je v izgubilo predhodnika, je implicitno vsebovana v sporočilu **connect_new**, podobno kakor smo to storili v algoritmu 3. Če katerokoli vozlišče prejme slednji tip sporočila od svojega direktnega predhodnika v drevesu, stori enako kot drugi primer v algoritmu 3 — izbriše znanega očeta z

oddaljenostjo od vozlišča do centra in o tem obvesti svoje sosede.

3.3.4 Začetna porazdelitev centrov

Ključni del našega algoritma je začetna porazdelitev centrov. Algoritem 4 prikazuje ta postopek s pomočjo žetonov, ki jih porazdelimo med k vozlišč. Algoritem deluje tako, da vozlišče v po prejetju k žetonov začne s propagacijo le-teh med svoje sosede, enega pa obdrži zase. Začetno sporočilo, ki nosi vseh k žetonov, pride s strani uporabnika kot zahteva za pričetek algoritma za izračun k centrov. Vsako vozlišče, ki prejme sporočilo s $k' <= k$ žetonov začetnega centra, postane center in preostalih $k' - 1$ žetonov porazdeli med svoje sosede. Na opisani način se porazdeli vseh k žetonov.

Definicija 3.3.7. *Sporočilo tipa* **alloc** *ima obliko*

$$\langle \textbf{alloc}, k' \rangle .$$

Parameter k' nosi število žetonov.

Z definicijo 3.3.7 podajamo tip sporočila, s katerim algoritem 4 širi začetne žetone. Po prejemu sporočila $\langle \textbf{alloc}, k' \rangle$ postane vozlišče v center (v primeru, če vozlišče še nima statusa centra) in nato porazdeli $k - 1$ žetonov naključno med svoje sosede (vrstice 2–3, 6–9). V primeru, da sporočilo, ki nosi začetne žetone, prejme vozlišče v s statusom center, v naključno porazdeli vseh k' prejetih žetonov med svoje sosede (preskoči vrstice 2–3). Vozlišče v s prejetjem žetona postane center (vrstica 2) in določi oznako, s katero bo poimenovana gruča vozlišču prirejenih vozlišč. Predpostavimo, da so izbrane oznake edinstvene v omrežju. Slednje lahko zagotovimo skoraj zagotovo, če so oznake izbrane na podlagi identitet vozlišč (ki so edinstvena v omrežju — osnovana so na naslovih IP).

Na opisani način se določi začetnih k vozlišč — centrov. Takoj, ko se žeton pretvori v center, center prične z izvajanjem gradnje drevesa najkrajših poti, s katerim zgradi svojo gručo (algoritem 1). To fazo center prične s propagacijo sporočil **connect** (definicija 3.3.2) med svoje sosede.

3.3.5 Premiki centrov

S pomočjo **premikov centrov** in **hevristike za premik** algoritem poskuša optimizirati lokacijo centrskih vozlišč po omrežju in se izogniti lokalnim minimumom. Vozlišča posameznih avtonomnih gruč so agnostična glede na sosednje gruče. S tem se izognemo potrebi po zahtevni sinhronizaciji med gručami

Algoritem 4 Začetni izbor vozlišč: vozlišče v prejme sporočilo $\langle$**alloc**, $k\rangle$ od vozlišča v'.

Node's variables:
center - id of the center
$Neighborhood$ - the set of all adjacent nodes

```
1: if center = undefined then
2:    center ← uniqueID()
3:    k ← k − 1
4: end if
5: while k > 0 do
6:    k' = random(k)
7:    i = random(|Neighborhood|)
8:    SEND ⟨alloc, k'⟩ TO v_i
9:    k ← k − k'
10: end while
```

in si olajšamo delo pri upoštevanju dinamičnosti omrežja (odvisnosti med vozlišči oz. gručami vpeljujejo dodatno kompleksnost [34]).

Center migrira v primeru izpolnitve pogoja 3.1. **Sporočanje nazaj** po izračunanem drevesu najkrajših poti se zdi primeren postopek pridobivanja podatkov za preverjanje tega pogoja. Postopek je opisan z algoritmom 5. Postopek premika vozlišča-centra na sosednje vozlišče je prikazano z algoritmom 6.

Sporočanje nazaj se torej odvija s sporočili tipa **update_leaf**. Ko vozlišče v v algoritmu 3 izračuna (vrstice 2–4), da je list v drevesu, sporoči podatek oddaljenosti do centra svojemu direktnemu predhodniku. V nadaljevanju bomo opisali postopek vozlišča v, ko prejme sporočila tega tipa (algoritem 5).

Definicija 3.3.8. *Sporočilo tipa* **update_leaf** *ima obliko*

$$\langle \textbf{update\_leaf}, \mathit{distance}, \mathit{center}, \mathit{generation}\rangle\,.$$

Parameter **distance** *predstavlja oddaljenost najbolj oddaljenega lista v poddrevesu. Par* **center** *in* **generation** *predstavlja kontekst centra.*

V algoritmu 5 prejemno vozlišče v v začetku preveri pravilnost konteksta centra. Pravilnost konteksta preverimo s primerjavo oznak generacije. Če spremenljivka **generation** v sporočilu nosi pretečeno informacijo (če je prejeta generacija manjša od vrednosti trenutne), se na tem mestu sporočilo zavrže brez posledic. Za tem v posodobi spremenljivko M, ki drži podatke o najbolj oddaljenih listih in njihovih korenih v posameznih poddrevesih vozlišča

Algoritem 5 Posodobitev: vozlišče v prejme sporočilo $\langle$**update_leaf**, $distance'$, $center'$, $generation'\rangle$ od vozlišča v'.

```
1:  if generation > generation' then
2:    return
3:  end if
4:  M ← M ∪ {(v', distance')}
5:  if |M| = |Children| then
6:    v'' ← max_dist_parent(M)
7:    second_max_distance ← max_dist(M \ {(v'', max_dist(M))})
8:    potential_radius ← second_max_distance + d(v, v'')
9:    if v.status is center then
10:     if max_dist(M) > potential_radius then
11:       SEND ⟨move, second_max_distance, center, generation⟩ TO v''
12:       center ← undefined
13:       M ← M \ {(v'', max_dist(M))}
14:       ∀v'' ∈ Neighborhood: SEND ⟨connect_new⟩ TO v''
15:     end if
16:   else
17:     max_distance ← max_dist(M)
18:     SEND ⟨update_leaf, max_distance, center, generation⟩ TO parent
19:   end if
20: end if
```

v. V primeru, ko je v prejelo enako število sporočil tipa **update_leaf**, kakor je število sosedov v drevesu, v izbere iz podatkovne strukture M naslednika v'' v drevesu, ki vodi do najbolj oddaljenega vozlišča (vrstica 6). V primeru, ko v ni center, propagira največjo oddaljenost lista v poddrevesu s sporočilom **update_leaf** po drevesu navzgor (vrstici 17–18). Če je vozlišče v center (vrstice 10–15), mora preveriti pogoj za premik 3.1 (vrstica 10). Če je pogoj izpolnjen, vozlišče pošlje sporočilo **move** vozlišču v''. Vozlišče v najbolj oddaljeno vozlišče izbriše iz poznanih naslednikov v drevesu. Nato vsem svojim sosedom pošlje zahtevek za ponovno priključitev drevesu. S časoma bo vozlišče dobilo odgovor **connect** in se tako priključilo drevesu.

Definicija 3.3.9. *Sporočilo tipa* **move** *ima obliko*

$$\langle \textbf{move}, distance, center, generation\rangle .$$

Parameter **distance** *nosi informacijo o najbolj oddaljenem listu poddrevesa*

Algoritem 6 Premiki centrov: vozlišče v prejme sporočilo $\langle$**move**, $distance'$, $center'$, $generation'\rangle$ od vozlišča v'

```
1: M ← M ∪ {v', distance}
2: v'' ← max_dist_parent(M)
3: second_max_distance ← max_dist(M \ (v'', max_dist(M)))
4: potential_radius ← second_max_distance + d(v, v'')
5: if max_dist(M) > potential_radius then
6:    SEND ⟨move, second_max_distance, center, generation⟩ TO v''
7:    center ← undefined
8:    M ← M \ {v'', max_dist(M)}
9:    ∀v'' ∈ Neighborhood: SEND ⟨connect_new⟩ TO v''
10: else
11:   distance ← 0
12:   M ← ∅
13:   Children ← ∅
14:   generation ← generation' + 1
15:   center ← center'
16:   ∀v'' ∈ Neighborhood: SEND ⟨connect, 0, center, generation⟩ TO v''
17: end if
```

korena, ki je poslal sporočilo. Dvojica **center** *in* **generation** *predstavljata kontekst centra.*

V primeru, da je vozlišče v sprožilo sporočilo tipa **move** (definicija 3.3.9), se bo sosednje vozlišče odzvalo z algoritmom 6. Sporočilo nosi podatek o najbolj oddaljenem listu prejetega poddrevesa s korenom v'. Tako najprej posodobi informacijo o najbolj oddaljenih listih in izračuna vse potrebno za preveritev pogoja 3.1 — najbolj oddaljeno vozlišče in radij po naivnem postopku. Če je pogoj izpolnjen, se premikanje centra še ne konča (vrstice 6–9 algoritma 6) in stori podobno, kakor je opisano z algoritmom 5 — posodobi informacijo sosednjih vozlišč in izda zahtevke za navezo vsem svojim sosedom. V primeru, da pogoj 3.1 ni izpolnjen, vozlišče postane center in začne s propagacijo sporočil **connect** (vrstice 11–16). V teh vrsticah algoritma vozlišče ponastavi svoje globalne spremenljivke razen spremenljivke **generation**. Slednjo le poveča za 1 in s tem sporoči vsem prejemnikom kasnejših sporočil, proženih s strani vozlišča v, da gre za novo generacijo centra in posledično novo generacijo gruče vozlišč.

Poglavje 4

Izdelava prototipa

Učinkovitost opisanih algoritmov želimo preveriti tudi v praksi. Vendar je delo z razpoložljivimi porazdeljenimi sistemi, kot je na primer PlanetLab, relativno zapleteno, zato se želimo najprej osredotočiti na preverjanje delovanja brez ukvarjanja z vzpostavitvijo seje in zaseganja vozlišč. Zato smo pri izdelavi prototipa najprej posegli po simulatorjih in orodjih za gradnjo modelov omrežij.

Pri svojem delu smo tako uporabili simulacijsko okolje **PeerSim** [27, 28], kjer smo algoritem realizirali in testirali. Množico testnih omrežij, ki smo jih uporabili tako v simulatorju kot kasneje v realnem sistemu, smo izdelali z orodjem **BRITE**. Ob prenosu algoritma na resnična vozlišča v omrežju TCP/IP nam je bilo pa v pomoč ogrodje **DCenter**. V nadaljevanju bomo opisana orodja ter okolje PlanetLab opisali podrobneje.

4.1 Simulator PeerSim

Izjemno razširljiv simulator PeerSim je orodje, ki omogoča postavljanje omrežij različnih topologij in z različnimi lastnostmi v pomnilniku razvojnega okolja. V svojem okolju prevzame nadzor nad simuliranimi vozlišči, vzdržuje povezave med njimi in izvaja izmenjavo sporočil.

Napisan je v programskem jeziku java (http://www.java.com) in vpeljuje različne vzorce programiranja za hitro realizacijo porazdeljenih algoritmov. Odlikuje ga majhna poraba pomnilniškega prostora, kar doseže z veliko mero deljenja javanskih predmetov brez njihovega podvajanja, zato je primeren za simulacijo velikih omrežij. Omogoča realizacijo algoritmov nad dinamičnimi omrežji s pomočjo več plasti ločenih protokolov. Plastovita arhitektura simulatorja omogoča preglednejšo realizacijo algoritmov. Ima tudi preprost način

nastavitve posameznih protokolov, tako da uporabniku ni potrebno pisati novih mehanizmov za nastavitev stanja posamezne plasti algoritma. Vsebuje dva načina simulacije: **ciklično** (sinhrono) in **dogodkovno** (asinhrono) simulacijo. Ciklična simulacija predstavlja zelo poenostavljen model, ki omogoča hitro in skalabilno realizacijo sinhronih algoritmov na račun neupoštevanja prenosne komunikacijske plasti. Po drugi strani pa dogodkovni model simulacije omogoča izbor dodatnih parametrov prenosne plasti. Taki parametri so npr. porazdelitev zakasnitve sporočil, določitev najmanjše in največje zakasnitve, verjetnost napake na komunikacijskem kanalu idr. Simulator omogoča simulacijo različnih modelov porazdeljenega sistema, ki se med seboj razlikujejo v sinhronosti oz. asinhronosti procesov in sinhronosti oz. asinhronosti omrežnih sporočil.

Tu smo naleteli na oviro, saj simulator v osnovi ne omogoča asinhronega — FIFO (angl. *first in first out*) — prenosa sporočil med posameznimi protokoli. Slednje smo rešili z majhnim popravkom razreda asinhronega načina simulacije, kjer za vsako vozlišče vodimo čas zadnjega sporočila. Simulator PeerSim v osnovi določi zakasnitev prejetega sporočila po normalni porazdelitveni funkciji, tako da lahko sporočilo s_1, ki je bilo poslano kasneje kot neko drugo sporočilo s_2, namenjeno istemu vozlišču v, prehiti sporočilo s_2. V spremenjeni izvedbi velja, da je v primeru prehitevanja kasneje poslanega sporočila s_2 sporočilo potisnjeno tik za zadnje prejeto sporočilo v vrsti vozlišča v.

PeerSim omogoča izredno modularno realizacijo in ponuja mehanizem nastavitev, s katerim lahko nastavimo vsak parameter izbranega modula simulatorja. Koraki za realizacijo simulacije so naslednji:

- s številom vozlišč izberemo velikost simulacije,
- izberemo enega ali več delovnih protokolov, ki jih bomo uporabili v simulaciji,
- izberemo nastavitvene in nadzorne protokole simulacije,
- poženemo simulator z nastavitveno datoteko.

Glavna vez, ki povezuje simulacijo v celoto, je nastavitvena datoteka. V njej opišemo vse izbore prejšnjih korakov.

4.2 Izgradnja testnih omrežij

Za izgradnjo testnih omrežij smo uporabili orodje BRITE [35]. To je univerzalni generator omrežij, ki omogoča izdelavo 6 različnih modelov omrežij: ravninska omrežja **samostojnih sistemov** (tipov **Waxman** in **Barabasi-Albert**), ravninska omrežja **z usmerjevalniki** (tipov **Waxman** in **Barabasi-Albert**) in **hierarhična omrežja** (model od **zgoraj-navzdol** in model od **spodaj-navzgor**).

Omrežja samostojnih sistemov (angl. *autonomous system*, AS) so omrežja, kjer vozlišča predstavljajo samostojne sisteme (v našem primeru odjemalce oz. strežnike, torej kar subjekte omrežja), povezave med njimi pa predstavljajo povezave med subjekti — samostojnimi subjekti.

Omrežja z usmerjevalniki so omrežja, kjer vozlišča nastopajo kot usmerjevalniki, povezave med vozlišči pa predstavljajo direktne povezave usmerjevalnik-usmerjevalnik ali usmerjevalniki-odjemalec. BRITE trenutno podpira dvostopenjska hierarhična omrežja usmerjevalnikov. Kljub temu je mogoče v več fazah ustvariti $2n$-stopenjska omrežja v n ponovitvah.

Dvostopenjska omrežja z usmerjevalniki si lahko predstavljamo kot omrežja omrežij, ki povezujejo več sistemov AS. Obstajata dva različna modela izgradnje hierarhičnih modelov omrežij: model od zgoraj-navzdol in model od spodaj-navzgor. Pri modelu od zgoraj-navzdol BRITE najprej ustvari omrežje G (tipa AS), kjer vsakemu vozlišču $v \in G$ priredi omrežje z usmerjevalniki U_v. Novo omrežje sestavljajo vozlišča

$$\bigcup_{0<i<|G|} U_{v_i} ,$$

kjer je U_{v_i} množica vozlišč ustvarjenega i-tega omrežja z usmerjevalniki. Za povezovanje med pari vozlišč z usmerjevalniki U_{v_i} in U_{v_j}, kjer $i \neq j$, BRITE uporablja različne strategije (naključno; najmanjša stopnja vozlišča podomrežij; najmanjša stopnja vozlišča, ki ni list; najmanjša stopnja k vozlišča podomrežij). Podrobne razlage presegajo zastavljen okvir magistrske naloge.

Alternativni pristop k strategiji gradnje hierarhičnega omrežja od zgoraj-navzdol je uporaba strategije od spodaj-navzgor. Tu BRITE najprej ustvari omrežje z usmerjevalniki (lahko z uporabo različnih tipov, npr. **Waxman** in **Barabasi-Albert**). V drugem koraku zgradi omrežje tipa AS, kjer vsakemu vozlišču iz prej zgrajenega omrežja priredi določeno število vozlišč (strategijo prirejanja lahko določi uporabnik). Nazadnje BRITE združi prirejena vozlišča zopet po strategiji, ki jo določi uporabnik (**naključni izbor**, kjer BRITE naključno izbira vozlišča za določen AS, dokler ne doseže prave velikosti; **na-**

ključni sprehod, kjer sistem naključno izbere vozlišče in naredi naključni sprehod po grafu iz izbranega vozlišča, vozlišča na poti pa prireja določenemu omrežju tipa AS, dokler ta ne doseže prave velikosti vozlišča).

4.2.1 Modeli omrežij

Z orodjem BRITE je možno ustvariti več tipov omrežij:

- Waxman,
- BA (Barabasi-Albertov model),
- BA2 (drugi Barabasi-Albertov model),

Model Waxman temelji na naključnem postavljanju vozlišč po prostoru, pri čemer je verjetnost povezanosti dveh poljubnih točk enaka

$$P(u, v) = \alpha e^{-d/(\beta\ L)}\,, \tag{4.1}$$

kjer velja $0 < \alpha, \beta \leq 1$, d je evklidska razdalja med vozliščema u in v, L pa je največja razdalja med poljubnima vozliščema v prostoru.

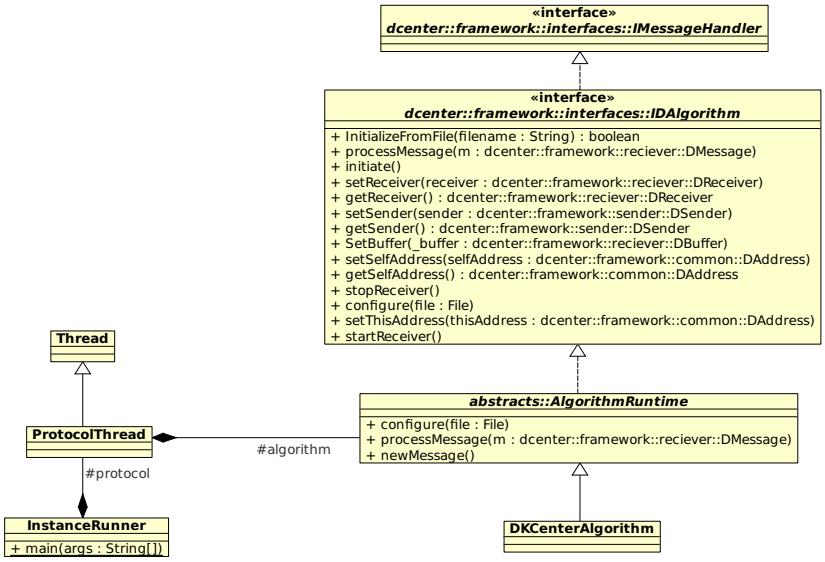
Model Barabasi-Albert je podrobneje opisan v delu [4]. Po tem modelu imajo novo pridružena vozlišča večjo verjetnost povezanosti z vozlišči, ki imajo večjo izhodno stopnjo. Verjetnost, da se bo pridruženo vozlišče u povezalo z že obstoječim vozliščem v, znaša

$$P(u, v) = \frac{d_v}{\sum_{k \in V} d_k}\,, \tag{4.2}$$

kjer je d_v stopnja ciljnega vozlišča, V je množica vozlišč, ki so se priključili omrežju, in $\sum_{k \in V} d_k$ je vsota izhodnih stopenj obstoječih vozlišč.

4.3 Ogrodje DCenter

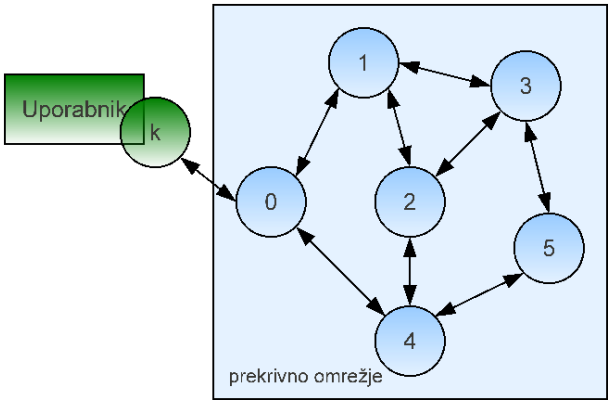
Zaradi hitrejšega razvoja in testiranja smo razvili ogrodje za realizacijo, kjer smo omogočili ponovno rabo izvorne kode simulatorja v našem ogrodju. Abstrahirali smo komunikacijsko plast tako, da razred z realiziranim algoritmom v simulacijskem okolju lahko preprosto prenesemo na mesto algoritma, ki je uporabljen na fizičnem vozlišču. Potrebni so le majhni popravki za nastavitev vmesnikov prenosa sporočil z nižjega, komunikacijskega nivoja, na nivo algoritma. Na sliki 4.1 si lahko ogledamo razredni diagram realizacije ogrodja.



Slika 4.1: Razredni diagram ogrodja za porazdeljene aplikacije, v katerem je realiziran algoritem za izračun problema NAJMANJŠI k-CENTER.

Izvedba porazdeljenega algoritma se nahaja v razredu **ProtocolThread**, ki vsebuje razred, ki razširja abstraktni razred **AlgorithmRuntime**. Razred vsebuje lastno nit izvajanja, ki skrbi za branje sporočil, izvajanje in pošiljanje novih sporočil sosednjim vozliščem. Razred **ProtocolThread** poskrbi tudi za začetno konfiguracijo razreda **DKCenterAlgorithm**, ki vsebuje izvedbo algoritma **DKCenter**. Vsaka izvedba algoritma vsebuje razrede za hranjenje (razred **DBuffer**), prejemanje (razred **DReceiver**) in pošiljanje (razred **DSender**) sporočil. Ti razredi vzdržujejo komunikacijo v ustvarjenem prekrivnem omrežju.

Na sliki 4.2 je prikazan diagram interakcije med uporabnikom, ki nadzoruje izvajanje algoritma preko uporabniškega vmesnika k in prekrivnim omrežjem, ki nastane z uporabo ogrodja **DCenter**. Ogrodje poskrbi za prenos sporočil med posameznimi vozlišči, ponuja razrede za nastavitev parametrov algoritma in orodja za nadzor sporočil. Vsako sporočilo med uporabnikovim delom ogrodja in vozliščem prekrivnega omrežja je del konteksta, ki je aktiven, dokler uporabnik ne prejme celotnega števila sporočil. Uporabnik lahko pregleduje kontekste in ima pregled nad sporočili, ki so izostali med izvajanjem algoritma in s tem dobi možnost pregleda (problematičnih) vozlišč. Slednje se je izkazalo kot zelo uporabno pri zagonu algoritma nad sistemom **PlanetLab**, saj so nekatera vozlišča zaradi previsoke izkoriščenosti zelo počasna. Uporabniški



Slika 4.2: Prekrivno omrežje z uporabo ogrodja **DCenter**.

vmesnik se lahko poveže na katerokoli vozlišče prekrivnega omrežja in uporabnik lahko preko vmesnika z ukazi nadzoruje izvajanje in pregleduje stanje vozlišč v omrežju. Ogrodju so dodana tudi prikazovalna orodja za pregled omrežja, tako da ogrodje lahko izriše trenutno stanje vozlišč s povezavami med njimi.

4.4 Raziskovalno omrežje PlanetLab

PlanetLab je raziskovalno omrežje, ki je namenjeno razvoju in testiranju novih omrežnih storitev, protokolov in, kot v našem primeru, tudi porazdeljenih aplikacij in algoritmov. Raziskovalno omrežje PlanetLab sestavlja 1138 vozlišč na 520 lokacijah po celotnem svetu[1]. Od začetka leta 2003 je že več kot 1000 raziskovalcev akademskih ustanov po celotnem svetu in industrijskih raziskovalnih organizacij, ki uporabljalo PlanetLab za razvoj in testiranje novih tehnologij.

Vsi dostopi do PlanetLab-a se vršijo preko t.i. **rezine** (angl. *slice*). Rezina je skupek virov, ki so porazdeljeni preko več strežnikov sistema PlanetLab. Gre za model, ki podpira porazdeljeno virtualizacijo (angl. *distributed virtualization*). Več rezin se sočasno izvaja nad viri porazdeljenega sistema, kjer delujejo kot **vsebnik** (angl. *container*) razmejujoč posamezne račune uporabnikov na strežnikih. Gre za množico navideznih računalnikov (angl. *virtual machines*, VM), ki imajo posebne zahteve pri izvajanju:

- VM so med seboj izolirane,

[1]Stanje zabeleženo na dan 27.11.2010.

- vsak uporabnik ima možnost varne prijave do rezin na prirejenih računalnikih,
- uporabniki lahko namestijo kakršnekoli programske pakete, ki ne morejo vplivati na druge rezine,
- vsako rezino (VM) je mogoče nadzorovati tako, da je možno izdelati poročila nad dogodki, prometom in drugimi statističnimi podatki, ki utegnejo biti zanimivi za uporabnike.

Arhitektura in delovanje celotnega porazdeljenega sistema PlanetLab je podrobneje opisana v delu [46]. Za potrebe testiranja svojega algoritma na sistemu PlanetLab smo napisali množico skript ukazne lupine, ki prevedejo izvorno kodo algoritma ter ogrodja in naredijo paket, ki je primeren za prenos in namestitev (angl. *deployment*) na vozliščih PlanetLab-a. Po prenosu in namestitvi paketa se povežemo na določeno vozlišče iz množice izbranih vozlišč sistema in izvedemo zagon uporabnikove strani ogrodja. Ta nam ponudi posebno ukazno lupino, s katero uporabnik komunicira med prekrivnim omrežjem, prekrivnemu omrežju pa ogrodje omogoča sporočanje stanja vozlišč uporabnikovi strani ogrodja (slika 4.2).

Poglavje 5

Rezultati eksperimentov

V tem poglavju predstavimo rezultate, ki smo jih dobili v eksperimentih z uporabo lokalne hevristike algoritma DKCENTER. Pri primerjavi optimalnih vrednosti rešitev smo si pomagali z delom [14]. Predstavimo različna okolja (statična in dinamična omrežja), kjer smo testirali svoj algoritem. Najprej predstavimo rezultate nad statičnimi omrežji reševanja problema 1-CENTER in nato splošnejšega problema NAJMANJŠI k-CENTER. Za tem predstavimo še rezultate nad dinamičnimi omrežji, ki smo jih ustvarili s pomočjo orodja BRITE na podoben način kot v primeru problema 1-CENTER. Dinamičnost omrežij smo simulirali znotraj simulacijskega okolja PeerSim.

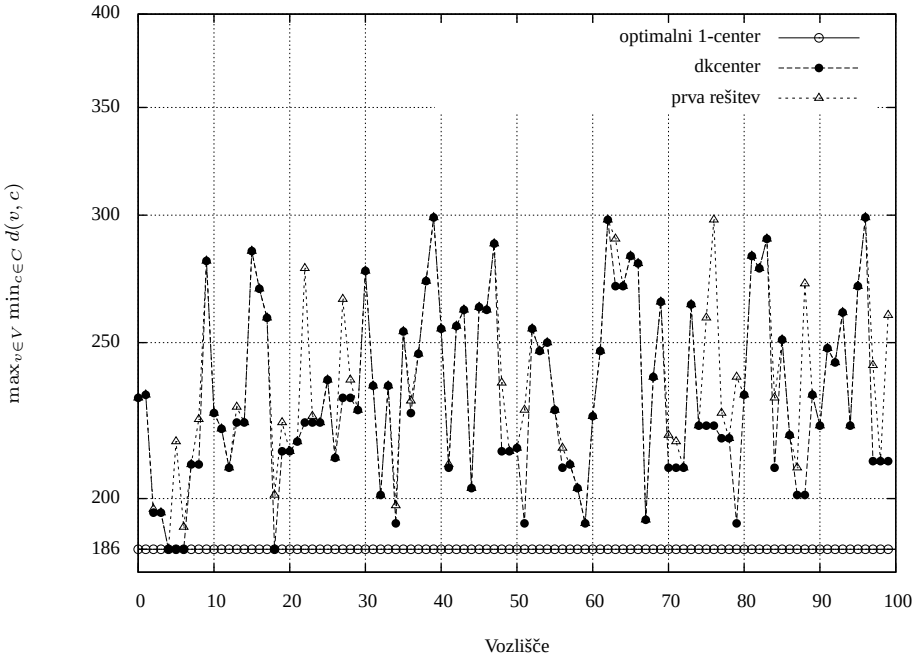
5.1 Simulacija statičnih omrežij

Simulacije smo izvedli v simulacijskem okolju PeerSim, ki omogoča simulacijo tako statičnih kot dinamičnih omrežij. Zaradi lažje primerjave že obstoječih rešitev, ki jih najdemo v literaturi [36, 38], smo si izbrali omrežja iz knjižnice OR-LIB[5]. Poleg tega smo uporabili univerzalni generator omrežij BRITE, s katerim smo ustvarili sintetična omrežja, ki smo jih prikazali in preverili tudi vizualno — BRITE priredi koordinate vozliščem in uteži povezavam med vozlišči tako, da prirejene uteži ponazarjajo oddaljenost med vozlišči in s tem ohranjajo trikotniško neenakost.

Na slikah v naslednjih podpoglavjih ordinatna os predstavlja vrednost cenilne funkcije, t.j. $\max_{v \in V} \min_{c \in C} d(v, c)$, kjer je V množica vseh vozlišč v omrežju in $C \subseteq V$ množica centrov v omrežju.

5.1.1 Rezultati problema 1-CENTER

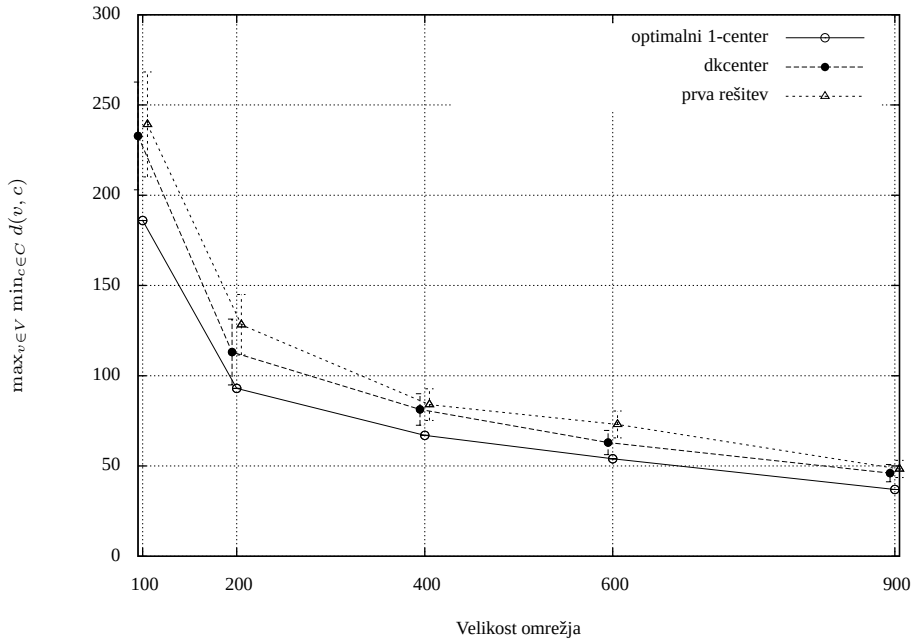
Najprej smo opravili teste v primeru reševanja problema 1-CENTER. Ta problem je optimalno rešljiv z naivnim zaporednim algoritmom v času $O(n^3)$. S porazdeljenim algoritmom DKCENTER je reševanje problema 1-CENTER ekvivalentno reševanju problemu NAJMANJŠI k-CENTER. Problem smo v našem primeru reševali z iterativnimi premiki centra na primernejša vozlišča, ki prevzamejo vlogo ponudnika storitve in s tem zmanjšajo cenilko pogoja 3.1 za premik. Na sliki 5.1 je prikazana primerjava rešitev problema 1-CENTER, pridobljenimi z algoritmom DKCENTER, z rešitvami, ki smo jih dobili z uporabo optimalnega algoritma za reševanje problema 1-CENTER in algoritmom z naključnimi postavitvami centrov. Na sliki 5.1 prikazujemo le meritve nad omrežjem `pmed1.txt` iz knjižnce OR-LIB. Omrežje `pmed1.txt` ima 100 vozlišč in 200 povezav ter je tipični primer prevladujoče oblike rezultatov v primeru reševanja problema 1-CENTER.



Slika 5.1: Slika prikazuje rezultate uporabe hevristike v algoritmu DKCENTER nad omrežjem `pmed1.txt` (100 vozlišč, 200 povezav) knjižnice OR-LIB. Črta **optimalni 1-center** prikazuje optimalne rezultate. Črti **prva rešitev** in **dkcenter** prikazujeta vrednosti, ki jih dobimo brez in z uporabo hevristike, torej brez in s premikom centra. Abscisna os predstavlja indeks vozlišča, ki je bil začetni center.

Za rezultate naključnega algoritma smo izbrali kar začetno postavitev centra v omrežje, torej prvo rešitev, ki jo dobimo z algoritmom DKCENTER. Opazimo, da se ta algoritem le malokrat približa optimalni vrednosti, to je le v 4 primerih od 100 različnih postavitev centra v omrežju s 100 vozlišči. V 25 primerih algoritem najde boljšo lokacijo za center kot naključni algoritem. V ostalih primerih je center obtičal v lokalnem minimumu — prvotni razmestitvi centra. Izkaže se, da algoritem DKCENTER ni primeren za izračun problema 1-CENTER. Za reševanje problema pri $k = 1$ bi bilo primerneje izbrati npr. algoritem, ki je opisan v delu [12] ali kombinacijo algoritmov za računanje najkrajših poti med vsemi pari vozlišč [8] in za izvolitev vodje [34].

Na sliki 5.2 so prikazane vrednosti rešitev nad različnimi omrežji iz knjižnice OR-LIB: `pmed1.txt`, `pmed10.txt`, `pmed20.txt`, `pmed30.txt` in `pmed40.txt`. Velikosti teh omrežij so po vrsti: 100, 200, 400, 600 in 900 vozlišč (200, 800, 3200, 9800 in 16200 povezav).

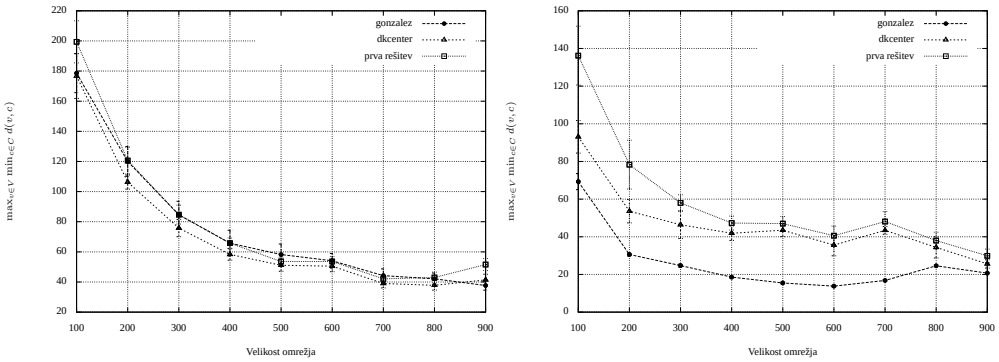


Slika 5.2: Slika prikazuje izboljšanje rezultatov z uporabo algoritma DKCENTER. Črta **optimalni 1-center** prikazuje optimalne rezultate. Črti **prva rešitev** in **dkcenter** prikazujeta vrednosti, ki jih dobimo brez in z uporabo hevristike, torej brez in s premikom centra. Meritve so narejene nad omrežji `pmed1.txt`, `pmed10.txt`, `pmed20.txt`, `pmed30.txt` in `pmed40.txt` knjižnice OR-LIB.

Narejenih je bilo 100 poskusov nad vsakem od omrežij. Algoritem DKCENTER je v vseh primerih po določenem številu iteracij prišel do boljšega rezultata, kakor smo ga dobili z naključnim razmeščanjem centra.

5.1.2 Rezultati problema NAJMANJŠI k-CENTER

V primeru razmeščanja k centrov v porazdeljenem okolju se pokažejo boljši rezultati naše hevristike z uporabo algoritma DKCENTER. Izkaže se, da so dobljeni rezultati primerljivi z 2-aproksimativnimi rešitvami Gonzalezovega algoritma [19]. Na sliki 5.3 imamo predstavljeno primerjavo med rezultati, ki jih dobimo z uporabo našega algoritma proti naključnim razmeščanjem centrov in razmeščanjem centrov po Gonzalezovem algoritmu. Slika 5.3a prikazuje vrednosti rešitve v primeru, kjer je razmerje med številom centrov in celotnim številom vozlišč majhno (število centrov je v vseh primerih 5, število vseh vozlišč se enakomerno povečuje z različnimi izbranimi vhodnimi datotekami omrežij — med 100 in 900). Omrežja, ki smo jih uporabili za slednje teste, so del knjižnice OR-LIB.

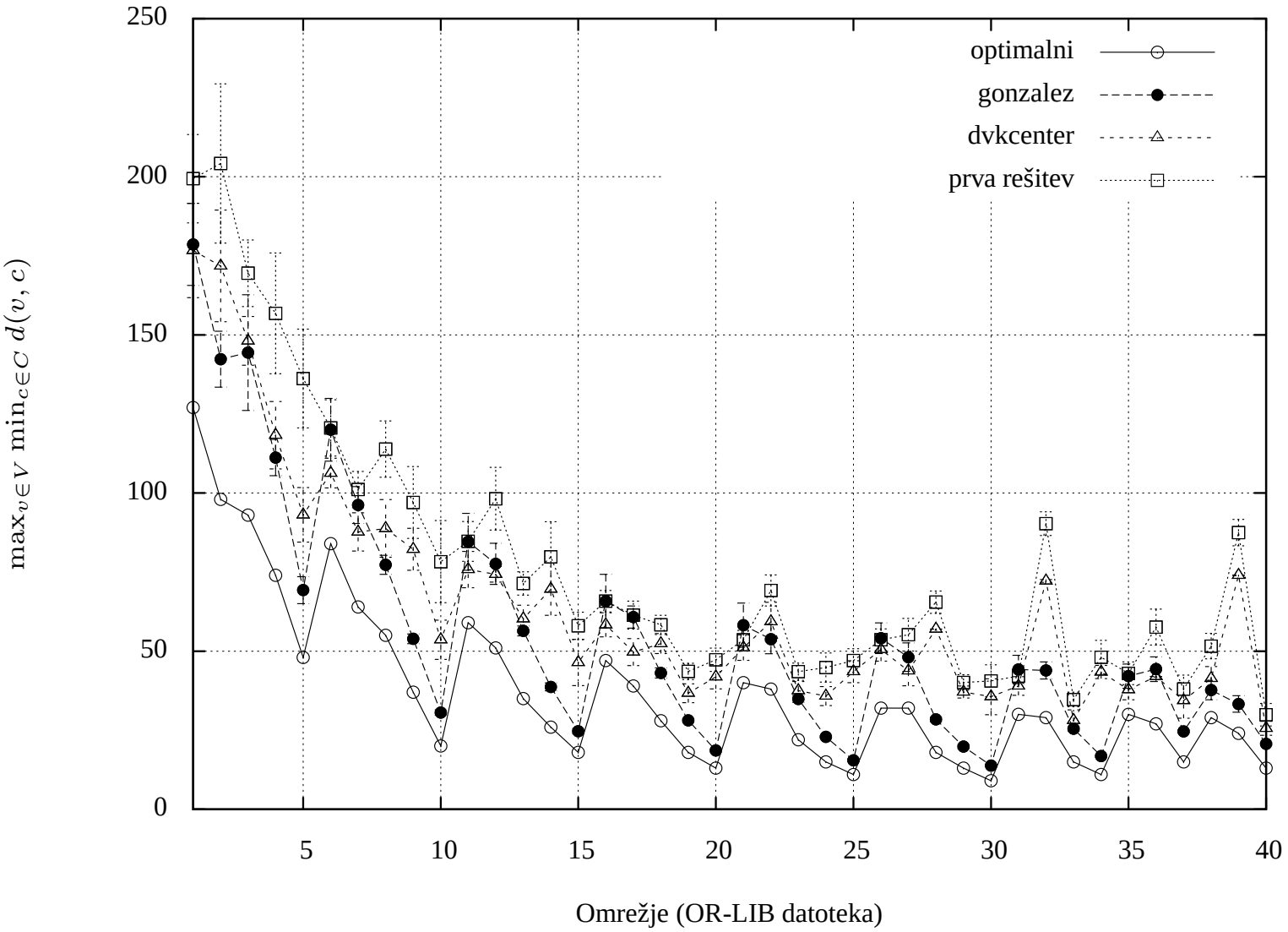


(a) Majhno število centrov — $k = 5$

(b) Veliko število centrov — največji k v seriji omrežij enake velikosti iz knjižnice OR-LIB

Slika 5.3: Sliki prikazujeta primerjavo med rešitvami zaporednega Gonzalezovega algoritma, rešitvami porazdeljenega algoritma DKCENTER in rešitvami tik pred prvim korakom našega algoritma (pred premiki centrov). Leva slika prikazuje rešitve nad majhnim številom centrov v omrežju (pri $k = 5$). Desna slika prikazuje rešitve z največjim številom centrov nad omrežji enake velikosti iz knjižnice OR-LIB.

Na sliki 5.3a lahko opazimo, da rešitve pri manjšem številu centrov sledijo trendu 2-aproksimativnega Gonzalezovega algoritma. V nekaterih primerih



Slika 5.4: Slika prikazuje izboljšanje rezultatov z uporabo hevristike DKCENTER. Črta **optimalni** prikazuje optimalne rezultate. Črti **prva rešitev** in **dkcenter** prikazujeta vrednosti, ki jih dobimo brez in z uporabo hevristike, torej brez in s premiki centrov.

so rešitve, ki jih dobimo po naključni razmestitvi centrov, celo boljše, kot po premikih z uporabo našega algoritma. Na sliki 5.3b lahko razberemo, da v primeru, ko je število centrov v omrežju večje, lahko praktično vedno izboljšamo naključne rešitve z uporabo našega algoritma. Število centrov se na sliki 5.3b spreminja: 33, 67, 100, 133, 167, 200, 140, 80 in 90 po vrsti. Nad vsakim omrežjem s slik 5.3a in 5.3b je bilo napravljeno 20 meritev. Pri tem moramo poudariti slabšo kakovost rešitev našega algoritma v primerjavi z 2-aproksimativnim algoritmom, kar prikazuje slika 5.3b.

Slika 5.4 prikazuje rezultate algoritma DKCENTER nad vsemi omrežji knjižnice OR-LIB. Na sliki so prikazani tudi rezultati, ki jih dobimo s prvo postavitvijo k centrov v omrežje (brez migracije), rezultati zaporednega Gonzalezovega algoritma in optimalni rezultati. Opazimo, da se pri majhnem številu centrov algoritem DKCENTER obnese podobno kot zaporedni Gonzalezov algoritem. Pri večjih omrežjih in večjem deležu centrov se algoritem DKCENTER obnese slabše kot zaporedni Gonzalezov algoritem.

5.2 Simulacija dinamičnih omrežij

S simulacijami nad dinamičnimi omrežji smo raziskali ustavljivost in komunikacijsko zahtevnost algoritma. Simulacije na dinamičnih omrežjih smo razdelili na različne scenarije dodajanja novih vozlišč in na simulacije odstranjevanja obstoječih vozlišč. Vstopanje novih vozlišč v obstoječe omrežje smo se lotili tako, da smo s pomočjo orodja BRITE ustvarili omrežje G velikosti N, kjer smo ob začetku simulacije inicializirali le določen odstotek omrežja. S tem smo ustvarili začetno omrežje $G' = (E', V') \subset G$ in $|N'| = d|N|$, kjer je $d \in [0, 1]$ odstotek vozlišč celotnega omrežja G.

Izbrali smo 4 različna omrežja, nad katerimi smo naredili različne poskuse:

- pri dodajanju vozlišč smo inicializirali začetni delež vozlišč (npr. 0,8) in za tem dodali različne deleže novih vozlišč (0,01, 0,05, 0,1, 0,15, 0,2) in beležili količino ustvarjenega prometa po omrežju,

- pri odstranjevanju smo ustvarili celotno omrežje in za tem odstranili delež obstoječih vozlišč (0,01, 0,05, 0,1, 0,15, 0,2). Ob tem smo beležili ustvarjen promet po omrežju.

Za primer vzemimo omrežje velikosti 100. V prvem primeru so vzpostavili omrežje velikosti 80 vozlišč in nato zaporedoma dodajali 1, 5, 10, 15 in 20 vozlišč. V primeru brisanja smo vzpostavili omrežje velikosti 100, nakar smo

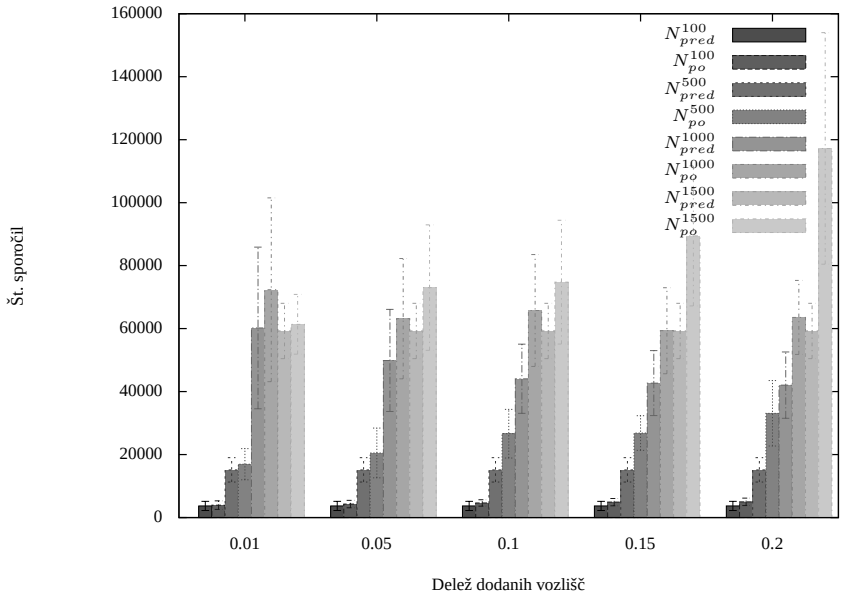
zaporedoma brisali 1, 5, 10, 15 ali 10 vozlišč (poskusi so bili med seboj neodvisni).

Omrežja smo ustvarili po modelu AS (model Waxman) s parametri $\alpha = 0{,}15$, $\beta = 0{,}2$, $m = 2$, kjer sta α in β parametra Waxmanove verjetnosti za povezanost vozlišč. Parameter m pomeni število povezav na novo vozlišče. Vsa tako ustvarjena omrežja imajo natanko $|E| = m|V| = 2|V|$.

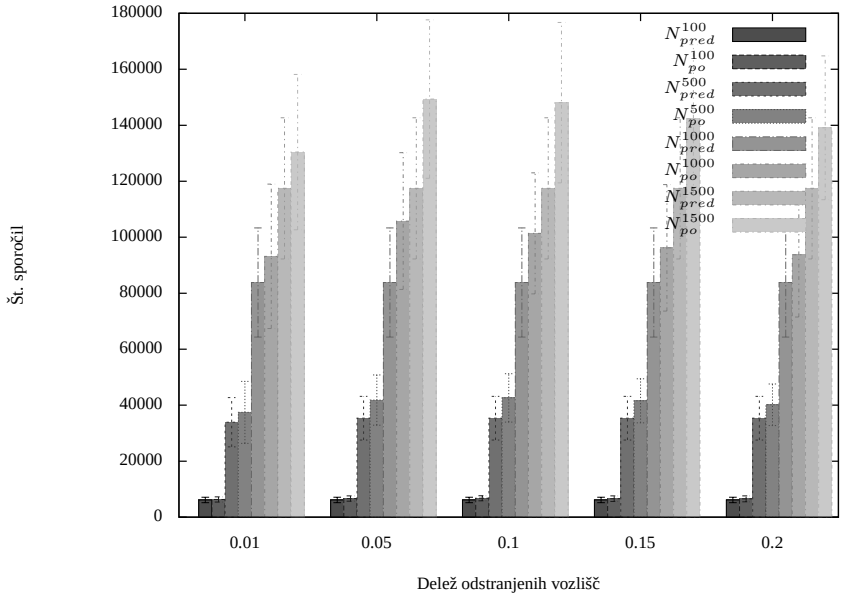
Slika 5.5 prikazuje število sporočil v različnih situacijah pri dodajanju vozlišč že obstoječemu omrežju. Postopek izmere števila sporočil je naslednji: najprej se po delnem omrežju G' porazdelijo centri. Ob pričetku simulacije imamo v omrežju določen delež končnega stanja omrežja, pri čemer velja $|G'| = 0{,}8|G|$. Ko se stanje izniha (po omrežju se prenehajo prenašati sporočila), se v določenem trenutku doda delež novih vozlišč. Tik pred tem dogodkom naredimo posnetek števila doslej poslanih sporočil v omrežju n_1. Po dodajanju vozlišč zopet v določenem trenutku, ko se stanje v omrežju umiri (vozlišča se povežejo na obstoječa vozlišča, centri spoznajo nova vozlišča in se novemu stanju primerno premaknejo na primernejša mesta), zopet naredimo posnetek števila doslej poslanih sporočil n_2 v omrežju. V omrežjih na slikah 5.6 imamo vedno $k = 5$ centrov. Za vsak odstotek novih sporočil (prikazan na abscisi) naredimo meritve za 4 različna omrežja: N^{100}, N^{500}, N^{1000}, N^{1500}, kar pomeni vozlišča z 100, 500, 1000 in 1500 vozlišči (200, 1000, 2000 in 3000 povezav po vrsti). Grafi, označeni z $N^n_{\{pred,\,po\}}$, predstavljajo število sporočil pred ali po dodajanju oz. odstranjevanju vozlišč v omrežju z n vozlišči. Vsako situacijo smo simulirali 20-krat, kar skupno pomeni 400 simulacij.

Na sliki 5.6a je prikazana primerjava deležev ustvarjenega prometa na različnih začetnih omrežjih pri dodajanju vozlišč. Delež dodatnih sporočil je enak razmerju med številom sporočil, ki je bilo potrebno za izgradnjo omrežja v trenutku tik pred dodajanjem vozlišč — n_1 — in tik pred koncem simulacije — n_2. Graf na sliki 5.6a prikazuje $\frac{n_2}{n_1}$. Opaziti je, da se delež dodatnih sporočil monotono veča z deležem dodanih vozlišč v primeru dodajanja, kar je pričakovano. Zmoti dejstvo, da se je razmerje pri N^{1000} z deležem 0,15 nekoliko zmanjšalo v primerjavi z deležem 0,1 vendar ugotovimo, da smo v primeru z deležem 0,1 imeli v povprečju več premikov centrov kot v primeru z deležem 0,15 ($11{,}5 \pm 3{,}5$ v primerjavi z $9{,}5 \pm 2$). Pričakujemo, da se ob dodajanju vozlišč ustvari več prometa pri večjem deležu dodanih vozlišč. To potrjujeta oba grafa 5.5a in 5.6a, ki prikazujeta dodajanje.

Na sliki 5.6b je prikazana primerjava deležev novoustvarjenega prometa pri odstranjevanju vozlišč na različnih omrežjih. Tu ugotovimo trend padanja deleža dodatnih sporočil, ki so potrebni za vzdrževanje rešitev problema NAJMANJŠI k-CENTER. Slednje razložimo s tem, da smo v vseh testih od-



(a) Dodajanje vozlišč

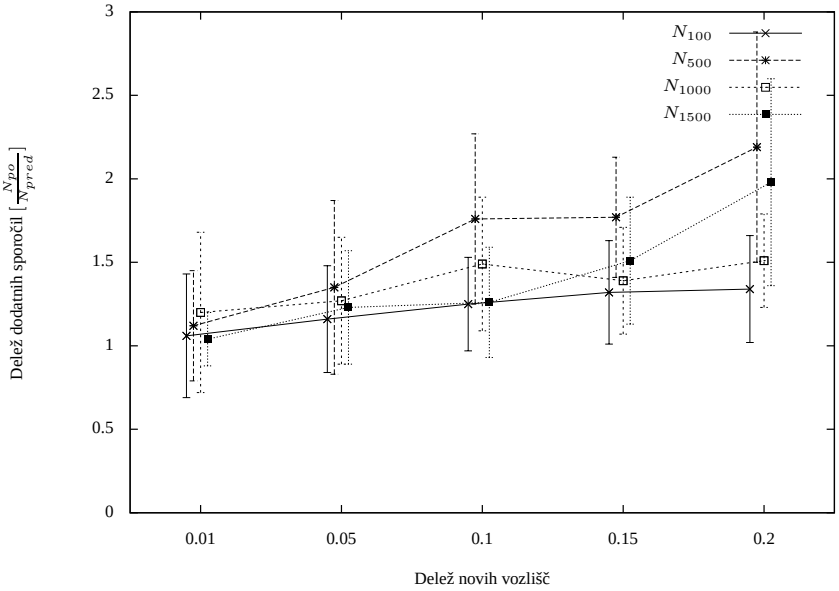


(b) Odstranjevanje vozlišč

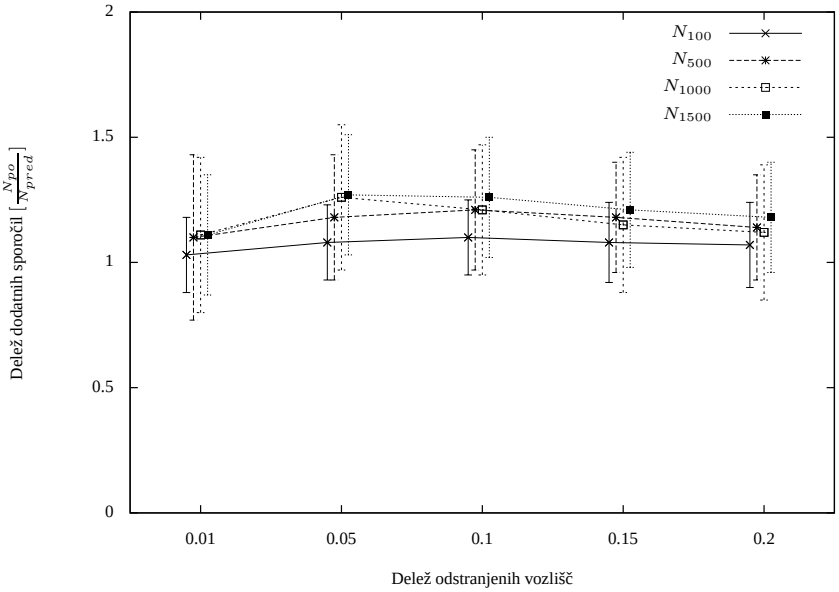
Slika 5.5: Sliki prikazujeta različne situacije pri dodajanju (zgornja slika) in odstranjevanju (spodnja slika) vozlišč v omrežjih različnih velikosti. Na abscisi je prikazanih 5 različnih deležev števila vozlišč, ki smo jih dodali oz. odstranili. Ordinatna os predstavlja število sporočil.

stranjevali naključna vozlišča. Z naključnim odstranjevanjem smo še vedno vzdrževali za naš algoritem sprejemljivo rešitev (s tem se situacija centrov v omrežju ni spreminjala). Opazili smo namreč, da je v teh primerih prišlo do manj premikov centrov.

Poskusimo oceniti, koliko prometa se ustvari pri 20-odstotnem deležu novih vozlišč (zadnji stolpec slike 5.5a). Povprečna velikost sporočila naj bo 1024 bajtov (1 kilobajt), velikost omrežja 1500 vozlišč s 3000 povezavami. Dodajanje povzroči približno 60.000 novih sporočil, kar v povprečju pomeni približno 20 novih sporočil na povezavo. Če predpostavimo, da je danes 1 Mbit/s ($\approx$ 125 kB/s) že kar standardna povezava, nam dodajanje povzroči 16-odstotno povečanje prometa po povezavi (če bi se sporočila zvrstila v sekundi). Ker pa se vsa dodatna sporočila ne prenesejo v trenutku, temveč preko daljšega časovnega obdobja (pri večjem omrežju in ob upoštevanju latenc okrog 60s), lahko trdimo, da predstavlja dodatni promet malo dodatnega bremena za obstoječo infrastrukturo.



(a) Dodajanje vozlišč - delež novih sporočil



(b) Odstranjevanje vozlišč - delež novih sporočil

Slika 5.6: Zgornja slika prikazuje razmerje med številom sporočil, ki so bili potrebni za izgradnjo začetnega omrežja, in med končnim številom sporočil. Podobno kakor zgornja slika tudi spodnja slika prikazuje razmerje med številom sporočil ob različnih časovnih trenutkih.

5.3 Rezultati na sistemu PlanetLab

Raziskovalno omrežje PlanetLab je podrobneje opisano v poglavju 4.4. Za testiranje algoritma smo izbrali množice vozlišč različnih velikosti: 10, 20, 30 in 40 vozlišč ($S_{\{10,20,30,40\}}$, $|S_i| = i$). Zanimal nas je čas izgradnje prekrivnega omrežja in reševanja problema NAJMANJŠI k-CENTER nad izbranem omrežjem. Preden smo izvedli meritve, smo s pomočjo največje množice vozlišč S_{40} pomerili **povprečni odzivni čas** (angl. *round trip time*) med vsemi pari vozlišč množice S_{40}. Slednji znaša

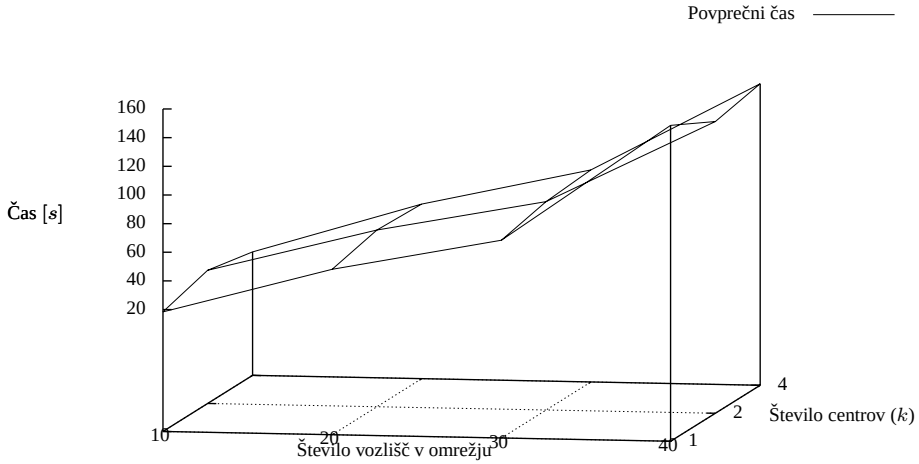
$$252ms \pm 597ms\,.$$

Opaziti je veliko standardno deviacijo, kar kaže na težavo uporabe raziskovalnega sistema PlanetLab. Dejstvo je, da na izbranih vozliščih vzporedno poteka veliko poskusov drugih uporabnikov omrežja, kar povzroča obremenitev vozlišč. Določena vozlišča so problematična, iskanje primernejših (hitrejših in manj obremenjenih vozlišč) pa je težavno.

N	K	Povprečje (ms)	St.dev. (ms)
10	4	21.224,67	2.212,67
20	4	46.893,34	22.720,85
30	4	83.032,65	21.987,34
40	4	145.445,55	44.639,45

Tabela 5.1: Meritve časa izvajanja algoritma na raziskovalnem omrežju PlanetLab.

Tabela 5.1 prikazuje čase izvajanja algoritma DKCENTER. Meritve smo izvedli nad vsemi množicami vozlišč $S_{\{10,20,30,40\}}$ pri $k = 4$, kjer smo ugotovili, da je čas izvajanja algoritma pri teh množicah vozlišč na sistemu PlanetLab premo sorazmeren z velikostjo omrežja, kar lahko opazimo na sliki 5.7.



Slika 5.7: Časi, ki so potrebni za izračun rešitev na raziskovalnem omrežju PlanetLab. Meritve so narejene nad omrežji $S_{\{10,20,30,40\}}$, kjer $k \in \{1, 2, 4\}$.

Poglavje 6

Sklepne ugotovitve

Omrežja enakovrednih subjektov so se v zadnjih nekaj letih izkazala za tehnologijo, ki je omogočila razvoj storitev, ki so jih uporabniki privzeli in jih uporabljajo praktično vsak dan doma in na delovnih mestih. Znanstveniki, podjetja, organizacije in posamezni uporabniki uporabljajo tako odprtokodne rešitve kot plačljive storitve za objavljanje svojih izdelkov in posodobitev paketov programske opreme v omrežju. Vse več je aplikacij, ki izkoriščajo najsodobnejše tehnologije omrežij enakovrednih subjektov za svoje delovanje delovanje na področju **deljenja virov in objavljanje datotek** (Gnutella, FastTrack, BitTorrent), **internetne telefonije** (Skype), **prostovoljno računstvo** (SETI@home, BOINC) in na drugih področjih (npr. multimedija). V teh aplikacijah ni zaslediti uporabe mehanizma za samodejno prilagajanje dostopnih točk streženja storitev, kar pušča veliko raziskovalnega prostora na tem področju.

V pričujočem delu smo razvili nov porazdeljen algoritem za izračun problema NAJMANJŠI k-CENTER nad porazdeljenimi omrežji tipa enakovrednih subjektov. Algoritem upošteva dinamičnost omrežja ter se skozi čas prilagaja tako, da vzdržuje status centra tistih vozlišč, ki so primernejši za gostitev porazdeljene aplikacije ali porazdeljene storitve. Z meritvami smo pokazali, da se algoritem DKCENTER obnese precej bolje pri majhnih k, kjer v primerjavi z obstoječimi 2-aproksimativnimi rešitvami daje podobne rešitve. Problemi nastanejo pri večjih omrežjih in večjih vrednostih parametra k, kjer se rešitev nekoliko izrodi. Nadgradili smo tudi delo [12], kjer smo dopolnili in izboljšali ogrodje DCenter, s katerim smo nov algoritem implementirali v realnem porazdeljenem okolju PlanetLab. V omrežju PlanetLab smo testirali omrežja do velikosti 40 vozlišč, kjer smo ugotovili, da izgradnja in migracija centrov v realnem sistemu do trenutka končne rešitve traja do 160 sekund. Dodajanje in

odstranjevanje komunikacijsko in posledično časovno ni zahtevno — odvisno je od števila novih oz. odhajajočih vozlišč. Izkazalo se je, da je algoritem stabilnejši in komunikacijsko manj zahteven v primeru odstranjevanja vozlišč (od 20 do 50 odstotkov več prometa sporočil v omrežju) kot v primeru dodajanja vozlišč (do dvakrat več prometa sporočil v omrežju).

V pregledani literaturi smo zasledili en sam porazdeljen algoritem, ki rešuje problem najmanjšega k-CENTRA, to je porazdeljeni Gonzalezov algoritem [9]. Avtoji so v tem delu privzeli uporabo usmerjevalnih tabel, kar zahteva predhodno komunikacijo pred zagonom algoritma. V primerjavi s to rešitvijo se naš algoritem obnese bolje v smislu komunikacije, saj v primeru odpovedi vozlišča naš algoritem ne poplavlja celotnega omrežja — algoritem obvesti samo sosednja vozlišča.

Sklepne ugotovitve in težave, na katere smo naleteli med implementacijo algoritma:

- obstajajo primeri, kjer z algoritmom lahko zaidemo v težave (lokalni ekstremi).
- Problemi nastajajo pri velikih omrežjih, zato je potrebno izboljšati lokalnost reševanja (npr. pristop z vzorčenji, naključnimi sprehodi, optimizacija z naključnimi sprehajalci).
- Na mestu bi bila tudi posplošitev reševanja problema k-CENTRA na splošnejši problem OSNOVNI PROBLEM RAZMEŠČANJA.
- Na realnih sistemih je potrebna sinhronizacija med centri, saj je potrebno stanja storitev sinhronizirati.
- Samodejno ugotavljanje števila centrov v omrežju in njihova optimizacija.
- Reševanje problema 1-CENTER: hevristika se obnese zelo slabo v tem primeru. Potrebno bi bilo poiskati način, kako najti boljšo lokacijo skozi čas npr. z naključnimi postavitvami centra v okolici (uporaba simuliranega ohlajanja ali metodo naključnih sprehajalcev in s tem raziskovanja okolice centrov).
- Izboljšava hevristike v dinamičnih omrežjih.

V nadaljnjem delu vidimo veliko izboljšav nad algoritmom DKCENTER. Z malce izpopolnjenim pristopom ocenjevanja smeri migracij centrov v omrežju bi se lahko izognili lokalnim minimumom in s tem izboljšali hevristiko. Da bi

dosegli izboljšanje, bi morali precej izpopolniti matematični model z novimi lastnostmi poleg globine poddreves. Model bi bilo potrebno tudi posplošiti in upoštevati tudi število odjemalcev v posamezni gruči centrov, obremenjenost centra (zahteve odjemalcev), kvaliteto povezav in cene postavitev [58].

Velikokrat se ne sprašujemo samo po rešitvah (primernih lokacijah), temveč tudi po potrebnem številu centrov v omrežju. V nadaljnjem delu bi se lotili tudi tega problema, kar zahteva mehanizem po sinhronizaciji centrov, s čimer bi v primeru preslabe pokritosti odjemalcev ustvarili nove centre in s tem izboljšali pokritost v primeru prevelikega števila centrov pa bi lahko posamezna centrska vozlišča izločili iz omrežja.

Slike

Tabele

Algoritmi

Literatura

[1] A compendium of np optimization problems. http://www.csc.kth.se/∼viggo/problemlist/.

[2] P. K. Agarwal, M. Sharir, E. Welzl. The discrete 2-center problem. *SCG '97: Proceedings of the thirteenth annual symposium on Computational geometry*, str. 147–155, New York, NY, USA, 1997.

[3] N. Antonopoulos, G. Exarchakos, M. Li, A. Liotta. *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies, and Applications*. Information Science Reference - Imprint of: IGI Publishing, Hershey, PA, 2010.

[4] A.L. Barabási, R. Albert. Emergence of scaling in random networks. *Science*, zv. 286, št. 5439, str. 509, 1999.

[5] J. E. Beasley. A note on solving large p-median problems. *European Journal of Operational Research*, zv. 21, št. 2, str. 270–273, August 1985.

[6] M. L. Brandeau, S. S. Chiu. An overview of representative problems in location research. *Management Science*, zv. 35, št. 6, str. 645–674, 1989.

[7] I. Brunkhorst, D. Olmedilla. Interoperability for peer-to-peer networks: Opening p2p to the rest of the world. *Innovative Approaches for Learning and Knowledge Sharing*, , str. 45–60, 2006.

[8] M. Bui, F. Butelle, C. Lavault. A distributed algorithm for constructing a minimum diameter spanning tree. *Journal of Parallel Distributed Computing*, zv. 64, št. 5, str. 571–577, 2004.

[9] S. Chimalwar, T. Pasrija, S. Rao. Finding k-centers in asynchronous distributed systems. *Computers and Their Applications*, str. 319–324, 2007.

[10] R. Cohen, B.V. Patel, F. Schaffa, M. Willebeek-LeMair. The sink tree paradigm: connectionless traffic support on atm lan's. *Networking, IEEE/ACM Transactions on*, zv. 4, št. 3, str. 363 –374, jun 1996.

[11] P. Crescenzi, V. Kann. A compendium of NP optimization problems, 1998.

[12] A. Černivec. Ogrodje za realizacijo porazdeljenih algoritmov v sistemu planetlab. Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2007. Diplomsko delo.

[13] A. Černivec, M. Artač, B. Slivnik. Ogrodje za zagotavljanje kakovosti odjemalcem v porazdeljenem okolju. *Sodobne tehnologije in storitve OTS2009 zbornik štirinajste konference*, Number 14, str. 179–191, junij 2009.

[14] A. Černivec, D. Vladušič, B. Slivnik. Load-balanced parallel solver for the minimum k-center and related problems. *v zborniku Proceedings of the IASTED International Conference*, Volume 641, Innsbruck, Austria, september 2009.

[15] Z. Drezner, H. W. Hamacher. *Facility location: Applications and Theory.* Springer Verlag, 2004.

[16] D. Eppstein. Faster construction of planar two-centers. *SODA '97: Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, str. 131–138, Philadelphia, PA, USA, 1997.

[17] C. Frank, K. Römer. Distributed facility location algorithms for flexible configuration of wireless sensor networks. *Proceedings of the 3rd IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS 2007)*, str. 124–141, Santa Fe, NM, USA, 2007.

[18] M. R. Garey, D. S. Johnson. *Computers and intractability.* Freeman San Francisco, 1979.

[19] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, zv. 38, str. 293–306, 1985.

[20] J. L. Guillaume, M. Latapy, D. Magoni. Relevance of massively distributed explorations of the internet topology: Qualitative results. *Computer Networks*, zv. 50, št. 16, str. 3197 – 3224, 2006.

[21] A. Gupta, K. Tangwongsan. Simpler analyses of local search algorithms for facility location, 2008.

[22] V. Hadzilacos, S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Tehnično poročilo, Citeseer, 1994.

[23] W. H. Hamacher, S. Nickel. Classification of location models. *Location Science*, zv. 6, št. 1-4, str. 229–242, 1998.

[24] D. S. Hochbaum. *Various notions of approximations: good, better, best, and more*. PWS Publishing Co., Boston, MA, USA, 1997.

[25] D. S. Hochbaum, D. B. Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of ACM*, zv. 33, št. 3, str. 533–550, 1986.

[26] R. Z. Hwang, R. C. T. Lee, R. C. Chang. The slab dividing approach to solve the euclidean p-center problem. *Algorithmica*, zv. 9, št. 1, str. 1–22, 1993.

[27] M. Jelasity, A. Montresor, G. P. Jesi, S. Voulgaris. The Peersim simulator. `http://peersim.sf.net`.

[28] G. P. Jesi. Peersim howto: Build a new protocol for the peersim 1.0 simulator, 2005.

[29] O. Kariv, S.L. Hakimi. An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics*, zv. 37, št. 3, str. 513–538, 1979.

[30] P. Karwaczyński, J. Močnik. Ip-based clustering for peer-to-peer overlays. *Journal of Software*, zv. 2, str. 30–37, 2007.

[31] D. Kodek. *Arhitektura in organizacija računalniških sistemov*. Bi-tim, Ljubljana, 2008.

[32] D. Krivitski, A. Schuster, R. Wolff. A local facility location algorithm for sensor networks. *Conference on Distributed Computing in Sensor Systems*, str. 149–158, 2005.

[33] N. Laoutaris, G. Smaragdakis, K. Oikonomou, I. Stavrakakis, A. Bestavros. Distributed placement of service facilities in large-scale networks. *Proceedings of IEEE INFOCOM 2007*, Anchorage, AK, May 2007.

[34] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[35] A. Medina, A. Lakhina, I. Matta, J. Byers. Brite: An approach to universal topology generation. *MASCOTS '01: Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, str. 346, Washington, DC, USA, 2001. IEEE Computer Society.

[36] J. Mihelič. *Hevristično reševanje problemov pokrivanja in razmeščanja*. Magistrsko delo, Fakulteta za računalništvo in informatiko, Ljubljana, 2004.

[37] J. Mihelič, B. Robič. Approximation algorithms for the k-center problem: an experimental evaluation. *Operations research proceedings 2002: selected papers of the International Conference on Operations Research (SOR 2002), Klagenfurt, September 2-5, 2002*, str. 371. Springer Verlag, 2003.

[38] J. Mihelič, B. Robič. Solving the k-center problem efficiently with a dominating set algorithm. *Journal of Computing and Information Technology*, zv. 13, št. 3, str. 225, 2005.

[39] A. Mirzaian. Lagrangian relaxation for the star-star concentrator location problem: Approximation algorithm and bounds. *Networks*, zv. 15, str. 1–20, 1985.

[40] N. Mladenovic, M. Labbé, P. Hansen, E. Commerciales. Solving the p-center problem with tabu search and variable neighborhood search, 2000.

[41] C. Morin. Xtreemos: A grid operating system making your computer ready for participating in virtual organizations. *IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, zv. 0, str. 393–402, 2007.

[42] D. Niculescu. Communication paradigms for sensor networks. *Communications Magazine, IEEE*, zv. 43, št. 3, str. 116 – 122, march 2005.

[43] L. K. Nozick. The fixed charge facility location problem with coverage restrictions. *Transportation Research Part E: Logistics and Transportation Review*, zv. 37, št. 4, str. 281 – 296, 2001.

[44] K. Oikonomou, I. Stavrakakis. Scalable Service Migration in Autonomic Network Environments. *IEEE Journal on Selected Areas in Communications*, zv. 28, št. 1, str. 84–94, 2010.

[45] M. E. O'Kelly. Hub facility location with fixed costs. *Papers in Regional Science*, zv. 71, št. 3, str. 293–306, 1992.

[46] L. Peterson. Planetlab architecture: An overview. Tehnično poročilo, Konzorcij PlanetLab, 2006.

[47] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.

[48] L. Ramaswamy, B. Gedik, L. Liu. A distributed approach to node clustering in decentralized peer-to-peer networks. *IEEE Transactions on Parallel and Distributed Systems*, zv. 16, št. 9, str. 814–829, September 2005.

[49] R. Rana, D. Garg. Heuristic approaches for k-center problem. *Advance Computing Conference, 2009. IACC 2009. IEEE International*, str. 332 –335, 6-7 2009.

[50] B. Robič. *Aproksimacijski algoritmi.* Založba FR in FRI, 2009.

[51] M. Sharir. A near-linear algorithm for the planar 2-center problem. *Discrete Computational Geometry*, zv. 18, str. 106–112, 1997.

[52] E. Sultanik, W. C. Regli. Stable service placement on dynamic peer-to-peer networks: A heuristic for the distributed k-center problem. *AAAI*, str. 196–201, 2005.

[53] A. Suzuki, Z. Drezner. The p-center location problem in an area. *Location Science*, zv. 4, str. 69–82, 1996.

[54] W. D. Tajibnapis. A correctness proof of a topology information maintenance protocol for a distributed computer network. *Communications of the ACM*, zv. 20, št. 7, str. 485, 1977.

[55] I. Taylor. *From P2P and Grids to Services on the Web.* Springer, 2009.

[56] G. Tel. *Introduction to Distributed Algorithms.* Cambridge University Press, 2nd edition, 2000.

[57] G. Wittenburg, J. Schiller. A survey of current directions in service placement in mobile ad-hoc networks. *PERCOM '08: Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, str. 548–553, Washington, DC, USA, 2008. IEEE Computer Society.

[58] G. Wittenburg, J. Schiller. Service Placement in Ad Hoc Networks. *PIK - Praxis der Informationsverarbeitung und Kommunikation*, zv. 33, št. 1, str. 21–25, 2010.

Stvarno kazalo