Otkrivanje moguće zlouporabe u informacijskim sustavima temeljeno na odstupanjima u vremenskim grafovima

Orel, Ognjen

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Ognjen Orel

OTKRIVANJE MOGUĆE ZLOUPORABE U INFORMACIJSKIM SUSTAVIMA TEMELJENO NA ODSTUPANJIMA U VREMENSKIM GRAFOVIMA

DOKTORSKI RAD

Zagreb, 2017.

Sveučilište u Zagrebu

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Ognjen Orel

OTKRIVANJE MOGUĆE ZLOUPORABE U INFORMACIJSKIM SUSTAVIMA TEMELJENO NA ODSTUPANJIMA U VREMENSKIM GRAFOVIMA

DOKTORSKI RAD

Mentorica: prof. dr. sc. Mirta Baranović

Zagreb, 2017.

University of Zagreb

FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Ognjen Orel

DETECTION OF POTENTIAL MISUSE IN INFORMATION SYSTEMS BASED ON TEMPORAL GRAPH ANOMALIES

DOCTORAL THESIS

Supervisor: Professor Mirta Baranović, PhD

Zagreb, 2017

Doktorski rad izrađen je na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva, na Zavodu za primijenjeno računarstvo

Mentorica: prof. dr. sc. Mirta Baranović

Doktorski rad ima: 159 stranica

Doktorski rad br.: _____

O mentorici

Mirta Baranović rođena je u Zagrebu 1952. godine. Diplomirala je, magistrirala i doktorirala u polju računarstva na Sveučilištu u Zagrebu Fakultetu elektrotehnike i računarstva (FER), 1976., 1983. odnosno 1997. godine.

Od kolovoza 1976. godine radi na ETF-u, kasnijem FER-u, na Zavodu za primijenjenu matematiku, u grupi predmeta Računarske znanosti, a 2005. godine prelazi na novoosnovani Zavod za primijenjeno računarstvo, gdje je danas redovita profesorica. Od 2010. do 2014. godine bila je predstojnica Zavoda za primijenjeno računarstvo. Izvodi nastavu iz predmeta vezanih uz baze podataka, na preddiplomskom, diplomskom i poslijediplomskom studiju te je kao mentor vodila 6 doktorata, 18 magistarskih radova te oko 260 diplomskih radova. Sudjelovala je na tri znanstvena projekta Ministarstva znanosti, obrazovanja i sporta Republike Hrvatske (MZOŠ) te je bila voditeljica istraživačkih projekata: "Postupci za povećanje kvalitete i iskoristivosti podataka" (2002 – 2006) i "Semantička integracija heterogenih izvorišta podataka" (2007 – 2013) koje je financiralo MZOŠ. Objavila je više od 70 radova u časopisima i zbornicima međunarodnih konferencija u području baza podataka, skladišta podataka i poslovne inteligencije, semantičkog weba i geoprostornih baza i tokova podataka.

Prof. Baranović članica je stručnih udruga IEEE, ACM, MIPRO i CIGRE. Sudjeluje u međunarodnom programskom odboru znanstvene konferencije te sudjeluje kao recenzentica u većem broju konferencija i časopisa. Od 2003. do 2005. godine bila je Počasna predsjednica European Federation of National Maintenance Societies (EFNMS), a od 2005 do 2009. članica Upravnog odbora EFNMS-a. Godine 1997. nagrađena je srebrnom plaketom "Josip Lončar" FER-a za posebno istaknutu doktorsku disertaciju. Godine 2014. nagrađena je zlatnom plaketom FER-a "Josip Lončar".

About the Supervisor

Mirta Baranović was born in Zagreb in 1952. She received B.Sc., M.Sc. and Ph.D. degrees in computing from the University of Zagreb, Faculty of Electrical Engineering and Computing (FER), Zagreb, Croatia, in 1976, 1983 and 1997, respectively.

From August 1976 she has been working at what is now the Faculty of Electrical Engineering and Computing, Department of Applied Computing, first as teaching assistant, and now as full professor and head of department. She teaches database-related subjects at the bachelor, graduate and post-graduate level, and she mentored 6 PhD theses, and about 260 graduation theses. She participated in 3 scientific projects financed by the Ministry of Science, Education and Sports of the Republic of Croatia (MZOŠ). Currently she is leader of the research project: "Semantic Integration of Heterogeneous Data Sources" financed by the MZOŠ. She has publi-

shed more than 70 papers in journals and conference proceedings in the area of databases, data warehouses and business intelligence, semantic web and geospatial databases and data streams.

Prof. Baranović is member of IEEE, ACM, MIPRO and CIGRE. She has participated in an international conference programme committee and she has been serving as technical reviewer for various conferences and journals. From the year 2003 to 2005 she was Honorary President of the European Federation of National Maintenance Societies (EFNMS), and from 2005 to 2009 she was a member of the Board of Directors of EFNMS. She received silver medal for an outstanding Ph.D. thesis in 1997. and she received gold medal "Josip Lončar" from FER in the year 2014.

posvećujem dragoj baki
Ivanki Štark Ivančić
koja me je naučila čitati, pisati i računati
i još mnogo toga

Zahvala

Dugotrajno istraživanje i proces izrade disertacije izvan radnog vremena zahtijevaju veliku količinu koncentracije, odricanja i vremena tijekom kojeg su, uz uspone i napretke, neminovni i brojni padovi, krize i pogreške. Stoga se najprije zahvaljujem svojoj obitelji na njihovoj podršci, razumijevanju i ljubavi kroz sve godine mog obrazovanja. Također se od srca zahvaljujem i svim prijateljima koji su me uz svoj učestali interes podržavali i hrabrili u ovom pothvatu.

Mentor nedvojbeno obavlja vrlo važan dio formiranja svakog studenta. Kroz našu suradnju, od dodiplomskog i magistarskog studija, višegodišnjeg rada na zajedničkim projektima, do ove disertacije, prof. dr. sc. Mirta Baranović je postala puno više od mentorice. Kao istinskoj prijateljici, ovim joj se putem zahvaljujem na svim razgovorima, razmišljanjima, podršci, sugestijama, teškim pitanjima i svemu što me je naučila još od ranih studentskih dana.

Iskreno se zahvaljujem i članovima povjerenstva za ocjenu rada, prof. dr. sc. Damiru Kalpiću, prof. dr. sc. Robertu Mangeru i doc. dr. sc. Igoru Mekteroviću na njihovim komentarima, pitanjima i prijedlozima koji su zasigurno učinili ovu disertaciju jasnijom i točnijom.

Na kraju, zahvaljujem se Sveučilišnom računskom centru Sveučilišta u Zagrebu na podršci tijekom studija kao i na ustupljenoj računalnoj infrastrukturi za istraživanje.

Sažetak

Informacijski sustavi često sadrže vrijedne podatke i procese koje je moguće zlouporabiti na različite načine. U višekorisničkom sustavu u kojem korisnici mogu imati različite uloge, putem kojih su im dodijeljene različite ovlasti, moguće su složene zlouporabe pri kojima nitko od korisnika ne prekoračuje svoje ovlasti. Ipak, zajedničkim djelovanjem koje je organizirano izvan sustava mogu prouzročiti štetu i steći direktnu ili indirektnu korist.

Ovakav oblik unutarnjih prijetnji sustavima, u kojima organizirano sudjeluje veći broj autoriziranih korisnika, a koji ne prekoračuju dodijeljene im ovlasti, uglavnom nije dovoljno istražen. Većina istraživanja se bavi identifikacijom individualnih unutarnjih prijetnji, dok se oni koji promatraju višekorisničke prijetnje uglavnom odnose na pronalazak poznatih uzoraka zlouporaba sustava. U ovom radu je predložena općenita metoda za pronalazak mogućih zlouporaba sustava neovisno o semantici podataka i poznavanju poslovnih procesa sustava. Metoda se temelji na postojanju snimljenog traga (povijesti podataka) relacijske baze podataka informacijskog sustava. Predloženi su algoritmi za konverziju relacijskih podataka u grafove, za konverziju snimljenog traga relacijske baze podataka u potpuno vremenski određeni graf, za pronalazak čestih podgrafova tog grafa te za pronalazak manjih odstupanja od tih čestih podgrafova. Česti podgrafovi ovog vremenskog grafa predstavljaju uzorke višekorisničkih ponašanja, a manja odstupanja od tih podgrafova koja ponavlja ista skupina korisnika predstavljaju moguće složene zlouporabe sustava koje je potrebno dodatno istražiti.

Temeljem implementacije i testiranja ocijenjeno je da predložena metoda prepoznaje moguće zlouporabe sustava. Predloženi model potpuno vremenski određenog grafa i algoritmi za konverziju relacijskih i vremenskih relacijskih podataka u grafove, pronalazak čestih vremenskih podgrafova i usporedbu vremenskih grafova su iskoristivi za opću namjenu.

Ključne riječi: vremenski graf, grafovska baza podataka, relacija-u-graf, česti podgraf, česti vremenski podgraf, usporedba grafova, zlouporaba informacijskog sustava, unutarnja prijetnja, algoritam

Detection of Potential Misuse In Information Systems Based On Temporal Graph Anomalies

Organizations, companies and systems (e.g. higher education system) often rely on the underlying information system in performing their businesses. These are multi-user information systems, where each user has their role which defines specific set of permissions. Valuable data and processes implemented in the information system might be misused in different ways. There are external threats to information systems, coming from outside of an organization or a system. Those threats are mostly covered using access control, user authorization and authentication etc. More interesting are internal threats, coming from inside of an organization. Employees or other insiders can misuse the system to gain more privileges, steal intellectual property, perform sabotage etc. Most of the related work regarding internal threats is aimed towards identification of one internal individual trying to gain more privileges. But what if that individual cooperates with another, who has different permissions on the system? In that way, they can "join" their privileges and perform a complex misuse without any of them overstepping their permissions. This kind of internal threats, where multiple users act coordinately and do not overstep their permissions, is not sufficiently researched. Most research work is considered with identification of individual internal threats, while the works regarding multi-user threats mainly try to identify known threat patterns.

This thesis proposes a general method for identification of potential organized multi-user misuses of information systems without overstepping any given privileges. Method is independent of data semantics and business rules familiarity. Research is aimed towards identification of multi-user behaviour patterns and deviations from those patterns. Those deviations, if repeated given number of times, in the same manner by the same group of users, represent potential complex misuse which needs to by further investigated by the system security officer/team.

Identification of multi-user behaviour patterns is formed as a frequent temporal subgraph mining problem, while deviation identification is formed as a temporal graph comparison problem. Given the presumption that information system uses relational database to store data and implement some business processes, the data source for the method is relational database audit trail, which contains information for each operation performed in the database. In order to be used in graph algorithms, this audit trail is first represented as form of a temporal graph.

Chapter 1 of this thesis gives the introductory information about the research subject and the problem at hand. The review of the current research in the field of internal threat detection is given and the direction of the research is set. Since lot of the work in the research concerns relational databases and graphs, main definitions of the terms regarding these two fields are given. Regarding relational databases, the terms *database*, *database management system*, *re-*

lational database, *entity*, *attribute*, *domain*, *relation*, *tuple*, *attribute value*, *database schema*, *relational database management system* as well as *audit trail*, are defined. Regarding graphs, the terms *graph*, *directed graph*, *graph element incidence*, *graph union*, *graph connectedness*, *adjacency matrix*, *graph isomorphism*, *subgraph*, *subgraph isomorphism*, and *minimum image based support* are defined.

Chapter 2 deals with graph databases and connection between graph and relational databases. Informal and formal definition of a graph database is given. Previous research of graph database models is presented. Proposed graph languages are presented, grouped in four categories: visual languages, SQL-like languages, formal languages and semantic languages. Some graph APIs (Application Programming Interfaces) are mentioned as well.

Attention is given to NoSQL databases and NoSQL idea in general. Main reasons for NoSQL database emergence are stated. Part of the graph databases in NoSQL ecosystem is discussed. Modern NoSQL graph databases are presented and data models they use are discussed, mainly property-graph model, which is used by many of those databases, and considered a model for a graph database in this dissertation.

Current research of connection between relational databases and graph databases is presented. Tools, algorithms and methods regarding topics such as relational data to graph visualization and exploration, graph generation from relational data, relational database to graph database conversion, and storing graphs in a relational database, are covered. Main algorithms for relational to graph database conversion are recognized, namely RDB2Graph, DB2Graph and R2G.

Since audit trail used in this research is in the form of a relational database itself, it is concluded that, for its conversion to a graph, it would be necessary to create a new relational-to-graph database approach, which will fully leverage the property-graph model.

Chapter 3 considers temporal graphs, recognizing the fact that audit trail contains temporal information which must not be ignored while identifying multi-user behaviour patterns. Dynamic graphs, as a form of graph which changes in time, and the opposite, static graphs, are discussed. Ways of incorporating temporal information in several kinds of graphs are presented. Current research of temporal graphs mainly considers time as a discrete variable, while the works considering time as a continuous variable are limited in a way that only edges are treated as time varying and nodes are not.

Main temporal graph models used in current works are graph series (a series of static graph snapshots in a discrete time moments), adjacency tensor (three dimensional tensor of static adjacency matrices in discrete time moments) and dynamic graphs with time varying edges. This research considers time as a continuous variable and approaches nodes' temporal information

as a vital data as well. Therefore, a new temporal graph model will be proposed, and it will extend the static property-graph model with temporal data.

In this chapter, ways of storing temporal graphs in graph databases and working with them are discussed as well.

Chapter 4 investigates the frequent subgraph mining state of research, given the idea of modelling multi-user behaviour pattern identification as a frequent temporal subgraph mining problem. A frequent subgraph mining algorithm, which will be modified to work with newly proposed temporal graph model, is looked for. Since this model will be the extension of a static graph, both frequent static subgraph mining algorithms and frequent dynamic subgraph mining algorithms may be considered. The main criteria of selecting an algorithm to be modified are: it can work with a single large graph (millions of elements or more), it can find subgraph precisely, it can work in a real-time, and it can be modified to work with temporal data.

Algorithm for frequent subgraph mining of static graphs and main differences between them are presented. Classification of such algorithms is shown: by search pattern (depth-first or breadth-first), by an entry graph (one large graph vs. set of multiple small graphs) and by precision (precise vs. approximate algorithms).

Algorithms for frequent subgraph mining of dynamic graphs are presented as well. Main differencing feature of such algorithms is the temporal graph model they use - dynamic graph with varying edges, temporal graph series, etc.

Given the stated criteria and performance shown, the GraMi algorithm by Elseidy et al. [110] is chosen to be modified for temporal subgraph mining. Hence the GraMi algorithm is described in more detail. Constraint satisfaction problem it employs, main functions and optimizations are presented. Identification of frequent subgraphs by GraMi is also shown in an example.

Chapter 5 deals with graph comparison, given the idea that deviations from multi-user behaviour pattern, under a certain circumstances, might be considered a potential misuse. As with frequent subgraph mining algorithms, various algorithms relating to anomaly detection in both static and dynamic graphs are considered. An algorithm to be modified, to work with a new temporal graph model, is looked for. Main criteria for selection of such an algorithm are: the ability to work with freely defined graph features, good performance regarding various differences between graphs, give a measure of similarity between graphs and the ability of modification.

Different static and dynamic graph comparison and anomaly detection algorithms are presented. Static graph anomaly detection algorithms use graphs with or without labels and properties and give different outputs, like the level of node or subgraph anomaly, or just a binary

classification of an element (anomalous or not). Algorithms that work with dynamic graphs mostly identify events which can be classified as anomalous.

Given the stated criteria and performance shown, the *signature similarity* algorithm for document comparison by Papadimitrou et al. [131] is chosen to be modified for temporal graph comparison. This algorithm is described in more detail and the example of comparing two sentences as a set of weighted characteristics is shown. The modification of this algorithm for use with graphs is discussed.

Chapter 6 brings the new relational-to-graph database conversion algorithm. This algorithm is also proposed and published in [141]. Main features and the contributions of the algorithm are presented: leverage of the property-graph model, independence of the data semantics and conversion without data loss. Ten key concepts and guidelines of the algorithm are stated. Relational database's metadata analysis and its effects are described and the relating functions are presented in detail: *get_migration_order*, *find_cycle_ref_for_table*, *find_all_cycle_ref*.

Creation of the graph database elements is described and the relating functions of the algorithm are presented as well: *table_to_graph* and *populate_gdb*. Complete example of converting relational database with the database schema and the data is shown, resulting with a generated property-graph in a graph database.

Chapter 7 introduces the new model of a temporal graph, *completely-timed graph*. This model is an extension of a property-graph model, where each element of the graph has one or two temporal marks - its start of activity (mandatory) and its end of activity (optional). In this way, this model contains both actual state of the data and its history as well. It also regards temporal information as continuous variable.

A formal definition of completely-timed graph is given and an example of such graph is presented. The means of storing and working with this model inside a modern NoSQL graph databases are also discussed.

Chapter 8 extends the relational to graph database conversion algorithm with the notion of time, thus introducing the audit trail to completely-timed graph conversion algorithm. The concept of auditing data operations using trigger modification method [8] is shortly explained and the example is given. This method produces audit trail in form of a temporal relational database, which should be converted to completely-timed graph.

Main guidelines and the concept of temporal relational database to completely-timed graph conversion algorithm is given. This algorithm uses the same relational metadata analysis to determine graph elements parameters and creation order. Special attention is paid to temporal data and temporal properties of completely-timed graph being generated.

Creation of the completely-timed graph database elements is described and the relating functions of the algorithm are presented as well: *audit_table_to_nodes*, *audit_table_to_edges* and *populate_pvog*. Complete example of converting temporal relational database with the database schema and the data is shown, resulting with a generated completely-timed graph in a graph database.

Algorithm in chapter 6, model of completely-timed graph in chapter 7 and algorithm in chapter 8 jointly represent an original contribution of converting temporal relational data to temporal graph data.

Chapter 9 brings the new frequent completely-timed subgraph mining algorithm. First, a notion of temporal neighbourhood is defined and example of it is given. It represents a connected completely-time subgraph where element's start or end of activity is within a defined interval of its neighbour's activity. *Temporal index*, *standardized temporal neighbourhood* and *standardized temporal neighbourhood of a graph element* are also introduced, defined and explained through examples. They represent a way to identify different temporal neighbourhoods as isomorphic temporal subgraphs.

The means of mapping completely-timed graph's labels, properties and standardized temporal properties of element's neighbourhood, in order to represent completely-timed graph as a graph with only numbered labels, is presented and thoroughly explained. Algorithm is then represented with functions *process_node*, *process_edge* and *find_FTSGs*, each of them being shown and explained in detail. These functions represent the extension of the GraMi algorithm core.

The complete graphic example of identifying frequent completely-timed subgraphs in a completely-timed graph is given at the end of the chapter. The extension of the GraMi algorithm with temporal neighbourhoods and mapping infrastructure represents an original contribution of frequent temporal subgraph mining performed on completely-timed graphs.

Chapter 10 defines the means to compare completely-timed graphs and find anomalies of frequent completely-timed subgraphs. In order to extend signature similarity algorithm, three types of temporal graph weighted characteristics are defined: topological, temporal and property characteristics. Each of these types is assigned with a specific weight. Different ratios of the weights are considered and discussed. Similarity measures are calculated on several slightly different graphs in order to explore the impact of the weight ratio variations. Based on these experiments, a weight ratio is chosen to be used in further completely-timed graph comparisons. A completely-timed graph comparison algorithm is represented with two functions, ϕ and *compare_pvog*, each of them being shown and explained in detail. These functions are the extension of the signature similarity algorithm.

Rest of the chapter focuses on identifying completely-timed subgraph deviations, or anomalies. The notion of *approximately equal completely-timed graphs* as a measure of graph similarity is defined. This notion is further explained by comparing one completely-timed graph to a series of similar graphs.

The completely-timed graph comparison algorithm, together with the measure of graph similarity represents an original contribution of frequent completely-timed subgraph anomaly detection.

Chapter 11 encompasses all new algorithms presented in this thesis to define the complete method for identification of potential misuses. Algorithms presented in chapters 6 to 10 are combined together and presented as a method - audit trail is converted into a completely-timed graph, frequent subgraphs (multi-user behaviour patterns) of that graph are found, and deviations (anomalies) from those subgraphs are found as well. Final step of this method is to analyse anomalies by a behaviour pattern and set of users and count the number each of those has occurred. If a number falls into a predefined range, then that set of anomalies represents a potential misuse, which needs to be further analysed by a system security team.

The terms *anomaly*, *anomaly frequency* and *potential misuse* are formally defined. Method is represented with three functions which use previously defined algorithms, *find_candidates*, *find_anomalies* and *find_misuses*, each of them being shown and explained in detail. Further analyses which need to be done by a security team are also discussed.

Previously proposed algorithms joined and expanded by these additional functions form an original contribution of method for identifying potential misuses in information systems.

Chapter 12 shows implementation and testing of all proposed algorithms and method. Algorithms are implemented in Java, SQL, SPL and Cypher languages. Testing environment contained a generated audit trail, made as a anonymised copy of a real one. In that trail, several different patterns were inserted by hand, representing anomalies which together form a potential misuse. Numbers, screenshots, code listings and textual outputs are presented in order to show the results of each of the proposed algorithms and the whole method. Testing proved the algorithms and the method function correctly.

Chapter 13 summarizes the conducted research and brings a series of conclusions. In the end, future work and possible enhancements are discussed.

In this thesis, method for potential complex misuse detection in information systems is proposed, with a series of underlying algorithms covering fields of relational-to-graph conversion, frequent subgraph mining, temporal graphs, graph similarity measurement and internal fraud

detection. During research, these fields were explored and as a result, some existing algorithms were expanded and new algorithms were proposed.

In conclusion, achieved original scientific contributions are:

- relational database to graph database conversion algorithm, with special emphasis on temporal relational database to completely-timed graph conversion
- frequent completely-timed subgraph mining algorithm
- frequent completely-timed subgraph anomaly detection algorithm
- potential information system complex misuse detection method based on frequent completely-timed subgraph anomalies.

Keywords: temporal graph, graph database, relation-to-graph, frequent subgraph, frequent temporal subgraph, information system misuse, internal threat, algorithm

Sadržaj

Poglavlje 1

Uvod

Organizacije, tvrtke i sustavi (u smislu npr. sustav javne uprave, sustav visokog obrazovanja) se u pravilu u svome poslovanju oslanjaju na informacijske sustave u kojima pohranjuju vitalne podatke svog poslovanja, provode svoje poslovne procese i uz pomoć kojih donose poslovne odluke. Ovakvi su informacijski sustavi uglavnom višekorisnički, pri čemu korisnici mogu imati jednu ili više uloga, putem kojih dobivaju ovlasti za korištenje pojedinog dijela sustava.

Zbog količine podataka i prirode poslovnih procesa, takvi se informacijski sustavi u pravilu temelje na bazama podataka u kojima su pohranjeni ne samo podaci, nego često i veliki dio poslovnih pravila. Ta se pravila implementiraju kroz pravila integriteta i druge aktivne objekte koje baze podataka mogu podržavati, poput okidača ili pohranjenih procedura. Također, unatoč sve većoj popularnosti alternativnih modela baza podataka, i dalje se većina informacijskih sustava oslanja uglavnom na relacijske baze podataka.

Podaci, ali također i uređeni procesi unutar samih organizacija, imaju vrijednost koja se može očitovati na različite načine. Oni mogu sadržavati povjerljive informacije koje imaju financijsku ili poslovnu vrijednost. Također opisuju poslovanje organizacije, načine na koji se pojedini procesi obavljaju, služe za održavanje uređenosti sustava, itd. Zbog tih vrijednosti podaci i povezani procesi unutar organizacija mogu biti izloženi raznim prijetnjama. Primjerice, druga strana (npr. konkurencija, zlonamjerni napadač, ...) može dobiti određenu prednost saznavanjem činjenica iz pojedinih podataka. Druga strana također može utjecati na procese unutar organizacije kojima bi organizacija možda pretrpila štetu, a ta druga strana dobila prednost. Moguće je kompromitiranje podataka i procesa organizacije iz različitih razloga. Pri tom druga strana može biti vanjski dionik (pojedinac, skupina, organizacija) koji nije dio organizacije, ili unutarnji dionik (zaposlenik organizacije, korisnik sustava ili skupina). Dok se u prvom slučaju govori o vanjskim prijetnjama, od kojih se uglavnom, na razini pripadnih informacijskih sustava, brani restrikcijama fizičkog i virtualnog pristupa, u potonjem slučaju se govori o unutarnjim prijetnjama, koje je često teže detektirati, jer dionici sustava ne moraju nužno prekršiti svoje ovlasti kako bi ga zlouporabili.

Unutarnje prijetnje organizacijama od strane autoriziranih korisnika pripadnih informacijskih sustava se mogu klasificirati kao zlonamjerne aktivnosti (sabotaža informacijskog sustava, krađa intelektualnog vlasništva, prijevara, zločin protiv nacionalne sigurnosti) i kao nenamjerni slučajni problemi nastali nepažnjom korisnika [1].

U ovoj disertaciji fokus je na unutarnjim prijetnjama organizacijama, u kojima organizirano sudjeluje veći broj autoriziranih korisnika pripadnog informacijskog sustava, i koji u svom djelovanju ne prekoračuju dane im ovlasti nad tim sustavom.

1.1 Pronalazak unutarnjih prijetnji

Istraživanja u području unutarnjih prijetnji (eng. *insider threat*) se uglavnom odnose na pronalazak mogućih zlouporaba pojedinačnih korisnika sustava [1, 2, 3, 4, 5], dok metode za pronalazak zlouporaba u kojima sudjeluje više korisnika sustava nisu dovoljno istražene. Algoritam Demids, predstavljen u [4] je fokusiran na pronalazak čestih skupova (eng. *Frequent Itemsets*), koji se sastoje od relacija, atributa i vrijednosti kojima korisnici uglavnom pristupaju u naredbama SQL (eng. *Structured Query Language*) jezika koje izvršavaju. U fazi nadzora stvarnog rada korisnika se uspoređuju aktualne SQL naredbe s često korištenima i računa se udaljenost među njima te se na taj način zaključuje je li aktualna naredba moguća anomalija koju je potrebno prijaviti. Izrada profila korisnika ili uloge (u slučajevima sustava koji korisničke dozvole temelje na ulogama) temeljem često obavljanih akcija i usporedba aktualnih akcija s tim profilima je također višekratno korištena ideja detekcije unutarnjih prijetnji sustavima, npr. [6, 7].

Ted i drugi [1] predlažu cijeli niz algoritama za detekciju unutarnjih prijetnji, od kojih se neki odnose i na reprezentaciju podataka u grafovima. U tim se slučajevima promatraju mjere lokaliteta kako bi se otkrile anomalije u grafovima, a anomalije se također odnose na pojedinačne korisnike.

U okviru ovog rada se istražuje mogućnost definiranja i implementiranja metode kojom će se detektirati moguće zlouporabe sustava od strane većeg broja organiziranih korisnika sustava, bez prekoračenja dodijeljenih im ovlasti. Istraživanje se obavlja u smjeru pronalaska uobičajenih obrazaca ponašanja skupina korisnika, što se može promatrati kao "skupni profil" te pronalaska odstupanja pojedinih skupina korisnika od tih obrazaca, pri čemu se za pronalazak uobičajenih obrazaca ponašanja korisnika koriste algoritmi za pronalazak čestih podgrafova unutar grafa, a za odstupanja algoritmi za usporedbu grafova.

Kao izvor podataka za ovu metodu, koristit će se snimljeni trag (eng. *audit trail*) relacijske baze podataka na kojoj se informacijski sustav koji se istražuje temelji. Snimljeni trag ima oblik vremenski obilježene relacijske baze podataka [8]. Istražit će se načini prikaza vremenskih podataka u grafovima te odabrati jedan od postojećih ili implementirati novi koji će biti prikladan

za prikaz ove vrste podataka. Istražit će se i postojeći algoritmi za konverziju podataka iz relacijskog modela u graf i temeljem jednog od njih ili prijedloga novog algoritma, predložit će se modeliranje snimljenog traga relacijske baze podataka u ovaj oblik vremenskog grafa.

U cilju pronalaska obrazaca ponašanja skupina korisnika, obavit će se analiza algoritama za pronalazak čestih podgrafova te prilagoditi jedan od njih za rad s vremenskim grafovima. Također će se analizirati mogućnosti usporedbe grafova u cilju pronalaska razine sličnosti, odnosno odstupanja grafova jednog od drugog, što će značiti odstupanja pojedinih podgrafova od pronađenih obrazaca, odnosno anomalije u ponašanju skupina korisnika. Predložit će se metoda koja će na temelju ovih odstupanja identificirati moguće zlouporabe sustava na koje je potrebno upozoriti osobu zaduženu za sigurnost organizacije, odnosno sustava.

Valja spomenuti dva pristupa koji su u dijelovima slični opisanome. U [9] se stvara graf iz dnevnika sustava (eng. *log*) te se unutar tog grafa traže oni podgrafovi koji odstupaju od unaprijed definiranog procesa (eng. *conformance checking*). Najvažnija razlika je što se postupak obavlja za poznati slučaj, u kojem su poznati poslovni procesi, dok je pristup koji se predlaže u sklopu ove disertacije općenit. Drugi pristup [10] pretražuje povijest grafa u cilju pronalaska zlouporaba u osiguranjima i bankama. Također su poznati uzorci u povijesti grafa koji se traže, a čiji pronalazak signalizira moguću zlouporabu.

1.2 Osnovni pojmovi vezani uz relacijske baze podataka

S obzirom da se kao izvorište podataka za istraživanje koristi snimljeni trag relacijske baze podataka, koji je i sam u obliku relacijske baze podataka te se istražuju i metode konverzije ovih podataka u grafove, ovdje je zbog konzistentnosti i cjelovitosti dan pregled osnovnih pojmova vezanih uz relacijske baze podataka.

Definicija 1. *Baza podataka* (eng. *database*) može se odrediti kao skup povezanih i postojanih podataka. ■

Definicija 2. *Sustav za upravljanje bazama podataka* (SUBP, eng. *Database Management System – DBMS*) je računalni program koji omogućava rad s bazama podataka – definiranje, upravljanje, zauzimanje fizičkog prostora za smještaj podataka, zauzimanje računalnih resursa potrebnih za rad, omogućavanje pristupa korisnicima, posluživanje korisnika, zaštitu od gubitka podataka, sigurnosne mjere zaštite baza podataka te obavljanje internih zadaća. ■

Definicija 3. *Relacijska baza podataka* je baza podataka temeljena na relacijskom modelu podataka (eng. *relational data model*). Relacijski model podataka razvijen je 70-tih godina 20-tog stoljeća i temelji se na formalnoj matematičkoj osnovi [11]. Informacije se u relacijskom modelu opisuju kroz tvrdnje korištenjem predikatne logike. Za opis strukture podataka u relacij-

skom modelu koriste se elementi: entitet, atribut, domena, relacijska shema, relacija, n-torka, shema baze podataka. ■

Definicija 4. *Entitet* (eng. *entity*) je nešto što ima suštinu ili bit i posjeduje značajke po kojima se može razlikovati od svoje okoline. ■

Definicija 5. *Atribut* (eng. *attribute*) je karakteristika entiteta i definira se nad domenom. Atribut se također naziva i stupcem. Atributi se označavaju slovima A, B, C, i sl., a skupovi atributa slovima X, Y, Z. ■

Definicija 6. *Domena* (eng. *domain*) je skup dopuštenih vrijednosti atributa i definira značenje podataka. Domena atributa A označava se s $D = DOM(A)$. ■

Definicija 7. *Relacijska shema* ili intenzija R, predstavlja imenovani skup atributa $\{A_1, A_2, ..., A_n\}$. ■

Definicija 8. *Relacija* (eng. *relation*) ili ekstenzija r definira se nad relacijskom shemom R i predstavlja konačan skup n-torki. Stupanj relacije je broj atributa relacije. Kardinalnost relacije je broj n-torki u relaciji. Relacija se također naziva i tablica. Relacija r definirana nad shemom $R = \{A_1, A_2, ..., A_n\}$ označava se s $r(R)$ ili $r(A_1, A_2, ..., A_n)$. ■

Definicija 9. *n-torka* (eng. *tuple*) t je istinita tvrdnja u kontekstu relacije i sadrži vrijednosti atributa koje odgovaraju atributima iz relacije. N-torka se također naziva i retkom. $t(A_1, A_2, ..., A_n)$ je n-torka definirana na shemi $R = \{A_1, A_2, ..., A_n\}$. $t[X]$ je restrikcija n-torke na skup atributa X. ■

Definicija 10. *Vrijednost atributa* je vrijednost konkretnog atributa u konkretnoj n-torki, pri čemu ta vrijednost mora pripadati domeni atributa. Vrijednost atributa A u n-torki t se označava s $t[A]$. ■

Definicija 11. *Shema baze podataka* je skup relacijskih shema i označava se s $\mathbf{R} = \{R_1, R_2, ..., R_n\}$. Instanca baze podataka $\mathbf{r}$ na shemi baze podataka $\mathbf{R}$ je skup relacija $\{r_1, r_2, ..., r_n\}$, pri čemu je relacija r_i definirana na relacijskoj shemi R_i. ■

Definicija 12. *Sustav za upravljanje relacijskim bazama podataka* (eng. *Relational Database Management System – RDBMS*) je sustav za upravljanje bazama podataka temeljenima na relacijskom modelu podataka. Omogućava izvođenje naredbi SQL jezika nad bazama podataka koje sadrži. ■

Definicija 13. *Snimljeni trag* (eng. *audit trail*) je produkt snimanja traga (eng. *auditing*) [8] - evidentiranja događaja u sustavu za upravljanje bazama podataka i samoj bazi podataka. Događaji mogu biti posljedica korisničkih operacija nad sustavom ili bazom podataka, a odnose se na cjelokupnu djelatnost sustava – pristupanje i izmjene postavki, objekata u bazama podataka, korisničkih podataka, samih podataka u bazama podataka itd. ■

1.3 Osnovni pojmovi vezani uz grafove

Većina postupaka u predloženoj metodi se odnosi na podatke predstavljene grafovima, pa je u ovom potpoglavlju, također radi konzistentnosti i cjelovitosti, dan pregled korištenih pojmova vezanih uz teoriju grafova.

Definicija 14. *Graf* $G = (V, E)$ se sastoji od nepraznog konačnog skupa V koji se naziva vrhovi (eng. *vertices*) i konačnog multiskupa E jednočlanih ili dvočlanih podskupova skupa V, koji se naziva lukovi ili bridovi (eng. *edges*). Lukovi, odnosno bridovi, predstavljaju spojnice vrhova grafa. Ovom definicijom su u grafu dopušteni vrhovi koji su povezani sami sa sobom, kao i višestruke spojnice između dva vrha. Elementi skupa V se označavaju slovima v i w, a elementi skupa E slovom e.

Preciznije, graf je uređena trojka $G = (V, E, \varphi)$, gdje je $\varphi : E \to V \times V$ funkcija incidencije koja svakom bridu e pridružuje dvočlani multiskup vrhova $\varphi(e) = \{v, w\}$. ■

U okviru ove disertacije će se često, zbog sličnih karakteristika koje će dijeliti, vrhovi grafa i njihove spojnice zajednički nazivati *elementi grafa*.

Definicija 15. *Usmjereni graf* $G = (V, E)$ sastoji se od nepraznog konačnog skupa vrhova V i konačnog multiskupa E uređenih parova vrhova iz V. Spojnice vrhova usmjerenog grafa dakle imaju početni vrh i završni vrh te se predstavljaju strelicama i nazivaju lukovi. Spojnice vrhova neusmjerenog grafa se nazivaju bridovi. ■

Definicija 16. Brid ili luk $e = \{v, w\}$ i vrhovi v i w koje on spaja su međusobno *incidentni*. Vrhovi v i w su u ovom slučaju *susjedni vrhovi*. ■

Definicija 17. Ako su $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$ grafovi čiji su skupovi V_1 i V_2 disjunktni, onda je *unija grafova* G_1 i G_2 graf $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$ [12]. ■

Definicija 18. Graf je *povezan* ako se ne može prikazati kao unija dva grafa. U suprotnom, graf je *nepovezan* [12] te postoje najmanje dva grafa od kojih se sastoji, bez bridova među vrhovima jednoga i drugog grafa. ■

Definicija 19. Označe li se vrhovi grafa G s $V = \{1, 2, 3, ..., n\}$, onda se može definirati *matrica susjedstva* $A = [a_{ij}]$ kao matrica $n \times n$ čiji je element a_{ij} u slučaju neusmjerenog grafa jednak broju bridova koji spajaju vrh i s vrhom j [12]. U slučaju usmjerenog grafa, element a_{ij} jednak je broju lukova usmjerenih iz vrha i u vrh j. ■

Definicija 20. Neusmjereni grafovi $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$ su *izomorfni* ako postoji bijektivna korespondencija (preslikavanje jedan na jedan) između skupova V_1 i V_2, takva da je broj bridova koji spajaju bilo koja dva izabrana vrha u V_1 jednak broju bridova koji spajaju dva vrha u V_2 na koje se izabrani vrhovi iz V_1 preslikavaju [12]. ■

Definicija 21. Usmjereni grafovi $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$ su *izomorfni* ako postoji bijektivna korespondencija (preslikavanje jedan na jedan) između skupova V_1 i V_2, takva da je broj lukova koji spajaju bilo koja dva izabrana vrha u V_1 u svakom smjeru između njih jednak broju lukova koji spajaju dva vrha u V_2 na koje se izabrani vrhovi iz V_1 preslikavaju, u odgovarajućem smjeru. ■

Definicija 22. Graf $S = (V_S, E_S)$ je *podgraf* grafa $G = (V, E)$ ukoliko je $V_S \subseteq V$ i $E_S \subseteq E$. ■

Definicija 23. Ako je graf $S = (V_S, E_S)$ podgraf grafa $G = (V, E)$, onda je *izomorfizam podgrafa* S grafa G injektivna funkcija $f : V_S \to V$ takva da za sve bridove ili lukove $(u, v) \in E_S$ vrijedi $(f(u), f(v)) \in E$. Drugim riječima, izomorfizam podgrafa S grafa G je funkcija koja identificira podgrafove S_i grafa G koji su izomorfni podgrafu S. ■

Izomorfizam podgrafa omogućava pronalaženje podgrafova s istim svojstvima, što je put za otkrivanje čestih podgrafova grafa.

Ukoliko se definicija grafa proširi oznakama koje se mogu pridijeliti elementima grafa, tako da se označeni graf prikaže kao $G = (V, E, L)$, gdje je L funkcija koja pridjeljuje oznake vrhovima i bridovima ili lukovima, onda u gornje definicije treba uključiti i te oznake, pa bi tako npr. izomorfizam grafova G_1 i G_2 uključio i uvjete $L(v_1) = L(v_2)$ za svaki vrh $v_1 \in V_1$ i $v_2 \in V_2$ te $L(e_1) = L(e_2)$ za svaki brid ili luk $e_1 \in E_1$ i $e_2 \in E_2$. Slično bi vrijedilo i za izomorfizam podgrafa S grafa G, gdje bi dodatni uvjeti bili $L_S(v) = L(f(v))$ za sve vrhove $v \in V_S$ i $L_S(u, v) = L(f(u), f(v))$ za sve bridove ili lukove $(u, v) \in E_S$.

Definicija 24. Neka je $f_1, ..., f_m$ skup izomorfizama podgrafa $S(V_S, E_S, L_S)$ označenog grafa $G = (V, E, L)$. Neka je $F(v) = \{f_1(v), ..., f_m(v)\}$ skup koji sadrži različite vrhove iz G koje preslikavaju funkcije $f_1, ..., f_m$ od vrha $v \in V_S$. *Minimalna podrška temeljena na prikazu* (eng. *Minimum image based support*) *MNI* podgrafa S u G se označava sa $s_G(S)$ i definira kao:
$s_G(S) = min\{t \mid t = |F(v)|,\ v \in V_S\}$ [13]. ■

Ova mjera omogućava prebrojavanje pojavnosti izomorfizma podgrafa unutar grafa, kako bi se utvrdilo zadovoljava li zadani prag za klasifikaciju kao česti podgraf.

Poglavlje 2

Grafovske baze podataka i veza s relacijskim bazama podataka

Pod pojmom grafovska baza podataka (koja će se u daljnjem tekstu označavati s $\mathcal{G}$) se podrazumijeva baza podataka koja podatke pohranjuje u obliku grafova. Angles [14] definira grafovsku bazu podataka kao bazu podataka čije su shema i/ili instance modelirane kao (označeni) (usmjereni) graf, ili generalizacija grafa kao podatkovne strukture, pri čemu je manipulacija podacima izražena operacijama i tipovima orijentiranima grafu te ima ograničenja koja se odnose na integritet podataka i prikladna su strukturi grafa.

Modeli grafovskih baza podataka i pripadni jezici za pristup i manipulaciju podacima se proučavaju dulje od 25 godina. U početku istraživanja ovog područja je razvijen veći broj modela grafovskih baza podataka i još veći broj pripadnih programskih jezika. Istraživanja su s vremenom zastala ali u posljednje vrijeme, pojavom NoSQL baza podataka i pripadnih načina pohrane podataka koji se razlikuju od relacijskih baza podataka te orijentacijom informacijskih sustava ka boljem iskorištavanju resursa u kraćem vremenu, ova tema dobiva novi zamah. Sve je veći broj podataka i domena koje se čini prirodno opisati grafovima i pohraniti u grafovske baze podataka, npr. semantički web, društvene mreže, lokacije i putanje navigacijskih sustava, itd.

U ovom poglavlju je dan pregled ranije predloženih modela i jezika za grafovske baze podataka, stanje suvremenih grafovskih baza podataka te veze s relacijskim bazama podataka u smislu istraživanja, prikazivanja, pohrane i konverzije podataka iz ovih baza podataka u grafovske.

2.1 Ranija istraživanja grafovskih baza podataka

2.1.1 Predloženi modeli

U literaturi su navedeni različiti modeli grafovskih baza podataka i očekivano, svi imaju formalno uporište u nekoj od varijacija osnovne definicije grafa (definicija 14). Većina ovih modela je nastala prije razvoja NoSQL baza podataka, aktualnih u posljednje vrijeme. Najstariji objavljeni model koji se može smatrati modelom grafovske baze podataka je R&M iz 1975. godine [15]. Ovaj je model nastao kao posljedica nemogućnosti tadašnjih sustava da uzmu u obzir semantiku podataka. Functional Data Model [16] iz 1981. godine predlaže da podaci u bazi podataka budu grafovi, a potpuno drugačiji pristup daje model Logical Data Model (LDM) [17] 1984. godine kojim se pokušalo modelom grafovske baze podataka generalizirati relacijski, hijerarhijski i mrežni model podataka. Model G-Base za modeliranje baza znanja predložen je 1987. godine [18].

Objektno orijentirani model O_2 temeljen na strukturi grafova je predložen 1988. godine [19]. 1990. godine je predložen jedan od najutjecajnih modela GOOD [20], grafovski orijentirani objektni model, kao teorijska osnova za sustav u kojem će manipulacija i prikaz podataka biti zasnovani na grafovima. Veći broj novih modela zasnovan je na ovom modelu: GMOD [21] koji donosi koncepte za korisnička sučelja zasnovana na grafovima; Gram [22], model grafovske baze za hipertekstualne podatke; PaMaL [23], koji proširuje GOOD model konceptom n-torki i skupova; GOAL [24] koji proširuje GOOD model konceptom povezujućih vrhova za prikaz n-arnih veza među entitetima; G-Log [25] koji predlaže deklarativni upitni jezik za grafove; GDM [26] koji dodaje višestruke simetrične veze.

Neki modeli predlažu generalizaciju grafova. Hypernode Model [27] je model koji se temelji na ugniježdenim grafovima, gdje vrh može biti drugi graf. Takvi se vrhovi nazivaju hipervrhovima, a grafovi koji ih sadrže hipergrafovima. Ovaj je model npr. korišten pri modeliranju genoma [28]. GROOVY [29] je objektno orijentirani model koji koristi hipergrafove temeljene na hipertekstualnim podacima.

Pregled modela nastalih do 2002. godine, kao i drugih alata i prijedloga vezanih uz grafovske baze podataka dan je u [14]. Napravljena je i analiza povezanosti pojedinih modela s drugima te su se referentnima pokazali Logical Data Model (LDM), GOOD i Hypernode. U istraživanju objavljenom 2011. godine [30] kao primarna razlika između navedenih modela promatra se složenost vrhova pri čemu složeni modeli omogućavaju pohranu grafova u vrhove drugih grafova. Nastavno na to istraživanje je predložen novi model SPIDER [31], prilagođen definiciji i manipulaciji grafovima temeljenim na ontologiji poslovanja.

2.1.2 Predloženi jezici

Upitni jezik baze podataka se temelji na operatorima i pravilima zaključivanja, koji se mogu primijeniti na bilo koju instancu podataka i koji obavljaju operacije nad podacima i pretražuju ih u bilo kojem željenom poretku [32]. To je također istina i za jezike koji se odnose na grafovske baze podataka, a uz modele grafovskih baza podataka je predložen i veliki broj pripadnih jezika za manipulaciju i pretraživanje podataka. Ovi se jezici mogu grupirati u četiri vrste: vizualni jezici, jezici nalik na SQL, formalni jezici i semantički jezici.

Vizualni jezici

Vizualni jezici nude funkcionalnost tekstualnih jezika onim korisnicima koji ne pripadaju kategoriji naprednih korisnika, ali isto tako i povećanu produktivnost stručnim korisnicima baza podataka. Korisnici mogu nacrtati upite u obliku uzorka iz baze podataka uz pomoć grafičkog sučelja, a dobiveni rezultat se prikazuje na isti način. Radi se dakle o pretraživanju po predlošku (eng. *query by example, QBE*).

G [33] je jezik temeljen na regularnim izrazima i omogućava jednostavnu formulaciju rekurzivnih upita koji se ne mogu jednostavno predstaviti relacijskim upitima. Ne podržava grafove sa zatvorenim petljama, pronalazak najkraćeg puta, izračun udaljenosti vrhova niti agregatne funkcije. Ovaj jezik je kasnije evoluirao u G+ [34] koji ispravlja navedene nedostatke. Na osnovu G+ je nastao GraphLog [35] koji omogućava korištenje negacije, podržava agregatne funkcije i sumiranje po putanjama, a uz to i praćenje izračunavanja upita.

Hyperlog [29] je deklarativni jezik za pretraživanje i ažuriranje definiran nad Hypernode modelom. Vizualizira shemu baze podataka, podatke i rezultate upita kao skupove ugniježdenih grafova, koji se na jedinstven način mogu pohranjivati, pregledavati i pretraživati. Programsko sučelje pomaže korisniku u radu s ovim jezikom ograničavajući ga samo na moguće akcije. Upiti u QGRAPH [36] jeziku se kreiraju kao označeni povezani graf koji se pretražuje u svojstvu podgrafa u modelu baze podataka. Praktičan je u slučajevima kad je potrebno definirati podupite, no ne podržava agregatne funkcije.

GOOD [20] je jezik s grafičkom sintaksom i semantikom, razvijen za istoimeni model. Temelji se na definiranju uzoraka grafova. Unos i brisanje elemenata grafa se također obavljaju kroz grafičko sučelje, a moguće je i grupiranje vrhova po određenim kriterijima, no ne može pronalaziti putanje proizvoljne duljine. Kako su razvoj GOOD modela baza podataka slijedili GMOD [21], PaMaL [23] i GOAL [24], tako su na osnovu GOOD jezika nastali i istoimeni jezici GMOD, PaMaL i GOAL, koji su odgovarali svakom od predloženih modela i donosili određena proširenja u odnosu na GOOD.

Jezici nalik na SQL

Ovi jezici su deklarativni jezici koji proširuju standardni SQL jezik s novim operatorima za rad s grafovima i objektima. Lorel [37] je upitni jezik definiran nad Object Exchange Model (OEM) modelom podataka. Omogućava korištenje regularnih izraza i pretraživanje bez točnog poznavanja putanje. GOQL [38] je proširenje OQL-a mogućnostima za kreiranje, manipulaciju i pretraživanje objekata koji su tipa *graf*, *putanja* i *brid*. Ova tri tipa zajedno s tipom *vrh* čine proširenje objektnog modela. GOQL sadrži i operatore koji olakšavaju pretraživanje grafova, a to su *next*, *until* i *connected*.

SoQL (Social networks Query Language) [39] je namijenjen pretraživanju i manipulaciji podataka u društvenim mrežama. Slično osnovnim objektima koji se koriste u standardnom SQL-u, uvedeni su objekti *putanja* (PATH) i *grupa* (GROUP) te operatori kojima je moguće odrediti uvjete nad putanjom ili grupom. Postoje i agregatne funkcije te egzistencijalni i univerzalni kvantifikatori koji se mogu primijeniti na elemente grafa u putanji ili grupi te na putanje unutar grupe.

GraphQL [40] namijenjen je grafovima kojima atributi nisu unaprijed definirani. Operandi koji se koriste su grafovi, tj. operatori koriste grafove kao ulaze i vraćaju grafove kao rezultate.

Cypher [41] je vrlo izražajan i jednostavan deklarativni upitni jezik novije generacije. Prvobitno je nastao u sklopu Neo4j [42] sustava za upravljanje grafovskim bazama podataka, nakon čega je otvoren i trenutno ga koristi više grafovskih baza podataka.

Formalni jezici

Formalni jezici su uz pripadne modele podataka u literaturi opisani formulama i izrazima pri čemu nije posebno definirana sintaksa jezika ili naveden neki drugi način njegova mogućeg korištenja u praksi.

Logical Database Model (LDM) [43] koristi istoimeni jezik za rad s podacima, koji je prikazan na način sličan relacijskoj algebri. Koristi osnovne formule za prikaz upita te prikazuje rezultate u obliku n-torki.

Gram [22] je algebarski jezik temeljen na regularnim izrazima. Podržava ograničenu rekurziju. Regularni izrazi se koriste za pronalazak putanja, a moraju se definirati kao naizmjenični slijed vrhova i bridova.

G-Log [25] je deklarativni nedeterministički jezik namijenjen složenim objektima s identifikatorom. Definiran je nad istim modelom podataka kao i GOOD, s time da je GOOD imperativni (proceduralni) jezik (program pisan imperativnim jezikom se izvodi slijedno, dok se onaj pisan deklarativnim izvodi vrednovanjem izraza). Osnovna jedinica programa u G-Logu je pravilo, koje se temelji na grafu. Skup pravila definira pojedinu operaciju, a cijeli program je slijed takvih skupova pravila.

HyperNode Query Language (HNQL) [44] je upitni jezik definiran nad Hypernode modelom podataka. Sastoji se od osnovnog skupa operatora za pretraživanje po predlošcima i ažuriranje hipervrhova. Također je proširen u proceduralnom smislu s mogućnostima grananja, iteracije i rekurzije.

Semantički jezici

Semantički jezik je upitni jezik koji je definiran za pretraživanje semantičkih grafova koji se temelje na nekoj ontologiji. Takav jezik prikazan je u [45]. Upiti koriste funkcije koje određuju uzorke i uvjete koje traženi grafovi u bazi trebaju ispunjavati (npr. *unija*, *razlika*, *uzorak*, ..).

Primjer semantičkog jezika je i SPARQL [46], često korišten jezik za pretraživanje podataka u RDF (eng. *Resource Description Framework*) formatu [47]. SPARQL je prepoznat kao jedna od ključnih tehnologija namijenjenih semantičkom webu.

Temeljit pregled jezika za pretraživanje i manipulaciju grafovskih baza podataka, prema specifičnoj namjeni (semantički web, društvene mreže,...), funkcionalnostima za pretraživanje, agregaciji te u konačnici uspješnosti implementacije jezika i složenosti izvršavanja naredbi, dan je u [48].

Uz navedene, postoje i jezici razvijeni u obliku aplikacijskog programskog sučelja (API, eng. *Application Programming Interface*). Ovakvo programsko sučelje je razvijeno u obliku biblioteka za razne programske jezike, poput Jave, Pythona ili C-a te omogućava da se u svim tim jezicima koristi na identičan način. Takvo korištenje za krajnjeg korisnika sučelja može biti izvedeno ulančavanjem funkcija, što je mogućnost koju podržavaju svi moderni programski jezici, pa se u tom slučaju sučelje može promatrati kao funkcijski jezik. Primjer ovakvog jezika je Gremlin [49], dio Apache TinkerPop programskog sustava za rad s grafovima [50].

2.2 NoSQL i suvremene grafovske baze podataka

Tijekom vremena su se pojavljivale manje ili više uspješne baze podataka koje nisu koristile relacijski model, a u posljednje vrijeme se ističe nastanak i porast korištenja takozvanih NoSQL (eng. *Not only SQL*) baza podataka, koje postupno dobivaju svoj udio na tržištu. To su baze podataka za koje većinom vrijedi da: nisu relacijske, distribuirane su, rade na načelu otvorenog koda i horizontalno su skalabilne [51].

Primarni razlozi za nastanak ovih baza su:

- Razlika između relacijskog modela podataka i podatkovnih struktura koje koriste suvremeni programski jezici (eng. *impedance mismatch*) [52]. Većina programskih jezika i alata današnjice su objektno orijentirani, pri čemu je u radu s podacima potrebna stalna pretvorba zapisa u relacijama (n-torki) u objekte i obratno, što se dodatno usložnjava

postojanjem referencijskog integriteta među relacijama i preslikavanjem istoga u objektni model. Ovaj problem djelomično rješavaju brojni programski okviri za objektno-relacijsko preslikavanje (eng. *object-relational mapping - ORM*) kao što su Hibernate [53] ili MyBatis [54], no često postaju problematični u slučajevima gdje inzistiranje na objektnom modelu dovodi do loših performansi relacijske baze. Ovaj problem može se formulirati kao razlika između relacijskog modela podataka i modela podataka koji se koristi za rješavanje tipičnog problema, kao što je npr. slučaj s podacima predstavljenim grafovima.

- Prikupljanje i obrada vrlo velikih količina podataka tvrtkama postaje skuplja u financijskom i vremenskom kontekstu ako ga obavljaju kroz relacijske baze podataka. Dok razni oblici skupina fizičkih i virtualnih računala (klasteri, oblaci) cijenom i performansama postaju vrlo konkuretni, većina relacijskih sustava za upravljanje bazama podataka trenutno prisutnih na tržištu nije dizajnirana imajući na umu performanse i skalabilnost na ovoj razini, ili uopće nije dizajnirana za rad u oblaku. Vlasnici informacijskih sustava koji u kratkom vremenu prikupljaju i obrađuju velike količine podataka radije koriste veliki broj cijenom pristupačnih poslužitelja umjesto investiranja u vrlo skupa superračunala, a za to su im potrebne one baze podataka koje će tu mogućnost podržavati.

NoSQL baze podataka podržavaju različite modele i fleksibilne sheme podataka te su stoga prikladan izbor pri bržem i jednostavnijem razvoju i održavanju aplikacija, a mnoge od njih su dizajnirane za rad u distribuiranim okruženjima (računalni klaster ili računalni oblak). Uglavnom se klasificiraju u četiri kategorije [51, 52, 55, 56]: ključ-vrijednost (eng. *key-value*) baze podataka, dokumentske baze podataka, baze podataka sa skupinama stupaca (eng. *column-family*, *wide-column*) i grafovske baze podataka.

Ključ-vrijednost baze podataka odnose se na podatke spremljene u parovima (ključ, vrijednost). Svi podaci unutar baze podataka su spremljeni na ovaj način. Pri tome vrijednost koja se pohranjuje uz pojedini ključ može biti bilo kojeg tipa podatka. Tipični predstavnici ove vrste su Riak [57], Redis [58], Berkeley DB [59], upscaledb [60].

Dokumentske baze podataka (npr. MongoDB [61]) temelje se na radu s raznim vrstama dokumenata (XML, JSON, BSON,...), uključujući i binarne dokumente. U ove baze podataka se podaci pohranjuju u formatu ključ-dokument, pri čemu je ključ identifikator podatka. Dokumenti se mogu dodatno indeksirati ovisno o svojoj strukturi, ali nije nužno da imaju istu strukturu niti da su dokumenti unutar jedne baze podataka istoga formata.

Baze podataka sa skupinama stupaca se temelje na BigTable modelu podataka predloženom od Googlea [62]. Radi se o proširenom modelu ključ-vrijednost. Ključ je neka vrsta identifikatora, a vrijednosti su podatkovne mape, koje sadržavaju skupine (stupce) vezanih podataka. U ovu grupu spadaju Cassandra [63], HBase [64], Amazon SimpleDB [65] i druge.

Kao što je već rečeno, grafovske baze podataka služe za pohranu podataka u obliku (us-

mjerenog) grafa. Entiteti i relacije među njima se također nazivaju vrhovima i bridovima ili lukovima. I jedni i drugi mogu imati svojstva, a lukovi također imaju i smjer. Trenutno je dostupno više grafovskih baza podataka koje su komercijalne ili otvorenog koda: AllegroGraph [66], Sparksee (ranije pod imenom DEX) [67, 68], HyperGraphDB [69, 70], Neo4j [42], Infinite Graph [71], Titan [72], Weaver [73], itd. Osim njih, postoji još niz programskih rješenja za pohranu grafova izvan grafovske baze podataka.

Iako se klasifikacija NoSQL baza podataka u većini slučajeva odnosi na navedene četiri vrste, u nekim radovima grafovske baze podataka nisu navedene među NoSQL baze podataka [74, 75]. Dvije su osnovne razlike koje do ovoga dovode. Prva su veće razlike u modelu podataka: ključ-vrijednost, dokumentske i baze podataka sa skupinama stupaca se sve oslanjaju na postojanje identifikatora s kojim su povezani ostali podaci koji su u većoj ili manjoj mjeri strukturirani, dok su mogući modeli grafovskih baza podataka potpuno drugačiji. Druga razlika je arhitekturne prirode: ključ-vrijednost, dokumentske i baze podataka sa skupinama stupaca su prvenstveno dizajnirane za rad u vrlo distribuiranim i skalabilnim rješenjima, dok je primarni cilj pri dizajnu grafovskih baza podataka upravo podržavanje pripadnog modela podataka. Dio grafovskih baza podataka trenutno još uvijek ne podržava skaliranje na više računala i distribuirani rad. Iz ove razlike proizlazi još jedna, a ta je obavljanje transakcija. Iako NoSQL baze podataka u pravilu ne podržavaju transakcije i njihova *ACID* svojstva (od eng. *Atomicity*, *Consistency*, *Isolation*, *Durability* - nedjeljivost, konzistencija, izolacija, trajnost), grafovske baze podataka često nauštrb distribuiranosti mogu implementirati transakcije i njihova svojstva u smislu relacijskih baza podataka.

Unatoč ovim razlikama, grafovske baze podataka treba promatrati u sklopu NoSQL-a baza podataka. Primjerice, jedna od važnih karakteristika NoSQL baza podataka je fleksibilnost sheme podataka. Stoga se aplikacije koje koriste ove baze podataka izgrađuju s idejom da se shema baze podataka može izmijeniti u bilo kojem trenutku, pa su u tom smislu prilagodljive. Ovo često nije slučaj s aplikacijama koje se izgrađuju nad relacijskim bazama podataka s jasno opisanom shemom podataka, čija se promjena reflektira na cijeli niz objekata u bazi podataka ali i aplikacija. Navedeno svakako vrijedi i za grafovske baze podataka koje redom nemaju unaprijed definirana ograničenja vezana uz shemu podataka, a posljedično i za aplikacije izgrađene za rad s njima.

Aktualna klasifikacija modela grafovskih baza podataka, prikazana u [55], se temelji na usporedbi ključnih mogućnosti trenutno dostupnih baza podataka, koje definiraju model podataka: struktura podataka, programski jezici i ograničenja integriteta. Struktura podataka odnosi se na samu vrstu grafova koji se pohranjuju u bazi (jednostavni, hipergrafovi, hipervrhovi, grafovi opisani atributima), vrhove (označeni, opisani atributima) i bridove ili lukove (usmjereni,

označeni, opisani atributima). Vezano uz programske jezike, promatra se implementira li pojedina baza podataka neki od jezika za pretraživanje, implementira li posebno programsko sučelje za pristup i manipulaciju podacima (API) te implementira li grafički prikaz i pretraživanje podataka. U području integritetskih ograničenja promatra se provjera domenskog integriteta, entitetskog integriteta (postojanje jedinstvenog identifikatora elementa grafa), referencijskog integriteta i sl. Osim navedenih, od aktualnih baza podataka se također zahtijeva korištenje trajne memorije i indeksiranje podataka.

Glede struktura podataka, sve baze podataka koje su uspoređivane u ovoj klasifikaciji su podržavale označene elemente grafa te usmjerene lukove, a dio njih je također podržavalo i atribuciju elemenata grafa, odnosno svojstva. U taj dio spadaju Sparksee, Infinite Graph i Neo4j.

U tom smislu, kao jedan od ključnih suvremenih modela grafova se može prepoznati model grafa sa svojstvima (eng. *property graph*). Graf sa svojstvima je usmjereni graf koji omogućuje da elementi budu označeni i da budu opisani svojstvima. Ovaj model nudi dovoljno veliku fleksibilnost za opis snimljenoga traga i istraživanje čestih podgrafova. Slijedom toga, u okviru ove disertacije, pod grafovskom bazom podataka se podrazumijeva ona koja kao model podataka koristi model grafa sa svojstvima.

Definicija 25. Grafovska baza podataka $\mathcal{G}$ se može opisati uređenom sedmorkom:

$$\mathcal{G} = (V, E, \varphi, L, \lambda, P, \pi) \tag{2.1}$$

gdje je:

- V skup vrhova,
- E skup lukova,
- $\varphi : E \to V \times V$ funkcija incidencije koja povezuje vrhove lukom,
- L skup oznaka elemenata grafa,
- $\lambda : V \cup E \to L$ funkcija označavanja elemenata grafa,
- $P = \{(p_i, v_i)\}$ skup svojstava p_i s pripadnim vrijednostima v_i,
- $\pi : V \cup E \to 2^P$ funkcija pridjeljivanja svojstava elementima grafa. ■

2.3 Grafovske i relacijske baze podataka

S obzirom na duboku ukorijenjenost relacijskih baza podataka u informacijskim sustavima, na relativno mladu povijest aktualnih grafovskih baza podataka i na činjenicu da je model grafa prikladan za prikaz i analizu mnogih domena, jasno je da postoje potrebe za povezivanjem ove dvije vrste baza podataka. Povezivanja idu u smjeru vizualizacije relacijskih podataka u grafovima, konverzije relacijskih podataka u grafove te pohrane grafova u relacijske baze podataka.

Soussi [31] tvrdi da je podatke iz relacijske baze moguće transformirati u neki međumodel iz kojeg je kasnije moguće na jednostavniji način stvoriti graf. Među moguće modele svrstava ER (eng. *Entity-Relationship*), RDF (eng. *Resource Description Framework*) i XML (eng. *Extensible Markup Language*) te daje prikaz radova iz tih domena. Iako postoji nekoliko proširenja XML-a namijenjenih opisivanju grafova (GraphXML, GXL i GraphML) trenutno ne postoji niti jedan predloženi algoritam za konverziju relacijske baze podataka u neki od ovih modela. Osnovna mana pristupa izgradnje grafova uz pomoć međumodela jest nemogućnost detekcije interakcije među generiranim vrhovima bez prvobitne izgradnje ontologije kojom će se opisati potrebiti podaci i odnosi.

Predloženo je nekoliko alata za prikaz informacija iz relacijske baze podataka u obliku grafa. Ovi alati omogućavaju korisniku, koji ne mora poznavati postojeću shemu relacijske baze podataka, postavljanje upita u obliku fraza ili ključnih riječi koje se mogu pronaći negdje u atributima n-torki, i dobivanje rezultata oblikovanih u grafove [76]. Ovisno o shemi baze podataka, stvaraju se složeni SQL upiti u kojima su relacije spojene prirodnim spajanjem, vanjskim spajanjem ili unijom te se na osnovu rezultantnih n-torki stvaraju grafovi u kojima relacije, n-torke i veze među njima mogu biti prikazane na različite načine. U tom smislu BANKS [77] modelira n-torke kao vrhove grafa. Ti su vrhovi spojeni lukovima koji predstavljaju strane ključeve ili druge veze među pojedinim n-torkama (cikličke ili tranzitivne). Među slična rješenja spadaju BLINKS [78], DBXplorer [79] i Discover [80].

Nedavno predloženi alati GraphGen [81] i Aster6 [82] također omogućavaju generiranje grafa iz relacijskih podataka. Ovi alati predstavljaju podatke iz relacijske baze podataka kao graf pohranjen u radnoj memoriji, a korisnik je taj koji treba definirati koji će se podaci prikazati. Iako omogućavaju provođenje tehnika dubinske analize grafova nad relacijskim podacima, nisu usmjereni na zadržavanje tih podataka za trajnije analize, a također je korisnik taj koji sam mora otkrivati koji su to zanimljivi podaci koje bi trebao podrobnije istraživati.

RDB2Graph [83] je pristup transformaciji relacijskih podataka u graf temeljem konceptualnog grafa (eng. *conceptual graph*) [84]. Stvara graf u kojem vrh predstavlja n-torku, a na njega se vežu drugi vrhovi koji predstavljaju vrijednosti atributa pojedine n-torke i koji nisu povezani s bilo kojim drugim vrhom. Ovaj algoritam ne podržava kompozitne primarne i strane ključeve. Iako je pogodan za istraživanje čestih podgrafova iz perspektive atributa, nije pogodan za istraživanje čestih podgrafova iz perspektive entiteta. Na primjer, n-torke s vrijednostima atributa koji se višestruko ponavljaju se mogu identificirati kao česti podgrafovi koji predstavljaju samo dio n-torke i na taj način producirati veliku količinu čestih podgrafova koji imaju malo veze s entitetima predstavljenim u relacijskoj bazi podataka i vezama među njima.

DB2Graph algoritam [85] se temelji na konceptu RR-grafa (eng. *relation-of-relations graph*), grafa u kojem vrhovi predstavljaju entitete odnosno n-torke relacije, svojstva vrha su vrijednosti atributa n-torke, a lukovi predstavljaju strane ključeve među n-torkama. Ovaj algoritam je prik-

ladan za korištenje s obzirom na utrošak resursa, ali je nedovoljno testiran na većim bazama podataka.

U smislu konverzije velike količine relacijskih podataka u grafovsku bazu podataka te također iskorištavanja svojstava elemenata grafa, algoritam koji je najprikladniji potrebama ovog istraživanja je R2G [86]. Ipak, iako ovaj pristup iskorištava svojstva vrhova, zanemaruje svojstva bridova te tako ne koristi model grafa sa svojstvima u potpunosti. Osim toga, ovaj pristup može agregirati pojedine n-torke u isti vrh, što ga čini manje prikladnim za istraživanje čestih podgrafova.

U kontekstu povezivanja relacijskih baza podataka i grafova, valja spomenuti da su predloženi i neki alati koji pokušavaju iskoristiti moćne sustave za upravljanje relacijskim bazama podataka i njihove mogućnosti kao prostor za pohranu grafova. U ove alate spadaju Vertexica [87] i Grail [88].

Slijedom istraženih algoritama za konverziju relacijskih podataka u grafovsku bazu podataka, može se zaključiti da je potrebno razviti novi pristup, koji će u potpunosti iskoristiti model grafa sa svojstvima, a na kojem će se temeljiti stvaranje vremenskog grafa iz snimljenoga traga relacijske baze podataka.

Poglavlje 3

Vremenski grafovi

Korištenjem grafova (bilo usmjerenih ili neusmjerenih) za trajnu pohranu podataka i razvojem sve uporabljivijih algoritama za analizu grafova, postupno dolazi do shvaćanja kako obični grafovi nisu dovoljno dobri za opis promjena podataka koje sadrže. U tom smislu se nepromjenjivi pohranjeni grafovi nazivaju statičkima, a ukoliko se grafovi mijenjaju kroz vrijeme ovisno o vanjskim utjecajima, onda se govori o dinamičkim grafovima. Dinamički grafovi su znatno prikladniji za prikaz promjenjivih domena. Pokušaj prikaza promjenjive domene u jednom statičkom grafu će rezultirati sadržavanjem višestrukih bridova ili lukova, odnosno agregiranih podataka u grafu. Ovo dovodi do precjenjivanja veza (bridova ili lukova) među vrhovima statičkog grafa, što predstavlja problem u analizi vremenskih podataka u grafovima.

Jednostavni primjer promjenjive domene za prikaz u dinamičkom grafu su telekomunikacije. U takvom grafu vrhovi mogu predstavljati pretplatnike mobilne telekomunikacijske mreže, a bridovi ostvarenu komunikaciju između njih. Ti će bridovi postojati samo za vrijeme trajanja komunikacije. Primjer precjenjivanja veza među vrhovima statičkog grafa u odnosu na dinamički je istraživanje mogućnosti širenja virusa. Neka vrhovi u grafu predstavljaju osobe koje sudjeluju na nekom događanju, a bridovi postojanje fizičke blizine među dvije osobe. U dinamičkom grafu će bridovi postojati samo za vrijeme kontakta dvaju osoba, a u statičkom će biti sadržana samo informacija da su te dvije osoba ostvarile kontakt. Proučavajući statički graf bi se moglo zaključiti da je moguće širenje virusa na tom događaju u znatno većoj mjeri nego promatrajući mogućnost širenja virusa kroz vrijeme putem dinamičkog grafa.

U istraživanju grafova u kojima se uzima u obzir koncept vremena, uz dinamičke, promjenjive grafove, također se promatraju i statički grafovi koji sadržavaju vremenske podatke. Sve grafove koji su promjenjivi kroz vrijeme ili sadrže vremenske podatke se objedinjenim imenom naziva *vremenskim grafovima*.

Holme [89] navodi primjere sustava, uz referentne radove, koje je moguće modelirati pomoću vremenskih grafova:

- fizička blizina ljudi, životinja, objekata

- ljudska komunikacija,
- mreže suradnje (npr. istraživača),
- citiranje radova,
- ekonomske mreže u poslovanju,
- neuronske mreže,
- putovanje i transport,
- distribuirano računarstvo,
- ekološke mreže (npr. interakcije među vrstama, prehrambeni lanci),
- biološki podaci (npr. genetika ili međudjelovanja proteina),
- ostalo (npr. stanja materije u složenim materijalima).

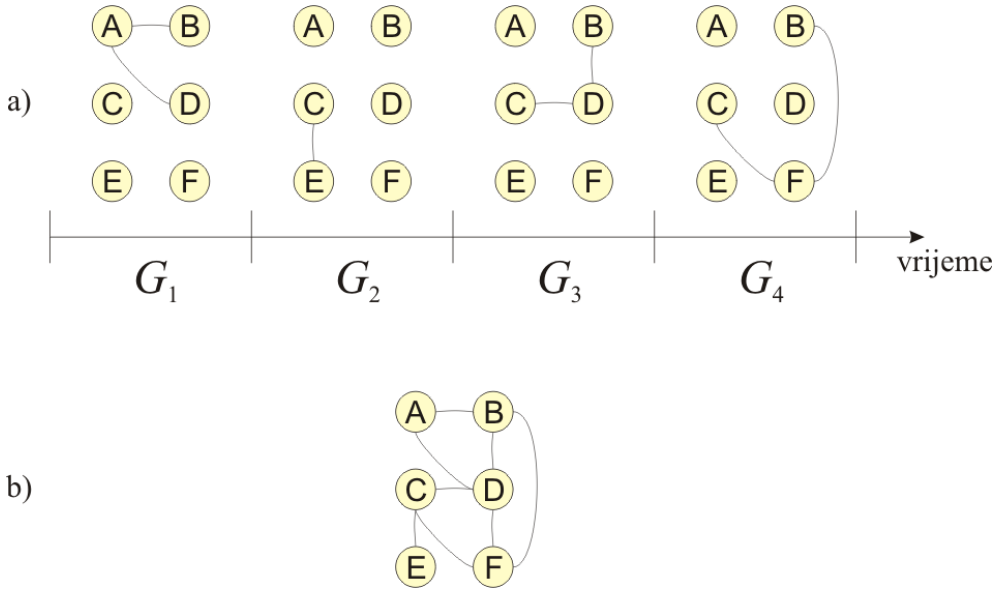
Koncept vremena je, često ovisno o domeni istraživanja, ugrađen u grafove na različite načine. Na primjer, u istraživanjima direktnih ili telekomunikacijskih kontakata koji traju kroz određeni vremenski period, vremenski grafovi imaju fiksne vrhove, koji jednako postoje kroz vrijeme i njihova promjenjivost se uopće ne promatra, dok su vremenski ovisni samo bridovi ili lukovi, npr. [90, 91, 92].

S obzirom na model, vremenski grafovi su u literaturi najčešće prikazani kao:

1. vremenski niz statičkih grafova, pri čemu svaki graf u nizu pripada određenom diskretnom vremenskom trenutku,
2. proširenje modela statičkog grafa konceptom vremenski ovisnih spojnica (bridova ili lukova), pri čemu svaka spojnica ima vrijeme nastanka i trajanje (vrijeme života) i/ili trajanje prijelaza te spojnice,
3. trodimenzionalni tenzor susjedstva (analogno matrici susjedstva statičkih grafova), gdje su prva i druga dimenzija vrhovi, a treća dimenzija su diskretni vremenski trenuci.

Pri definiranju vremenskog grafa u obliku vremenskog niza [91, 93], taj se graf promatra kao uređeni skup $\mathcal{G} = \{G_1, G_2, ..., G_T\} = \{G_t\}_{t=1,2,...T}$ koji se sastoji od T usmjerenih ili neusmjerenih grafova. Svaki od ovih grafova pripada određenom diskretnom vremenskom trenutku t. Na slici 3.1 je prikazan jednostavni niz grafova i njegova projekcija u statički graf. Iz prikaza se jasno vidi kako statički graf precjenjuje spojnost. Na primjer, informacija iz vrha A nikako nije mogla doći do vrha E, što se vidi iz vremenskog niza, ali se ne vidi iz statičkog grafa. Iako je načelno moguće da u ovakvom vremenskom nizu grafovi imaju različite vrhove, često je zadan dodatni uvjet: ako je $G_i = (V_i, E_i)$, onda treba biti $V_1 = V_i$ za svaki $i = 1, ..T$ (npr. u [94]). Ovo omogućava međusobnu usporedbu matrica susjedstva grafova u različitim vremenskim trenucima, što algoritmi za analizu vremenskih grafova često koriste.

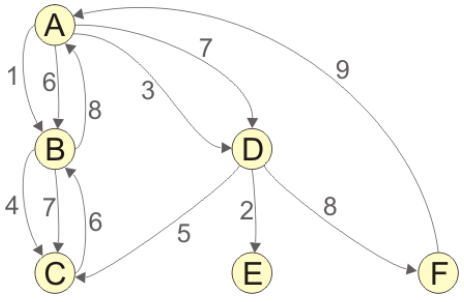
U slučaju proširenja modela statičkog grafa konceptom vremenski ovisnih spojnica (bridova ili lukova), vrhovi se promatraju kao postojani u vremenu, a spojnice su te koje u određenim vremenskim trenucima postoje a u određenima ne [90, 94, 95, 96, 97]. Spojnicama se u ovom



Slika 3.1: Jednostavni vremenski niz grafova (a) i njegova projekcija u statički graf (b)

smislu može pridjeliti određeno trajanje nakon kojeg opet nestaju, ili im se može pridjeliti trajanje "prelaska" spojnice (npr. u slučaju da vrhovi predstavljaju geografske točke, spojnica među njima može imati "težinu" koja je zapravo trajanje puta od jedne točke do druge). U kontekstu ovakvih vremenskih grafova se put (eng. *path*) između dva vrha promatra kao putovanje (eng. *journey*).

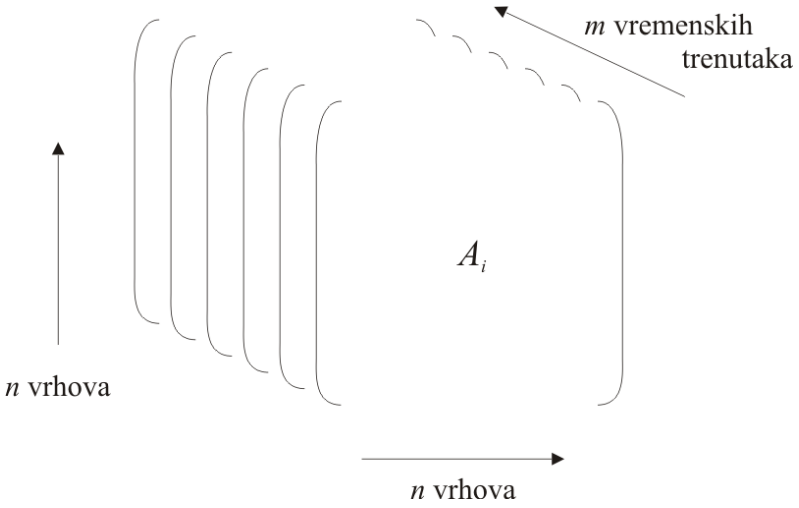
Usmjereni graf s vremenski ovisnim lukovima se može prikazati slikom 3.2, a najjednostavnije ga se može opisati kao graf $G = (V, E)$, gdje je V skup vrhova a E skup lukova, pri čemu je luk $e = (u, v, t) \in E$ vremenski luk među vrhovima u i v u vremenu t i za svaka dva vremenska luka (u, v, t_1) i (u, v, t_2) vrijedi $t_1 \neq t_2$, odnosno u jednom trenutku u vremenu može postojati samo jedan luk između dva ista vrha. Također se u ovakvim grafovima vrh promatra kao aktivan u određenom trenutku ako u tom trenutku postoji makar jedan incidentni luk tom vrhu.



Slika 3.2: Vremenski graf s vremenski ovisnim lukovima

Vremenski graf se može prikazati i trodimenzionalnim tenzorom susjedstva, koji se sastoji od m matrica susjedstva veličine $n \times n$, gdje je n broj vrhova u grafu, a m broj diskretnih vremenskih trenutaka za koje se čuvaju podaci [92]. Ovakva reprezentacija vremenskog grafa prikazana je na slici 3.3, a karakteriziraju ju dva očita ograničenja - stalan i nepromjenjiv skup

vrhova u vremenu i diskretno poimanje vremenske dimenzije.



Slika 3.3: Vremenski graf prikazan tenzorom susjedstva $\mathcal{A}$. Matrica susjedstva A_i je odsječak tenzora u trenutku t_i.

Vremenski grafovi koji su prikazani vremenskim nizom i tenzorom susjedstva su najiskoristiviji pri izračunavanju raznih mjera vremenskih lokaliteta, centraliteta, pronalaska najkraćeg i najbržeg puta i slično. Vremenski grafovi koji su proširenje statičkih grafova vremenski promjenjivim spojnicama su prikladniji za istraživanja čestih podgrafova, odnosno vremenskih motiva. Od navedenih radova, samo je u [95] ostavljena mogućnost definicije vremenskog grafa koja dodaje i vrijeme nastanka i trajanje pojedinog vrha, ali ju daljnja razrada teme nije koristila.

Model vremenskog grafa u kojem bi i vrhovi i bridovi ili lukovi imali vremenske odrednice početka i kraja aktivnosti omogućio bi potpuno definiranje grafa, uključujući i njegovu povijest, s pristupom vremenu kao kontinuiranoj, a ne diskretnoj varijabli, što često u istraživanjima nije slučaj. Stoga će takav model biti predložen u okviru ove disertacije (u poglavlju 7), a odnosit će se na proširenje statičkog grafa konceptom vremenski određenih elemenata grafa. Najbliži ovom modelu je STINGER [98], gdje je predložen detaljan način implementacije podatkovne strukture grafa s težinama i svojstvima vrhova i bridova, vremenskim odrednicama bridova te osnovnih operacija nad njom.

3.1 Vremenski grafovi u grafovskim bazama podataka

Uzevši u obzir moderne grafovske baze podataka i spomenute modele vremenskih grafova, zanimljivo je vidjeti i mogućnosti pohrane tih modela u ove baze podataka.

Načelno, i statički i dinamički grafovi se na jednak način pohranjuju u grafovsku bazu podataka. Dinamički graf se razlikuje po tome što je sama baza podataka dio aktivnog sustava u kojem je moguća promjena grafa u svakom trenutku, pa je to nešto što je u analizama potrebno

uzeti u obzir.

Vremenski graf koji je u obliku vremenskog niza se može pohraniti kao više grafova unutar grafovske baze podataka. Svaki graf iz vremenskog niza je potrebno posebno označiti kako bi se poznavala njegova vremenska odrednica, a ako pojedini graf unutar vremenskog niza nije povezan, onda je potrebno tu odrednicu primijeniti na svaki odvojeni podgraf. Ukoliko grafovska baza podataka koristi model grafa sa svojstvima, najjednostavniji način za vremensko označavanje grafa jest primjena dodatne oznake elementa grafa, koja će se odnositi na točno određeni trenutak u vremenu ili primjena dodatnog svojstva koje će biti dodano svakom elementu grafa. Ovaj način prikaza vremenskog grafa je prostorno najzahtjevniji za grafovsku bazu podataka.

Proširenje modela statičkog grafa konceptom vremenski ovisnih spojnica je znatno jednostavnije za implementaciju, jer su vrhovi vremenski konstantni, pa je njih u ovom slučaju potrebno samo jednom pohraniti u bazu podataka. Spojnice među njima pak treba pohraniti na način da sadržavaju i vremensku informaciju aktivnosti, odnosno početak i kraj aktivnosti svake spojnice. Ovo je moguće učiniti uvođenjem dodatnih svojstava spojnice. Ukoliko su važne i druge vremenske karakteristike spojnice, kao npr. vrijeme prolaska, ovaj je podatak također moguće modelirati kao dodatno svojstvo spojnice.

Vremenske grafove koji su prikazani tenzorom susjedstva u tom obliku nije moguće pohraniti u grafovsku bazu podataka sa svojstvima zbog same činjenice da moderne baze podataka ne omogućavaju taj način pohrane. Ovaj model je znatno prilagođeniji direktnim izračunima u raznim programskim jezicima. Ipak, uz unaprijed definirana pravila o vremenskim podacima i odnošenju prema promjenjivim spojnicama, ovaj je prikaz moguće konvertirati u vremenski niz ili statički graf s vremenski aktivnim spojnicama te iskoristiti način pohrane jednog od tih modela u grafovsku bazu podataka.

Poglavlje 4

Pronalazak čestih podgrafova

Pronalazak čestih podgrafova unutar grafa je postupak koji će se, u okviru ove disertacije, provesti na modelu vremenskog grafa, kako bi se dobili česti vremenski podgrafovi. S obzirom da će se predložiti model koji se zasniva na prilagodbi statičkog grafa konceptom vremenski određenih vrhova i lukova, tako za pronalazak čestih vremenskih podgrafova unutar tog grafa dolaze u obzir za razmatranje za prilagodbu i algoritmi za pronalazak čestih podgrafova unutar statičkog grafa i algoritmi za pronalazak čestih vremenskih podgrafova.

Ključni kriteriji za odabir algoritma koji se dalje može koristiti za pronalazak čestih vremenskih podgrafova su:

- podaci su pohranjeni u jednom velikom grafu,
- mogućnost preciznog pronalaska čestih podgrafova,
- mogućnost pronalaska čestih podgrafova velikog grafa u stvarnom vremenu,
- mogućnost prilagodbe algoritma za pronalazak čestih vremenskih podgrafova grafa s konceptom vremenski određenih elemenata.

U kontekstu ovog rada, velikim grafom se smatraju grafovi reda veličine milijun elemenata ili više.

4.1 Pronalazak čestih podgrafova u statičkome grafu

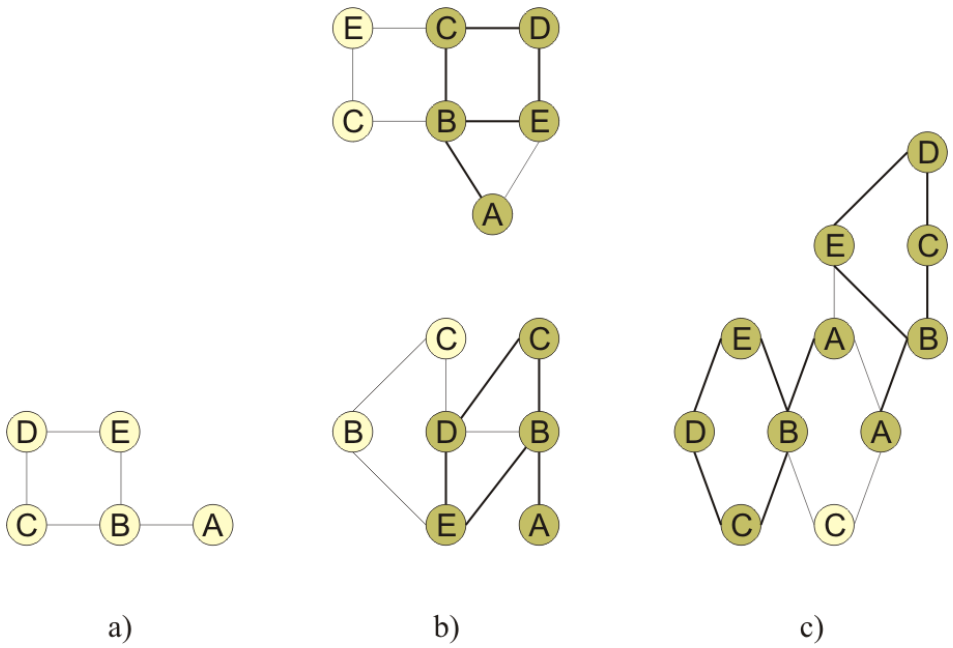
Pronalazak čestih podgrafova (eng. *Frequent Subgraph Mining - FSM*) je jedan od složenijih postupaka u dubinskoj analizi grafova. Postupak se svodi na rješavanje dva najveća problema, usporedbe podgrafova, koji spada u kategoriju NP-potpunih problema, i generiranja mogućih kandidata za česte podgrafove.

U rješavanju problema usporedbe podgrafova često se prilazi strategiji reprezentacije grafa nekim jednostavnijim modelom, kao što su matrica susjedstva, ili kanonsko označavanje (kodiranje) grafa, metoda koja je preuzeta iz slične grane istraživanja, pronalaska čestih uzoraka u skupovima (eng. *Frequent Itemset Mining - FIM*). U ovim postupcima pomaže jedinstveno

označavanje elemenata grafa kroz cijeli graf.

Pri generiranju kandidata za česte podgrafove ključno je sistematično generirati kandidate bez redundancija. Koriste se postupci spajanja jednostavnijih kandidata u složenije, kao i proširenja kandidata u širinu ili dubinu. Ako je moguće na neki način klasificirati kandidate, onda je moguć i postupak spajanja jednostavnijih kandidata istih klasa da bi se dobili složeniji za ispitivanje frekvencije pojavljivanja.

Prema načinu pretraživanja, algoritmi za pronalazak čestih podgrafova se mogu podijeliti na one koji pretražuju grafove u širinu i one koji pretražuju grafove u dubinu. Prema ulaznim podacima, mogu se podijeliti u one koji obrađuju ulazne podatke kao jedan veliki graf ili kao više manjih grafova (tzv. transakcijski grafovi). Načelno, u situacijama u kojima se promatraju manji grafovi, često takav graf može sadržavati samo jednu instancu podgrafa koji se kao kandidat istražuje, dok se pri promatranju jednog velikog grafa ili više njih mora uzeti u obzir višestruka pojavnost svakog od kandidata. Ova je razlika ilustrirana na slici 4.1. Na kraju, prema načinu usporedbe podgrafova, s obzirom na složenost tog problema, algoritmi mogu biti precizni i neprecizni. Neprecizni algoritmi mogu propustiti prijaviti neke česte podgrafove, ali su zauzvrat višestruko brži. Upravo zbog složenosti problema, općenito su algoritmima za pronalazak čestih podgrafova slaba točka brzina i utrošak radne memorije.



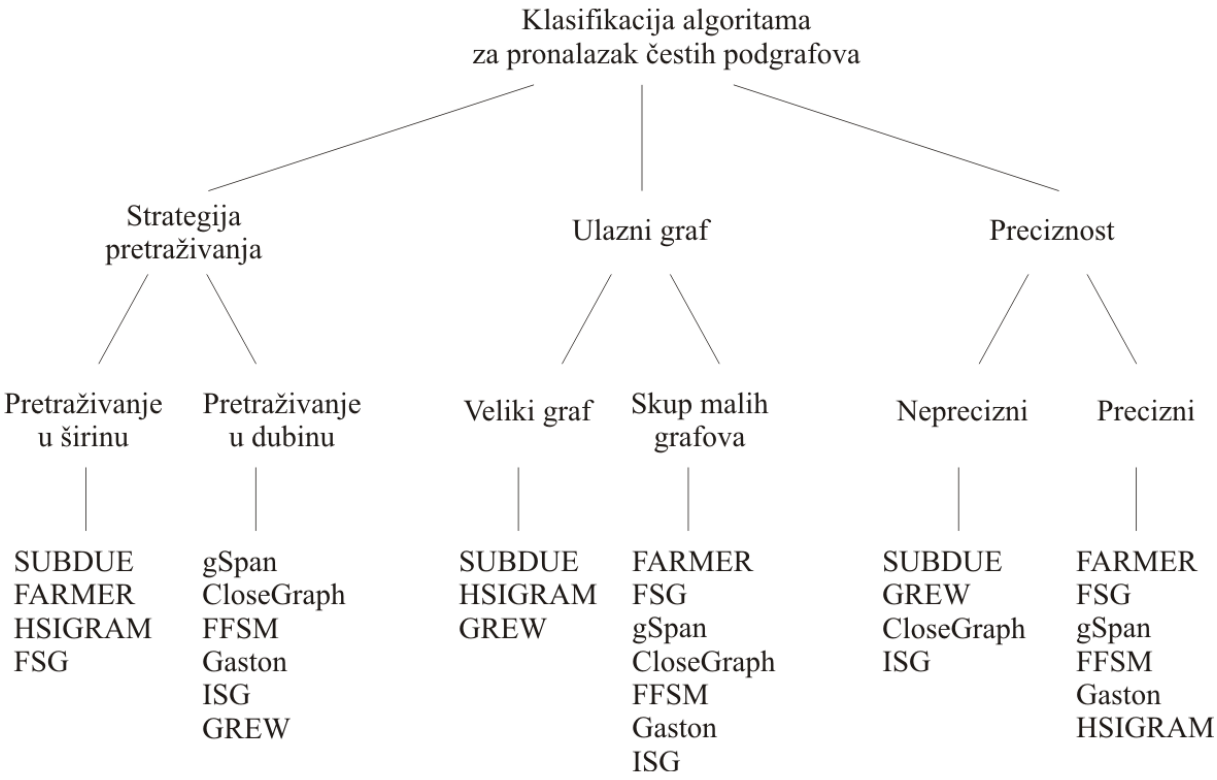
Slika 4.1: Pojavnost podgrafa kandidata (a) u više manjih grafova (b) i u jednom većem grafu (c)

U recentnim pregledima dostupnih algoritama [99, 100, 101] je uglavnom prepoznato nekoliko najčešćih i najuspješnijih u rješavanju problema pronalaska čestih podgrafova:

- SUBDUE, Substructure Discovery [102],
- AGM, Apriori-based Graph Mining [103],
- FSG, Frequent Subgraph Mining [104],

- MoFa, Molecular Fragment Miner [105],
- gSpan, Graph-based Substructure Pattern Mining [106],
- HSIGRAM, Horizontal Single Graph Miner [107],
- FFSM, Fast Frequent Subgraph Mining [108].

Na slici 4.2 je prikazana klasifikacija ovih i još nekih algoritama spram načina pretraživanja, oblika ulaznih podataka i preciznosti.



Slika 4.2: Klasifikacija poznatijih algoritama za pronalazak čestih podgrafova (prema [109])

Algoritam GraMi (eng. *GRAph MIning*) koji se temelji na gSpan algoritmu, i prema prikazanim rezultatima pokazuje značajna ubrzanja je prikazan u [110]. Ovaj algoritam ne pronalazi sve instance određenog podgrafa, nego samo minimalni broj kako bi se dokazalo da se promatrani podgraf pojavljuje dovoljno frekventno. Također je predstavljena i aproksimativna varijanta ovog algoritma, koja u određenim slučajevima postiže znatno bolje vrijeme pronalaska čestih podgrafova. Naknadno je algoritam proširen kombiniranjem s izračunom vjerojatnosti [111], a predstavljena je i verzija ovog algoritma koja omogućuje paralelno pretraživanje grafa [112].

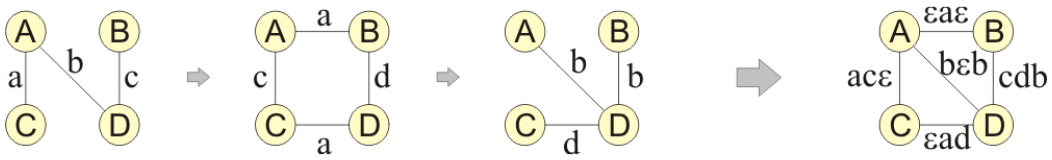
4.2 Pronalazak čestih vremenskih podgrafova

Povećanjem svjesnosti o potrebi inkorporiranja vremenskih podataka u grafove, dolazi i do istraživanja mogućnosti pronalaska čestih vremenskih podgrafova unutar vremenskih grafova.

Predloženi algoritmi se međusobno razlikuju najviše po modelu vremenskog grafa koji proučavaju. S obzirom da je veliki dio spomenutih modela vremenskih grafova ograničen spram vremenske promjenjivosti vrhova (samo se bridovi ili lukovi promatraju kao promjenjivi u vremenu), tako se i predloženi algoritmi fokusiraju upravo na takve vremenske grafove.

Iako se vremenska dimenzija vrha može činiti nevažnom u kontekstu čestih podgrafova, jer sam taj vrh nije dio nijednog (pod)grafa dok god nije povezan s drugim vrhom, nju treba promatrati kroz prizmu domene koju graf opisuje. Na primjer, u kontekstu promatranja korisničke aktivnosti, mogu biti zanimljive činjenice da je vrh nastao u točno određenom intervalu prije nego je nastao njemu incidentni luk, da su najprije nastali vrhovi u određenom redoslijedu, a nakon njih lukovi u određenom redoslijedu, i sl. Stoga se za primjenu unutar ovog istraživanja promatra i mogućnost predloženih algoritama za pronalazak čestih vremenskih podgrafova s vremenski određenim vrhovima, a također i mogućnost prilagodbe algoritama u tom smislu.

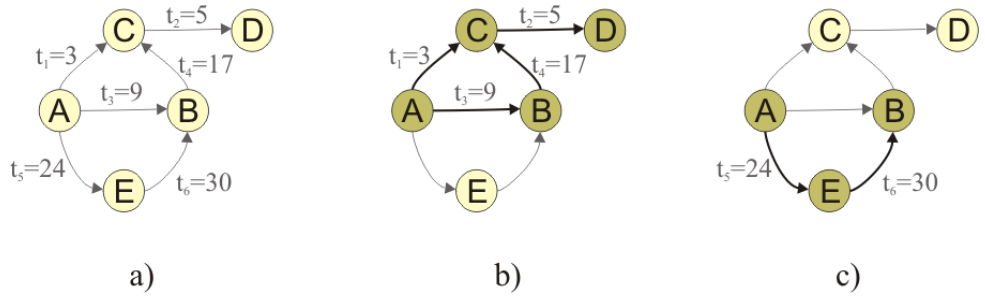
Problem pronalaska čestih vremenskih podgrafova, tj. podgrafova u vremenskim grafovima je sličan pronalasku čestih podgrafova "običnog" grafa u dijelu usporedbe podgrafova, ali znatno složeniji u dijelu pronalaska kandidata za česte podgrafove. Jedan od pristupa, koji se odnosi na neusmjereni graf s vremenski određenim bridovima, vremenski postojanim vrhovima i diskretnim poimanjem vremena, prikazanog u vremenskom nizu, je generiranje kanonske oznake svakog brida, koja u sebi sadržava stanja brida kroz vrijeme [94]. Sumiranjem svih stanja bridova u vremenskom nizu se dobiva jedinstveni dinamički graf (slika 4.3) koji se nakon toga dalje analizira. U primjeru sa slike su ta različita stanja bridova označena slovima *a*, *b*, *c*, i *d*, a nepostojanje brida u nekom vremenskom trenutku je označeno s ϵ.



Slika 4.3: Transformacija vremenskog niza grafova u dinamički graf (prema [94])

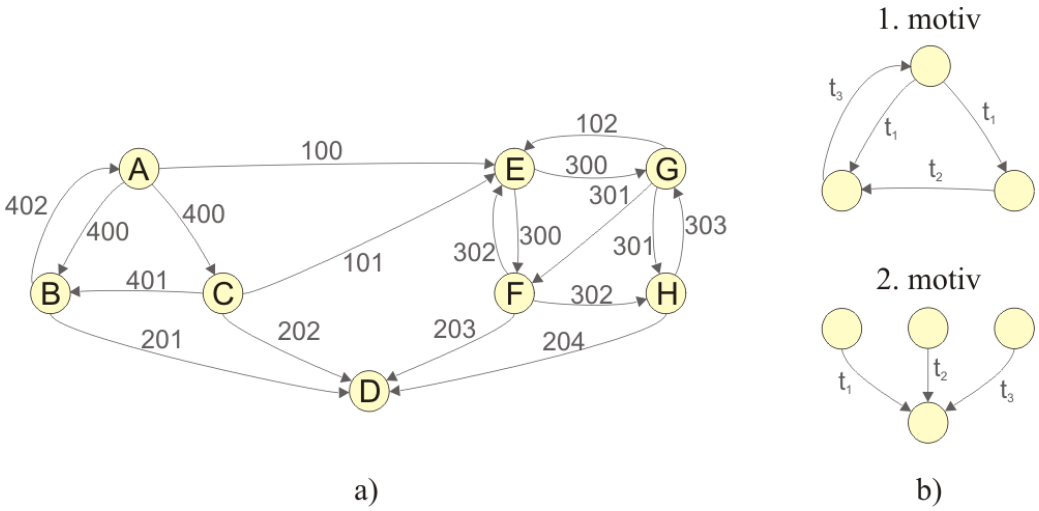
U slučaju kontinuiranog poimanja vremena u grafovima s vremenski određenim bridovima, česti vremenski grafovi se promatraju kao *vremenski motivi* [96, 97, 113]. U tom smislu se promatraju međusobni odnosi među bridovima ili lukovima i grupiraju prema svojoj udaljenosti u vremenski ograničena susjedstva. Na slici 4.4 je prikazan usmjereni graf s vremenski određenim lukovima (uz luk je prikazano vrijeme nastanka luka) (a) te dva vremenski ograničena susjedstva (b) i (c) unutar kojih su lukovi koji nastaju u razmaku od 10 vremenskih jedinica u odnosu na drugi luk iz susjedstva.

Na slici 4.5 je u dijelu (a) prikazan usmjereni vremenski graf s vremenski određenim lukovima, pri čemu je oznaka luka vrijeme njegova nastanka. U dijelu (b) su prikazana dva najveća vremenska podgrafa koja se pojavljuju najmanje tri puta u grafu i gdje je vremenski razmak



Slika 4.4: Vremenski graf (a) i dva vremenski ograničena susjedstva (b) i (c) (deset vremenskih jedinica; prema [96])

među događajima 1 vremenska jedinica, odnosno vrijedi $t_1 + 1 = t_2$, $t_2 + 1 = t_3$. Prvi vremenski motiv se odnosi na vrhove označene s $\{A, B, C\}$, $\{E, F, G\}$ i $\{G, H, F\}$, a drugi na vrhove označene s $\{B, C, D, F\}$, $\{C, D, F, H\}$ i $\{A, C, E, G\}$.



Slika 4.5: Vremenski graf (a) i dva najveća vremenska podgrafa (b), s razmakom događaja jednakim jednoj vremenskoj jedinici (prema [113])

Algoritam COMMIT [113] pokazuje znatno bolje rezultate od nekih drugih algoritama u pronalasku čestih statičkih podgrafova, no ograničen je u smislu da ne koristi oznake vrhova niti lukova te su samo lukovi vremenski određeni. Algoritam TGMiner [114] se također bavi pronalaskom čestih motiva u skupovima malih vremenskih grafova kojima su vremenski određeni lukovi, a samo vrhovi su označeni. Chronos [115] je mogućnostima bogat alat za analizu vremenskih grafova u kojima su također vremenski određeni lukovi. Algoritam predstavljen u [116] je učinkovit algoritam za pronalazak vremenskih podgrafova koji su vrlo blisko vremenski povezani, pri čemu graf također ima samo vremenski određene lukove.

Slijedom istraženih najrelevantnijih dostupnih radova vezanih uz pronalazak čestih vremenskih podgrafova, može se doći do zaključka da do sada prikazani algoritmi ne donose dovoljno prikladna svojstva koja bi bila iskoristiva u analizi grafa s vremenski označenim svim elemen-

tima grafa. Konkretno, ovi algoritmi se odnose na vremenske grafove s vremenski promjenjivim bridovima ili lukovima u obliku vremenskih nizova ili s vremenskim oznakama bridova ili lukova, ali bez mogućnosti postojanja vremenski označenih vrhova. Zbog ovoga, kao i činjenice da su algoritmi za pronalazak čestih podgrafova statičkog grafa znatno više istraženi i optimirani, u okviru ovog istraživanja će se odabrati neki od učinkovitih algoritama za pronalazak čestih podgrafova statičkog grafa i prilagoditi uporabi s grafovima s vremenski označenim vrhovima i lukovima.

4.3 Opis algoritma GraMi

S obzirom na do sada prikazane radove i tvrdnje u ovom poglavlju, i na osnovne kriterije za odabir algoritma koji će se koristiti ili prilagoditi za pronalazak čestih vremenskih podgrafova, dane u samom uvodu poglavlja, odabran je algoritam GraMi [110]. Ovaj algoritam treba prilagoditi radu s grafovima koji sadrže vremenski određene vrhove i lukove. Algoritam je namijenjen radu s velikim grafovima, usmjerenima i neusmjerenima, može raditi u preciznom i aproksimativnom načinu rada, a u testiranjima s grafovima većim od milijun elemenata su prezentirani najbolji rezultati među aktualnim algoritmima iste namjene. Osim toga, programski kôd je javno dostupan [117], što čini prilagodbu algoritma, implementaciju i testiranje znatno lakšim. Algoritam se djelomično oslanja na gSpan algoritam i koristi dio njegove programske implementacije.

U nastavku ovog odjeljka je prenesen kratki opis algoritma GraMi i njegovih ključnih svojstava.

4.3.1 Problem zadovoljavanja ograničenja

Specifičnost algoritma GraMi je pristup problemu pronalaska čestih podgrafova, pri kojem se ne pobrojavaju svi pronađeni jednaki podgrafovi. GraMi ovom problemu pristupa kao *problemu zadovoljavanja ograničenja* (eng. *Constraint Satisfaction Problem - CSP*). CSP je predstavljen trojkom $(\mathcal{X}, \mathcal{D}, C)$, gdje su:

- $\mathcal{X}$ uređeni skup varijabli,
- $\mathcal{D}$ skup domena koje odgovaraju varijablama u $\mathcal{X}$,
- C skup ograničenja među varijablama u $\mathcal{X}$.

Rješenje problema zadovoljavanja ograničenja je pridjeljivanje vrijednosti iz $\mathcal{D}$ varijablama iz $\mathcal{X}$, takvo da su ograničenja iz C zadovoljena.

U sklopu algoritma GraMi, predložen je sljedeći način preslikavanja problema izomorfizma grafova u problem zadovoljavanja ograničenja. Neka je $S(V_S, E_S, L_S)$ podgraf grafa $G(V, E, L)$, pri čemu su V i E vrhovi odnosno bridovi ili lukovi grafa i podgrafa, a L funkcija pridjeljivanja

oznaka vrhovima i bridovima ili lukovima. Definira se problem zadovoljavanja ograničenja podgraf-S-grafa-G kao $CSP(\mathcal{X}, \mathcal{D}, C)$ gdje:

1. $\mathcal{X}$ sadrži varijablu x_v za svaki vrh $v \in V_S$.
2. $\mathcal{D}$ je skup domena za svaku varijablu $x_v \in \mathcal{X}$. Svaka domena je podskup od V.
3. C sadrži sljedeća ograničenja:
   (a) $x_v \neq x_{v'}$, za svake dvije različite varijable $x_v, x_{v'} \in \mathcal{X}$.
   (b) $L(x_v) = L_S(v)$, za svaku varijablu $x_v \in \mathcal{X}$.
   (c) $L(x_v, x_{v'}) = L_S(v, v')$, za svaki par $x_v, x_{v'} \in \mathcal{X}$ takav da je $(v, v') \in E_S$.

Slijedom ovako definiranog problema, proizlazi da rješenje problema zadovoljavanja ograničenja podgraf-S-grafa-G odgovara izomorfizmu podgrafa S grafu G (definicija 23), za što su ključna ograničenja (b) i (c) iz skupa C. Još je potrebno definirati pridjeljivanje vrijednosti i način utvrđivanja je li neki podgraf čest ili ne. U tu svrhu se definira τ kao minimalna frekvencija koju podgraf treba imati da bi se smatrao čestim.

Pridjeljivanje vrha u varijabli v je *ispravno* ako i samo ako postoji rješenje koje pridjeljuje u varijabli v. Temeljem toga, ako je $(\mathcal{X}, \mathcal{D}, C)$ problem zadovoljavanja ograničenja podgraf-S-grafa-G, onda vrijedi sljedeće: MNI podrška S u G zadovoljava τ, ako i samo ako svaka varijabla u $\mathcal{X}$ ima najmanje τ različitih ispravnih pridjeljivanja, odnosno izomorfizama S u G. Ovo je preduvjet za korištenje navedenog načina evaluacije čestih podgrafova u sljedećim opisima algoritma GraMi.

4.3.2 Algoritam za pronalazak čestih podgrafova

Ovdje su prikazane osnovne funkcije koje čine algoritam GraMi.

Algoritam 1 FrequentSubgraphMining

Input: graf G
Input: minimalna frekvencija τ
Output: svi podgrafovi S od G takvi da je $s_G(S) \geq \tau$

```
1. result ← ∅
2. fEdges ← svi česti bridovi ili lukovi u G
3. for all e ∈ fEdges do
4.     result ← result ∪ SubgraphExtension(e, G, τ, fEdges)
5.     ukloni e iz G i fEdges
6. end for
7. return result
```

Varijante funkcija $FrequentSubgraphMining$ i $SubgraphExtension$ se koriste u sličnim algoritmima za pronalazak čestih podgrafova [106, 107]. Funkcija $FrequentSubgraphMining$ (algoritam 1) započinje identificiranjem skupa $fEdges$ koji sadrži sve česte bridove ili lukove u grafu, one koji se pojavljuju τ ili više puta u grafu. Za svaki od ovih bridova ili lukova, poziva se

funkcija $SubgraphExtension$ (algoritam 2) koji pokušava proširiti podgraf S čestim bridovima ili lukovima iz $fEdges$.

Algoritam 2 SubgraphExtension

Input: podgraf S grafa G
Input: minimalna frekvencija τ
Input: skup čestih bridova ili lukova $fEdges$
Output: svi česti podgrafovi od G koji proširuju S

```
1.  result ← ∅, candidateSet ← ∅
2.  for all e ∈ fEdges ∧ u ∈ S do
3.     if e se može iskoristiti za proširenje u then
4.        ext ← proširenje S sa e
5.        if ext nije već generirano then
6.           candidateSet ← candidateSet ∪ ext
7.        end if
8.     end if
9.  end for
10. for all c ∈ candidateSet do
11.    if IsFrequentCSP(S, G, τ) then
12.       result ← result ∪ SubgraphExtension(c, G, τ, fEdges)
13.    end if
14. end for
15. return result
```

Skup $candidateSet$ sadrži moguća proširenja koja još uvijek nisu promatrana. Provjera generiranih proširenja koristi kanonsku reprezentaciju podgrafa pri pretraživanju po dubini, $DFScode$ iz algoritma gSpan. Nakon toga se eliminiraju oni članovi iz $candidateSet$ skupa koji ne zadovoljavaju zadanu frekvenciju τ i funkcija $SubgraphExtension$ se dalje rekurzivno poziva, kako bi se česti podgrafovi dodatno proširili.

Testiranje je li podgraf česti se obavlja uz pomoć funkcije $IsFrequentCSP$ (algoritam 3). Pravila konzistentnosti koja se primjenjuju u algoritmu se odnose na isključivanje pojedinih vrhova iz domena (koji se prerijetko pojavljuju ili imaju različite oznake) i provjeru konzistentnosti pri pridjeljivanju dvaju varijabli. To znači da se za svako ograničenje $C(v,v')$ provjerava da za svaki vrh u domeni vrha v postoji vrh u domeni vrha v' koji zadovoljava $C(v,v')$. Nakon što su pravila konzistentnosti nametnuta, ako je kardinalnost domene manja od τ, funkcija vraća negativnu vrijednost. Inače se obavlja provjera svih rješenja i za svako se označavaju svi vrhovi u odgovarajućim domenama. Ukoliko svaka domena nakon toga ima najmanje τ označenih vrhova, onda to rješenje predstavlja česti podgraf i funkcija vraća pozitivnu vrijednost, a inače negativnu.

Algoritam 3 IsFrequentCSP

Input: podgraf S grafa G
Input: minimalna frekvencija τ
Output: *true* ako je S česti podgraf od G, a inače $false$

```
1. primijeni podgraf-S-grafa-G CSP
2. primijeni pravila konzistentnosti
3. if veličina ijedne domene je manja od τ then
4.    return false
5. end if
6. for all rješenje Solution podgraf-S-grafa-G do
7.    označi sve vrhove iz Solution u odgovarajućim domenama
8.    if sve domene imaju najmanje τ označenih vrhova then
9.       return true
10.   end if
11. end for
12. return false
```

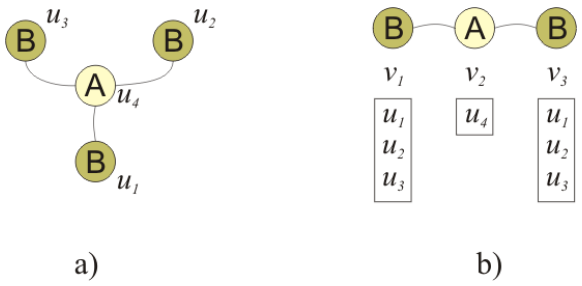
4.3.3 Optimizacije

U sklopu prezentacije algoritma GraMi su predložene i optimizacije u cilju ubrzanja postupka pronalaska čestih podgrafova. Optimizacije se odnose na opisanu funkciju $IsFrequentCSP$.

Potisnuto pročišćavanje (eng. *push-down pruning*) je tehnika koja se koristi za optimiranje liste kandidata za česte podgrafove. Kandidati za česte grafove se stvaraju proširenjem podgrafa-roditelja jednim bridom ili lukom. To znači da je podgraf-roditelj podstruktura podgrafova-djece. Ona pridjeljivanja varijabli koja su već eliminirana (pročišćena) za kandidata-roditelja ne mogu biti ispravna niti za kandidate-djecu, tako ih se može odmah eliminirati (pročistiti) prije nego se kandidati-djeca uopće promatraju.

Ukoliko vrhovi grafa sadrže *jedinstvene oznake* (eng. *unique labels*) to omogućava dodatnu optimizaciju. Preduvjet za ovu optimizaciju je da vrhovi sadrže samo jednu oznaku i da su te oznake jedinstvene za vrhove u grafu. U tom slučaju je olakšan pronalazak rješenja problema zadovoljavanja ograničenja podgraf-S-grafa-G. Ako svaki vrh ima jednu oznaku i te se oznake ne pojavljuju više puta u grafu, onda se vrh može pojaviti samo u jednoj domeni, pa je dovoljno primijeniti pravila konzistentnosti na ovaj problem zadovoljavanja ograničenja i nakon toga prebrojati veličine domena.

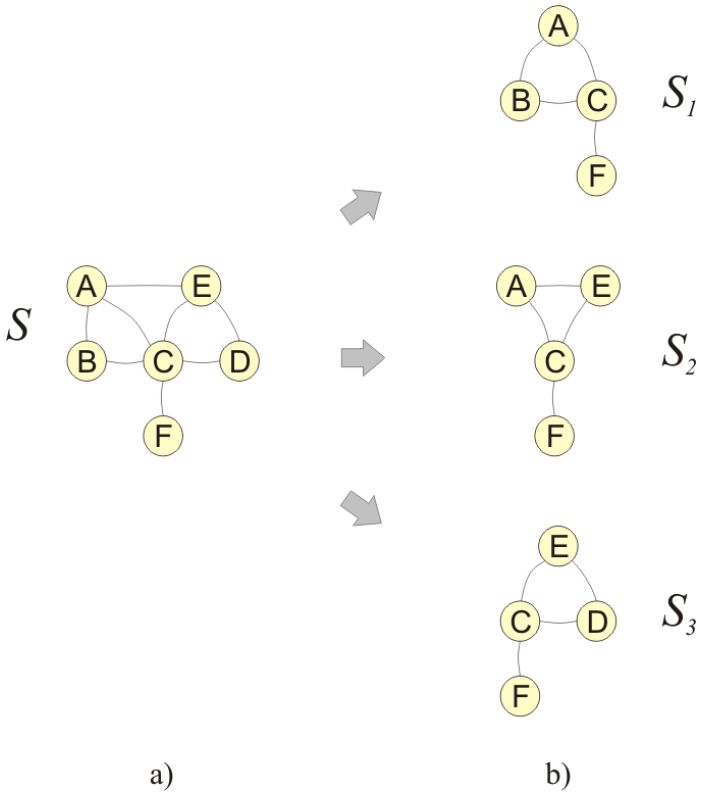
Moguće je provesti optimiranje vezano za *automorfizme* (eng. *automorphisms*). Automorfizam je izomorfizam grafa samome sebi, a pojavljuje se zbog simetrije. Jedan takav je prikazan na slici 4.6, gdje je za zadani graf G u dijelu slike (a) pokazan njegov podgraf S i moguća pridjeljivanja vrhova u_1, u_2 i u_3 varijablama tog podgrafa u dijelu slike (b). Pronalazak automorfizama pojedinog podgrafa reducira naknadna istraživanja i time ubrzava cijeli proces. Iako je složenost funkcije za pronalazak automorfizama $O(n^n)$, gdje je n broj vrhova u podgrafu, ovaj je utjecaj na složenost cijelog algoritma jako mali, jer je broj elemenata podgrafa uglavnom znatno manji



Slika 4.6: Graf (a), njegov podgraf i automorfizmi (b) (prema [110])

od broja elemenata grafa.

U određenim slučajevima, uvedeno je *lijeno pretraživanje* (eng. *lazy search*). U slučaju velikih grafova i velikog broja vrhova s istim oznakama, potraga za pojedinim djelomičnim pridjeljivanjima vrijednosti domena može trajati dulje vrijeme. U tu svrhu su autori uveli vremensko ograničenje takvih pretraga, s idejom da, ukoliko nije pronađeno u kratkom vremenu, vjerojatno neće utjecati na konačni ishod. Preduga se pretraživanja stoga prekidaju i pohranjuju u privremenu memoriju. Pretraživanje se dalje obavlja za druga pridjeljivanja i očekuje se da će tih pridjeljivanja biti dovoljno kako bi konkretni podgraf bio proglašen čestim, što znači da se prekinuta pretraživanja mogu zanemariti jer više nisu bitna. Ako nije pronađeno dovoljno pridjeljivanja da bi podgraf bio česti, a prebrojavanjem se utvrdi da prekinutih pretraživanja ima onoliko koliko bi trebalo da se prijeđe granica za česte podgrafove, onda će se ta pretraživanja nastaviti do konačnog utvrđivanja je li podgraf česti ili ne.



Slika 4.7: Podgraf (a) i njegova dekompozicija (b) (prema [110])

Pročišćavanje dekompozicijom (eng. *decomposition pruning*) se koristi upravo pri nastavljanju prekinutih pretraživanja. Pretraživanja u smislu pronalaska pridjeljivanja mogućih vrhova se optimiraju tako da se podgraf kandidat rastavi na nekoliko manjih grafova, kandidata-djece koji sadržavaju novododani brid ili luk, ali im nedostaju neki drugi bridovi (lukovi). Ideja iza ove optimizacije je da je brže pokazati da kandidati-djeca koji imaju manji broj vrhova ne zadovoljavaju problem zadovoljavanja ograničenja i na taj način eliminirati i kandidata-roditelja iz pretraživanja. Na dijelu (a) slike 4.7 je prikazan kandidat-roditelj kojem je posljednji dodan brid između vrhova C i F te u dijelu slike (b) tri kandidata-djeteta koji su nastali dekompozicijom roditelja.

Izvan originalnog algoritma je predloženo i pročišćavanje kandidata koje uzima u obzir bridove ili lukove koje kandidati međusobno dijele, a s kojim se dobiva ubrzanje algoritma od nekoliko postotaka ukupnog vremena [118].

4.3.4 Složenost

Složenost algoritma se odnosi na složenost optimirane funkcije $IsFrequentCSP$. U tom kontekstu se promatra broj vrhova grafa N i broj vrhova podgrafa n. Pojedine optimizacije i dijelovi algoritma imaju sljedeće složenosti:

- potisnuto pročišćavanje: $O(n^2)$,
- jedinstvene oznake: $O(n)$,
- automorfizam: $O(n^n)$,
- primjena konzistentnosti: $O(Nn)$,
- lijeno pretraživanje: $O(N)$,
- nastavak prekinutih pretraživanja: $O(N^{n-1})$.

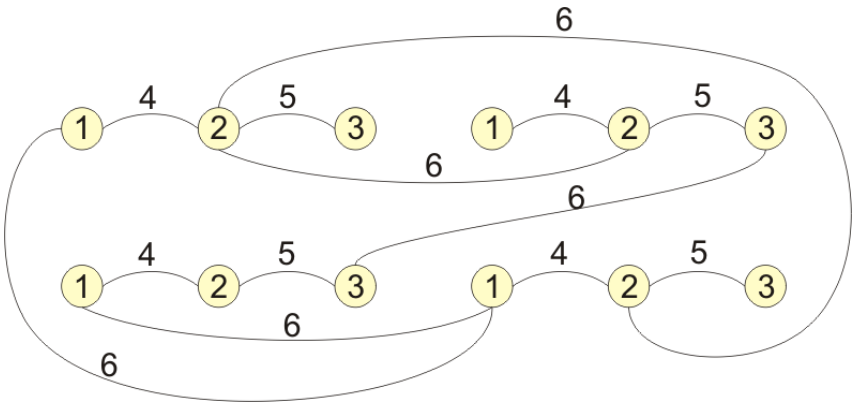
S obzirom da je veličina podgrafa n znatno manja od veličine grafa N, tako su prve tri stavke zanemarive, a složenost algoritma je određena nastavkom prekinutih pretraživanja $O(N^{n-1})$. Autori nadalje navode da, ukoliko je p vjerojatnost da je vrh u domeni varijable ispravan, onda je složenost algoritma $O(n \cdot \tau / p \cdot N^{n-1})$.

4.3.5 Primjer pronalaska čestih podgrafova

Način funkcioniranja algoritma GraMi bit će prikazan na jednostavnom neusmjerenom grafu s dvanaest vrhova i trinaest bridova.

Primjer 1. Graf sa slike 4.8 će se pretražiti algoritmom GraMi za pronalazak čestih grafova.

Graf sadrži vrhove označene brojčanim oznakama 1, 2 i 3 te bridove označene brojčanim oznakama 4, 5 i 6. Brojčane oznake se koriste iz praktičkih razloga: usporedbe brojčanih podataka su u velikoj većini programskih jezika i pripadnih platformi znatno brže i zahtijevaju



Slika 4.8: Graf za pronalazak čestih podgrafova u primjeru

manje resursa nego usporedbe slovčanih podataka. Implementacija algoritma GraMi također koristi tu prednost.

Ovaj je graf potrebno opisati unutar tekstualne datoteke, na način da se najprije opišu vrhovi (početno slovo u retku je *v*) a nakon njih bridovi (početno slovo u retku je *e*). Svaki vrh slijedi njegov jedinstveni identifikator (brojevi koji počinju od nula) i oznaka, a svaki brid slijede jedinstveni identifikatori vrhova koje spaja i oznaka brida. Ako se radi o usmjerenom grafu, onda redoslijed vrhova u tom retku određuje smjer luka. Ako pak postoji više bridova među istim vrhovima, oni se višekratno navode. Redoslijed bridova u datoteci nije relevantan. Datoteka s ovakvim opisom grafa sa slike ima sljedeći sadržaj:

```
v 0 1
v 1 2
v 2 3
v 3 1
v 4 2
v 5 3
v 6 1
v 7 2
v 8 3
v 9 1
v 10 2
v 11 3
e 0 1 4
e 1 2 5
e 1 4 6
e 3 4 4
e 4 5 5
e 5 8 6
e 6 7 4
e 7 8 5
e 6 9 6
e 9 10 4
```

```
e 10 11 5
e 1 10 6
e 0 9 6
```

U stvaranju gornjeg sadržaja, bilo je potrebno vrhovima pridijeliti identifikatore, pa su vrhovima u gornjem retku grafa pridjeljeni identifikatori 0, 1, 2, 3, 4 i 5, a vrhovima u donjem retku grafa 6, 7, 8, 9, 10 i 11.

Algoritmu se pri izvođenju predaju osnovni parametri kao što su lokacija datoteke s ulaznim grafom, minimalna frekvencija (prag) za česte podgrafove i tip grafa (u ovom slučaju neusmjereni) te drugi parametri koji se odnose na korištenje programa. Pronađeni podgrafovi se ispisuju u unaprijed definiranu datoteku, u sličnom formatu kao što je zapisan ulazni graf. Prvi redak u toj datoteci je broj sekundi koliko je trajalo izvršavanje programa, a drugi broj pronađenih čestih podgrafova, nakon čega se ispisuju sami česti podgrafovi.

U slučaju poziva programa s parametrom $f = 5$ izlazna datoteka ima ovakav sadržaj:

```
0.031
0
```

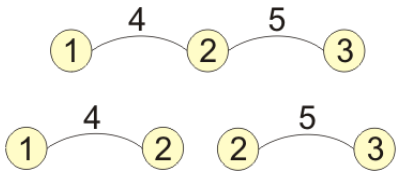
što znači da je pretraživanje trajalo 0,031 sekundi i nije pronađen niti jedan dovoljno česti podgraf.

Prilikom poziva programa s parametrom $f = 4$ sadržaj izlazne datoteke je:

```
0.114
3
0:
v 0 1
v 1 2
v 2 3
e 0 1 4
e 1 2 5
1:
v 0 1
v 1 2
e 0 1 4
2:
v 0 2
v 1 3
e 0 1 5
```

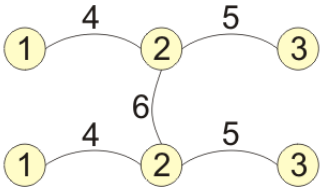
što pokazuje tri pronađena česta podgrafa koji se ponavljaju minimalno četiri puta, onaj pod brojem 0 s tri vrha i dva brida među njima, a ostala dva s po dva vrha i bridom među njima. Ova se tri grafa mogu prevesti u podgrafove prikazane na slici 4.9. Vidljivo je da su donja dva

podgrafa zapravo podgrafovi gornjeg čestog podgrafa.



Slika 4.9: Pronađeni podgrafovi s frekvencijom 4

Pri pozivu programa s pragom frekvencije 3, pronalazi se 19 čestih podgrafova. Najveći, podgraf sa 6 vrhova, je prikazan na slici 4.10.



Slika 4.10: Najveći pronađeni podgraf s frekvencijom 3

Pri pozivu programa s pragom frekvencije 2 pronalazi se 52 česta podgrafa. ■

Poglavlje 5

Usporedba grafova

Usporedba grafova se obavlja u cilju određivanja razine sličnosti dvaju grafova. U sklopu metode koja se u disertaciji istražuje, ponavljajuća razina sličnosti između čestog podgrafa i drugog podgrafa koji nije klasificiran kao česti može implicirati anomaliju koju je potrebno detaljnije istražiti. U ovom poglavlju će biti prikazani načini usporedbe grafova, a s obzirom da se usporedba obavlja s ciljem pronalaska anomalija, prikazat će se i ranije predloženi algoritmi za pronalazak anomalija u grafovima.

5.1 Pronalazak anomalija u grafovima

Pronalazak anomalija u grafovima ima jednaku svrhu kao i pronalazak anomalija u svim drugim podacima. No, i ovo područje istraživanja postaje aktivno tek u posljednje vrijeme, aktualizacijom raznih načina trajne pohrane podataka u grafovima. S druge strane, dio je vrlo aktivnog područja istraživanja pronalaska anomalija u podacima općenito, pa u tom smislu ima i neke zajedničke pristupe, kao što su na primjer statistička odstupanja pojedinih dijelova grafa i sl.

Poimanje anomalija u podacima koji su u obliku grafa je često različito od poimanja anomalija u podacima u kojima nedostaju dodatne informacije o povezanosti entiteta, gdje se uglavnom govori o pronalasku odstupajućih vrijednosti (eng. *outlier detection*). Također, zbog različitih modela grafova i sama poimanja anomalija u grafovima se međusobno znatno razlikuju. Anomalije u grafovima mogu varirati od jednostavnih, poput vrhova s većim brojem incidentnih bridova (lukova) od uobičajenoga ili bridova (lukova) koji se među vrhovima vremenskog grafa pojavljuju u neočekivano vrijeme, do složenih kao što su posebno jako povezani podgrafovi ili velike cikličke veze u pretežno acikličnom grafu. U recentnim preglednim radovima [119, 120, 121, 122] autori daju temeljit pregled algoritama za pronalazak anomalija u statičkim i dinamičkim grafovima, fokusirajući se na grafove bez svojstava ili oznaka elemenata i na grafove sa svojstvima i oznakama elemenata. U tom smislu, u [119] definiraju anomalije u statičkom grafu na sljedeći način: ako je dano trenutno stanje grafovske baze podataka (s ozna-

kama i svojstvima ili bez njih), anomalije su vrhovi i/ili bridovi (lukovi) i/ili podgrafovi koji su "rijetki i različiti" ili znatno odstupaju od uzoraka koji se mogu pronaći u grafu. Anomalije u dinamičkom grafu pak definiraju ovako: ako je dan vremenski niz grafova (s oznakama i svojstvima ili bez njih), anomalije su vremenski trenutci u kojima se događaju najveće promjene te oni vrhovi, bridovi (lukovi) ili dijelovi grafa koji najviše doprinose promjeni.

Pri proučavanju anomalija u statičkim grafovima, kako je već spomenuto, promatraju se dvije vrste algoritama, ovisno o vrsti grafa koji analiziraju:

- grafovi čiji elementi nemaju oznaka niti svojstva
- grafovi čiji elementi imaju oznake i svojstva.

Kod proučavanja anomalija u grafovima bez oznaka i svojstava jedina je raspoloživa informacija njihova struktura, pa su stoga i algoritmi koji se primjenjuju nad njima orijentirani prema toj informaciji. Ovi algoritmi mogu biti temeljeni samo na lokalnoj strukturi grafa ili na zajednicama (eng. *communities*) tj. skupinama gusto povezanih vrhova. Algoritmi koji se odnose na lokalnu strukturu grafa zasnivaju se na mjerama lokaliteta grafa kako bi identificirali uobičajene vrhove i podgrafove te temeljem toga zaključili koji od njih su neuobičajeni. Algoritmi temeljeni na zajednicama pronalaze one elemente grafa koji spajaju zajednice a ne pripadaju nijednoj od njih te takve elemente grafa smatraju anomalijama.

Algoritmi koji mogu analizirati grafove s oznakama i svojstvima osim strukture mogu koristiti i ove dodatne informacije koje pripadaju elementima grafa i pobliže ih opisuju. I ovi algoritmi mogu biti temeljeni na lokalnoj strukturi grafa ili na zajednicama. Algoritmi orijentirani ka lokalnoj strukturi uglavnom pronalaze vrhove ili podgrafove koji su rijetki (ne pojavljuju se često u grafu) u strukturnom smislu (povezanost vrhova ili podgrafova) i svojstveno (njihova svojstva ili oznake nisu česte). Algoritmi koji su temeljeni na zajednicama pokušavaju identificirati one vrhove čije vrijednosti svojstava znatno odstupaju od ostalih članova zajednice kojoj pripadaju (npr. automobil s dizelskim motorom koji je u zajednici u kojoj svi ostali automobili imaju benzinske motore).

Neki od algoritama za pronalazak anomalija u statičkim grafovima su prikazani u tablici 5.1.

Tablica 5.1: Neki od algoritama za pronalazak anomalija u statičkim grafovima

algoritam	vrsta ulaznog grafa	moguća parametrizacija	vrsta izlaznih informacija
OddBall [123]	bez oznaka i svojstava	ne	razine anomalija vrhova
Autopart [124]	bez oznaka i svojstava	ne	binarna klasifikacija lukova
Subdue [125]	s oznakama i svojstvima	da	razine anomalija podgrafova
CODA [126]	s oznakama i svojstvima	da	binarna klasifikacija vrhova
gOutRank [127]	s oznakama i svojstvima	da	razine anomalija vrhova

Iako se vremenski graf u okviru ove disertacije planira prikazati kao statički graf proširen vremenskim konceptom koji se odnosi na vrhove i lukove, algoritmi za pronalazak anomalija

u statičkim grafovima nisu primjenjivi na slučaj pronalaska odstupanja u ovakvim vremenskim grafovima. Najprije, u obzir mogu ući samo oni algoritmi koji analiziraju grafove s oznakama i svojstvima elemenata, a od tih su algoritmi koji se temelje na zajednici konceptualno neprimjenjivi. Od ostalih, također se mogu eliminirati oni algoritmi čiji je rezultat binarna klasifikacija vrha ili podgrafa (jest/nije anomalija) i oni koji su fokusirani na anomalije vrhova, a ne podgrafova. Sukladno prezentiranim algoritmima za pronalazak anomalija u statičkim grafovima [119], preostaje samo još jedan, Subdue [125]. Ovaj pak algoritam ne nudi dovoljne mogućnosti prilagodbe za selektivni pronalazak uzoraka i usporedbu potencijalnih anomalija spram pojedinih grupa korisnika.

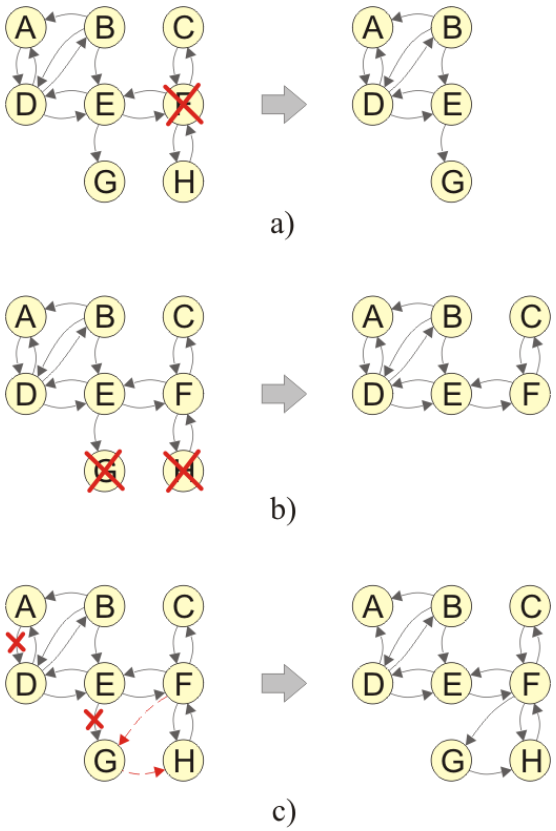
Pri istraživanju anomalija u dinamičkim grafovima, najveća pažnja se posvećuje identifikaciji događaja koji se mogu karakterizirati kao anomalije. Istraživanja anomalija u dinamičkim grafovima se uglavnom odnose na grafove bez dodatnih oznaka i/ili svojstava elemenata [119] pa se tako algoritmi odnose na razne načine utvrđivanja razlika u strukturi grafa u raznim vremenskim trenucima. Algoritmi se mogu grupirati u one koji promatraju razlike u odlikama grafova (npr. [128, 129, 130, 131]), one koji se temelje na dekompoziciji matrica ili tenzora susjedstva (npr. [132, 133]), one koji ne promatraju promjene cijelog grafa nego se fokusiraju na promjene u zajednicama ili klasterima unutar grafa (npr. [134, 135]) te one koji pretpostavljaju da je u određenom vremenskom razdoblju stanje grafa "normalno", iz kojeg mogu zaključiti što je normalno, da bi usporedili nova stanja grafa i zaključili postoje li odstupanja (npr. [136, 137]).

U sklopu ove disertacije će se koristiti grafovi koji posjeduju neograničene kombinacije oznaka, svojstava elemenata i njihovih vremenskih karakteristika, što se sve, uz strukturu grafa, može smatrati odlikama grafova koje mogu biti slobodno zadane. Stoga su, od navedenih algoritama za istraživanje anomalija u dinamičkim grafovima, ovdje posebno zanimljivi oni koji se temelje na odlikama grafova koje mogu biti slobodno zadane i koje daju mjeru sličnosti između dva grafa, kako bi se razina anomalije mogla kvantificirati.

Osnovna ideja ovih algoritama je da je moguće napraviti skup odlika (kao npr. raspodjelu stupnja vrhova, dijametar grafa i slično) koji opisuje normalno stanje grafa a temeljem kojeg se izračunava razlika, odnosno sličnost među dva vremenski susjedna grafa u vremenskom nizu. Temeljem zadane ili izračunate granice sličnosti ili razlike, pojedini se graf u vremenskom nizu proglašava anomalijom. Pri tom se istraživanja razlikuju po onom što ulazi u skup odlika grafa, jer ne postoji jedan univerzalan skup za svaku primjenu.

Među algoritmima koji se temelje na odlikama grafova koje mogu biti slobodno zadane i koji daju mjeru sličnosti, posebno su zanimljivi oni predloženi od Papadimitrioua i drugih [131]. U tom radu autori promatraju nekoliko mjera sličnosti i primjenjuju ih na različite razlike među grafovima, kako bi zaključili koja je od tih mjera najprimjerenija. U istraživanju se fokusiraju na sličnosti tzv. web-grafa, odnosno približne trenutne slike internetskih stranica koju kreiraju

alati za pretraživanje. Stranice predstavljaju vrhove, lukovi veze (linkove) među njima, a vrhovi imaju i težinu, odnosno kvalitetu, koja je određena PageRank algoritmom [138].



Slika 5.1: Razlike među grafovima kojima se testiraju izračuni sličnosti u [131]

Izračuni sličnosti koji se testiraju su: rangiranje vrha (eng. *vertex ranking*), sličnost vektora (eng. *vector similarity*), preklapanje vrha/luka (eng. *vertex/edge overlap*), sličnost slijeda (eng. *sequence similarity*) i sličnost potpisa (eng. *signature similarity*). Razlike u grafovima koje se uspoređuju su prikazane na slici 5.1. Na dijelu slike (a) su prikazane razlike među grafovima u kojima nedostaje jedan dio grafa, na dijelu slike (b) nedostaju slučajni vrhovi, dok na dijelu slike (c) dolazi do promjene u lukovima. Različiti izračuni sličnosti imaju različite performanse na testovima spram ove tri vrste razlika u grafovima, a u tablici 5.2 su prikazani dobiveni rezultati. Autori su ocijenili performanse algoritama trima opisnim ocjenama (loše, dobro, vrlo dobro).

Tablica 5.2: Usporedba različitih izračuna sličnosti i razlika među grafovima u [131]

	nedostajući podgraf	nedostajući slučajni vrhovi	promjena u lukovima
rangiranje vrha	vrlo dobro	loše	loše
sličnost vektora	vrlo dobro	dobro	dobro
preklapanje vrha/luka	vrlo dobro	dobro	loše
sličnosti slijeda	dobro	loše	vrlo dobro
sličnosti potpisa	vrlo dobro	vrlo dobro	vrlo dobro

Kako je vidljivo iz prikazane tablice, algoritam sličnosti potpisa je pokazao najbolje rezul-

tate u usporedbama grafova, pa je stoga, i činjenice da je znatno prilagodljiv usporedbi vremenskih grafova, odabran kao algoritam koji će se prilagoditi za usporedbu i određivanje razine sličnosti grafova u sklopu ove disertacije. Ovaj je algoritam opisan u sljedećem potpoglavlju.

5.2 Opis algoritma sličnosti potpisa

Algoritam sličnosti potpisa se temelji na ideji da su dva objekta slična ukoliko su njihovi potpisi slični. Pri korištenju ovog algoritma, graf se pretvara u skup odlika, koje se po načelu slučajnih brojeva projiciraju na prostor odlika manje dimenzije (potpise) koji se onda mogu uspoređivati.

Ovaj se algoritam, kako je opisan u [131], oslanja na *SimHash* algoritam [139], i njegovu primjenu na usporedbu dokumenata [140].

Algoritam *SimHash* se koristi za preslikavanje visokodimenzionalnih vektora u otiske, odnosno potpise, manjih dimenzija. Algoritam pretpostavlja da se, uzevši u obzir Hammingovu udaljenost (u ovom smislu, broj razlika) među njima, slični vektori preslikavaju u slične potpise. Stoga je predložena njegova primjena za utvrđivanje sličnosti dokumenata, i to na sljedeći način.

Dokument se transformira u skup težinskih karakteristika $L = \{(c_i, w_i)\}$ (u originalnom opisu algoritma se umjesto oznake karakteristike c_i koristi oznaka t_i, za englesku riječ *token*, no ovdje je izbjegnuta zbog uporabe iste oznake za n-torku i vremenske oznake). Karakteristika, odnosno odlika c_i može biti riječ ili fraza koja se pojavljuje u dokumentu, a njegova težina w_i je pojavnost te riječi u dokumentu. Riječi, odnosno karakteristike se pojavljuju jednom u cijelom skupu L, i ovaj težinski skup se može promatrati kao višedimenzionalni vektor, pri čemu su dimenzije različite jedinstvene karakteristike koje se mogu pronaći u svim dokumentima. Svaka karakteristika se projicira u b-dimenzionalni prostor, na način da se slučajno odabire b brojeva iz skupa $\{-w_i, +w_i\}$.

U [131] je ovo projiciranje obavljeno tako da je za svaku karakteristiku stvoren slučajni bit-vektor (koji sadrži samo nule i jedinice) veličine b. Na mjesto svake jedinice u tom vektoru postavlja se vrijednost $+w_i$, a na mjesto svake nule vrijednost $-w_i$ (w_i je pojavnost te karakteristike u dokumentu). Konačni otisak, odnosno potpis dokumenta se dobiva zbrajanjem svih b-dimenzionalnih težinskih vektora stvorenih za sve karakteristike tog dokumenta (zbrajaju se vrijednosti na istim pozicijama svih vektora). Rezultantni vektor također ima duljinu b i konvertira se u bit-vektor h, na način da se i-ta vrijednost postavlja na 1 ako je vrijednost na tom mjestu nenegativna, a na 0 ako je vrijednost na tom mjestu negativna. Bit-vektor h se naziva potpisom dokumenta.

Postupak se obavlja za oba dokumenta L i L' koji se uspoređuju. Na kraju, sličnost između

njih je jednaka postotku jednakih bitova u njihovim odgovarajućim potpisima h i h':

$$sim_{SimHash}(L, L') = 1 - \frac{Hamming(h, h')}{b} \tag{5.1}$$

pri čemu funkcija *Hamming* vraća broj razlika među ulaznim bit-vektorima.

S obzirom da je uključena funkcija slučajnih brojeva u generiranje pojedinih težinskih vektora, veličina potpisa b treba biti velika koliko je to praktično za implementaciju. Također treba primijetiti da će algoritam vratiti rezultate u intervalu $[0, 1]$, pri čemu je 0 vrijednost potpune različitosti, a 1 vrijednost jednakosti.

Složenost algoritma za usporedbu ovisi o veličini bit-vektora b, što zapravo znači $O(1)$, jer je b konstanta.

Primjer 2. Izračun sličnosti algoritmom $sim_{SimHash}$ prikazat će se na usporedbi dvaju skupova težinskih karakteristika koji predstavljaju dvije rečenice:

R_1 = "Bach je najbolji skladatelj svih vremena"
R_2 = "Mozart je najbolji austrijski skladatelj"

Ako se svaka riječ promatra kao jedna karakteristika, a težine se pridjeljuju sukladno rečeničnom dijelu kojem riječ pripada, tako da subjekt i predikat vrijede 4, objekt 2, a ostali rečenični dijelovi 1, onda se rečenice mogu prikazati skupovima:

$L_1 = \{(Bach, 4),\ (je, 4),\ (najbolji, 1),\ (skladatelj, 2),\ (svih, 1),\ (vremena, 1)\}$
$L_2 = \{(Mozart, 4),\ (je, 4),\ (najbolji, 1),\ (austrijski, 1),\ (skladatelj, 2)\}$

S obzirom da sve karakteristike ne postoje u oba skupa, u onom skupu u kojem ne postoje se dodaju s negativnom težinom, pa se na taj način "kažnjava" njihov nedostatak u skupu. Nakon tog proširenja, skupovi izgledaju ovako:

$L_1 = \{(Bach, 4),\ (je, 4),\ (najbolji, 1),\ (skladatelj, 2),\ (svih, 1),$
$(vremena, 1),\ (Mozart, -4),\ (austrijski, -1)\}$

$L_2 = \{(Mozart, 4),\ (je, 4),\ (najbolji, 1),\ (austrijski, 1),$
$(skladatelj, 2),\ (Bach, -4),\ (svih, -1),\ (vremena, -1)\}$

Ukupno dakle postoji osam različitih karakteristika, za koje je potrebno stvoriti slučajne bit-vektore. Neka je za ovaj primjer $b = 10$ (dok bi, za smanjenje utjecaja generatora slučajnih brojeva, u stvarnom izračunu taj broj bio znatno veći) te neka su za karakteristike stvoreni pri-

padni slučajni bit-vektori kako je navedeno u tablici 5.3.

Tablica 5.3: Karakteristike i pripadni slučajni bit-vektori

karakteristika	bit-vektor
Mozart	0 1 1 1 0 1 0 0 0 1
Bach	0 0 1 1 0 1 1 1 0 0
je	0 0 0 1 1 1 0 0 0 1
najbolji	1 0 1 0 1 0 1 0 1 0
skladatelj	1 0 0 1 0 0 1 0 1 1
svih	0 1 1 0 0 0 1 1 1 0
vremena	1 0 0 0 0 1 1 0 0 1
austrijski	0 0 1 1 0 1 1 1 0 0

Postupak stvaranja potpisa rečenice R_1 je prikazan u tablici 5.4. Na pripadne slučajne bit-vektore su primijenjene težine karakteristika, nakon čega su sumirani svi težinski vektori i od toga stvoren bit-vektor koji predstavlja potpis rečenice R_1.

Tablica 5.4: Stvaranje potpisa rečenice R_1

karakteristika	težina	težinski vektor									
Mozart	-4	4	-4	-4	-4	4	-4	4	4	4	-4
Bach	4	-4	-4	4	4	-4	4	4	4	-4	-4
je	4	-4	-4	-4	4	4	4	-4	-4	-4	4
najbolji	1	1	-1	1	-1	1	-1	1	-1	1	-1
skladatelj	2	2	-2	-2	2	-2	-2	2	-2	2	2
svih	1	-1	1	1	-1	-1	-1	1	1	1	-1
vremena	1	1	-1	-1	-1	-1	1	1	-1	-1	1
austrijski	-1	1	1	-1	-1	1	-1	-1	-1	1	1
suma težina		0	-14	-6	2	2	0	8	0	0	-2
bit-vektor h_1		1	0	0	1	1	1	1	1	1	0

Postupak stvaranja potpisa rečenice R_2 je prikazan u tablici 5.5.

Tablica 5.5: Stvaranje potpisa rečenice R_2

karakteristika	težina	težinski vektor									
Mozart	4	-4	4	4	4	-4	4	-4	-4	-4	4
Bach	-4	4	4	-4	-4	4	-4	-4	-4	4	4
je	4	-4	-4	-4	4	4	4	-4	-4	-4	4
najbolji	1	1	-1	1	-1	1	-1	1	-1	1	-1
skladatelj	2	2	-2	-2	2	-2	-2	2	-2	2	2
svih	-1	1	-1	-1	1	1	1	-1	-1	-1	1
vremena	-1	-1	1	1	1	1	-1	-1	1	1	-1
austrijski	1	-1	-1	1	1	-1	1	1	1	-1	-1
suma težina		-2	0	-4	8	4	2	-10	-14	-2	12
bit-vektor h_2		0	1	0	1	1	1	0	0	0	1

S obzirom na određene potpise $h_1 = 1001111110$ i $h_2 = 0101110001$, vidljiva je njihova razlika na 6 mjesta te i je $Hamming(h_1, h_2) = 6$, a $sim_{SimHash}(R_1, R_2) = 1 - 6/10 = 0,4$. ■

5.3 Primjena algoritma sličnosti potpisa na sličnost grafova

Opisani algoritam sličnosti potpisa za usporedbu dokumenata može se primijeniti na sličnost grafova. U tu svrhu definira se funkcija $\phi : G \rightarrow L$, kao funkcija koja će od ulaznog grafa G stvoriti skup težinskih karakteristika L. Nakon što su dva grafa prikazana kao skupovi težinskih karakteristika, može se primijeniti algoritam sličnosti potpisa na jednak način kako je opisan u prethodnom potpoglavlju te dobiti mjera sličnosti između njih.

Sličnost potpisa grafova G i G' se dakle može pisati kao:

$$sim_{SS}(G, G') = sim_{SimHash}(\phi(G), \phi(G')) \tag{5.2}$$

pri čemu je $sim_{SimHash}$ dana u formuli 5.1, a oznaka SS se odnosi na kraticu sličnosti potpisa (eng. *signature similarity*).

Funkcija ϕ može biti takva da predstavlja odlike grafa prema potrebi, što čini ovaj algoritam jednostavno prilagodljivim usporedbi vremenskih grafova s vremenski određenim vrhovima i lukovima. Moguće je odlučiti na koji način će se težinama nagraditi ili kazniti postojanje odnosno nepostojanje određenih elemenata grafa i njihovih svojstava, kao i drugih osobina grafa, što će biti tema pri usporedbi grafova u okviru ove disertacije.

Poglavlje 6

Konverzija relacijskih baza podataka u grafovske baze podataka

U ovom poglavlju je opisan postupak koji je osnova pretvaranja snimljenog traga relacijske baze podataka u vremenski graf sa svojstvima svih elemenata grafa, s obzirom da je snimljeni trag također u obliku relacijske baze podataka, a vremenski graf sa svojstvima svih elemenata grafa će se moći pohraniti u grafovsku bazu podataka, kako će biti pokazano u sljedećem poglavlju. Taj postupak je konverzija podataka iz relacijskog modela u model grafa sa svojstvima. Koncept, algoritmi, doprinosi i usporedba sa sličnim pristupima su objavljeni u [141].

6.1 Opis koncepta

Pod konverzijom podataka iz relacijske baze podataka u grafovsku bazu podataka podrazumijeva se čitanje svih podataka iz relacijske baze i stvaranje odgovarajućeg sadržaja grafovske baze podataka. Relacijska baza podataka pri tom procesu biva neizmijenjena, a grafovska baza podataka se nadopunjuje novim elementima - vrhovima i lukovima koji predstavljaju podatke iz relacijske baze podataka i njihove odnose.

Ključni aspekti algoritma za konverziju podataka iz relacijske baze podataka u grafovsku bazu podataka su:

1. Graf koji nastaje tijekom konverzije je usmjeren, a ne mora nužno biti povezan (odnosno, može se raditi o više odvojenih grafova unutar prostora grafovske baze podataka). Povezanost i acikličnost ovisi o strukturi relacijske baze podataka i podataka koje sadrži.
2. Svaki vrh predstavlja jednu n-torku iz relacije u relacijskoj bazi podataka. Neki lukovi također predstavljaju n-torke, a drugi predstavljaju strane ključeve.
3. Smjer luka koji predstavlja strani ključ je od vrha koji predstavlja referencirajuću n-torku prema vrhu koji predstavlja referenciranu n-torku.
4. Oznake (tipovi) se koriste za označavanje različitih vrsta vrhova i lukova. Svaki vrh

i svaki luk dobivaju točno jednu oznaku, iako neki sustavi za upravljanje grafovskim bazama podataka podržavaju i mogućnost da elementi imaju više oznaka. Oznake pomažu odrediti vrstu vrha ili luka i vrlo su važne za ovaj algoritam.

5. Oznaka vrha je ime relacije u relacijskoj bazi podataka čiju n-torku vrh predstavlja. Isto vrijedi i za luk ako on predstavlja n-torku. Ako luk predstavlja strani ključ, onda se ime stranog ključa u relacijskoj bazi koji predstavlja koristi kao oznaka luka.
6. Svojstva vrhova i lukova su parovi ključ-vrijednost koji predstavljaju atribute n-torke i njihove vrijednosti. Ime atributa postaje ime svojstva, a njegova vrijednost u n-torki postaje vrijednost svojstva. Lukovi koji predstavljaju strane ključeve nemaju svojstva.
7. Svaki vrh ili luk koji predstavlja n-torku također dobiva svojstvo *id*, koje dobiva vrijednost atributa primarnog ključa n-torke. Ako je primarni ključ kompozitni, onda se spajaju (konkateniraju) vrijednosti svih atributa primarnog ključa. Ovo znači da će svi elementi grafa (vrhovi ili lukovi) koji predstavljaju n-torke iz iste relacije relacijske baze, imati istu oznaku i jedinstvene identifikatore.
8. Relacije u relacijskoj bazi, koje imaju točno dva strana ključa, a čiji primarni ključ nije referenciran od ijedne tablice, su one čije će n-torke biti predstavljene lukovima. Ovi će lukovi spajati vrhove koji predstavljaju dvije referencirane n-torke putem stranih ključeva. Smjer ovakvog luka je od vrha koji predstavlja n-torku koja je prva referencirana u toj relaciji prema vrhu koji predstavlja n-torku koja je druga referencirana.
9. N-torke svih ostalih tablica će biti predstavljene vrhovima.
10. Atributi n-torki koji su dio stranog ključa ne formiraju svojstva. Informacija koju oni nose je zapisana u vezi na drugi element grafa, referencirani vrh ili luk. Vrijednosti primarnog ključa su sadržane u svojstvu *id*, strani ključevi su sadržani u odnosima spram drugih elemenata grafa, a samo su vrijednosti zavisnih atributa upisani kao svojstva elemenata grafa.

6.2 Određivanje podataka o konverziji

U cilju stvaranja učinkovitog algoritma za popunjavanje grafovske baze podataka podacima iz relacijske baze podataka, potrebno je najprije analizirati metapodatke relacijske baze podataka. Osnovni cilj je smanjiti broj ponavljanja u radu s podacima svake relacije.

Sustavi za upravljanje relacijskim bazama podataka se općenito pridržavaju Coddovih pravila o relacijskom modelu, tako da su metapodaci baze podataka (opis svih relacija, primarnih i stranih ključeva, opis atributa i sl.) dostupni na isti način kao i sami podaci unutar baze podataka.

Kroz analizu metapodataka baze podataka, pronalaze se:

- relacije čije n-torke će postati lukovi,

- relacije čije n-torke će postati vrhovi,
- cikličke relacije koje formiraju strani ključevi, koje će formirati cikličke veze u grafu,
- atributi u svakoj relaciji koji su dijelovi primarnog ključa, koji će se koristiti kao *id* svojstva,
- atributi u svakoj relaciji koji su dijelovi stranih ključeva, koji će se koristiti u stvaranju lukova,
- atributi u svakoj relaciji koji nisu dio nijednog ključa, koji će se koristiti u stvaranju svojstava vrhova i lukova
- redoslijed stvaranja vrhova i lukova.

Algoritmi koji obavljaju ove zadaće rade s metapodacima baze podataka te njihove performanse ovise o složenosti relacijske baze podataka (primarno broju relacija i stranih ključeva), ali ne i o veličini same baze podataka.

Za njihov jednostavniji opis, a s obzirom da su metapodaci relacijske baze podataka dostupni i putem programskog sučelja, npr. korištenjem SQL (*Structured Query Language*) jezika, pretpostavlja se postojanje ili mogućnost implementacije sljedećih jednostavnih funkcija unutar relacijske baze podataka:

- $pk(r)$, koja daje skup atributa primarnog ključa relacije r,
- $fks(r)$, koja daje skup stranih ključeva relacije r,
- $refd(fk)$, koja pronalazi relaciju referenciranu stranim ključem fk (relaciju primarnog ključa),
- $refing(fk)$, koja pronalazi relaciju koja referencira stranim ključem fk (relaciju stranog ključa).

6.2.1 Određivanje redoslijeda konverzije

Prvi korak pri definiranju procesa konverzije je određivanje budućih vrhova i lukova i redoslijeda konverzije relacija, kako bi se minimiziralo učitavanje podataka u grafovsku bazu podataka. Za ovu svrhu se koristi funkcija *get_migration_order* prikazana u algoritmu 4. Podaci koji su dio relacija koje referenciraju dvije druge relacije a čiji primarni ključ nije referenciran od nijedne druge relacije će biti transformirani u lukove na samom kraju postupka. Ove relacije se bilježe u skupu *ToEdges*.

Nakon toga se promatraju relacije koje ne referenciraju niti jednu drugu relaciju. Njihovi podaci će biti obrađeni prvi, a nakon njih, podaci onih relacija koje referenciraju samo one relacije čiji su podaci već obrađeni. Za svaku n-torku, stvorit će se novi vrh, a za svaki od njezinih stranih ključeva, luk od tog vrha prema vrhu koji već postoji u grafovskoj bazi podataka i koji predstavlja n-torku iz primarne relacije (referencirane stranim ključem). Kako se povećava broj relacija čiji će podaci biti obrađeni, tako će biti moguće obraditi i sve više složenih relacija (onih s većim brojem stranih ključeva). Ovaj se postupak ponavlja dok se ne iscrpi popis svih

Algoritam 4 Određivanje redoslijeda konverzije: **get_migration_order**(**r**)

Input: relacijska baza podataka **r**
Output: skup relacija koje će biti izvor podataka za lukove
Output: skup relacija koje će biti izvor podataka za vrhove

1. $ToEdges \leftarrow \emptyset, ToNodes \leftarrow \emptyset$
2. **for all** $r \in \mathbf{r} \mid |fks(r)| = 2 \wedge r \notin \{refing(fks(s)), s \in \mathbf{r}\}$ **do**
3. $\quad ToEdges \leftarrow ToEdges \cup r$
4. **end for**
5. **repeat**
6. $\quad$ {kandidati su u svakoj iteraciji one relacije koje još uvijek nisu odabrane i koje referenciraju samo one relacije koje već jesu odabrane, uključujući i samu sebe}
7. $\quad candidates \leftarrow \{r \in \mathbf{r} \mid r \notin ToNodes \wedge r \notin ToEdges \wedge refing(fks(r)) \subset ToNodes \cup r\}$
8. $\quad$ **for all** $r \in candidates$ **do**
9. $\quad\quad ToNodes \leftarrow ToNodes \cup r$
10. $\quad$ **end for**
11. **until** $candidates \neq \emptyset$
12. **return** $ToEdges, ToNodes$

relacija iz relacijske baze podataka. Relacije za obradu se bilježe u uređeni skup *ToNodes*, koji je na početku prazan, te se u algoritmu jednostavno moguće referencirati na njega kada se dodaju složenije relacije za konverziju. Skupovi *ToEdges* i *ToNodes* se dalje koriste u postupku popunjavanja grafovske baze podataka i vidljivi su svim algoritmima, pa ih se u tom smislu može promatrati kao globalne varijable programa koji bi objedinio sve algoritme za konverziju relacijskih podataka u graf.

Ukoliko se pretpostavi kako relacijska baza podataka sadrži n relacija, čiji su metapodaci u konačnom (konstantnom) broju metarelacija, moguće je analizirati složenost algoritma na sljedeći način. Sadržaj metarelacija je potrebno pročitati jednom kako bi se dobio popis relacija čiji će podaci biti korišteni za stvaranje lukova te je složenost tog dijela algoritma $O(n)$. U najgorem slučaju, u kojem samo jedna relacija nema stranih ključeva, samo jedna ima jedan strani ključ, samo jedna ima dva strana ključa, i tako dalje do n-te relacije koja ima $n-1$ stranih ključeva, složenost drugog dijela algoritma je $O(n^2)$, što je i ukupna najveća složenost algoritma. Treba, međutim, napomenuti kako relacije u bazama podataka imaju konačan, relativno mali, broj stranih ključeva (u prosjeku manji od 10) te se u realnom slučaju složenost algoritma približava linearnoj, $O(n)$.

6.2.2 Pronalazak cikličkih referenci

Neki sustavi za upravljanje relacijskim bazama podataka dopuštaju kreiranje cikličkih referenci među relacijama, odnosno takvog oblikovanja modela podataka da dvije relacije imaju strane ključeve koji referenciraju primarni ključ one druge. U kontekstu popunjavanja grafovske baze podataka iz relacijske, ovakve cikličke veze impliciraju da neće biti moguće odmah stvoriti sve

lukove koji predstavljaju vrijednosti stranog ključa, jer možda u tom trenutku neće postojati referencirani vrh, s obzirom da ta relacija možda još nije došla na red u tijeku popunjavanja, niti je, s obzirom na cikličku vezu moguće odrediti ispravan redoslijed obrade relacija. Zbog toga je potrebno pronaći cikličke veze, i odrediti jedan strani ključ u njima koji zatvara krug u toj vezi te lukove koji predstavljaju vrijednosti tog ključa kreirati naknadno - nakon što su stvoreni svi vrhovi koji predstavljaju n-torke referenciranih relacija.

Slučaj cikličke veze u relacijskoj bazi podataka može biti znatno složeniji od dvaju međusobno referencirajućih relacija. U općenitom slučaju, relacija r_1 može referencirati relaciju r_2 preko svog stranog ključa fk_1, relacija r_2 može referencirati relaciju r_3 preko svog stranog ključa fk_2, i tako dalje, nastavno do relacije r_{n-1} koja može referencirati relaciju r_n putem svog stranog ključa fk_{n-1} te relacije r_n koja referencira r_1 putem svog stranog ključa fk_n i na taj način zatvara referencirajući krug. Za potrebe ovog algoritma, strani ključ fk_n se promatra kao ključ koji zatvara cikličku referencu.

Najjednostavniji oblik cikličke reference je za vrijednost $n = 1$, a ujedno i najčešći u praksi. Relacije koje same sebe referenciraju se često koriste u relacijskom modelu baza podataka za predstavljanje npr. hijerarhijskih podataka, kao što su organizacijske jedinice, struktura zaposlenika, timova, projekata i slično.

Algoritam 5 Pronalazak cikličkih referenci za referenciranu relaciju: **find_cycle_ref_for_table**($referenced, fk$)

Input: referencirana relacija $referenced$
Input: strani ključ koji se obrađuje fk
1. $observed \leftarrow refing(fk)$
2. **if** $observed = referenced$ **then**
3. $\quad CyclicFKs \leftarrow CyclicFKs \cup observed$
4. **end if**
5. **if** $fk \in VisitedFKs$ **then**
6. $\quad$ **return**
7. **else**
8. $\quad VisitedFKs \leftarrow VisitedFKs \cup observed$
9. **end if**
10. **for all** $fk_i \in fks(observed)$ **do**
11. $\quad find\_cycle\_ref\_for\_table(referenced, fk_i)$
12. **end for**

Ispitivanje je li pojedina relacija ciklički referencirana obavlja se promatrajući njezine strane ključeve i relacije koje oni referenciraju. Toj svrsi služi rekurzivna funkcija *find_cycle_ref_for_table* prikazana algoritmom 5. Ukoliko funkcija pronađe da je relacija referencirana stranim ključem zapravo ona od koje je pretraživanje započelo, onda je to znak da je pronađen ključ koji zatvara cikličku vezu. On se dodaje u skup *CyclicFKs* koji sadrži sve strane ključeve koji zatvaraju cikličke reference u cijeloj bazi podataka, i koristi se u daljnjim postupcima punjenja grafovske baze podataka. Osim toga, funkcija također održava popis stranih ključeva koji su već posjećeni

i pregledani za vrijeme ispitivanja pojedine relacije, kako bi se osigurao izlazak iz rekurzivnih poziva. Funkcija *find_all_cycle_ref*, koja je prikazana u algoritmu 6, se koristi kako bi se pronašli svi strani ključevi koji zatvaraju cikličke reference u cijeloj promatranoj bazi podataka.

Skupovi *CyclicFKs* i *VisitedFKs* se također u ovim algoritmima tretiraju kao globalne varijable objedinjujućeg programa za konverziju podataka.

Algoritam 6 Pronalazak cikličkih referenci u relacijskoj BP: **find_all_cycle_ref**(**r**)

Input: relacijska baza podataka **r**
Output: skup stranih ključeva koji zatvaraju cikličke reference

```
1. CyclicFKs ← ∅
2. for all r ∈ r do
3.     VisitedFKs ← ∅
4.     for all fk ∈ fks(r) do
5.         find_cycle_ref_for_table(r, fk)
6.     end for
7. end for
8. return CyclicFKs
```

U analizi složenosti ovih dvaju algoritama najgori slučaj bi bio da svaka od n relacija ima strane ključeve na sve ostale relacije, odnosno $n-1$ stranih ključeva (pri čemu je moguća i situacija višestrukih ključeva na istu relaciju, no za ovu analizu to može biti konstantan broj). Time bi rekurzivna funkcija $find\_cycle\_ref\_for\_table$ imala složenost $O(n^n)$, što je onda i ukupna složenost funkcije $find\_all\_cycle\_ref$. Uzevši u obzir konstantni, prosječno mali broj stranih ključeva po relaciji, u realnom slučaju se složenost ovog algoritma približava $O(n)$.

6.3 Popunjavanje grafovske baze podataka

Nakon što je obavljena priprema analizom metapodataka relacijske baze podataka, ispunjeni su uvjeti za postupak stvaranja grafovske baze podataka iz relacijske baze podataka. Pretpostavlja se da je grafovska baza podataka $\mathcal{G}$ već kreirana u sustavu za upravljanje grafovskim bazama podataka te se tijekom ovog postupka podaci čitaju iz relacijske baze podataka i stvaraju se novi vrhovi i lukovi unutar baze podataka $\mathcal{G}$.

Za jednostavniji prikaz postupka popunjavanja grafovske baze podataka, pretpostavlja se da su unutar sustava za upravljanje grafovskim bazama podataka implementirane sljedeće jednostavne funkcije:

- $\mathcal{G}.add\_node(:label, id\_value)$, koja stvara vrh s oznakom *label* i svojstvom *id* koje ima vrijednost *id_value*,
- $\mathcal{G}.add\_edge(v_1, v_2, : label, [id\_value])$, koja stvara luk s oznakom *label* i opcionalnim svojstvom *id* vrijednosti *id_value* od vrha v_1 prema vrhu v_2,

- $\mathcal{G}.get\_node(:label, id\_value)$, koja pronalazi vrh s oznakom *label* i svojstvom *id* koje ima vrijednost *id_value*,
- $x.add\_property(K, V)$, koja dodaje svojstvo K s vrijednosti V elementu grafa x.

Funkcija *table_to_graph*, opisana algoritmom 7, obavlja učitavanje podataka iz jedne relacije u grafovsku bazu podataka. Ako je ranije utvrđeno da će ta relacija biti konvertirana u vrhove (njezino je ime u *ToNodes* skupu), stvara se novi vrh za svaku n-torku relacije. Svaki novi vrh dobiva oznaku jednaku imenu relacije i svojstvo *id* s vrijednosti jednakoj spojenim vrijednostima atributa primarnog ključa. Kao što je ranije rečeno, atributi koji nisu dio primarnog niti stranih ključeva, a imaju zapisanu vrijednost (nije nepostojeća vrijednost, *null*), postaju svojstva vrha s imenom jednakim imenu atributa i vrijednosti jednakoj vrijednosti atributa u n-torki.

Algoritam 7 Konverzija podataka relacije: **table_to_graph**$(r, \mathcal{G})$

Input: relacija r sa shemom R
Input: grafovska baza podataka $\mathcal{G}$
Output: elementi grafa koji predstavljaju n-torke dodani u grafovsku bazu podataka

1. **for all** $t \in r$ **do**
2. $id \leftarrow t[pk(r)]$
3. **if** $r \in ToNodes$ **then**
4. $v \leftarrow \mathcal{G}.add\_node(:r, id)$
5. **for all** $A_i \in R \mid A_i \notin pk(r) \wedge A_i \notin fks(r)$ **do**
6. $v.add\_property(A_i, t[A_i])$
7. **end for**
8. **for all** $fk \in fks(r) \mid fk \notin CyclicFKs$ **do**
9. $r_{refd} \leftarrow refd(fk)$
10. $id_{refd} \leftarrow t[fk]$
11. $v_{refd} \leftarrow \mathcal{G}.get\_node(:r_{refd}, id_{refd})$
12. $\mathcal{G}.add\_edge(v, v_{refd}, :fk)$
13. **end for**
14. **end if**
15. **if** $r \in ToEdges$ **then**
16. $fk_1, fk_2 \leftarrow fks(r)$
17. $r_{refd1} \leftarrow refd(fk_1)$
18. $r_{refd2} \leftarrow refd(fk_2)$
19. $id_{refd1} \leftarrow t[fk_1]$
20. $id_{refd2} \leftarrow t[fk_2]$
21. $v_{refd1} \leftarrow \mathcal{G}.get\_node(:r_{refd1}, id_{refd1})$
22. $v_{refd2} \leftarrow \mathcal{G}.get\_node(:r_{refd2}, id_{refd2})$
23. $e \leftarrow \mathcal{G}.add\_edge(v_{refd1}, v_{refd2}, :r, id)$
24. **for all** $A_i \in R \mid A_i \notin pk(r) \wedge A_i \notin fks(r)$ **do**
25. $e.add\_property(A_i, t[A_i])$
26. **end for**
27. **end if**
28. **end for**

Za vrijeme obrade n-torke svi strani ključevi koji ne zatvaraju cikličke reference (ne pojavljuju se u skupu *CyclicFKs*) su također odmah obrađeni. Za svakog od njih, u grafovskoj bazi podataka se pronalazi ranije kreirani vrh s oznakom jednakom imenu referencirajuće relacije i s vrijednosti svojstva *id* jednakoj spojenoj vrijednosti atributa stranog ključa. Stvara se novi luk od novog vrha prema tom vrhu te mu se daje oznaka jednaka imenu stranoga ključa.

Ako je ranije utvrđeno da će promatrana relacija biti konvertirana u lukove (njezino je ime u *ToEdges* skupu), novi luk se stvara za svaku n-torku relacije. Vrhovi koje će taj luk spajati se pronalaze na već opisani način, korištenjem oznaka koje odgovaraju imenima referenciranih relacija i jedinstvenih vrijednosti svojstva *id* koje odgovaraju vrijednostima atributa stranih ključeva u n-torki. Novi luk dobiva oznaku jednaku imenu relacije, svojstvo *id* s vrijednosti atributa primarnog ključa i usmjeren je od vrha koji je prvi referenciran stranim ključem u promatranoj relaciji prema vrhu koji je drugi referenciran stranim ključem u promatranoj relaciji. Na kraju, luku se dodaju svojstva, za sve zavisne atribute čija vrijednost postoji (nije nepostojeća vrijednost *null*), na isti način kao i za vrhove.

Algoritam 8 Popunjavanje grafovske BP iz relacijske BP: **populate_gdb**(**r**)

Input: relacijska baza podataka **r**
Output: grafovska baza podataka $\mathcal{G}$

1. $CyclicFKs \leftarrow find\_all\_cycle\_ref(\mathbf{r})$
2. $ToEdges, ToNodes \leftarrow get\_migration\_order(\mathbf{r})$
3. $\mathcal{G} \leftarrow \emptyset$
4. **for all** $r \in ToNodes$ **do**
5. $\quad table\_to\_graph(r, \mathcal{G})$
6. **end for**
7. **for all** $r \in ToEdges$ **do**
8. $\quad table\_to\_graph(r, \mathcal{G})$
9. **end for**
10. **for all** $fk \in CyclicFKs$ **do**
11. $\quad r_{refing} \leftarrow refing(fk)$
12. $\quad r_{refd} \leftarrow refd(fk)$
13. $\quad$ **for all** $t \in r_{refing}$ **do**
14. $\quad\quad id_{refing} \leftarrow t[pk(r)]$
15. $\quad\quad id_{refd} \leftarrow t[fk]$
16. $\quad\quad v_{refing} \leftarrow \mathcal{G}.get\_node(:r_{refing}, id_{refing})$
17. $\quad\quad v_{refd} \leftarrow \mathcal{G}.get\_node(:r_{refd}, id_{refd})$
18. $\quad\quad \mathcal{G}.add\_edge(v_{refing}, v_{refd}, :fk)$
19. $\quad$ **end for**
20. **end for**

Funkcija *populate_gdb* prikazana u algoritmu 8, opisuje glavni proces popunjavanja grafovske baze podataka. Nakon određivanja stranih ključeva koji zatvaraju možebitne cikličke veze i određivanja redoslijeda obrade relacija, one se obrađuju i za njihove vrijednosti i strane ključeve se stvaraju vrhovi i lukovi.

Posljednji dio popunjavanja grafovske baze podataka je kreiranje lukova koji proizlaze iz stranih ključeva koji zatvaraju cikličke veze. Ovo je jedini dio procesa u kojem je potrebno ponovno čitati dio podataka iz relacijske baze podataka, da bi se ponovno prošlo kroz one relacije koje te ključeve sadrže i zaključilo koje vrhove novi lukovi, koji će zatvarati cikličku vezu, trebaju povezivati. Ovi se usmjereni lukovi stvaraju na identičan način kao i drugi lukovi koji predstavljaju strane ključeve. Novi luk dobiva oznaku jednaku imenu stranoga ključa i nema dodatnih svojstava.

Složenost algoritma u funkciji *table_to_graph* treba promatrati kroz količinu n-torki u relaciji koja se obrađuje te broj atributa i stranih ključeva te relacije. Općenito se može reći da je broj n-torki, n, znatno veći od broja atributa i stranih ključeva relacije koji se mogu promatrati kao konstante pa je složenost ovog algoritma $O(n)$. Proširujući ovaj kontekst i na funkciju *populate_gdb*, gdje je ključna iteracija po svim relacijama baze podataka, može se, prihvaćajući n kao ukupan broj n-torki u svim relacijama, također reći da je složenost tog algoritma linearna.

6.4 Primjer popunjavanja grafovske baze podataka iz relacijske baze podataka

Ilustracija načina rada prikazanih algoritama bit će obavljena na jednostavnom primjeru relacijske baze podataka koja sadrži podatke o studentima, predmetima, nastavnicima, ispitima te organizacijskoj strukturi visokoškolske ustanove.

Primjer 3. Neka u relacijskoj bazi podataka **studentska_sluzba** postoji sljedećih šest relacija:

- *student*, koja sadrži podatke o studentima,
- *nastavnik*, koja sadrži podatke o nastavnicima, uključujući i organizacijsku jedinicu u kojoj su zaposleni,
- *predmet*, koja sadrži podatke o predmetima,
- *nastavnik_predaje*, koja sadrži informacije o tome koji nastavnik od koje godine do koje godine predaje koji predmet,
- *ispit*, koja sadrži podatke o ispitima studenta na predmetu kod nastavnika i postignutoj ocjeni,
- *org_jed*, koja sadrži podatke o organizacijskim jedinicama ustanove.

Shema relacijske baze podataka je sljedeća:

STUDENT = student_id, student_ime, student_prezime
$K_{STUDENT}$ = student_id

ORG_JED = org_jed_id, org_jed_naziv, nadredjeni_org_jed_id

K_{ORG_JED} = org_jed_id

$K_{ORG\_JED \rightarrow ORG\_JED}$:= fk_nadorgjed = nadredjeni_org_jed_id

NASTAVNIK = nastavnik_id, nastavnik_ime, nastavnik_prezime, org_jed_id

$K_{NASTAVNIK}$ = nastavnik_id

$K_{NASTAVNIK \rightarrow ORG\_JED}$:= fk_nastavnik_orgjed = org_jed_id

PREDMET = predmet_id, predmet_naziv

$K_{PREDMET}$ = predmet_id

NASTAVNIK_PREDAJE = nastavnik_id, predmet_id, predaje_od, predaje_do

$K_{NASTAVNIK\_PREDAJE}$ = nastavnik_id, predmet_id

$K_{NASTAVNIK\_PREDAJE \rightarrow NASTAVNIK}$:= fk_nastpred_nastavnik = nastavnik_id

$K_{NASTAVNIK\_PREDAJE \rightarrow PREDMET}$:= fk_nastpred_predmet = predmet_id

ISPIT = student_id, predmet_id, nastavnik_id, ocjena

K_{ISPIT} = student_id, predmet_id, nastavnik_id

$K_{ISPIT \rightarrow STUDENT}$:= fk_ispit_student = student_id

$K_{ISPIT \rightarrow PREDMET}$:= fk_ispit_predmet = predmet_id

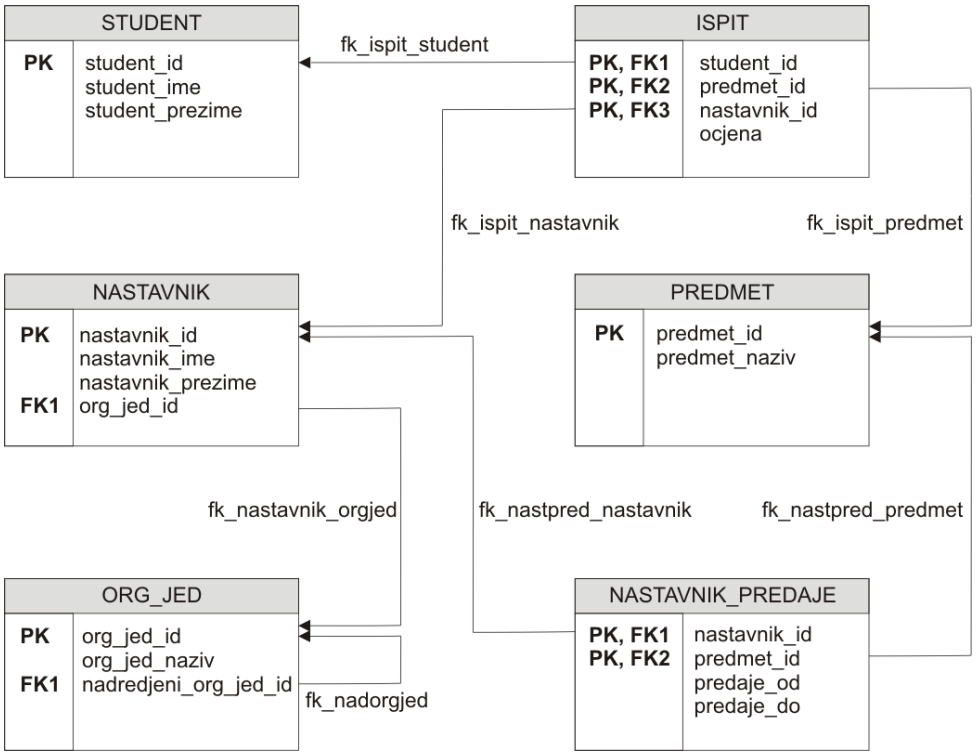
$K_{ISPIT \rightarrow NASTAVNIK}$:= fk_ispit_nastavnik = nastavnik_id

Radi bolje preglednosti, ova je shema prikazana i relacijskim dijagramom na slici 6.1.

U pripremnoj fazi postupka punjenja grafovske baze, najprije se izvršava funkcija *get_migration_order*, koja će popuniti skupove *ToEdges* i *ToNodes*. Funkcija će najprije pronaći relacije koje imaju dva strana ključa a same nisu referencirane, a to je u ovom slučaju samo relacija *nastavnik_predaje*. Nakon toga će biti pregledane relacije koje ne referenciraju druge relacije, što znači da se u skup *ToNodes* dodaju redom relacije *student*, *org_jed* i *predmet*. U sljedećoj iteraciji se može u ovaj skup dodati i relacija *nastavnik*, a u posljednjoj i relacija *ispit*.

Funkcija *find_all_cycle_ref* će obraditi sve relacije u potrazi za cikličkim referencama. U ovom primjeru postoji samo jedna takva veza, ona najjednostavnija u kojoj relacija *org_jed* referencira samu sebe, jer se na taj način omogućava opis hijerarhije organizacijskih jedinica unutar učilišta - npr. katedre i grupe mogu pripadati zavodima, a zavodi učilištu. Dakle, nakon izvođenja funkcija *get_migration_order* i *find_all_cycle_ref*, ključni skupovi za popunjavanje grafovske baze imaju sljedeći sadržaj:

$ToEdges = \{nastavnik\_predaje\}$,



Slika 6.1: Relacijski dijagram baze podataka u primjeru

$ToNodes = \{student, org\_jed, predmet, nastavnik, ispit\}$,
$CyclicFKs = \{fk\_nadorgjed\}$.

Sam postupak popunjavanja grafovske baze podataka obavlja funkcija *populate_gdb*, odnosno funkcija *table_to_graph* koju ona koristi. Temeljem do sada navedenog zaključuje se da će, na primjer, n-torke relacije *student* stvoriti vrhove s istoimenom oznakom, vrijednostima svojstva *id* jednakim vrijednostima atributa *student_id* te dvama svojstvima *student_ime* i *student_prezime* koja će sadržavati vrijednosti istoimenih atributa. N-torke relacije *ispit* će stvoriti vrhove čije će svojstvo *id* imati spojenu vrijednost atributa *student_id*, *predmet_id* i *nastavnik_id*. Osim njega, ovi će vrhovi imati još i svojstvo *ocjena* koje će imati vrijednost istoimenoga atributa te lukove prema referenciranim vrhovima s oznakama *student*, *predmet* i *nastavnik*.

U tablici 6.1 je prikazan sadržaj navedenih relacija temeljem kojeg se stvara sadržaj grafovske baze podataka.

Tablica 6.1: Sadržaj relacija u primjeru 3

org_jed

org_jed_id	org_jed_naziv	nadredjeni_org_jed_id
1000	Visoka škola	
1001	Zavod za prirodne znanosti	1000
1002	Katedra za matematiku	1001
1003	Katedra za fiziku	1001

nastavnik

nastavnik_id	nastavnik_ime	nastavnik_prezime	org_jed_id
II	Ivan	Ivanović	1002
JJ	Josip	Josipović	1002
KK	Karlo	Karlović	1003

predmet

predmet_id	predmet_naziv
1	Algebra
2	Fizika 1

nastavnik_predaje

nastavnik_id	predmet_id	predaje_od	predaje_do
II	1	2000	2010
JJ	1	2010	
KK	2	2005	

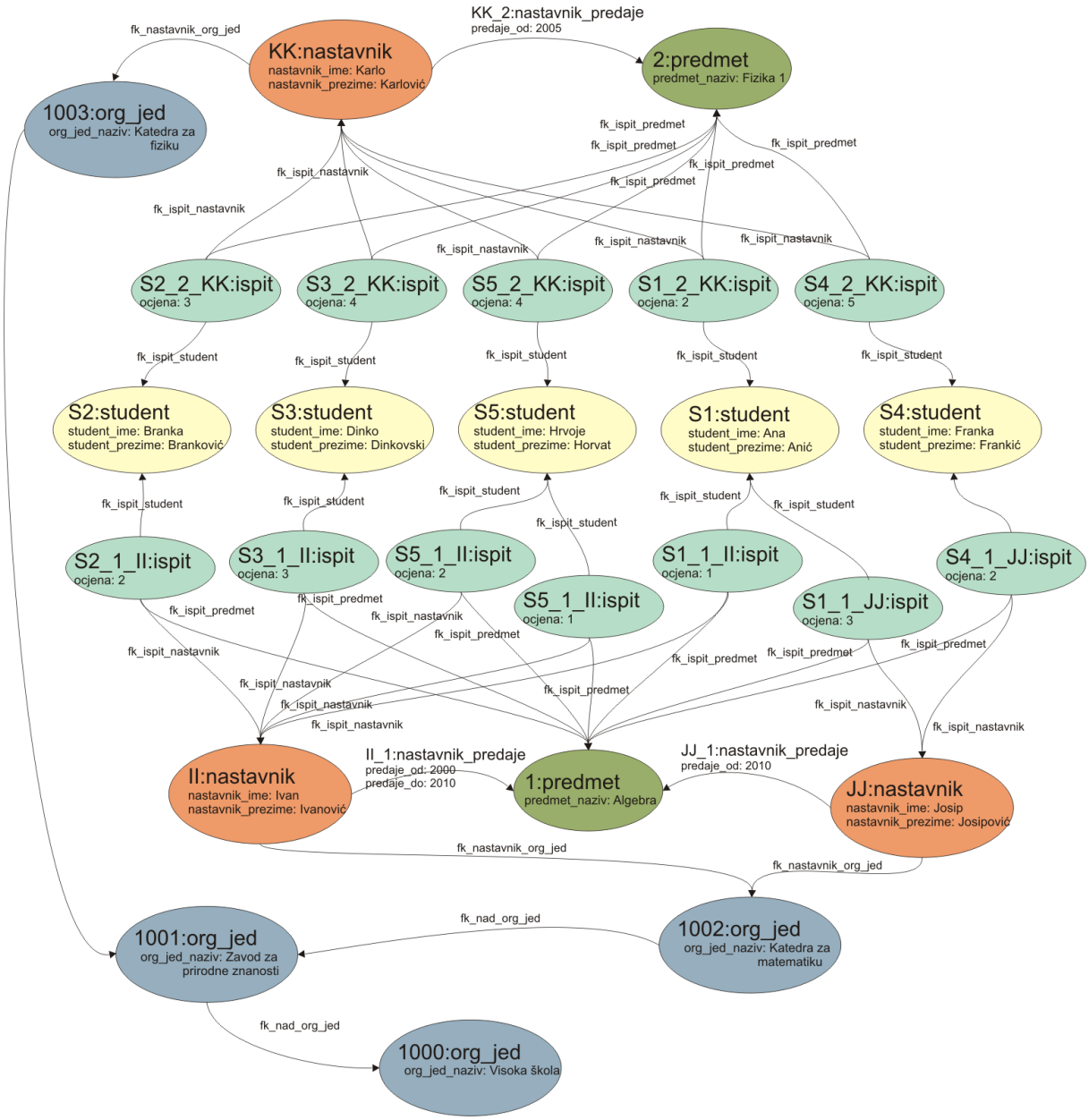
student

student_id	student_ime	student_prezime
S1	Ana	Anić
S2	Branka	Branković
S3	Dinko	Dinkovski
S4	Franka	Frankić
S5	Hrvoje	Horvat

ispit

student_id	predmet_id	nastavnik_id	ocjena
S1	1	II	1
S1	1	JJ	3
S2	1	II	2
S3	1	II	3
S4	1	JJ	2
S5	1	II	1
S5	1	II	2
S1	2	KK	2
S2	2	KK	3
S3	2	KK	4
S4	2	KK	5
S5	2	KK	4

Na slici 6.2 je prikazan stvoreni sadržaj grafovske baze podataka.



Slika 6.2: Popunjeni podaci iz primjera 3 u grafovskoj bazi podataka

Identifikator vrha (i luka, ako postoji), odnosno vrijednost svojstva *id* je napisana većim slovima, a iza dvotočke ga slijedi oznaka vrha, odnosno luka. Manjim slovima su ispod toga napisana svojstva elementa, u formatu "naziv svojstva: vrijednost svojstva". Zbog lakšeg snalaženja, vrhovi s istim oznakama su prikazani istom bojom. ■

Poglavlje 7

Potpuno vremenski određeni graf

Kao što je pokazano u poglavlju 3, postoje različiti koncepti pohrane vremenskih informacija unutar samog grafa. Dinamički grafovi, na primjer, u nekim primjenama prikazuju samo aktualno stanje grafa, bez vremenske dimenzije kroz koju je moguće promatrati i analizirati promjene grafa u vremenu. Ukoliko je potrebno imati vremensku informaciju o postojanju, odnosno aktivnosti, pojedinih elemenata grafa unutar samog grafa, tada su prikladniji prikazi grafa pomoću vremenskih nizova ili tenzora susjedstva, no oba prikaza imaju ograničenje koje se odnosi na vremensku dimenziju, a koja je u ovom slučaju prikazana diskretnim vremenskim razmacima. Ukoliko se u grafu želi prikazati precizno vrijeme, u kojem svaki element grafa može nastati ili biti obrisan u bilo kojem trenutku, i uz to imati jasno određene vremenske udaljenosti među događajima, ove vrste prikaza nisu praktične, jer bi njihovo korištenje impliciralo pohranu stanja grafa u previše vremenskih trenutaka, odnosno diskretni vremenski razmak među pohranjenim grafovima u vremenskom nizu ili tenzoru bi trebao biti premali, a prostor za pohranu takvog grafa postao prevelik.

Također, u nekim radovima se proučavaju vremenski grafovi u kojima su vrhovi konstantni u vremenu, a samo su bridovi ili lukovi vremenski ovisni, odnosno pojavljuju se i nestaju u vremenskim periodima. Dok ovaj pristup nudi pohranu vremenske dimenzije kao kontinuirane varijable, za primjenu analize snimljenog traga je također previše ograničavajući, s obzirom da i vrhovi također mogu nastajati i nestajati u vremenu, odnosno postati aktivni i prestati to biti.

Slijedom navedenoga, a iz potrebe prikaza vremenskog grafa koji će sadržavati vremenske odrednice svih elemenata grafa, uvodi se pojam potpuno vremenski određenog grafa.

Definicija 26. Potpuno vremenski određeni graf G_T je usmjereni graf u kojem svaki vrh i luk može imati oznaku (tip vrha ili luka), svojstva (skup podataka u formatu ključ-vrijednost) i period aktivnosti (svoj početak i kraj u vremenu). Takav graf se može opisati kao uređena četvorka:

$$G_T = (V_T, E_T, \varphi, \mathcal{T}) \tag{7.1}$$

u kojoj su V_T vremenski određeni vrhovi grafa, E_T vremenski određeni lukovi grafa, $\varphi : E \rightarrow$

$V \times V$ funkcija incidencije između lukova i vrhova grafa, a $\mathcal{T}$ je interval životnog vijeka grafa, vremenski odsječak unutar domene vremena $\mathbb{T}$.

Vremenski određeni vrhovi grafa se opisuju s:

$$V_T = (V, \mathcal{T}_V, \psi, L_V, \lambda_V, P_V, \pi_V) \tag{7.2}$$

pri čemu je:

- V skup vrhova grafa,
- $\mathcal{T}_V$ ukupan interval životnog vijeka vrhova,
- $\psi : V \times \mathcal{T}_V \to \{0, 1\}$ funkcija postojanja (aktivnosti) vrha u vremenu,
- L_V skup oznaka vrhova,
- $\lambda_V : V \to L_V$ funkcija označavanja vrhova,
- $P_V = \{(p_i, v_i)\}$ skup svojstava vrhova p_i s vrijednostima v_i,
- $\pi : V \to 2^{P_V}$ funkcija pridjeljivanja svojstava vrhu.

Vremenski određeni lukovi se opisuju s:

$$E_T = (E, \mathcal{T}_E, \rho, L_E, \lambda_E, P_E, \pi_E) \tag{7.3}$$

pri čemu je:

- E skup lukova grafa,
- $\mathcal{T}_E$ ukupan interval životnog vijeka lukova,
- $\rho : E \times \mathcal{T}_E \to \{0, 1\}$ funkcija postojanja (aktivnosti) luka u vremenu,
- L_E skup oznaka lukova,
- $\lambda_E : E \to L_E$ funkcija označavanja lukova,
- $P_E = \{(p_i, v_i)\}$ skup svojstava lukova p_i s vrijednostima v_i,
- $\pi : E \to 2^{P_E}$ funkcija pridjeljivanja svojstava luku.

Ukupan životni vijek grafa je zajednički interval postojanja vrhova i lukova:

$$\mathcal{T} = \mathcal{T}_V \cup \mathcal{T}_E,\ \mathcal{T} \subseteq \mathbb{T} \tag{7.4}$$

U slučaju primjene potpuno vremenski određenog grafa za prikaz vremena kao kontinuirane varijable, može se uzeti $\mathbb{T} = \mathbb{R}^+$, a u slučaju prikaza vremena kao niza diskretnih trenutaka, može se uzeti $\mathbb{T} = \mathbb{N}$, pri čemu se u oba slučaja određuje nula kao početni vremenski trenutak. ■

Promatraju li se, u smislu postojanja oznaka, vremenske aktivnosti i svojstava, vremenski određeni vrhovi i lukovi generalizirano, kao elementi grafa koji imaju zajedničke kvalitete, onda

je moguće jednakosti iz definicije 26 objediniti te potpuno vremenski određeni graf jednostavnije opisati s

$$G_T = (V, E, \varphi, \mathcal{T}, \psi, L, \lambda, P, \pi) \tag{7.5}$$

gdje je:

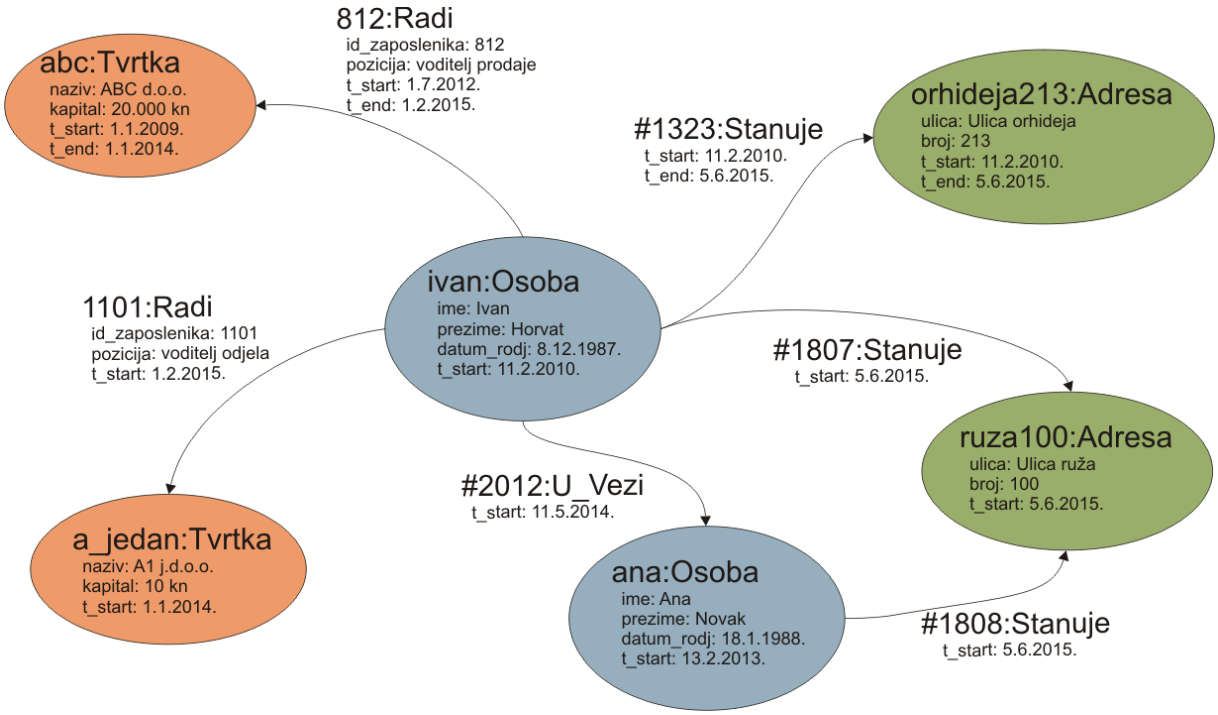
- V skup vrhova,
- E skup lukova,
- $\varphi : E \to V \times V$ funkcija incidencije koja povezuje lukove i vrhove grafa,
- $\mathcal{T}$ ukupan interval životnog vijeka vrhova i lukova grafa, $\mathcal{T} \subseteq \mathbb{T}$,
- $\psi : (V \cup E) \times \mathcal{T} \to \{0, 1\}$ funkcija postojanja (aktivnosti) elementa grafa u vremenu,
- L skup oznaka elemenata grafa,
- $\lambda : V \cup E \to L$ funkcija označavanja elemenata grafa,
- $P = \{(p_i, v_i)\}$ skup svojstava p_i s pripadnim vrijednostima v_i,
- $\pi : V \cup E \to 2^P$ funkcija pridjeljivanja svojstava elementu grafa.

Dakle, svaki element grafa ima pridijeljen najmanje jedan, a najviše dva događaja koji se odnose na njegovu aktivnost u vremenu. Svaki element mora imati početak aktivnosti, a oni elementi koji nisu aktivni u aktualnom trenutku imaju pridijeljen i završetak aktivnosti.

Potpuno vremenski određeni graf, slično kao i statički graf, prikazuje stanje koje je akumulirano u vremenu, ali uz bitnu razliku, a to je da, ovisno o primjeni, može ujedno prikazivati i cijelu povijest trenutnoga stanja, što je posebno zanimljivo za vremensku analizu snimljenog traga.

Primjer 4. Na slici 7.1 je primjer potpuno vremenski određenog grafa. Zbog jednostavnijeg pregleda, vrhovi istih oznaka su označeni istim bojama (osobe plavom, adrese zelenom a tvrtke crvenom bojom). Identifikatori vrhova i lukova su vrijednosti ključnih svojstava grafa (npr. ime osobe je identifikator vrha koji predstavlja osobu, naziv tvrtke je identifikator vrha koji predstavlja tvrtku a vrijednost svojstva *id_zaposlenika* je identifikator luka s oznakom "Radi") ili su generičke vrijednosti (npr. lukovi s oznakom "Stanuje" imaju generičke identifikatore).

Graf sadrži podatke o dvije osobe koje su u nekoj vrsti veze, njihovoj trenutnoj adresi, adresi na kojoj je jedna od njih živjela ranije i zaposlenjima te osobe u dvjema tvrtkama. Vrhovi i lukovi sadrže informaciju o početku aktivnosti (prikazano kao svojstvo *t_start*), a neki od njih i o završetku aktivnosti (prikazano kao svojstvo *t_end*). Vremenski trenutci su u ovom primjeru diskretizirani na razinu dana (datuma). ■



Slika 7.1: Potpuno vremenski određeni graf

7.1 Potpuno vremenski određeni graf u grafovskoj bazi podataka

Formalno opisani potpuno vremenski određeni graf je moguće implementirati u nekom od sustava za upravljanje grafovskim bazama podataka koji podržava model grafa sa svojstvima, odnosno, moguće ga je pohraniti kao grafovsku bazu podataka u takvom sustavu.

Sama priroda grafovske baze podataka podrazumijeva brigu o vrhovima i lukovima među njima, odnosno funkciji incidencije φ. Vrhovi i lukovi u takvoj bazi podataka mogu imati oznake, te su skup L i pripadna funkcija označavanja λ interno implementirani u samom sustavu. Jednaka činjenica vrijedi i za svojstva koja se mogu pridružiti vrhovima i lukovima te su tako i skup P i pripadna funkcija π interno implementirani u sustavu.

Vremenske oznake početka i eventualnog završetka aktivnosti pojedinog vrha ili luka je moguće također evidentirati kao svojstva čije su vrijednosti konkretne vremenske oznake, bilo da su normirane na diskretne vremenske odnose ili zapisane kao potpuni vremenski zapisi. U tom slučaju, ukupan interval životnog vijeka grafa $\mathcal{T}$ traje od najranijeg vremenskog zapisa početka aktivnosti vrha do najkasnijeg vremenskog zapisa početka ili završetka aktivnosti nekog od elemenata grafa, a funkcija aktivnosti elemenata grafa u vremenu ψ je zapravo funkcija pridjeljivanja svojstava elementima grafa π.

Poglavlje 8

Snimljeni trag relacijskih baza podataka i potpuno vremenski određeni graf

8.1 Snimanje traga relacijskih baza podataka

Pod snimanjem traga relacijskih baza podataka (*auditing*) podrazumijeva se bilježenje promjena nad podacima unutar relacija koje izvode korisnici tijekom redovitog rada sa sustavom. Opis mogućeg snimanja traga metodom prilagodbe okidača detaljno je opisan u magistarskom radu [8], a u poglavljima 8.1.1 i 8.1.2 slijedi sažeti prikaz ove metode.

8.1.1 Koncept snimanja traga unutar relacijske baze podataka

Za razliku od brojnih vanjskih sustava za snimanje traga, osnovna ideja ove metode jest snimanja traga koji ostavljaju DML (*Data Modifying Language*) operacije - unosa, izmjene i brisanja podataka - tamo gdje se operacije i događaju, unutar same baze podataka. Treba napomenuti da je snimanje traga DML operacija unutar baze podataka mogućnost koju nude mnogi komercijalni sustavi za upravljanje bazama podataka, ali i da su u provedenom istraživanju tih mogućnosti [8] otkrivene njihove razne manjkavosti, kako u području prilagodbe i jasnog definiranja što će se sve snimati, tako i u sigurnosnim aspektima definiranja snimanja traga.

Snimanje traga metodom prilagodbe okidača se obavlja na razini pojedine relacije te je za svaku relaciju potrebno osigurati prostor za pohranu njezina traga. Stoga se za svaku relaciju r sa shemom R čiji se trag snima stvara posebna relacija za njezin trag, $\_r$ sa shemom $\_R$, koja sadrži isti skup atributa kao i sama relacija r, ali i dodatne atribute. Radi jednostavnijeg razlikovanja, relacija r se u ovom kontekstu naziva radnom relacijom, a relacija $\_r$ koja sadrži povijest relacije r se naziva povijesnom relacijom. Svaki zapis, n-torka, odnosno redak, u povijesnoj relaciji odgovara jednoj izvršenoj operaciji nad jednim zapisom u radnoj relaciji. Stoga povijesna relacija uz originalne atribute sadrži i dodatne atribute koji opisuju operaciju:

- $OpUser$ – korisnik koji je obavio operaciju,
- $OpTimestamp$ – vremenska oznaka operacije,
- $TxTimestamp$ – vremenska oznaka transakcije unutar koje je operacija obavljena,
- $Operation$ – oznaka operacije: I - unos (*insert*), U - izmjena (*update*) ili D - brisanje (*delete*),
- $SessionID$ – oznaka korisničke sjednice u kojoj je korisnik obavio operaciju.

Slijedom akcije unosa podataka u radnu relaciju, u atribute povijesne relacije koji odgovaraju radnoj se zapisuju vrijednosti atributa kakve su unesene u radnoj relaciji. Prilikom ažuriranja (izmjene) podataka, također se u te atribute povijesne relacije upisuju nove vrijednosti atributa n-torke radne relacije, one na koje su postavljene, dok se prilikom brisanja evidentiraju vrijednosti atributa cijele obrisane n-torke iz radne relacije. Za sve tri akcije, vrijednosti dodatnih atributa se popunjavanju automatizmom iz stanja sustava za upravljanje bazama podataka i njegovih internih varijabli.

8.1.2 Prilagodba okidača i druge sigurnosne i organizacijske akcije

Kako bi se svaka operacija nad podacima zabilježila u povijesne relacije, potrebno je stvoriti okidače (*triggers*) nad radnim relacijama koji će po svakoj obavljenoj operaciji izvršiti upis podataka u pripadnu povijesnu relaciju. Za svaku relaciju čiji se trag snima, potrebno je implementirati okidače nad akcijama unosa, izmjene i brisanja zapisa. Pri tom treba imati na umu da se okidači u bazi podataka koriste i za osiguranje poslovnih pravila sustava te se njihovo djelovanje u procesu stvaranja ovih okidača ne bi smjelo narušiti.

S obzirom da su okidači ključni za zapis snimljenog traga, potrebno je poduzeti mjere zaštite relacija čiji se trag snima, ali i cijele baze podataka od neovlaštenog pristupa i manipulacije okidačima. Uz to je potrebno omogućiti i sistemsko snimanje traga kritičnih akcija nad bazom podataka, ukoliko sustav za upravljanje bazama podataka posjeduje takvu mogućnost. Osim osnovnih najvažnijih akcija nad cijelim sustavom za upravljanje bazama podataka, kritičnim akcijama se u ovom kontekstu smatraju izmjene samih objekata u bazi podataka (kao što su relacije i okidači), dozvola za takve akcije, kao i izmjene u rječniku podataka.

Također, kako je pokazano u [8], moguće je automatizirati procese uključivanja snimanja traga za pojedine relacije, stvaranje povijesnih relacija, stvaranje posebne povijesne baze podataka, procese eventualnog brisanja zastarjele povijesti itd. Nad ovako snimljenim tragom moguće je obavljati jednostavne forenzičke analize direktnim upitima u bazu podataka, a ta platforma omogućava i izvođenje složenih analiza poput pronalaska mogućih zlouporaba informacijskih sustava, što je predmet istraživanja ove disertacije.

8.1.3 Primjer snimanja traga unutar relacijske baze podataka

Snimanje traga i rezultat operacija nad relacijskim podacima će biti ukratko prikazan na primjeru koji se nastavlja na primjer 3 (str. 52), jednostavnoj relacijskoj bazi podataka koja sadrži podatke o studentima, predmetima, nastavnicima, ispitima te organizacijskoj strukturi visokoškolske ustanove.

Primjer 5. Ukoliko se, na ranije spomenutom primjeru relacijske baze podataka **studentska_sluzba** primijene opisane metode stvaranja objekata namijenjenih snimanju traga, to će rezultirati stvaranjem povijesne baze podataka **_studentska_sluzba** i u njoj pripadnih povijesnih relacija koje sadrže trag podataka radnih relacija: *_student*, *_nastavnik*, *_predmet*, *_nastavnik_predaje*, *_ispit* i *_org_jed*. Za ilustraciju, shema relacije *_student* će biti:

_STUDENT = student_id, student_ime, student_prezime, OpUser, OpTimestamp, TxTimestamp, Operation, SessionID

Također će biti modificirani okidači nad radnim relacijama, kako bi osigurali punjenje relacija tragom obavljenih operacija, te kreirani dodatni sigurnosni i korisnički objekti. Rezultat svega navedenog je bilježenje traga operacija koje korisnici izvode.

U tablici 8.1 je primjer snimljenog traga koji je rezultat dijela rada korisnika nad informacijskim sustavom učilišta. Zbog jednostavnosti, uz osnovne atribute radnih relacija, povijesne relacije su prikazane samo s tri dodatna atributa (OpUser, OpTimestamp i Operation).

Iz podataka u tablici se mogu dobiti sljedeće informacije:

- djelatnik s korisničkim imenom "josip" evidentirao je dio organizacijske strukture učilišta i unio tri zapisa o organizacijskim jedinicama u relaciju *org_jed* (sa šiframa 1000, 1001 i 1002)
- djelatnik s korisničkim imenom "josip" evidentirao je nastavnike koji su zaposleni na nekoj od tih organizacijskih jedinica u relaciju *nastavnik* (nastavnici sa šiframa "II" i "MM")
- naknadno je isti djelatnik obrisao nastavnika sa šifrom "MM"
- predmet sa šifrom 1 se u početku zvao "Matematika 1", zatim "Linearna algebra" a nekada nakon toga u "Algebra"
- dvoje djelatnika je obavljalo unos i dvije promjene tog predmeta
- djelatnica s korisničkom imenom "ivana" je evidentirala da nastavnik sa šifrom "II" predaje predmet sa šifrom 1 od 2000-te godine.

U prikazanim relacijama se kao oznaka vremena koristi jednostavna vremenska odrednica, broj sekundi od nekog fiksnog trenutka u vremenu, umjesto pune oznake datuma i vremena. ■

Tablica 8.1: Sadržaj relacija sa snimljenim tragom u primjeru

_org_jed

org_jed_id	org_jed_naziv	nadredjeni_org_jed_id	OpUser	OpTimestamp	Operation
1000	Visoka škola		josip	123000	I
1001	Zavod za prirodne znanosti	1000	josip	123100	I
1002	Katedra za matematiku	1001	josip	123200	I

_nastavnik

nastavnik_id	nastavnik_ime	nastavnik_prezime	org_jed_id	OpUser	OpTimestamp	Operation
MM	Mirko	Mirković	1002	josip	124000	I
II	Ivan	Ivanović	1002	josip	125800	I
MM	Mirko	Mirković	1002	josip	130000	D

_predmet

predmet_id	predmet_naziv	OpUser	OpTimestamp	Operation
1	Matematika 1	petra	111000	I
1	Linearna algebra	lucija	122000	U
1	Algebra	petra	126800	U

_nastavnik_predaje

nastavnik_id	predmet_id	predaje_od	predaje_do	OpUser	OpTimestamp	Operation
II	1	2000		ivana	127300	I

8.2 Stvaranje potpuno vremenski određenog grafa temeljem snimljenog traga

Kako je već spomenuto, snimljeni trag pojedine relacije sadrži potpuni zapis svake operacije koja je nad n-torkom relacije provedena, uključujući podatke o samoj akciji i n-torku nad kojom je operacija izvršena. Također, trag se snima za više relacija odjednom, koje mogu biti međusobno povezane stranim ključevima, što, zbog postojanja identičnih atributa i same semantike podataka, implicira iste veze među povijesnim relacijama sa snimljenim tragom. Vodeći se istim načelima za konverziju relacijskih baza podataka u grafovske baze podataka, opisanima u poglavlju 6.1, moguće je ostvariti i konverziju snimljenog traga relacija u potpuno vremenski određeni graf koji će biti pohranjen unutar grafovske baze podataka temeljene na modelu grafa sa svojstvima (definirane u definiciji 25). Važnu ulogu u tom procesu igraju primarni i strani ključevi relacija, a u tom aspektu se snimljeni trag razlikuje od radne baze podataka.

Povijesne relacije mogu sadržavati više n-torki koje su identične u onom dijelu atributa koji je zajednički povijesnoj i radnoj relaciji, jer može biti zapisano obavljanje više operacija nad istom n-torkom, primarni ključ povijesne relacije se razlikuje od primarnog ključa radne relacije. Uz sve atribute koji čine primarni ključ radne relacije, primarni ključ povijesne relacije

treba sadržavati i sve dodatne atribute. Naime, isti zapis u radnoj relaciji je moguće promijeniti na isti način više puta u istom trenutku od strane istog korisnika na bazi podataka, ali ne u istoj korisničkoj sjednici, pa stoga svi ti atributi također trebaju biti dio primarnog ključa svake povijesne relacije.

S druge strane, stvaranje stranih ključeva među povijesnim relacijama koje sadrže navedeni skup dodatnih atributa, u uobičajenom smislu relacijskih baza podataka nije moguće. Sustav za upravljanje relacijskim bazama podataka ne može referencirati jednu n-torku na drugu ako nisu poznati svi atributi stranoga ključa. S obzirom da se govori o povijesnim relacijama, to bi značilo da bi u referencirajućoj relaciji trebao postojati atribut koji bi sadržavao i vrijeme nastanka posljednje aktivne referencirane n-torke, a to nije slučaj. Snimanje traga bi postalo znatno složenije ako bi se htjelo znati točne n-torke u onim povijesnim relacijama koje odgovaraju referenciranim radnim relacijama na koje se strani ključ u n-torki nad kojom se obavlja snimanje traga odnosi. To bi značilo proširenje svake relacije vremenskim zapisima za svaki strani ključ unutar te relacije te također izvršavanje složenih dohvata iz ranije snimljenog traga prilikom snimanja svake operacije nad svakom n-torkom, što bi znatno opteretilo sustav za upravljanje bazama podataka. Umjesto toga, u ovom dijelu snimljenog traga se vjeruje da postoji ispravan strani ključ nad radnom relacijom u trenutku snimanja operacije.

Međutim, iako strani ključ s jedne povijesne relacije na drugu u povijesnoj relacijskoj bazi podataka ne postoji, u implementaciji potpuno vremenski određenoga grafa u grafovskoj bazi podataka će biti moguće realizirati lukove koji će predstavljati strani ključ s n-torke u radnoj relaciji nad kojom se obavlja operacija u nekom trenutku na drugu takvu koja je u tom trenutku bila aktualna. Vrh koji treba referencirati je onaj koji ima odgovarajuću oznaku i vrijednost *id* svojstva i koji je najmlađi među takvima a nastao je prije referencirajućeg vrha. Iz tog razloga se svojstvo *id* vrhova i lukova tretira jednako kao pri konverziji relacijske baze podataka u grafovsku u svim koracima, uz napomenu da u grafovskoj bazi podataka koja predstavlja potpuno vremenski određeni graf, element grafa koji je nositelj informacije o n-torci jedinstveno određuje njegova oznaka, vrijednost *id* svojstva i početak aktivnosti u vremenu.

Pri stvaranju potpuno vremenski određenog grafa temeljem snimljenog traga, vrijede dakle ranije izrečeni ključni aspekti stvaranja grafa iz relacijske baze podataka (poglavlje 6.1). Uz njih, također vrijede i sljedeća proširenja:

1. Analizom metapodataka radnih relacija algoritmima prikazanim u poglavlju 6 određuje se redoslijed obrade povijesnih relacija. Također se određuje n-torke kojih povijesnih relacija će biti prikazane u grafu kao vrhovi, a kojih kao lukovi.
2. Svaki element grafa, vrh ili luk, dobiva dodatna svojstva koja opisuju njegovu aktivnost u vremenu: *t_start* - početni vremenski trenutak u kojem je element nastao; *t_end* - trenutak u kojem element prestaje biti aktivan; *username* - korisnička oznaka korisnika sustava koji je obavio vezanu akciju nad elementom.

3. Vrijednost svojstva *id* je jednaka vrijednosti atributa primarnog ključa n-torke radne relacije. Ako se taj primarni ključ sastoji od više atributa, njihove vrijednosti se spajaju (konkateniraju). Ukoliko u snimljenom tragu postoji više zapisa o operacijama unosa i izmjene nad istom n-torkom, za njih će biti stvoreno isto toliko elemenata grafa, koji će svi imati istu vrijednost svojstva *id*, no imat će različite vrijednosti početka aktivnosti.
4. Zapis u povijesnoj relaciji koji se odnosi na unos nove n-torke u radnoj relaciji rezultira stvaranjem novog elementa u potpuno vremenski određenom grafu. Uz svojstva koja opisuju n-torku na identičan način kao pri konverziji relacije u graf, zapisuje se i vrijeme operacije kao početno vrijeme aktivnosti elementa (vrijednost svojstva *t_start*), i korisnička oznaka korisnika koji je obavio operaciju (vrijednost svojstva *username*).
   Ako se n-torka prikazuje vrhom i sadrži strane ključeve koji će biti prikazani lukovima, pronalaze se referencirani vrhovi prema kojima će ti lukovi biti usmjereni. Referencirani vrh treba imati odgovarajuću oznaku i vrijednost svojstva *id* te treba nastati u vremenu ranijem, ali najbližem vremenu početka ovog vrha. Lukovi koji nastaju kao prikaz stranog ključa, također dobivaju istu vremensku oznaku početka aktivnosti i korisničkog imena kao i novonastali vrh.
   Ako se n-torka prikazuje lukom, vrijedi ista logika. Pronalaze se oni vrhovi koje će luk povezati, s odgovarajućim oznakama i vrijednostima *id* svojstava, a koji su nastali u vremenu ranijem, ali najbližem vremenu nastanka luka koji se stvara.
5. Zapis u povijesnoj relaciji koji se odnosi na brisanje n-torke iz radne relacije rezultira dodavanjem svojstva *t_end* elementu grafa na koji se odnosi, preciznije onoj instanci elementa s odgovarajućom oznakom i svojstvom *id* i vremenom početka aktivnosti ranijim, ali najbližim operaciji brisanja. Ne stvaraju se dodatni elementi i veze, nego se samo dodaje vremenska oznaka prestanka aktivnosti elementa. Ukoliko je element vrh, onda se svojstvo *t_end* dodaje i svim njegovim pripadnim lukovima - svi su nastali temeljem nekih stranih ključeva koji u trenutku prestanka aktivnosti zapisa također bivaju obrisani.
6. Zapis u povijesnoj relaciji koji se odnosi na izmjenu (ažuriranje) n-torke iz radne relacije se promatra kao kombinacija brisanja (dodaje se svojstvo *t_end* posljednjoj instanci elementa s odgovarajućom oznakom i svojstvom *id*) i umetanja nove n-torke s istom oznakom i svojstvom *id*, izmijenjenim svojstvima koja predstavljaju ovisne atribute, i novim početkom aktivnosti. Također, ako se radi o relaciji koja se predstavlja vrhovima, dodaje se i novi luk s posebnom oznakom koja označava da se radi o novoj instanci istog elementa (koristi se oznaka *prethodi*). Taj luk nema drugih svojstava osim *t_start* i *username*.

8.2.1 Postupak popunjavanja grafovske baze podataka potpuno vremenski određenim grafom

Iz snimljenog traga relacijske baze podataka se stvara potpuno vremenski određeni graf koji se pohranjuje u grafovsku bazu podataka. Sukladno rečenome, postupak stvaranja potpuno vremenski određenog grafa temeljem opisanog snimljenoga traga i njegove pohrane u grafovsku bazu podataka značajno ovisi o strukturi radnih relacija čiji se trag snima. Struktura povijesnih relacija je identična strukturi odgovarajućih radnih relacija, uz dodatne atribute, te se analizom metapodataka o radnim relacijama dobivaju relevantne informacije za popunjavanje potpuno vremenski određenoga grafa: koje povijesne relacije će biti predstavljene lukovima a koje vrhovima; ispravan redoslijed popunjavanja; postojanje cikličkih veza i identifikacija stranih ključeva koji ih zatvaraju; identifikacija atributa koji postaju svojstva; identifikacija atributa koji formiraju *id* svojstvo.

Zbog ovoga se postupak popunjavanja potpuno vremenski određenog grafa najprije svodi na obavljanje funkcija *get_migration_order* i $find\_all\_cycle\_ref$, prezentiranih u poglavlju 6.2, nad radnim relacijama, odnosno radnom relacijskom bazom podataka. Nakon toga su popunjeni skupovi $ToNodes$, $ToEdges$ i $CyclicFKs$, što koriste funkcije za popunjavanje potpuno vremenski određenog grafa.

Za opis funkcija za popunjavanje potpuno vremenski određenog grafa unutar grafovske baze podataka koriste se iste pomoćne funkcije unutar relacijske baze podataka opisane u poglavlju 6.2. Pri tom treba obratiti pozornost da se te funkcije obavljaju nad radnom relacijom *r*, dok se dobiveni atributi (u slučaju funkcija *pk* i fks) uporabljuju nad n-torkom povijesne relacije *_r*. Također se pretpostavlja da su unutar grafovske baze podataka implementirane jednostavne funkcije *add_node*, *add_edge* i *add_property* opisane u poglavlju 6.3, ali i dodatne, koje se koriste u svrhu pronalaska vrha, odnosno luka s vremenskim ograničenjima:

- $\mathcal{G}.get\_youngest\_node\_older\_than(:label, id\_value, timestamp)$, koja pronalazi vrh s oznakom *label* i svojstvom *id* koje ima vrijednost *id_value*, koji ima vrijeme nastanka *t_start* starije od vrijednosti *timestamp*, a od svih takvih je najbliže tom vremenu,
- $\mathcal{G}.get\_youngest\_edge\_older\_than(:label, id\_value, timestamp)$, koja pronalazi luk s oznakom *label* i svojstvom *id* koje ima vrijednost *id_value*, koji ima vrijeme nastanka *t_start* starije od vrijednosti *timestamp*, a od svih takvih je najbliže tom vremenu,
- $\mathcal{G}.get\_youngest\_edge\_older\_than(v_1, v_2, :label, timestamp)$, koja pronalazi luk s oznakom *label* između vrhova v_1 i v_2, koji ima vrijeme nastanka *t_start* starije od vrijednosti *timestamp*, a od svih takvih je najbliže tom vremenu,
- $\mathcal{G}.get\_edges(v)$, koja pronalazi sve lukove vrha *v*.

Zbog jednostavnijeg prikaza, proces konverzije povijesne relacije je razdvojen u dvije funk-

cije, ovisno o tome nastaju li temeljem te relacije vrhovi (funkcija *audit_table_to_nodes*) ili lukovi (funkcija *audit_table_to_edges*).

Algoritam 9 Konverzija povijesne relacije u vrhove: **audit_table_to_nodes**($\_r, \mathcal{G}$)

Input: povijesna relacija $\_r$ koja se odnosi na radnu relaciju $r(R)$
Input: grafovska baza podataka $\mathcal{G}$
Output: vrhovi koji predstavljaju n-torke dodani u grafovsku bazu podataka

1. **for all** $t \in \_r$ **do**
2. $id \leftarrow t[pk(r)]$
3. **if** $t[Operation] \in \{'D', 'U'\}$ **then**
4. $v_{old} \leftarrow \mathcal{G}.get\_youngest\_node\_older\_than(:r, id, t[OpTimestamp])$
5. $v_{old}.add\_property('t\_end', t[OpTimestamp])$
6. **for all** $e_i \in \mathcal{G}.get\_edges(v_{old})$ **do**
7. $e_i.add\_property('t\_end', t[OpTimestamp])$
8. **end for**
9. **end if**
10. **if** $t[Operation] \in \{'I', 'U'\}$ **then**
11. $v \leftarrow \mathcal{G}.add\_node(:r, id)$
12. **for all** $A_i \in R \mid A_i \notin pk(r) \wedge A_i \notin fks(r)$ **do**
13. $v.add\_property(A_i, t[A_i])$
14. **end for**
15. $v.add\_property('t\_start', t[OpTimestamp])$
16. $v.add\_property('username', t[OpUser])$
17. **for all** $fk \in fks(r) \mid fk \notin CyclicFKs$ **do**
18. $r_{refd} \leftarrow refd(fk)$
19. $id_{refd} \leftarrow t[fk]$
20. $v_{refd} \leftarrow \mathcal{G}.get\_youngest\_node\_older\_than(:r_{refd}, id_{refd}, t[OpTimestamp])$
21. $e \leftarrow \mathcal{G}.add\_edge(v, v_{refd}, :fk)$
22. $e.add\_property('t\_start', t[OpTimestamp])$
23. $e.add\_property('username', t[OpUser])$
24. **end for**
25. {dodatno, za izmjenu dodajemo i vezu na prethodnu instancu vrha}
26. **if** $t[Operation] =' U'$ **then**
27. $e \leftarrow \mathcal{G}.add\_edge(v_{old}, v, :'prethodi')$
28. $e.add\_property('t\_start', t[OpTimestamp])$
29. $e.add\_property('username', t[OpUser])$
30. **end if**
31. **end if**
32. **end for**

Funkcija *audit_table_to_nodes*, opisana algoritmom 9, obavlja učitavanje podataka iz jedne povijesne relacije za koju je prethodno određeno da će biti konvertirana u vrhove (njezino ime je u *ToNodes* skupu) u grafovsku bazu podataka. N-torke se obrađuju redom kojim se nalaze u povijesnoj relaciji, odnosno redoslijedom nastanka zapisa. Ukoliko je operacija koju n-torka predstavlja brisanje ili izmjena, podrazumijeva se da je već obrađen zapis o unosu ili naknadnim izmjenama zapisa u radnoj relaciji koji predstavlja. Stoga je potrebno najprije pronaći vrh

koji predstavlja taj zapis i dodati mu vrijeme završetka aktivnosti. Također se vrijeme završetka aktivnosti dodaje i svim pripadnim lukovima tog vrha. Ovo je ujedno i sve što će funkcija obaviti u slučaju obrade operacija brisanja, dok se za operaciju izmjene, jednako kao za operaciju unosa, stvara novi vrh. Stvaranje novog vrha se obavlja kao i pri konverziji bilo koje relacije u graf, dodaje se oznaka, svojstvo *id* i druga svojstva koja predstavljaju vrijednosti nezavisnih atributa koji nisu dio niti jednog ključa. Jedina je razlika pri referenciranju relacija lukovima koji predstavljaju strane ključeve, gdje se pronalazi najmlađi vrh koji je nastao prije ovog koji je upravo stvoren te se do njega stvara novi luk. Ovom se vrhu također dodaju i svojstva koja predstavljaju vrijeme početka aktivnosti i korisničku oznaku. Dodatno, ukoliko se radi o operaciji izmjene, funkcija dodaje i luk s oznakom *prethodi* od vrha koji predstavlja prethodnu instancu zapisa radne baze, kojemu je na početku funkcije dodano vrijeme završetka aktivnosti, prema novom vrhu.

Algoritam 10 Konverzija povijesne relacije u lukove: **audit_table_to_edges**($\_r, \mathcal{G}$)

Input: povijesna relacija $\_r$ koja se odnosi na radnu relaciju $r(R)$
Input: grafovska baza podataka $\mathcal{G}$
Output: lukovi koji predstavljaju n-torke dodani u grafovsku bazu podataka

1. **for all** $t \in \_r$ **do**
2. $id \leftarrow t[pk(r)]$
3. **if** $t[Operation] \in \{'D', 'U'\}$ **then**
4. $e_{old} \leftarrow \mathcal{G}.get\_youngest\_edge\_older\_than(:r, id, t[OpTimestamp])$
5. $e_{old}.add\_property('t\_end', t[OpTimestamp])$
6. **end if**
7. **if** $t[Operation] \in \{'I', 'U'\}$ **then**
8. $fk_1, fk_2 \leftarrow fks(r)$
9. $r_{refd1} \leftarrow refd(fk_1)$
10. $r_{refd2} \leftarrow refd(fk_2)$
11. $id_{refd1} \leftarrow t[fk_1]$
12. $id_{refd2} \leftarrow t[fk_2]$
13. $v_{refd1} \leftarrow \mathcal{G}.get\_youngest\_node\_older\_than(:r_{refd1}, id_{refd1}, t[OpTimestamp])$
14. $v_{refd2} \leftarrow \mathcal{G}.get\_youngest\_node\_older\_than(:r_{refd2}, id_{refd2}, t[OpTimestamp])$
15. $e \leftarrow \mathcal{G}.add\_edge(v_{refd1}, v_{refd2}, :r, id)$
16. **for all** $A_i \in R \mid A_i \notin pk(r) \wedge A_i \notin fks(r)$ **do**
17. $e.add\_property(A_i, t[A_i])$
18. **end for**
19. $e.add\_property('t\_start', t[OpTimestamp])$
20. $e.add\_property('username', t[OpUser])$
21. **end if**
22. **end for**

Funkcija *audit_table_to_edges*, opisana algoritmom 10, obavlja učitavanje podataka iz jedne povijesne relacije za koju je prethodno određeno da će biti konvertirana u lukove (njezino ime je u $ToEdges$ skupu) u grafovsku bazu podataka. Slično kao i u slučaju obrade relacija temeljem kojih se stvaraju vrhovi, za n-torke koje predstavljaju operacije brisanja ili izmjene se prona-

lazi luk koji predstavlja pripadni zapis radne relacije koji je najmlađi, ali stariji od n-torke koja se upravo obrađuje. Tom se luku dodaje vrijeme završetka aktivnosti. Također se i ovdje, za operaciju izmjene, kao i za operaciju unosa stvara novi luk među najmlađim starijim vrhovima koje referencira. Njemu se dodaju svojstva koja predstavljaju vrijednosti nezavisnih atributa koji nisu dio ključeva, odnosno svojstva s oznakom vremena početka aktivnosti i korisničkom oznakom.

Funkcija *populate_pvog* prikazana u algoritmu 11, opisuje glavni proces stvaranja potpuno vremenski određenog grafa unutar grafovske baze podataka. Nakon određivanja stranih ključeva koji zatvaraju možebitne cikličke veze i određivanja redoslijeda obrade povijesnih relacija temeljem analize radne baze podataka, povijesne se relacije obrađuju i temeljem njihovih n-torki se stvaraju vrhovi i lukovi.

Algoritam 11 Popunjavanje PVOG iz povijesne relacijske BP: **populate_pvog**(**_r**)

Input: povijesna relacijska baza podataka **_r** koja se odnosi na radnu bazu podataka **r**
Output: grafovska baza podataka $\mathcal{G}$ koja sadrži potpuno vremenski određeni graf

1. $CyclicFKs \leftarrow find\_all\_cycle\_ref(\mathbf{r})$
2. $ToEdges, ToNodes \leftarrow get\_migration\_order(\mathbf{r})$
3. $\mathcal{G} \leftarrow \emptyset$
4. **for all** $r \in ToNodes$ **do**
5. $\quad audit\_table\_to\_nodes(\_r, \mathcal{G})$
6. **end for**
7. **for all** $r \in ToEdges$ **do**
8. $\quad audit\_table\_to\_edges(\_r, \mathcal{G})$
9. **end for**
10. **for all** $fk \in CyclicFKs$ **do**
11. $\quad r_{refing} \leftarrow refing(fk)$
12. $\quad r_{refd} \leftarrow refd(fk)$
13. $\quad$ **for all** $t \in \_r_{refing}$ **do**
14. $\quad\quad id_{refing} \leftarrow t[pk(r)]$
15. $\quad\quad id_{refd} \leftarrow t[fk]$
16. $\quad\quad v_{refing} \leftarrow \mathcal{G}.get\_youngest\_node\_older\_than(:r_{refing}, id_{refing}, t[OpTimestamp])$
17. $\quad\quad v_{refd} \leftarrow \mathcal{G}.get\_youngest\_node\_older\_than(:r_{refd}, id_{refd}, t[OpTimestamp])$
18. $\quad\quad$ **if** $t[Operation] \in \{'D', 'U'\}$ **then**
19. $\quad\quad\quad e_{old} \leftarrow \mathcal{G}.get\_youngest\_edge\_older\_than(v_{refing}, v_{refd}, :fk, t[OpTimestamp])$
20. $\quad\quad\quad e_{old}.add\_property('t\_end', t[OpTimestamp])$
21. $\quad\quad$ **end if**
22. $\quad\quad$ **if** $t[Operation] \in \{'I', 'U'\}$ **then**
23. $\quad\quad\quad e \leftarrow \mathcal{G}.add\_edge(v_{refing}, v_{refd}, :fk)$
24. $\quad\quad\quad e.add\_property('t\_start', t[OpTimestamp])$
25. $\quad\quad\quad e.add\_property('username', t[OpUser])$
26. $\quad\quad$ **end if**
27. $\quad$ **end for**
28. **end for**

Kao i u već opisanom procesu stvaranja grafovske baze podataka iz relacijske, posljednji se

obrađuju strani ključevi koji zatvaraju cikličke reference među relacijama, a to je također dio funkcije *populate_pvog*. U ovom se dijelu procesa ponovno čita dio snimljenog traga, da bi se opet prošlo kroz one povijesne relacije koje sadrže podatke tih stranih ključeva. U slučaju da referentna n-torka predstavlja unos ili izmjenu podataka, zaključuje se koje vrhove novi luk, koji će zatvarati cikličku vezu, treba povezivati. Taj se luk stvara na identičan način kao i drugi lukovi koji predstavljaju strane ključeve u povijesnoj bazi podataka - pronalaze se najmlađi stariji vrhovi koje će povezivati i dodaju im se svojstva koja opisuju vrijeme početka aktivnosti i korisničku oznaku. U slučaju da referentna n-torka predstavlja izmjenu ili brisanje podataka, funkcija pronalazi postojeći takav luk, koji je najmlađi i stariji od referentne n-torke te mu se dodaje vrijeme završetka aktivnosti.

Složenost algoritama u funkcijama *audit_table_to_nodes* i *audit_table_to_edges* treba promatrati kroz količinu n-torki u povijesnoj relaciji koja se obrađuje te broj atributa i stranih ključeva pripadne radne relacije. Kao i u analizi algoritama za konverziju relacija u graf, može se doći do zaključka da je broj n-torki znatno veći od broja atributa i stranih ključeva radne relacije, tako da se oni mogu promatrati kao konstante pa je složenost ovih algoritama $O(n)$. Ovaj se kontekst također može proširiti i na funkciju *populate_pvog*, u kojoj dominiraju iteracije po povijesnim relacijama koje su izvor podataka za vrhove i lukove, odnosno svim relacijama povijesne baze podataka. Stoga se, uzimajući n kao ukupan broj n-torki u svim povijesnim relacijama, također može reći da je složenost tog algoritma linearna.

Sama količina stvorenog sadržaja grafovske baze podataka ovisi prvenstveno o količini sadržaja povijesnih relacija, ali i o strukturi, ponajprije povezanosti relacija u radnoj bazi podataka. Broj stvorenih vrhova odgovarat će ukupnom broju zabilježenih operacija unosa i izmjena podataka u onim relacijama čiji sadržaj stvara vrhove (skup $ToNodes$). Broj lukova će približno biti jednak umnošku broja vrhova u tim relacijama i prosječnog broja stranih ključeva po radnoj relaciji, uz dodatak broja operacija izmjena u tim relacijama (lukovi s oznakom *prethodi*) te broja zabilježenih operacija unosa i izmjena u onim relacijama čiji sadržaj stvara lukove (skup $ToEdges$). Ukupan broj stvorenih svojstava u grafu odgovarat će približno zbroju dvaju umnožaka. Jedan je umnožak broja zabilježenih operacija unosa i izmjena u svim relacijama i prosječnog broja atributa po radnoj relaciji koji nisu dio nijednog ključa uvećanog za tri (svojstva *id*, *t_start*, *username*). Drugi je umnožak ukupnog broja zabilježenih operacija brisanja i prosječnog broja stranih ključeva po radnoj relaciji uvećanog za jedan (svojstvo *t_end* koje se dodaje samom elementu koji se briše, plus svi strani ključevi).

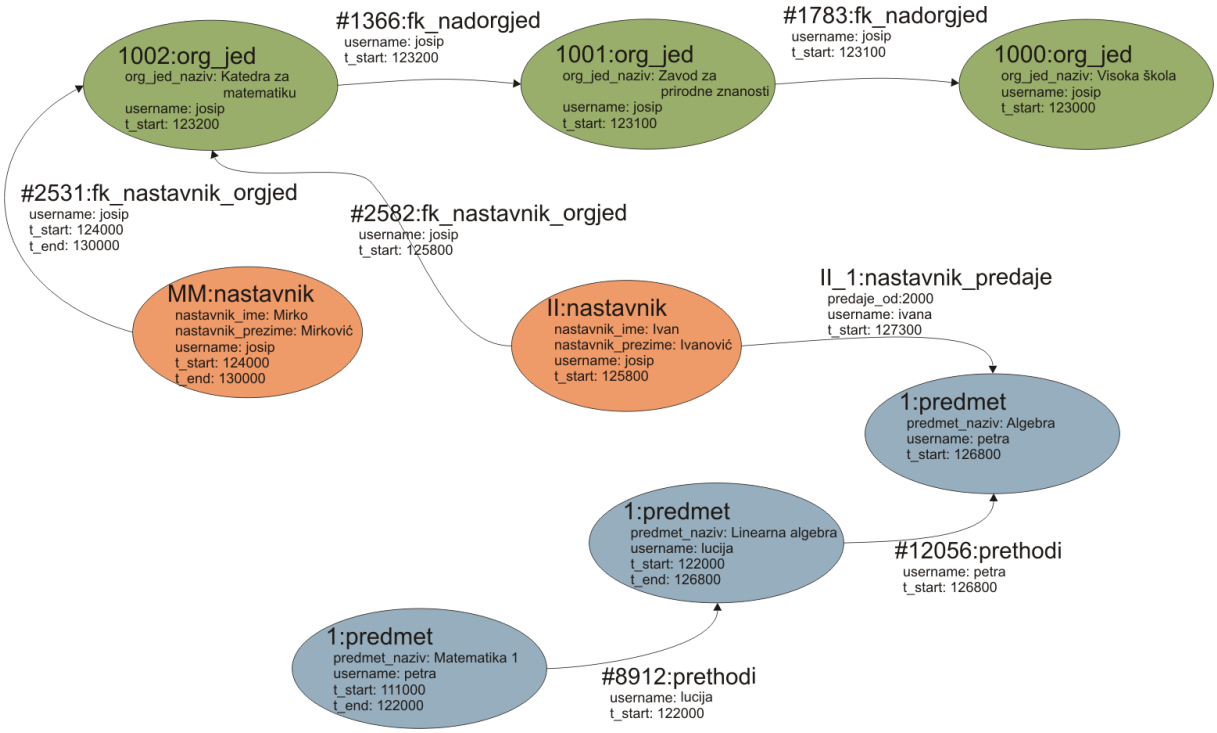
8.2.2 Primjer popunjavanja potpuno vremenski određenog grafa

Popunjavanje potpuno vremenski određenog grafa bit će prikazano kao nastavak primjera 5, korištenjem snimljenoga traga iz tog primjera.

Primjer 6. Temeljem sheme relacijske baze podataka **studentska_sluzba** iz primjera 3 (str. 52) i sadržaja pripadnih povijesnih relacija sa snimljenim tragom iz primjera 5 (tablica 8.1), ukratko će se pokazati postupak popunjavanja grafovske baze podataka s potpuno vremenski određenim grafom.

Kao i kod punjenja grafovske baze podataka sadržajem iz relacijske, u pripremnoj fazi postupka se izvršavaju funkcije *get_migration_order* i *find_all_cycle_ref*, koje će popuniti skupove *ToNodes*, *ToEdges* i *CyclicFKs*. S obzirom da je snimljeni trag iz primjera 5 ograničen samo na četiri relacije, sadržaj ovih skupova će biti:

$$ToEdges = \{nastavnik\_predaje\},$$
$$ToNodes = \{org\_jed, predmet, nastavnik\},$$
$$CyclicFKs = \{fk\_nadorgjed\}.$$



Slika 8.1: Potpuno vremenski određeni graf nastao temeljem podataka iz tablice 8.1

Izvršavanje funkcije *populate_pvog* će rezultirati stvaranjem vrhova i lukova temeljem sadržaja četiriju pripadnih povijesnih relacija. Najprije se stvaraju tri vrha za operacije unosa u relaciju *org_jed*, bez ikakvih stranih ključeva, jer je jedini ključ koji ta relacija posjeduje ujedno i onaj koji zatvara cikličku vezu. Nakon toga se stvaraju vrhovi za relaciju *predmet*. Prvi zapis u relaciji *_predmet* se odnosi na unos te se najprije stvara jedan vrh s vrijednosti svojstva *id* 1. Sljedeći zapis je zapis izmjene, te se najprije pronalazi posljednji najmlađi vrh s istom vrijednosti svojstva *id* i njemu se dodaje vrijeme završetka aktivnosti, dodaje se novi vrh koji

predstavlja novu instancu tog zapisa, i dodaje se luk s oznakom "prethodi" kako bi se povezale ove dvije instance. Treći zapis iz ove relacije obavlja istu stvar, samo je prethodni vrh onaj u kojem se predmet zove "Linearna algebra". Nakon toga se obrađuju relacije koje imaju više stranih ključeva, u ovom slučaju relacija *nastavnik* s jednim stranim ključem. Stvaraju se dva vrha za unose novih zapisa u tu relaciju i povezuju se na posljednju instancu referencirajućeg vrha koji predstavlja organizacijsku jedinicu u kojoj su zaposleni. Povezivanje se obavlja lukom koji predstavlja strani ključ *fk_nastavnik_orgjed* koji također dobiva jednake vrijednosti svojstava *username* i *t_start*. Zadnji zapis iz relacije *_nastavnik* je operacija brisanja te se pronalazi odgovarajući vrh i dodaje mu se svojstvo *t_end*. Isto se obavlja i s pripadnim lukovima tog vrha.

Sljedeći korak u funkciji je konverzija podataka iz relacija koje se predstavljaju lukovima, a to je u ovom primjeru relacija *_nastavnik_predaje*. Relacija sadrži samo jedan zapis unosa podataka, tako da se unutar funkcije *audit_table_to_edges* pronalaze najmlađi stariji vrhovi koje treba povezati novim lukom te se on stvara.

U posljednjem dijelu popunjavanja se stvaraju oni lukovi koji predstavljaju strane ključeve koji zatvaraju cikličke veze u relacijskoj bazi podataka, a to je u ovom slučaju ključ *fk_nadorgjed*. Za tri zapisa u relaciji *_org_jed* se pronalaze odgovarajući najmlađi stariji referencirajući vrhovi i prema njima se stvaraju novi lukovi s vrijednostima svojstava *username*, *t_start* i eventualno *t_end*. Time je postupak popunjavanja potpuno vremenski određenog grafa ovim podacima završio, a konačan rezultat je prikazan na slici 8.1. ■

8.3 Osvrt

U poglavlju 6 prikazani su originalni algoritmi koji objedinjeni služe transformaciji podataka iz relacijskih baza podataka u grafovske baze podataka. Uvođenjem pojma potpuno vremenski određenog grafa u poglavlju 7 i proširenjem spomenutih algoritama u ovom poglavlju, dolazi se do originalnih algoritama za transformaciju vremenskih relacijskih podataka u potpuno vremenski određene grafove.

Svaki od tri navedena segmenta je moguće koristiti zasebno. Algoritmi za transformaciju relacijskih baza podataka u grafovske omogućavaju preglede i analize podataka kakve ranije često nisu bile dostupne, ali i način za trajnu konverziju relacijskih podataka u graf. Potpuno vremenski određeni graf je moguće koristiti za prikaz promjenjivih entiteta i njihovih odnosa u kontinuiranom vremenu, pri čemu je u grafu sadržana njegova povijest, što je model vremenskih grafova koji nije dovoljno korišten u dosadašnjim istraživanjima. Treće, algoritmi za konverziju snimljenog traga relacijskih baza podataka u potpuno vremenski određeni graf omogućavaju analitičko i vizualno istraživanje uzoraka nastajanja i izmjena podataka u relacijskim bazama podataka, primjenama metoda dubinskog istraživanja grafova.

U sljedećim poglavljima će se ova tri segmenta koristiti objedinjeno, u svrhu pronalaska mogućih složenih zlouporaba u informacijskim sustavima.

Poglavlje 9

Česti vremenski podgrafovi unutar potpuno vremenski određenog grafa

Pojam čestog vremenskog podgrafa, kako je objašnjeno u poglavlju 4, podrazumijeva ne samo uzorak vrhova i lukova koji se kao takav pojavljuje u grafu dovoljno često, nego taj uzorak mora zadovoljavati i određene vremenske odnose tih elemenata. Iako je moguće zahtijevati da elementi podgrafa nastaju u točno određenim, različitim vremenskim razmacima jedan u odnosu na drugi (npr. vrh B nastaje pet sekundi nakon vrha A, a luk među njima sedam sekundi nakon toga itd.), u praksi bi taj pristup rezultirao s vrlo malim brojem pronađenih podgrafova, jer bi tražio preveliku preciznost, pogotovo u slučaju kontinuiranog promatranja vremena. Zbog toga je najprije potrebno definirati vremensku toleranciju između promatranih događaja unutar uzorka, koja treba biti dovoljno mala da bi pronađeni vremenski podgrafovi imali smisla za konkretnu primjenu, ali i dovoljno velika da bi ih se uopće moglo identificirati. Ovisno o području primjene potpuno vremenski određenog grafa, ova tolerancija može značajno varirati - od razina sekundi do dana pa ju je potrebno ponavljanjem analiza empirijski odrediti.

Nakon što je utvrđen pojam vremenske tolerancije između elemenata grafa, moguće je promatrati vremenske podgrafove potpuno vremenski određenog grafa i definirati postupak pronalaska takvih podgrafova koji se unutar grafa pojavljuju dovoljno često. Te su teme detaljno opisane u nastavku poglavlja.

9.1 Vremensko susjedstvo

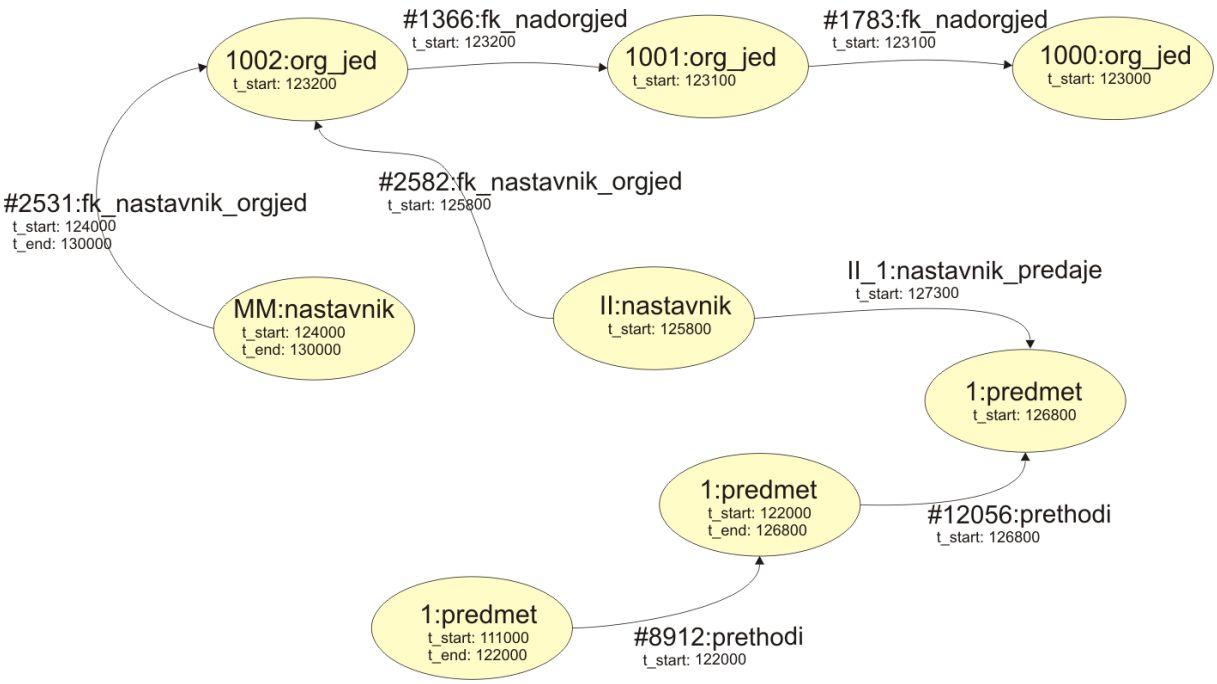
Vremensku toleranciju među promatranim elementima grafa je moguće definirati fiksno, na način da se između svaka dva elementa grafa promatra je li razlika između vremena njihova nastajanja manja od iznosa tolerancije, ili varijabilno, gdje bi ta razlika mogla varirati ovisno o tome o kakvim je susjednim elementima riječ. U drugom slučaju, različitost koja bi implicirala eventualne varijacije u vrijednosti tolerancije može biti npr. oznaka vrha ili luka, tako da se

elementima s jednom oznakom pridaje jedna, a s drugom druga vrijednost tolerancije. Ta različitost također može biti i vrijednost određenog svojstva ili lokalna karakteristika grafa, kao što je stupanj vrha i sl. Određivanje varijabilnih iznosa vremenske tolerancije unosi dodatne složenosti u postupak i nije predmet daljnjeg interesa u sklopu ovog istraživanja te se u nastavku promatra samo slučaj jednake vremenske tolerancije za cijeli potpuno vremenski određeni graf.

Svaki element potpuno vremenski određenog grafa (vrh ili luk) može imati pridijeljenu jednu ili dvije vremenske oznake, odnosno jedan ili dva događaja koji definiraju njegovu aktivnost. Jedan je početak aktivnosti, i taj događaj sigurno imaju svi elementi grafa, a drugi je kraj aktivnosti, koji ne moraju imati svi elementi. Vremenska tolerancija odnosa između dva elementa se može definirati samo kao odnos njihovih početaka aktivnosti, ili kao odnos bilo kojih događaja među njima (npr. početak aktivnosti jednoga i završetak aktivnosti drugoga, ili završetak aktivnosti oba).

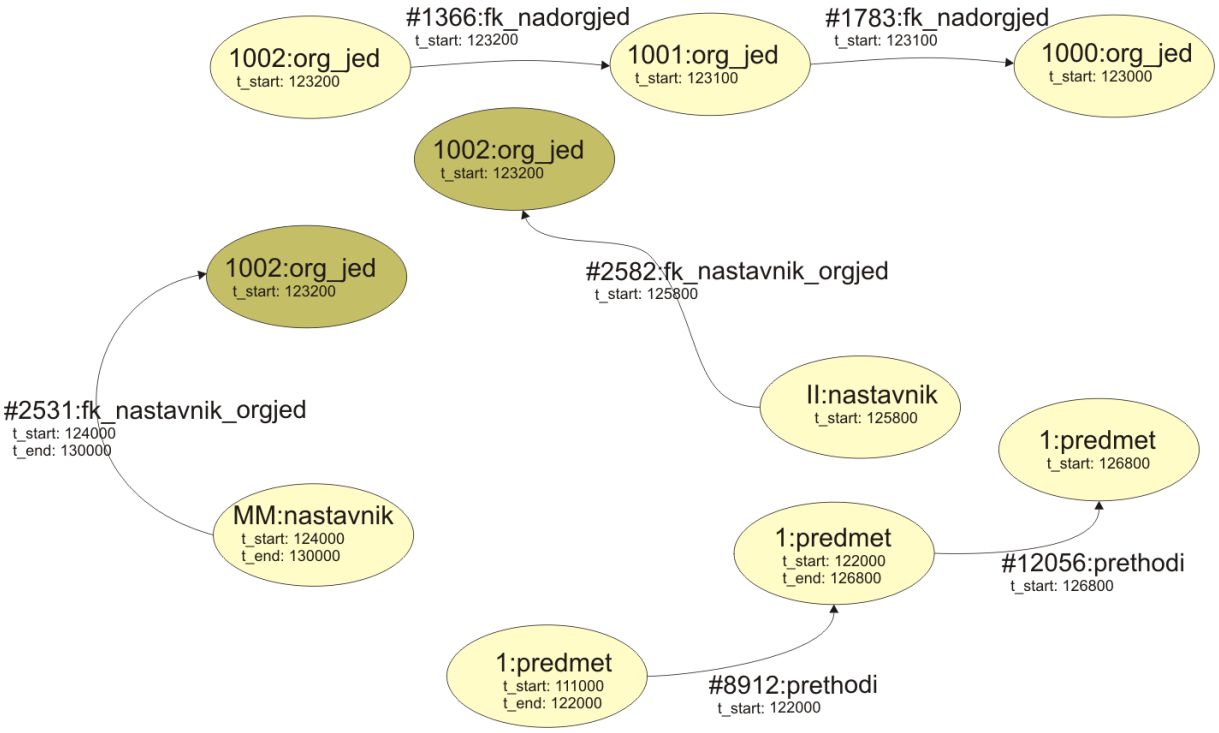
Definicija 27. *Vremensko susjedstvo* je povezani podgraf potpuno vremenski određenog grafa u kojem su se bilo koja dva događaja u aktivnosti vrha i incidentnog mu luka dogodila unutar unaprijed određenog vremenskog intervala. Taj se interval označava s δ i naziva *širina vremenskog susjedstva*.

S obzirom da lukovi u grafu moraju biti povezani na oba kraja, vrhovi koji su na drugom kraju lukova koji su dio vremenskog susjedstva, a svi njihovi događaji aktivnosti su udaljeni više od δ od događaja aktivnosti pripadnog luka, se također pridodaju u vremensko susjedstvo, a nazivaju se *rub vremenskog susjedstva*. ■



Slika 9.1: Pojednostavljeni prikaz potpuno vremenski određenog grafa sa slike 8.1

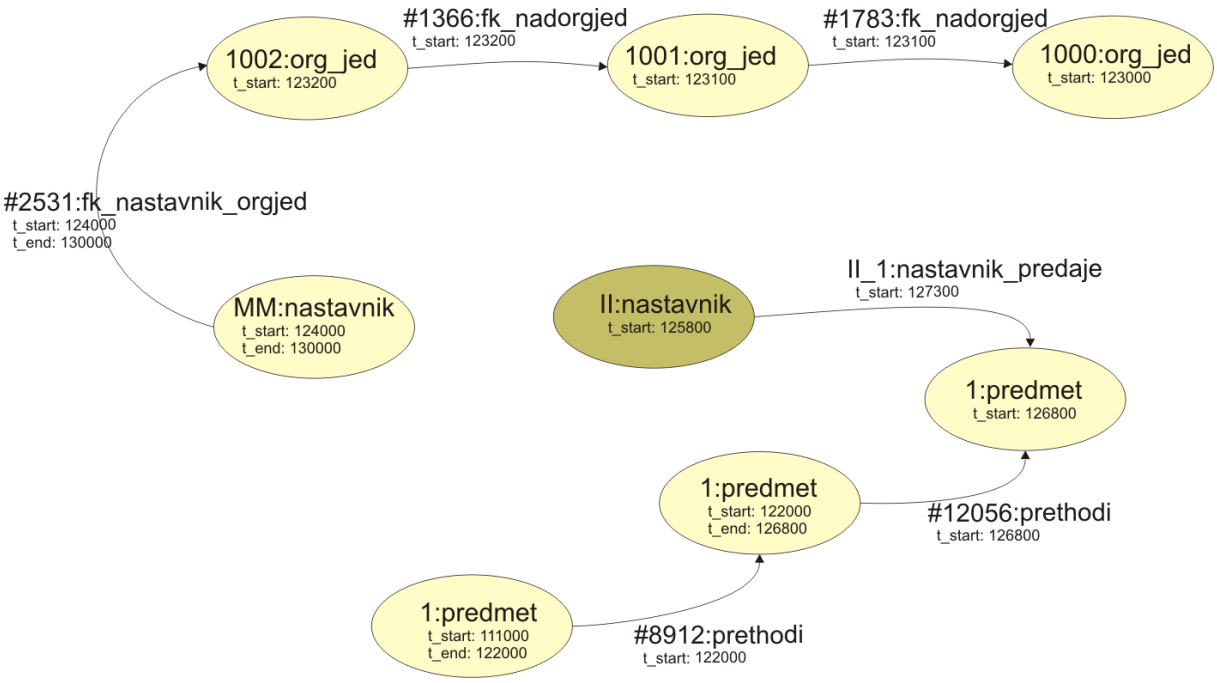
Primjer 7. Na slici 9.1 je pojednostavljeni prikaz potpuno vremenski određenog grafa sa slike 8.1. U ovom prikazu su uklonjena svojstva koja se ne odnose na vremena aktivnosti elemenata te su vrhovi prikazani istom bojom, neovisno o oznaci. Fokus je na svojstvima koja se odnose na vremena aktivnosti elemenata. Ukoliko se, npr. promatra širina vremenskog susjedstva $\delta = 100$ vremenskih jedinica, tada je unutar prikazanog potpuno vremenski određenog grafa moguće izolirati četiri vremenska susjedstva, kako je to prikazano na slici 9.2.



Slika 9.2: Vremenska susjedstva grafa sa slike 9.1 uz $\delta = 100$

Jedno vremensko susjedstvo predstavljaju vrhovi s identifikatorima 1000, 1001 i 1002 i lukovi među njima. Početak aktivnosti luka s identifikatorom #1783 je točno 100 vremenskih jedinica udaljen od početka vrha s identifikatorom 1000. Vrh koji se nalazi na drugom kraju tog luka je nastao u vrijeme kad i luk, tako da je njihova vremenska udaljenost nula. Tom vrhu je incidentan luk s identifikatorom #1366, čiji je početak aktivnosti 100 vremenskih jedinica udaljen od početka aktivnosti vrha, a na njegovom drugom kraju je vrh s identifikatorom 1002 koji je nastao u isto vrijeme. Vrh s identifikatorom 1002 sudjeluje u još dva vremenska susjedstva, no kao rubni vrh vremenskog susjedstva, s obzirom da je u oba slučaja njegovo vrijeme početka aktivnosti udaljeno više od δ spram aktivnosti incidentnog luka koji je dio vremenskog susjedstva. Taj je vrh u ovim vremenskim susjedstvima prikazan tamnijom bojom. U odnosu na izvorni potpuno vremenski određeni graf na slici 9.1, može se primijetiti da nedostaje luk s identifikatorom "II_1", koji je vremenski previše udaljen da bi bio dio ijednog vremenskog susjedstva. Svođenje potpuno vremenski određenog grafa na vremenska susjedstva pojednostavljuje graf, jer eliminira one elemente grafa koji su vremenski izolirani, odnosno nisu unutar

vremenske tolerancije u odnosu na okolne elemente.



Slika 9.3: Vremenska susjedstva grafa sa slike 9.1 uz $\delta = 1000$

Na slici 9.3 su prikazana vremenska susjedstva unutar potpuno vremenski određenog grafa sa slike 9.1 uz širinu vremenskog susjedstva $\delta = 1000$ vremenskih jedinica. U ovom slučaju se mogu identificirati dva vremenska susjedstva, niti jedan vrh se ne pojavljuje u više njih, a vrh s identifikatorom "II" je dio ruba drugog vremenskog susjedstva, s obzirom da je njegov početak aktivnosti više od 1000 vremenskih jedinica udaljen od aktivnosti incidentnog luka s identifikatorom "II_1". ■

9.1.1 Poredak događaja u vremenskom susjedstvu

Uvođenjem koncepta vremenskog susjedstva ispunjen je prvi uvjet za identifikaciju smislenih uzoraka u vremenskom grafu, a to je ograničavanje veličine podgrafa na određenu vremensku dimenziju. Osim toga, uzorke u vremenskom grafu određuje i redoslijed događaja nad elementima u njemu. Nije svejedno je li najprije nastao jedan vrh, a nakon njega drugi te luk koji ih povezuje, ili je najprije nastao drugi vrh a nakon njega prvi i luk koji ih povezuje. Na primjer, to može značiti razliku između činjenica da se osoba prijavila za predodobreni kredit ili se najprije prijavila za kredit, koji joj je kasnije odobren. Sa stajališta sigurnosti, ovisno o ponavljanju, oba uzorka mogu indicirati potencijalne probleme.

S obzirom da širina vremenskog susjedstva δ definira unaprijed određeni najveći razmak među dva susjedna elementa vremenskog susjedstva, konkretni vremenski razmak među njima

nije od pomoći pri identificiranju čestih podgrafova, i može ga se zanemariti, no vremenske odrednice početka i završetka aktivnosti elementa treba iskoristiti prilikom definiranja redoslijeda događaja unutar vremenskog susjedstva. U svrhu definiranja redoslijeda događaja unutar vremenskog susjedstva, uvodi se pojam vremenskog indeksa.

Definicija 28. *Vremenski indeks* je redni broj događaja unutar vremenskog susjedstva u odnosu na referentni događaj tog vremenskog susjedstva.

Referentni događaj ima vremenski indeks nula. Događaji koji su se dogodili nakon njega imaju pozitivne, a događaji prije njega imaju negativne vremenske indekse.

Događaji koji pripadaju rubu vremenskog susjedstva i događaji koji su u odnosu na drugi događaj vrha ili luka (početak ili završetak aktivnosti) udaljeni više od δ, indeksirani su s odmakom ξ u odnosu na druge događaje. Ako se takav događaj dogodio nakon referentnoga, taj je odmak pozitivan, a ako se dogodio prije, onda je taj odmak negativan. ■

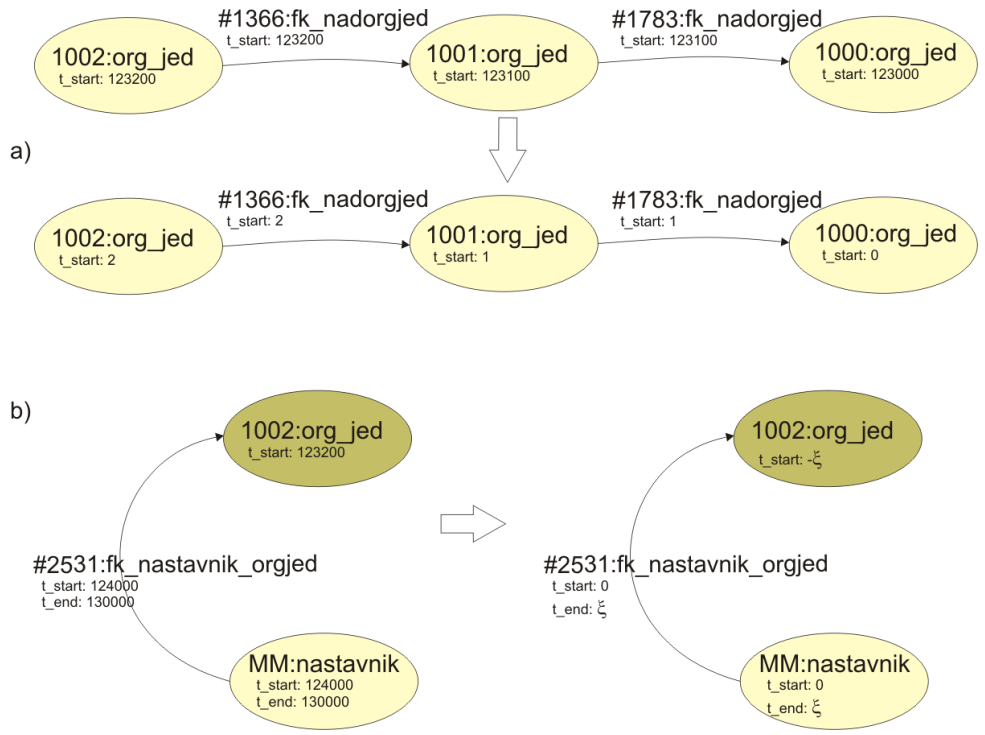
Ukoliko je referentni događaj vremenskog susjedstva početak aktivnosti određenog vrha, svi drugi događaji koji su se dogodili u isto vrijeme, također dobivaju vremenski indeks nula. Dva susjedna vremenska indeksa u vremenskom susjedstvu su udaljena najviše δ vremenskih jedinica jedan od drugoga, osim ako ne predstavljaju rubne događaje, odnosno one koje su vremenski daleko od drugog događaja vrha ili luka. Oni međusobno, i u odnosu na referentni događaj susjedstva mogu biti proizvoljno daleko udaljeni, no u smislu promatranja samog vremenskog susjedstva, važan je samo njihov redoslijed. Pri tom je ξ dovoljno velik broj koji ne interferira s vremenskim indeksima susjedstva.

Definicija 29. *Normirano vremensko susjedstvo* je vremensko susjedstvo širine δ u kojem su vremenske odrednice događaja početka i završetka aktivnosti elementa svedene na vremenske indekse tih događaja unutar vremenskog susjedstva. ■

Primjer 8. Na slici 9.4 je primjer normiranja dvaju vremenskih susjedstava grafa sa slike 9.1 uz $\delta = 100$.

U vremenskom susjedstvu na dijelu slike a) je referentni događaj stvaranje najranijeg vrha (desni na slici). Sljedeći događaji su stvaranje vrha s identifikatorom 1001 i pripadnog luka #1783 te oni dobivaju vremenski indeks 1, a slično je i s trećim vrhom.

U vremenskom susjedstvu na dijelu slike b) referentni događaj je stvaranje vrha s identifikatorom "MM". Događaj prestanka njegove aktivnosti je već predaleko u budućnosti, te dobiva vremenski indeks ξ, kojem odgovara i prestanak aktivnosti luka s identifikatorom #2531. Vrh koji je rubni ovom vremenskom susjedstvu, s identifikatorom 1002, je nastao ranije u odnosu na vremensko susjedstvo te kao prvi među takvima, taj događaj dobiva vremenski indeks $-\xi$. ■.



Slika 9.4: Normirana vremenska susjedstva grafa sa slike 9.1 uz $\delta = 100$

9.2 Podgraf potpuno vremenski određenog grafa

Podgrafove potpuno vremenski određenog grafa se može promatrati na sličan način kao podgrafove običnog, statičkog grafa. Razlog leži u činjenici da je potpuno vremenski određeni graf sličniji statičkom nego dinamičkom u smislu postojanja vrhova i lukova kroz vrijeme. Jednom stvoreni elementi grafa ne nestaju, nego im se samo dodaju svojstva i pristižu novi elementi grafa. Isto tako se i pronalazak čestih vremenskih podgrafova u potpuno vremenski određenom grafu može svesti na pronalazak čestih podgrafova običnog, statičkog grafa, uz određene pripremne radnje nad samim potpuno vremenski određenim grafom. Stoga će biti prikazana prilagodba potpuno vremenski određenog grafa i algoritma GraMi (prezentiranog u poglavlju 4.3) u cilju pronalaska čestih podgrafova potpuno vremenski određenog grafa.

Pronalazak čestih vremenskih podgrafova uz pomoć algoritma GraMi ostvarit će se bez modifikacije same jezgre tog algoritma. Prilagodba jezgrenog dijela algoritma GraMi podacima koji su u obliku potpuno vremenski određenog grafa implicira znatno širu domenu istraživanja u odnosu na opseg ovog rada. Naime, algoritam GraMi kao svoju jezgru koristi gSpan algoritam [106], koji kandidate za česte podgrafove pronalazi šetnjom po grafu u dubinu. Uvođenje vremenske komponente u grafove donosi novu dimenziju koju je potrebno uzeti pri prolasku grafom, pa time i sami pronalazak kandidata za česte vremenske podgrafove postaje znatno složeniji. Način prolaska po vremenskim grafovima je zasebna tema istraživanja [142].

S obzirom da se potpuno vremenski određeni graf sastoji od elemenata koji osim vremenskih odrednica aktivnosti sadrže i druga svojstva te oznake, tako i elementi podgrafa trebaju moći

sadržavati svojstva i oznake. U isto vrijeme, potrebno je moći iskoristiti optimirani proces pronalaska čestih podgrafova algoritmom GraMi, što je u određenoj mjeri u kontradikciji jedno s drugim. Naime, algoritam GraMi, kao i mnogi drugi iz područja pronalaska čestih podgrafova, je ograničen s obzirom na značajke elemenata grafa koji obrađuje. Konkretno, vrhovi i lukovi mogu imati samo jednu, i to brojčanu, oznaku, i prema toj oznaci ih se klasificira i pretražuje česte podgrafove. Da bi se ovaj algoritam uspješno primijenio na potpuno vremenski određeni graf, potrebno je:

- kvalitete pojedinih elemenata ovog grafa (oznaku, svojstva i vrijednosti, vremensku aktivnost) predstaviti jednom zamjenskom brojčanom oznakom,
- osigurati da elementi koji sadrže iste kvalitete (istu oznaku, ista svojstva s istim vrijednostima, istu vremensku situaciju) dobiju istu zamjensku oznaku,
- osigurati da je moguć povrat informacija o kvalitetama elementa iz zamjenske oznake.

Za uspješan pronalazak čestih podgrafova je potrebno pojednostavniti informacije o vremenskim susjedstvima, odnosno sve informacije o situaciji vremenskog susjedstva treba predstaviti jednom zamjenskom oznakom pojedinog elementa (uključujući i dodatne informacije o svojstvima i izvornoj oznaci). Ovaj se problem stoga rješava korištenjem *map-reduce* tehnika, odnosno preslikavanja (mapiranja) istih vrijednosti (svojstava, njihovih vrijednosti, oznaka, pa i vremenskih oznaka). Na taj način se bez gubitaka reducira količina informacija koja se koristi u daljnjim algoritmima. Jednom reducirane informacije je često moguće ponovno preslikavati u nove vrijednosti i na taj način znatno reducirati informacije, sve do razine zamjenske oznake.

Metoda preslikavanja i reduciranja podataka elemenata potpuno vremenski određenog grafa u cilju stvaranja zamjenskih oznaka će biti opisana u nastavku ovog poglavlja, a najvažniji korak u stvaranju zamjenske oznake vrha ili luka jest uključivanje informacija o vremenskom susjedstvu u tu oznaku.

9.2.1 Vremensko susjedstvo elementa potpuno vremenski određenog grafa

Kako bi se informacije o vremenskom susjedstvu prenijele u algoritam za pronalazak čestih podgrafova, za svaki element potpuno vremenski određenog grafa se promatra samo njegovo vremensko susjedstvo, koje se kodira u dodatna svojstva elementa i prenosi u postupak preslikavanja potpuno vremenski određenog grafa. Iz perspektive pojedinog elementa grafa, vremensko susjedstvo je znatno jednostavnije od vremenskog susjedstva cijelog grafa.

Definicija 30. *Vremensko susjedstvo vrha* je vremensko susjedstvo širine δ čiji je referentni događaj početak aktivnosti vrha, a ograničeno je samo na incidentne lukove tog vrha. ■

Definicija 31. *Vremensko susjedstvo luka* je vremensko susjedstvo širine δ čiji je referentni događaj početak aktivnosti luka, a ograničeno je samo na vrhove kojima je luk incidentan. ■

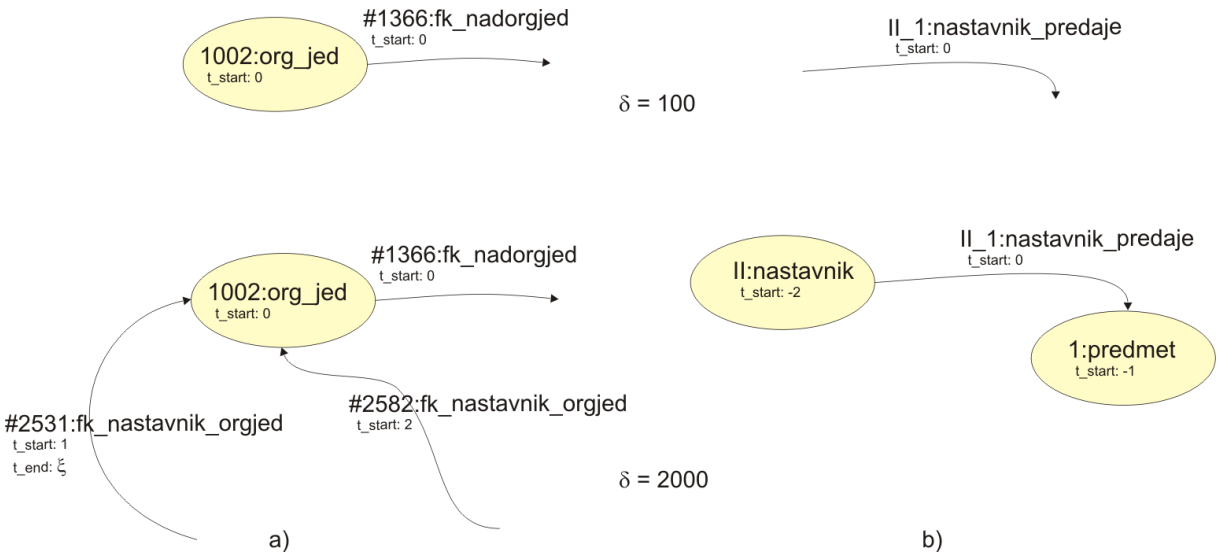
Slijedom ovih definicija, uvodi se i definicija normiranog susjedstva elementa.

Definicija 32. *Normirano vremensko susjedstvo elementa grafa* je vremensko susjedstvo elementa širine δ u kojem su vremenske odrednice svih događaja svedene na vremenske indekse unutar tog susjedstva, pri čemu se drugi elementi eliminiraju iz susjedstva ako su njihovi događaji više od δ udaljeni od bilo kojeg drugog događaja. ■

Također, u promatranju odnosa događaja elemenata grafa u kontekstu vremenskog susjedstva, koristit će se i termini poput *daleko* i *blizu* koji se odnose na činjenicu jesu li događaji izvan ili unutar vremenskog intervala zadanog širinom vremenskog susjedstva.

Dvije su važne činjenice pri definiranju najužeg susjedstva elemenata kao što je učinjeno definicijom 32. Najprije, promatraju se samo neposredni dodirni elementi, što su za vrh incidentni lukovi, i sa stajališta vrha, zanemaruje se informacija što je sadržano u vrhu koji se nalazi na drugoj strani luka. Ta informacija će biti sadržana u sklopu vremenskog susjedstva luka, tako da nije izgubljena, a na ovaj način se minimizira skup informacija koje je potrebno preslikati da bi se dobio vremenski određeni vrh koji se koristi u pronalasku čestih podgrafova. Drugo, uvodi se eliminiranje pripadnih elemenata koji su na rubu vremenskog susjedstva, a nisu u vremenskom odnosu unutar δ spram bilo kojeg drugog elementa. Na ovaj način se fokus istraživanja zadržava na vremenskim odnosima elemenata koji, iako su daleko od relevantnog događaja, mogu biti vremenski povezani na način koji je zanimljiv za istraživanje, a zanemaruju se oni elementi koji su vremenski predaleko od ostalih u susjedstvu. Ukoliko ti elementi nakon pripremne faze analize normiranih vremenskih susjedstava elemenata ne budu ni na koji način povezani, uklanjaju se iz grafa i na taj način smanjuju ukupnu količinu informacija potrebnih za obradu. Elementi, odnosno njihovi pripadni događaji, koji ostaju unutar normiranog vremenskog susjedstva elementa, a daleko su od relevantnog događaja i njemu bliskih događaja, indeksirani su odmakom ξ koji može biti pozitivan ili negativan, ovisno o vremenu događaja. U slučaju normiranog vremenskog susjedstva vrha, svi vezani događaji unutar susjedstva mogu imati samo pozitivan odmak, jer lukovi mogu nastati tek nakon vrha.

Primjer 9. Na slici 9.5 su prikazana normirana vremenska susjedstva dvaju elementa iz potpuno vremenski određenog grafa prikazanog na slici 9.1, uz dvije različite vrijednosti širine vremenskog susjedstva δ: 100 i 2000 vremenskih jedinica. Na dijelu slike a) je prikaz vremenskih susjedstava vrha s identifikatorom 1002 i oznakom *org_jed*. U slučaju $\delta = 100$, iako ovaj vrh u grafu ima tri incidentna luka, svi događaji vezani uz dva od njih su udaljeni više od δ u odnosu na sve ostale događaje, te su ti lukovi eliminirani, i preostao je jedan, koji ima jednako vrijeme početka aktivnosti kao i sam vrh, te oba ta događaja imaju vremenski indeks 0. U slučaju $\delta = 2000$, događaji su unutar δ jedan u odnosu na drugi, nema eliminiranih lukova, a jedan od događaja incidentnog luka s oznakom #2351 je izvan ostalih te on, s obzirom na to da taj luk nije eliminiran, dobiva vremenski indeks ξ.



Slika 9.5: Normirana vremenska susjedstva elemenata s identifikatorima 1002 i "II_1" iz grafa 9.1 uz $\delta = 100$ i $\delta = 2000$

Na dijelu slike b) je prikaz vremenskih susjedstava luka s identifikatorom "II_1" i oznakom *nastavnik_predaje*, a referentni je događaj početak aktivnosti luka. U slučaju $\delta = 100$, događaji vezani uz oba vrha kojima je incidentan su daleko od svih drugih unutar susjedstva, te su ti vrhovi eliminirani, i luk je sam u susjedstvu. S obzirom da u susjedstvu nema vrhova kojima je incidentan, i ovaj će luk biti eliminiran iz daljnjeg promatranja grafa i identifikacije podgrafova. U slučaju $\delta = 2000$, događaji vezani uz oba vrha kojima je incidentan su unutar susjedstva, a s obzirom da su se dogodili ranije od referentnog događaja, dobivaju negativne vremenske indekse. ■

Nakon što je određeno normirano vremensko susjedstvo relevantnog elementa, moguće je, uvođenjem dogovorene nomenklature, početak i završetak aktivnosti elemenata susjedstva i pripadne vremenske indekse proglasiti svojstvima relevantnog elementa i preslikati ih zajedno s ostalim svojstvima kako bi se dobila jedinstvena zamjenska oznaka tog elementa.

9.2.2 Preslikavanje oznaka i svojstava elemenata i redukcija na zamjensku oznaku

Kako bi se smanjio broj svojstava koja pri usporedbi podgrafova treba usporediti, najjednostavnije ih je preslikati u zajedničku mapu, tako da u konačnici svako svojstvo i njegova vrijednost mogu biti zamijenjeni brojem iz kojeg je moguće zaključiti o kojem se svojstvu i vrijednosti radi. U tu svrhu se koristi podatkovna struktura mapa, koja sadrži ključeve i vrijednosti te činjenica da i ključevi i vrijednosti mogu biti druge složene podatkovne strukture, npr. mape ili skupovi.

Prilikom preslikavanja svojstava potrebno je održavati sadržaj mape s nazivima svih svojstava. Neka je to

$$M_{Properties} = \{(PropIndex_i, PropName_i)\} \tag{9.1}$$

U toj mapi je jedinstveni ključ redni broj svojstva $PropIndex_i$ (svojstva se mogu proizvoljnim redoslijedom upisivati u mapu), a vrijednost je naziv svojstva $PropName_i$. Druga mapa sadrži sve upisane vrijednosti svih svojstava. Ključ u ovoj mapi je također redni broj svojstva $PropIndex_i$, a vrijednost je mapa u kojoj su sve vrijednosti tog svojstva - ključ je redni broj vrijednosti svojstva $PropValueIndex_j$, a vrijednost je vrijednost svojstva $PropValue_j$:

$$M_{Values} = \{(PropIndex_i, \{(PropValueIndex_j, PropValue_j)\})\} \tag{9.2}$$

Na ovaj je način, u dvije mape $M_{Properties}$ i M_{Values} moguće upisati sva svojstva i sve njihove vrijednosti, ne dupliciraju̇ći nijedno od toga, i na taj način, jedinstveno odrediti svojstvo i njegovu vrijednost koji se ponavljaju u grafu pomoću dva broja, $PropIndex$ i $PropValueIndex$. Uz pretpostavku da se broj različitih vrijednosti svojstava može ograničiti, odnosno da ih u grafu neće biti beskonačno, ova je dva broja moguće kombinirati u jedan broj:

$$CombinedPropIndex = PropIndex \times MAX\_DISTINCT\_PROPS + PropValueIndex \tag{9.3}$$

pri čemu je $MAX\_DISTINCT\_PROPS$ najveći podržani broj različitih vrijednosti svojstva, npr. 10^9. Time je svaka kombinacija svojstvo-vrijednost koja se može pojaviti proizvoljan broj puta u grafu u svedena na brojku koju je moguće koristiti u daljnjem postupku.

Primjer 10. Neka postoji pet elemenata grafa sa svojstvima kao u tablici 9.1.

Tablica 9.1: Svojstva elemenata za preslikavanje u primjeru

svojstvo	element 1	element 2	element 3	element 4	element 5
ime	Josip	Matija	Matija	Josip	Matija
spol	M	M	Ž	M	M
godine	33	32	33		32
radni odnos	da	da		ne	da

Prilikom obrade svojstava pojedinih elemenata popunjavaju se mape $M_{Properties}$ i M_{Values}. Najprije se obrađuje prvi element, tako da se u mape dodaje svojstvo "ime" i njegova vrijednost "Josip" tako da je sadržaj mapa u tom trenutku:
$M_{Properties} = \{(1, ime)\}$
$M_{Values} = \{(1, (1, Josip))\}$

Nakon završetka obrade svih svojstava prvog elementa, sadržaj mapa sadrži sva svojstva i po jednu vrijednost za njih:

$M_{Properties} = \{(1, ime), (2, spol), (3, godine), (4, radni\ odnos)\}$
$M_{Values} = \{(1, \{(1, Josip)\}), (2, \{(1, M)\}), (3, \{(1, 33)\}), (4, \{(1, da)\})\}$

Obrada drugog elementa donosi novu vrijednost za svojstvo "ime" i novu vrijednost za svojstvo "godine", dok se za druga dva svojstva koriste već preslikana svojstva i njihove vrijednosti:
$M_{Values} = \{(1, \{(1, Josip), (2, Matija)\}), (2, \{(1, M)\}), (3, \{(1, 33), (2, 32)\}), (4, \{(1, da)\})\}$

Obrada trećeg elementa donosi novu vrijednost za svojstvo "spol", a obrada četvrtog elementa novu vrijednost za svojstvo "radni odnos". Konačno, obrada petog elementa ne donosi nove vrijednosti u mape, te je konačni sadržaj mape s vrijednostima:
$M_{Values} = \{$
$(1, \{(1, Josip), (2, Matija)\}),$
$(2, \{(1, M), (2, Z)\}),$
$(3, \{(1, 33), (2, 32)\}),$
$(4, \{(1, da), (2, ne)\})\}$

Elementi, dakle, sadrže ukupno četiri svojstva od kojih svako ima po dvije različite vrijednosti. Uz pretpostavku da neće biti više od 100 različitih vrijednosti pojedinog svojstva, mogu se stvoriti kombinirani indeksi koji predstavljaju indeks svojstva pomnožen brojem 100 zbrojen s indeksom vrijednosti tog svojstva, što u konačnici omogućava jednostavniji prikaz svojstava elemenata (tablica 9.2).

Tablica 9.2: Preslikana svojstva elemenata iz tablice 9.1

element	svojstva
element 1	101, 201, 301, 401
element 2	102, 201, 302, 401
element 3	102, 202, 301
element 4	101, 201, 402
element 5	102, 201, 302, 401

Ovim su postupkom svojstva iz oblika prilagođenijeg ljudskom pogledu transformirana u skupove brojeva nad kojima je računalno jednostavnija obrada. ■

Kao što je već elaborirano, normirani vremenski događaji početka i završetka aktivnosti elementa grafa i njemu dodirnih elemenata se također mogu, uz unaprijed definiranu nomenklaturu, predstaviti svojstvima. Isto vrijedi i za oznake elemenata, koje mogu postati vrijednosti svojstva unaprijed određenog imena, npr. "node_label" za vrhove, i "edge_label" za lukove.

Sva pobrojana svojstva pojedinog elementa se dalje mogu preslikati u zajedničku mapu, kako bi se dobila jedinstvena brojčana zamjenska oznaka, s kojom je moguće računati česte

podgrafove. U tu svrhu se koristi mapa čiji je ključ sama zamjenska oznaka *LabelIndex* (zapravo je to redni broj kombinacije svojstava) a vrijednost skup svih kombinacija svojstava i njihovih vrijednosti, $CombinedPropIndex$, koje element grafa ima:

$$M_{Labels} = \{(LabelIndex_i, \{CombinedPropIndex_j\})\} \tag{9.4}$$

Na ovaj način se osigurava da, ukoliko dva elementa imaju ista svojstva i njihove vrijednosti, posljedično imaju isti skup kombiniranih indeksa svojstava te u konačnici dobiju istu zamjensku varijablu. Ovo je važna činjenica za nastavak postupka pronalaska čestih vremenskih podgrafova, jer znači da će npr. dva vrha s istim relevantnim svojstvima i istim vremenskim susjedstvom, biti promatrani kao identični vrhovi.

Primjer 11. Preslikana svojstva iz prethodnog primjera, prikazana u tablici 9.2 se za svaki element bilježe u skupove te se svaki skup evidentira u mapi M_{Labels}, s pripadnim indeksom, koji predstavlja novu, zamjensku oznaku elementa, koja u sebi sadrži sve relevantne informacije o njemu. Ukoliko identičan skup već postoji u mapi, elementu se pridjeljuje indeks koji pripada tom skupu.

Elementi iz prethodnog primjera se redom obrađuju te se u mapu dodaju skupovi za prvi, drugi, treći i četvrti element, s obzirom da su svi različiti, a za peti element već postoji identičan skup, i taj se ne dodaje u mapu. Sadržaj mape nakon obrade svih elemenata je:
M_{Labels} = {
(1,{101, 201, 301, 401}),
(2,{102, 201, 302, 401}),
(3,{102, 202, 301}),
(4,{101, 201, 402})}

a elementi dobivaju nove zamjenske oznake prikazane u tablici 9.3.

Tablica 9.3: Zamjenske oznake elemenata iz tablice 9.2

element	zamjenska oznaka
element 1	1
element 2	2
element 3	3
element 4	4
element 5	2

Time se drugi i peti element smatraju identičnima u daljnjoj analizi grafa. ■

9.3 Postupak pronalaska čestih vremenskih podgrafova potpuno vremenski određenog grafa

Postupak pronalaska čestih vremenskih grafova svodi se na spajanje ideja prezentiranih u ovom poglavlju uz uporabu algoritma GraMi prezentiranog u poglavlju 4.3. Ukratko, u algoritmu je potrebno obraditi elemente grafa i za njih odrediti normirana vremenska susjedstva, uz koja dodati i relevantna svojstva i oznaku elementa te odrediti zamjensku oznaku svakog od elemenata. Nad tako pripremljenim elementima grafa treba primijeniti algoritam GraMi, čiji konačni skup pronađenih čestih podgrafova je na kraju potrebno prilagoditi tako da se pronađeni česti podgrafovi zapišu s elementima u izvornom formatu, kao elementi potpuno vremenski određenog grafa s relevantnim svojstvima. Tako zapisan česti podgraf je česti vremenski podgraf potpuno vremenski određenog grafa.

9.3.1 Pomoćne programske strukture

U cilju jednostavnijeg opisa algoritama koji se koriste pri obradi elemenata grafa i pronalaska čestih vremenskih podgrafova, uvode se pomoćne programske strukture koje opisno sažimaju i objedinjuju ranije pojašnjenje procese.

Najprije se uvodi razred $TemporalAdjacency$. Ova programska struktura održava skup događaja jednog vremenskog susjedstva, obavlja osnovne operacije nad tim skupom i za konkretne vremenske oznake pronalazi pripadne vremenske indekse. Skup događaja nad kojima se obavljaju operacije može se promatrati kao $D = \{t_i\}$, pri čemu je $t_i \in \mathcal{T}$ vrijeme pojedinog događaja unutar vremenskog susjedstva. Nad ovim razredom su definirane sljedeće procedure i funkcije:

- $add\_event(t)$, koja u skup događaja dodaje događaj s vremenom t,
- $set\_reference(t)$, postavlja događaj t za referentni događaj, tako da se u odnosu na njega može izračunati nulti indeks te, uz poznavanje širine susjedstva, pozitivni i negativni indeksi ostalih događaja, uključujući i one koji su indeksirani s odmakom $\pm\xi$,
- $set\_width(\delta)$, postavlja širinu vremenskog susjedstva δ,
- $sort()$, koja reda elemente skupa D u kronološki redoslijed,
- $remove\_distant\_neighbors()$, koja iz skupa D uklanja elemente koji su od svojih susjeda udaljeniji od δ vremenskih jedinica, pri čemu se referentni događaj ne uklanja,
- $remove\_duplicates()$, koja iz skupa D uklanja sve elemente koji se višestruko pojavljuju, ostavljajući jedan od njih,
- $is\_empty()$, koja daje informaciju da li je skup nakon uklanjanja elemenata ostao prazan,
- $get\_index(t)$, koja temeljem referentnog događaja i širine susjedstva δ pronalazi indeks događaja koji ima vrijeme t.

Sljedeća pomoćna programska struktura koja se uvodi je razred *PropMapper*, koji je programska implementacija mapa $M_{Properties}$ i M_{Values} i pripadnih akcija nad njima. Osim internog sadržaja ovih mapa, nad ovim razredom se definira i funkcija:

- *get_property*(*prop_name*, *prop_value*), koja u navedenim mapama pronalazi kombinirani indeks za svojstvo *prop_name* s vrijednosti *prop_value*, a ako ta kombinacija ne postoji, upisuje ju te u daje pripadni novi kombinirani indeks.

U skladu s time, definira se i razred *ReplacementLabelMapper*. Ova programska struktura je implementacija mape M_{Labels} koju sadrži, a nad ovim razredom definiraju se funkcije:

- *get_label*(*properties*), koja za skup kombiniranih indeksa svojstava *properties* daje pripadnu zamjensku oznaku,
- *expand*(x), koja elementu grafa x pridjeljuje sva svojstva, vremensko susjedstvo i oznaku, koristeći mape M_{Labels}, $M_{Properties}$ i M_{Values} i njegovu početnu oznaku kao zamjensku; koristi se pri reverznom postupku dobivanja podataka o potpuno vremenski određenom grafu iz zamjenskih oznaka.

Pri samom pronalasku čestih podgrafova se koristi ranije opisani algoritam GraMi. Za prezentaciju algoritma za pronalazak čestih vremenskih podgrafova, uvodi se GraMi kao programska struktura koja sadrži sljedeće procedure i funkcije:

- *add_node*(v, *label*), koja dodaje vrh v sa zamjenskom oznakom *label* u interni opis grafa za GraMi,
- *add_edge*(v_1, v_2, *label*), koja dodaje luk od vrha v_1 do vrha v_2 sa zamjenskom oznakom *label* u interni opis grafa za GraMi,
- $find\_FSGs(f)$, koja pronalazi sve česte podgrafove među zabilježenim vrhovima i lukovima, s minimalnom frekvencijom f i daje ih kao skup grafova.

Pri opisu algoritama koristi se nomenklatura *x.prop* koja se odnosi na vrhove i lukove grafovske baze podataka i označava dohvat vrijednosti svojstva *prop* elementa x, odnosno njezino postavljanje, ovisno o kontekstu. Također se pretpostavlja postojanje sljedećih mogućnosti sustava za upravljanje grafovskim bazama podataka:

- $\mathcal{G}$.*get_nodes*(), koja daje sve vrhove u grafovskoj bazi podataka $\mathcal{G}$,
- $\mathcal{G}$.*get_edges*(), koja daje sve lukove u grafovskoj bazi podataka $\mathcal{G}$,
- *x.get_label*(), koja pronalazi oznaku elementa x,
- *x.get_properties*(), koja daje skup svojstava p_i elementa x u formatu (*name*, *value*),
- *p.get_name*(), koja daje naziv svojstva p,
- *p.get_value*(), koja daje vrijednost svojstva p.

9.3.2 Algoritmi za obradu elemenata i pronalazak čestih vremenskih podgrafova

U ovom su odjeljku algoritamski opisane funkcije za obradu elemenata potpuno vremenski određenog grafa te objedinjujuća funkcija za obavljanje postupka pronalaska čestih vremenskih podgrafova.

Funkcija *process_node* (algoritam 12) obavlja obradu jednog vrha potpuno vremenski određenog grafa i njegovih incidentnih lukova u cilju normiranja njegova vremenskog susjedstva širine δ, preslikavanja tog susjedstva, njegovih svojstava i oznake te konačnog određivanja zamjenske oznake vrha.

Korištenjem opisanih pomoćnih programskih struktura, proces koji funkcija obavlja je pravolinijski. Svi događaji vezani uz vrh i njemu incidentne lukove su dodani u skup događaja, kao referentni događaj se postavlja nastanak vrha, događaji se redaju kronološki te se izbacuju oni koji su više od δ vremenskih jedinica udaljeni od najbližih događaja u poretku. Nakon uklanjanja duplikata se ponovno obrađuju svi događaji i dodjeljuju im se pripadni vremenski indeksi, čime je susjedstvo vrha normirano.

U zajedničku programsku strukturu koju dijele svi elementi grafovske baze podataka, a koja održava mape s nazivima i svim vrijednostima svojstava se nakon normiranja susjedstva dodaju svojstva koja predstavljaju to susjedstvo. Za naziv svojstva koje predstavlja kraj aktivnosti vrha se koristi unaprijed definirano ime "node_end", a za naziv svojstava koje predstavljaju početak odnosno kraj aktivnosti incidentnih lukova se koriste oznake luka sa sufiksom "_start" za početak aktivnosti, odnosno "_end" za kraj aktivnosti. U ovu se strukturu također kao svojstvo dodaje i sama oznaka vrha, s unaprijed definiranim imenom "node_label" te sva druga originalna svojstva vrha sa svojim imenima i vrijednostima. Sve kombinirane vrijednosti indeksa dodanih ili korištenih svojstava koja su dodana u ranijoj obradi elemenata pohranjuju se u skup $NodeProps$, temeljem kojeg se, uz pomoć programske strukture za preslikavanje tih skupova u zamjenske oznake, dolazi do zamjenske oznake za vrh, koju funkcija daje natrag u pozivajući algoritam.

Funkcija *process_edge* (algoritam 13) obavlja obradu jednog luka potpuno vremenski određenog grafa i vrhova kojima je incidentan, u cilju normiranja njegova vremenskog susjedstva, preslikavanja susjedstva, njegovih svojstava i oznake te konačnog određivanja zamjenske oznake luka.

Proces koji ova funkcija obavlja je vrlo sličan onom koji obavlja funkcija *process_node*. Razlike se odnose na broj elemenata koje je za luk potrebno obraditi, s obzirom na to da luk može imati najviše dva vrha kojima je incidentan, dok vrh može imati praktički neograničen broj incidentnih lukova. Vremensko susjedstvo ima vrlo ograničen broj članova i moguće je da nakon uklanjanja svih događaja koji su od drugih, kronološki najbližih događaja udaljeni više od

Algoritam 12 Obrada vrha i incidentnih lukova: **process_node**(v, δ)

Input: vrh v koji se obrađuje
Input: širina vremenskog susjedstva δ
Output: zamjenska oznaka vrha

```
1. {za vrh i sve njegove lukove}
2. for all x ∈ {v, v.get_edges()} do
3.     TemporalAdjacency.add_event(x.t_start)
4.     if ∃x.t_end then
5.         TemporalAdjacency.add_event(x.t_start)
6.     end if
7. end for
8. TemporalAdjacency.set_reference(v.t_start)
9. TemporalAdjacency.set_width(δ)
10. TemporalAdjacency.sort()
11. TemporalAdjacency.remove_distant_neighbors()
12. TemporalAdjacency.remove_duplicates()
13. NodeProps ← ∅
14. {preslikavanje oznake}
15. NodeProps ← PropMapper.get_property("node_label", v.get_label())
16. {preslikavanje svojstava vremenskog susjedstva}
17. if ∃ v.t_end ∧ ∃ i ← TemporalAdjacency.get_index(v.t_end) then
18.     NodeProps ← NodeProps ∪ PropMapper.get_property("node_end", i)
19. end if
20. for all e ∈ v.get_edges() do
21.     if ∃ i ← TemporalAdjacency.get_index(e.t_start) then
22.         NodeProps ← NodeProps ∪ PropMapper.get_property(e.get_label() + "_start", i)
23.     end if
24.     if ∃ e.t_end ∧ ∃i ← TemporalAdjacency.get_index(e.t_end) then
25.         NodeProps ← NodeProps ∪ PropMapper.get_property(e.get_label() + "_end", i)
26.     end if
27. end for
28. {preslikavanje ostalih svojstava vrha}
29. for all p ∈ v.get_properties() | p ∉ {"t_start","t_end"} do
30.     NodeProps ← NodeProps ∪ PropMapper.get_property(p.get_name(), p.get_value())
31. end for
32. return ReplacementLabelMapper.get_label(NodeProps)
```

δ vremenskih jedinica, skup događaja u vremenskom susjedstvu ostane prazan. U tom slučaju luk nije od interesa za daljnju obradu i neće biti uzet u obzir pri pronalasku čestih podgrafova. Istu stvar nije ispravno napraviti i prilikom obrade vrha u funkciji *process_node*, jer u tom trenutku još nije poznato hoće li postojati neki incidentni luk tom vrhu kojem će on biti na rubu vremenskog susjedstva. Ukoliko vrh bez incidentnih lukova bude dio grafa za pronalazak čestih podgrafova, on će na početku procesa GraMi algoritma biti izuzet iz daljnje obrade.

Daljnje razlike postupka obrade luka spram postupka obrade vrha se odnose na činjenicu da je referentni događaj početak aktivnosti luka i na nazive svojstava koja se koriste pri predstav-

Algoritam 13 Obrada luka i vrhova kojima je incidentan: **process_edge**(e, δ)

Input: luk e koji se obrađuje
Input: širina vremenskog susjedstva δ
Output: zamjenska oznaka luka

```
{za luk i njegova dva vrha}
for all x ∈ {e, e.get_nodes()} do
    TemporalAdjacency.add_event(x.t_start)
    if ∃x.t_end then
        TemporalAdjacency.add_event(x.t_start)
    end if
end for
TemporalAdjacency.set_reference(e.t_start)
TemporalAdjacency.set_width(δ)
TemporalAdjacency.sort()
TemporalAdjacency.remove_distant_neighbors()
if TemporalAdjacency.is_empty() then
    return ∅
end if
TemporalAdjacency.remove_duplicates()
EdgeProps ← ∅
{preslikavanje oznake}
EdgeProps ← PropMapper.get_property("edge_label", e.get_label())
{preslikavanje svojstava vremenskog susjedstva}
if ∃ e.t_end ∧ ∃ i ← TemporalAdjacency.get_index(e.t_end) then
    EdgeProps ← EdgeProps ∪ PropMapper.get_property("edge_end", i)
end if
for all v ∈ v.get_nodes() do
    if ∃ i ← TemporalAdjacency.get_index(v.t_start) then
        EdgeProps ← EdgeProps ∪ PropMapper.get_property(v.get_label() + "_start", i)
    end if
    if ∃ v.t_end ∧ ∃ i ← TemporalAdjacency.get_index(e.t_end) then
        EdgeProps ← EdgeProps ∪ PropMapper.get_property(v.get_label() + "_end", i)
    end if
end for
{preslikavanje ostalih svojstava luka}
for all p ∈ e.get_properties() | p ∉ {"t_start","t_end"} do
    EdgeProps ← EdgeProps ∪ PropMapper.get_property(p.get_name(), p.get_value())
end for
return ReplacementLabelMapper.get_label(EdgeProps)
```

ljanju vremenskog susjedstva: za naziv svojstva koje predstavlja kraj aktivnosti luka se koristi unaprijed definirano ime "edge_end", a za naziv svojstava koje predstavljaju početak odnosno kraj aktivnosti vrhova kojima je incidentan se koriste oznake vrha sa sufiksom "_start" za početak aktivnosti, odnosno "_end" za kraj aktivnosti. Također se u dijeljenu strukturu za mape svojstava dodaje oznaka luka s unaprijed definiranim imenom "edge_label" te sva druga originalna svojstva vrha sa svojim imenima i vrijednostima. Sve kombinirane vrijednosti indeksa

dodanih ili korištenih svojstava iz ranije obrade elemenata pohranjuju se u skup *EdgeProps*. Kao i kod obrade vrhova, temeljem ovog skupa se dolazi do zamjenske oznake za luk, koju funkcija daje natrag u pozivajući algoritam.

Funkcija $find\_FTSGs$ (algoritam 14) predstavlja glavni programski prikaz pronalaska čestih vremenskih podgrafova u potpuno vremenski određenom grafu, o čemu najbolje govore ulazni argumenti: grafovska baza podataka koja sadrži potpuno vremenski određeni graf, širina vremenskog susjedstva po kojoj će se određivati dopuštene vremenske udaljenosti među elementima podgrafova te minimalna frekvencija čestog vremenskog podgrafa. Nakon obrade svih vrhova i lukova potpuno vremenski određenog grafa, čime ga se svodi na jednostavni statički graf, nad tim se grafom primjenjuje algoritam GraMi i dobiveni se jednostavni statički česti podgrafovi ponovno predstavljaju elementima koji su vremenski određeni i sadrže svojstva i izvornu oznaku. Izlaz iz ovoga algoritma je skup svih čestih vremenskih podgrafova koji, sa stajališta analize snimljenoga traga relacijske baze podataka, predstavljaju obrasce ponašanja korisnika sustava koji tu relacijsku bazu podataka koristi.

Algoritam 14 Pronalazak čestih vremenskih podgrafova: **find_FTSGs**$(\mathcal{G}, \delta, f)$

Input: grafovska baza podataka $\mathcal{G}$ koja sadrži potpuno vremenski određeni graf
Input: širina vremenskog susjedstva δ
Input: minimalna frekvencija čestog podgrafa f
Output: skup čestih vremenskih podgrafova

1. **for all** $v \in \mathcal{G}.get\_nodes()$ **do**
2. $\quad GraMi.add\_node(v, process\_node(v, \delta))$
3. **end for**
4. **for all** $e \in \mathcal{G}.get\_edges()$ **do**
5. $\quad$ **if** $\exists\ label \leftarrow process\_edge(e, \delta)$ **then**
6. $\quad\quad GraMi.add\_edge(e.get\_nodes, label)$
7. $\quad$ **end if**
8. **end for**
9. **for all** $FSG \in GraMi.find\_FSGs(f)$ **do**
10. $\quad FTSG \leftarrow \emptyset$
11. $\quad$ **for all** $x \in FSG$ **do**
12. $\quad\quad FTSG \leftarrow FTSG \cup ReplacementLabelMapper.expand(x)$
13. $\quad$ **end for**
14. $\quad$ **return** $FTSG$
15. **end for**

Složenost algoritama za obradu vrhova i lukova potpuno vremenski određenog grafa ovisi o broju dodirnih elemenata grafa koji se obrađuju za svaki element te o broju svojstava koje pojedini element ima. U najgorem slučaju je moguće da je vrh povezan sa svim drugim vrhovima u grafu te da ima vrlo velik broj svojstava, pa bi tada složenost algoritma predstavljenoga funkcijom *process_node* bila bliska $O(N)$. Slično vrijedi i za obradu lukova i njihova svojstva. Međutim, treba uzeti u obzir da se tijekom pripreme potpuno vremenski određenog grafa obav-

lja obrada svih njegovih elemenata, a da su prosječan broj svojstava elemenata i prosječan broj lukova po vrhu i dalje znatno manji brojevi od samog broja elemenata u grafu. Promatrajući stoga N kao ukupan broj elemenata potpuno vremenski određenog grafa, složenost dijela algoritma funkcije $find\_FTSGs$ za pripremnu obradu elemenata bi bila $O(N)$. Naknadna obrada pronađenih čestih podgrafova se svodi na linearnu obradu elemenata pronađenih podgrafova, što također znači složenost $O(N)$, i dovodi najveću ukupnu složenost algoritma pronalaska čestih vremenskih podgrafova na razinu složenosti algoritma GraMi, što je, kako je rečeno u poglavlju 4.3.4 $O(N^{n-1})$, gdje je n veličina podgrafa.

9.3.3 Primjer pronalaska čestih vremenskih podgrafova

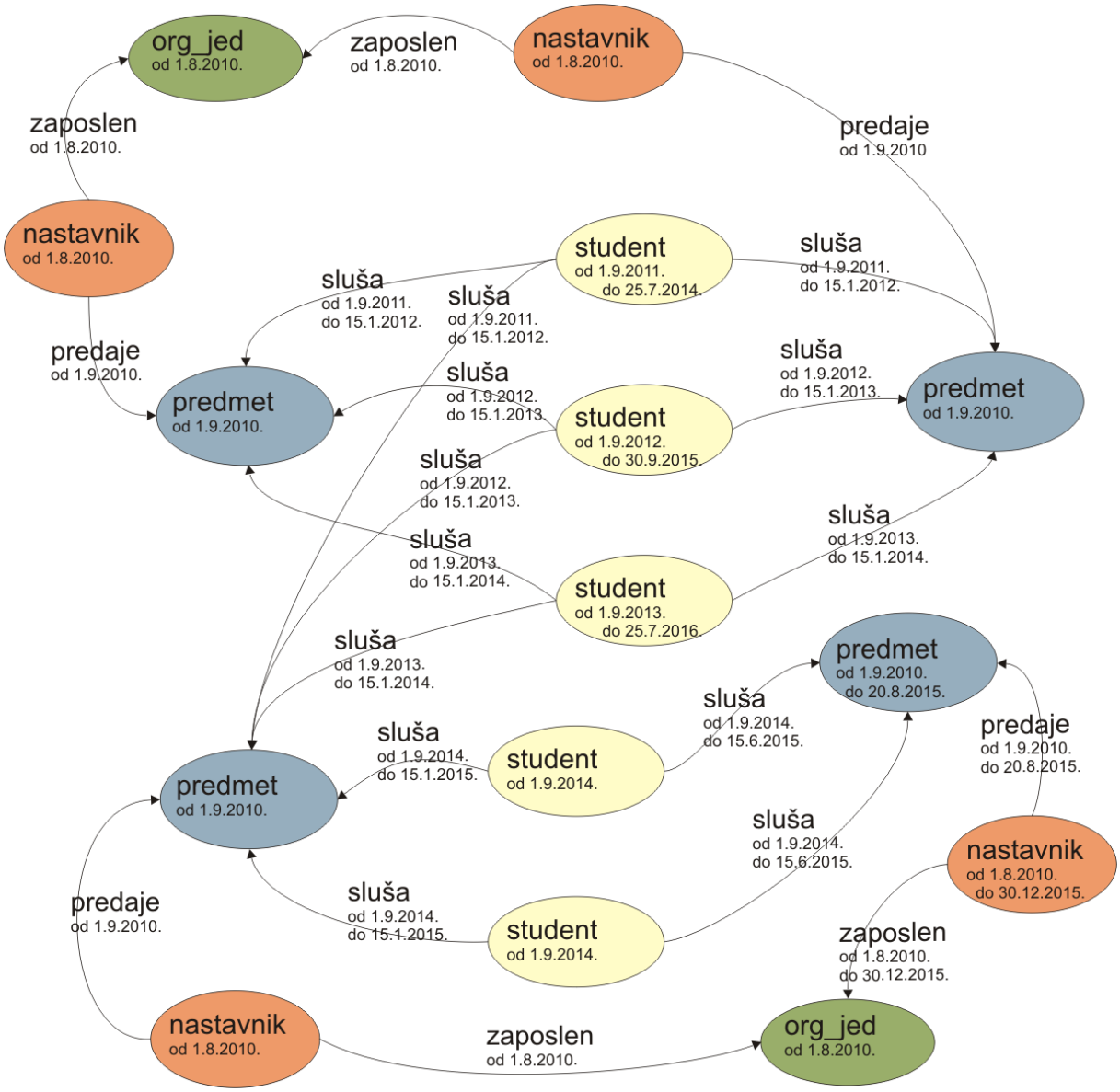
Postupak pronalaska čestih podgrafova demonstrira se na potpuno vremenski određenom grafu koji sadrži podatke o nastavnicima i organizacijskim jedinicama u kojima su zaposleni te predmetima i studentima.

Primjer 12. Potpuno vremenski određeni graf na slici 9.6 sadrži podatke o nekoliko organizacijskih jedinica unutar visokog učilišta, predmeta, nastavnika koji ih izvode i studenata koji ih slušaju. Zbog jednostavnosti prikaza, uklonjeni su svi jedinstveni identifikatori elemenata grafa, kao i njihova svojstva. Prikazane su samo oznake i vremenske odrednice elemenata. Vrhovi koji su prikazani istom bojom su oni koji imaju iste oznake. Vremenske odrednice su u ovom primjeru datumi, i njihova su značenja logički povezana s oznakom elementa:

- za vrhove s oznakom *org_jed*: od kada do kada organizacijska jedinica postoji na učilištu,
- za vrhove s oznakom *nastavnik*: od kada do kada nastavnik radi u nastavnom zvanju,
- za lukove s oznakom *zaposlen*: od kada do kada je nastavnik zaposlen u organizacijskoj jedinici,
- za vrhove s oznakom *predmet*: od kada do kada se predmet predaje na učilištu,
- za lukove s oznakom *predaje*: od kada do kada nastavnik predaje predmet,
- za vrhove s oznakom *student*: od kada do kada student studira na učilištu,
- za lukove s oznakom *sluša*: od kada do kada student sluša predmet.

U primjeru sa slike, može se naslutiti da je učilište osnovano 1.8.2010. godine, prvi semestar je započeo 1.9.2010., studenti slušaju predmete jedan semestar, osim u jednom slučaju dvosemestralnog predmeta; jedan od nastavnika je otišao u mirovinu i predmet koji je predavao se više ne predaje. Prikazani su studenti iz četiri različite generacije i slušaju predmete u četiri različite akademske godine.

Na slici 9.7 su prikazana normirana vremenska susjedstva svih elemenata grafa, uz širinu vremenskog susjedstva $\delta = 6$ mjeseci. Uz oznaku se nalaze vremenski indeksi početka i eventualnog završetka aktivnosti elementa a u samoj blizini i vremenski indeksi koji se odnose na

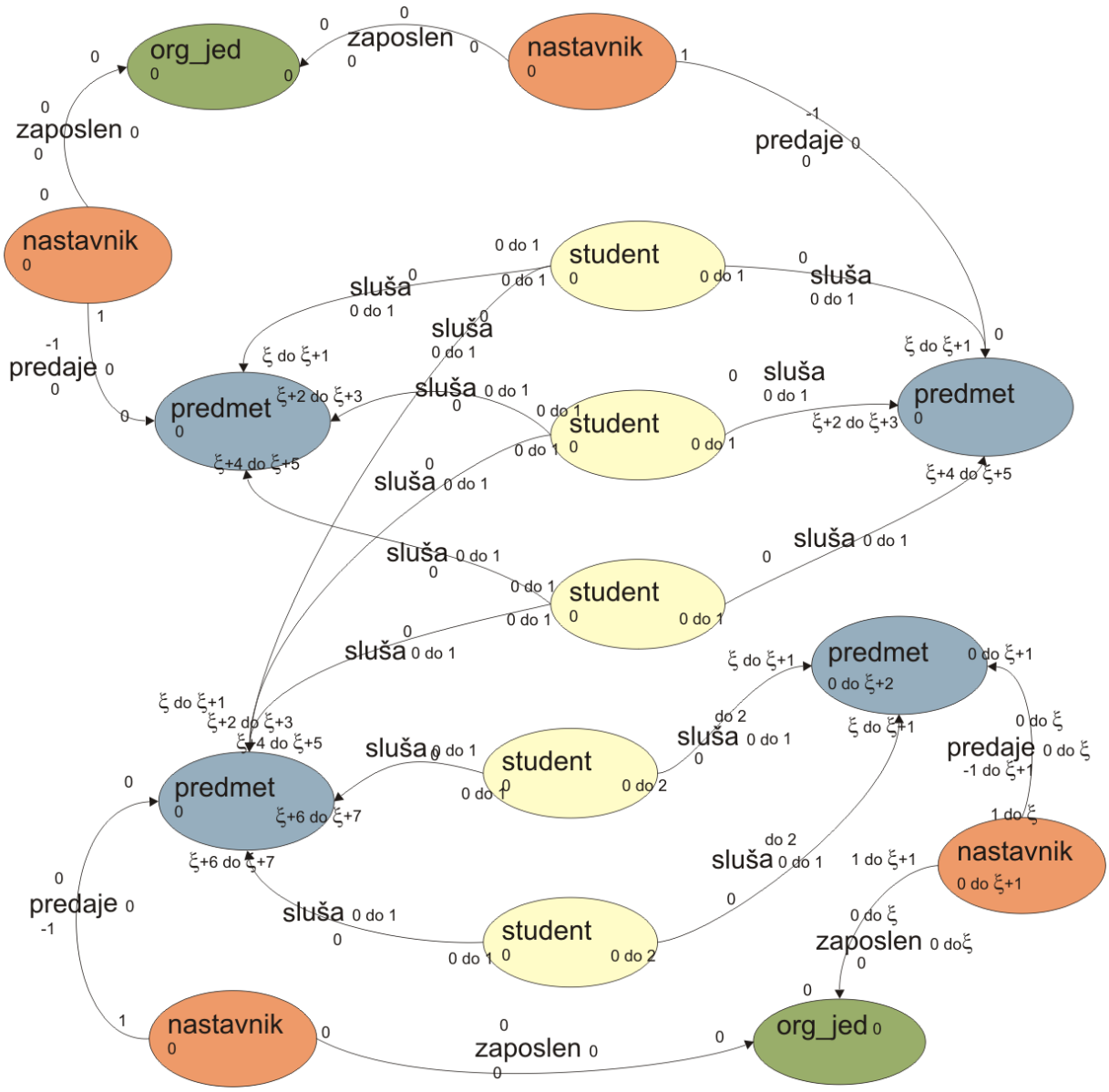


Slika 9.6: Primjer potpuno vremenski određenog grafa

početak i eventualni završetak susjednog elementa. Ukoliko je neka vremenska odrednica bila nevažna za pojedino vremensko susjedstvo, ona je zanemarena i ne nalazi se na ovom prikazu.

Temeljem vremenskih susjedstava elemenata i njihovih oznaka, stvaraju se svojstva koja će biti preslikana kako bi se dobile pripadne zamjenske oznake. Ova su svojstva, zajedno sa svim vrijednostima koja mogu poprimiti i konačnim kombiniranim indeksima svojstva i njegove vrijednosti, prikazana u tablici 9.4. Pri izračunu vrijednosti udaljenih vremenskih indeksa koristi se vrijednost $\xi = 100$, a pri izračunu kombiniranog indeksa podrazumijeva se da neće biti više od 100 različitih vrijednosti svojstva.

U konačnici se, preslikavanjem svih zajedničkih kombiniranih indeksa svojstava i njihovih



Slika 9.7: Normirana susjedstva svih elemenata sa slike 9.6 uz $\delta = 6$ mjeseci

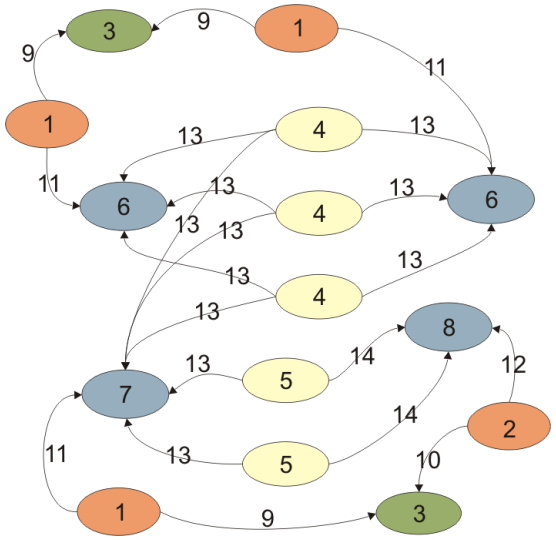
vrijednosti, kako je prikazano u ranijim primjerima, popunjava i mapa zamjenskih oznaka, čime svaki element grafa dobiva zamjensku oznaku temeljem koje će GraMi algoritam pronaći česte podgrafove.

Tablica 9.4: Preslikana svojstva i njihove vrijednosti grafa sa slike 9.7

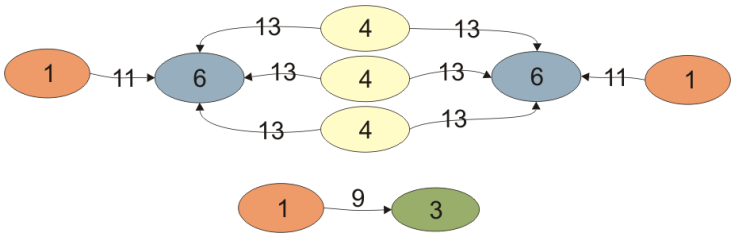
svojstvo	indeks svojstva	vrijednost	indeks svojstva i vrijednosti
node_label	1	nastavnik	101
		org_jed	102
		student	103
		predmet	104
edge_label	2	zaposlen	201
		predaje	202
		sluša	203
node_end	3	101	301
		102	302
edge_end	4	1	401
		100	402
nastavnik_start	5	-1	501
		0	502
nastavnik_end	6	100	601
		101	602
org_jed_start	7	0	701
predmet_start	8	0	801
predmet_end	9	2	901
		100	902
student_start	10	0	1001
zaposlen_start	11	0	1101
		1	1102
zaposlen_end	12	101	1201
predaje_start	13	0	1301
		1	1302
predaje_end	14	100	1401
		101	1402
sluša_start	15	0	1501
		100	1502
		102	1503
		104	1504
		106	1505
sluša_end	16	1	1601
		2	1602
		101	1603
		103	1604
		105	1605
		107	1606

Na slici 9.8 je prikazan zamjenski graf dobiven ovim algoritmom.

S obzirom na malu veličinu grafa u primjeru, najveća frekvencija ponavljanja podgrafa s kojom algoritam GraMi pronalazi česte podgrafove je 2. U tom slučaju, algoritam pronalazi



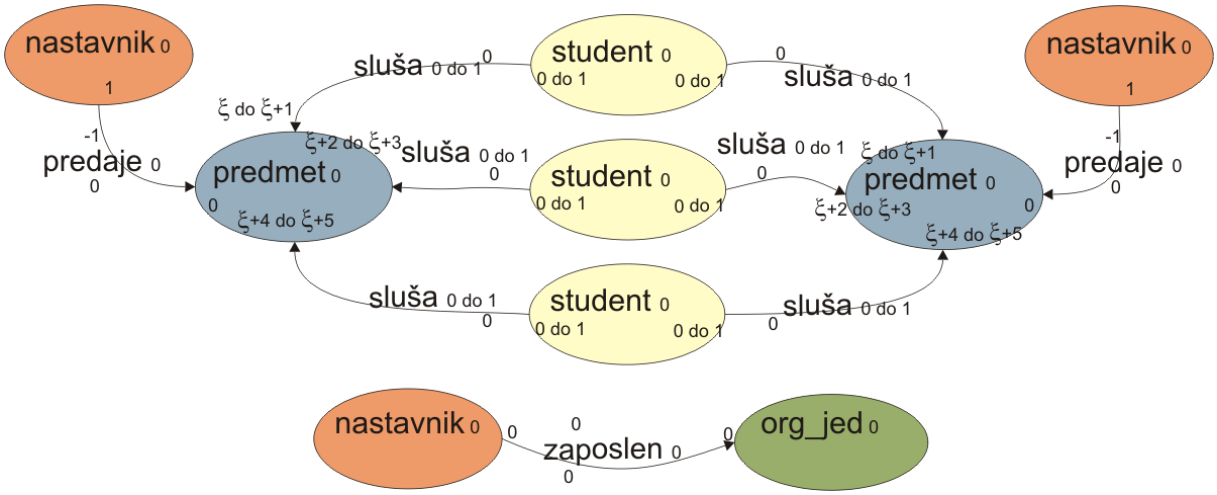
Slika 9.8: Zamjenski graf grafa sa slike 9.6



Slika 9.9: Pronađeni maksimalni česti podgrafovi grafa sa slike 9.8

ukupno 32 česta podgrafa, no samo su dva maksimalna česta podgrafa (slika 9.9), od kojih jedan nema podgrafova, a iz drugoga se može pronaći 30 dodatnih podgrafova koji su također česti u odnosu na početni graf koji se analizira.

Zaključno, na slici 9.10 su prikazani pronađeni česti maksimalni vremenski podgrafovi potpuno vremenski određenog grafa sa slike 9.6. Ovi su podgrafovi prikazani u formatu normiranih vremenskih susjedstava elemenata, koji je najbliži grafu, a na koji se mogu svesti drugi grafovi ili proširenja ovoga grafa. ■



Slika 9.10: Pronađeni maksimalni česti vremenski podgrafovi grafa sa slike 9.6

9.4 Osvrt

U ovom poglavlju prikazan je originalni algoritam za pronalazak čestih vremenskih podgrafova potpuno vremenski određenog grafa. Kao što je slučaj i s ranije prikazanim algoritmima, i ovaj algoritam je moguće koristiti zasebno, kao jedan od osnovnih algoritama za dubinsku analizu potpuno vremenski određenog grafa.

Pronalazak podgrafova u svakom grafu je početni korak u razumijevanju podataka koje predstavlja, a u slučaju vremenskih grafova koji sadrže povijesti podataka, kao što je to potpuno vremenski određeni graf, ti podgrafovi su uzorci u vremenu. Ovisno o domeni koju potpuno vremenski određeni graf prikazuje, ovi uzorci mogu pokazati različite korisne informacije. Na primjer, mogu otkriti načine širenja virusnih bolesti s vremenom inkubacije, načine i vrijeme širenja vijesti među internetskim portalima, ili mogu, kao što će biti slučaj u narednim poglavljima, pokazati uzorke ponašanja korisnika višekorisničkih informacijskih sustava.

Poglavlje 10

Pronalazak odstupanja u potpuno vremenski određenom grafu

Pronalazak anomalija u potpuno vremenski određenom grafu temelji se na opisanom algoritmu za pronalazak sličnih grafova, sličnosti potpisa (poglavlje 5.2, str. 40). U ovom poglavlju opisano je proširenje tog algoritma u cilju usporedbe sličnih vremenskih grafova, koje se zasniva na određivanju funkcije preslikavanja potpuno vremenski određenog grafa u skup težinskih karakteristika. Također je prikazano mjerenje manjih odstupanja između potpuno vremenski određenih grafova u cilju identifikacije odstupanja nekih podgrafova od pronađenih čestih vremenskih podgrafova koje će se u nastavku promatrati kao anomalije unutar grafa.

10.1 Sličnost potpuno vremenski određenih grafova

Uporaba algoritma sličnosti potpisa grafova za usporedbu dvaju grafova svodi se na određivanje težinskih karakteristika koje predstavljaju ta dva grafa. Kako bi se usporedili potpuno vremenski određeni grafovi, potrebno je najprije odrediti što su to njihove karakteristike, a zatim i funkciju $\phi : G_T \rightarrow \{(c_i, w_i)\}$ koja preslikava svaki graf u skup pripadnih težinskih karakteristika.

10.1.1 Karakteristike potpuno vremenski određenog grafa

Ilustracija karakteristika potpuno vremenski određenog grafa bit će prikazana pomoću primjera takvog grafa sa slike 7.1, str. 60.

Definicija 33. *Topološke karakteristike potpuno vremenski određenog grafa* su opisnice postojanja vrhova i lukova unutar grafa i njihove incidencije, pri čemu se vrhovi i lukovi opisuju samo pripadnom oznakom. Topološke karakteristike potpuno vremenski određenog grafa označavaju se s $c_{top,i}$. ■

Topološke karakteristike dakle opisuju sam izgled grafa, incidenciju lukova s vrhovima kroz njihovu osnovnu karakteristiku, oznaku. Topološke karakteristike potpuno vremenski određenog grafa su ujedno i sve karakteristike statičkog grafa s oznakama, jer on nema svojstva niti vrijeme aktivnosti elemenata. Grafu sa slike 7.1 topološke karakteristike bi bile "postoji vrh s oznakom *Osoba*", ili "vrhu s oznakom *Osoba* je incidentan luk s oznakom *Radi*" itd.

Definicija 34. *Vremenske karakteristike potpuno vremenski određenog grafa* su opisnice vremenskih događaja elemenata tog grafa. Vremenske karakteristike potpuno vremenski određenog grafa označavaju se s $c_{time,i}$. ■

Vremenske karakteristike opisuju najvažnije činjenice vezane uz potpuno vremenski određeni graf, početak i završetak aktivnosti svakog elementa. Grafu sa slike 7.1 vremenske karakteristike mogu biti "vrh s oznakom *Osoba* ima početak aktivnosti 11.2.2010." ili "luk s oznakom *Radi* ima trajanje aktivnosti od 1.7.2012. do 1.2.2015." itd.

Definicija 35. *Svojstvene karakteristike potpuno vremenski određenog grafa* su opisnice svojstava elemenata tog grafa i njihovih vrijednosti. Svojstvene karakteristike potpuno vremenski određenog grafa označavaju se s $c_{prop,i}$. ■

Svojstvene karakteristike se odnose na postojanje pojedinog svojstva određenog elementa grafa i vrijednost koju to svojstvo ima. Grafu sa slike svojstvene karakteristike su "vrh s oznakom *Osoba* posjeduje svojstvo *ime* koje ima vrijednost *Ivan*", "luk s oznakom *Radi* posjeduje svojstvo *pozicija* koje ima vrijednost *voditelj prodaje*", itd.

Na ovaj način je moguće opisati potpuno vremenski određeni graf i njegova svojstva kroz niz karakteristika koje će se kasnije koristiti u usporedbi s drugim takvim grafom. Karakteristike ujedno sadrže i nazive oznaka ili svojstava i vrijednosti svojstava i odnose među elementima grafa. Za sami opis karakteristika, koje su u gornjim primjerima navedene opisnim rečenicama, potrebno je definirati računalno primjereniju konvenciju:

- topološka karakteristika koja opisuje postojanje vrha s oznakom *label* u grafu, uz tu informaciju koristi nepromjenjivi predmetak "node_" te ima oblik: "node_*label*",
- topološka karakteristika koja opisuje postojanje luka s oznakom *eLabel* od vrha s oznakom *n*1*Label* do vrha s oznakom *n*2*Label*, uz tu informaciju koristi nepromjenjivi predmetak "edge_" nakon čega spaja sve nazive oznaka prvog vrha, drugog vrha i samog luka koristeći znak "_" te ima oblik "edge_*n*1*Label*_*n*2*Label*_*eLabel*",
- vremenska karakteristika koja opisuje početak aktivnosti vrha s oznakom *label*, dobiva predmetke koji se sastoje od informacije da se radi o vrhu ("node_") i početku aktivnosti ("t_start_") uz samu vrijednost vremena početka aktivnosti elementa *tStamp*, pa ima oblik "node_*label*_t_start_*tStamp*",

- jednako se odnosi na vremensku karakteristiku početka aktivnosti luka (koristi se predmetak "edge_" umjesto "node_") odnosno završetka aktivnosti vrha ili luka (koristi se predmetak "t_end_" umjesto "t_start_"),
- svojstvena karakteristika vrha ili luka s oznakom *label*, čije svojstvo *propName* ima vrijednost *propValue*, za opis te činjenice koristi predmetak koji nosi informaciju o kojoj vrsti elementa grafa se radi ("node_" ili "edge_) i spojenu oznaku elementa, naziv i vrijednost svojstva, pa u slučaju vrha ima oblik "node_*label_propName_propValue*" te analogno za slučaj luka.

Potrebno je naglasiti da se, pri usporedbi čestih vremenskih podgrafova potpuno vremenski određenoga grafa, kao vremenske karakteristike dodaju i druge opisnice koje predstavljaju cijelo normirano vremensko susjedstvo elemenata grafa. Gornji opis vremenskih karakteristika se odnosi primarno na karakterizaciju samog potpuno vremenski određenog grafa, a pri usporedbi vremenskih podgrafova se kao opisnice dodaju odnosi elementa s incidentnim elementima, prilikom čega se elementi imenuju svojim oznakama, a opisnice sadrže i oznaku početka ili završetka te vremenski indeks.

Algoritam 15 Transformacija PVOG u težinske karakteristike: $\phi(G_T)$

Input: G_T PVOG čije se težinske karakteristike traže
Output: skup težinskih karakteristika C

1. $C \leftarrow \emptyset$
2. **for all** $c \in dohvati\_topoloske\_karakteristike(G_T)$ **do**
3. $\quad C \leftarrow C \cup (c, w_{top})$
4. **end for**
5. **for all** $c \in dohvati\_vremenske\_karakteristike(G_T)$ **do**
6. $\quad C \leftarrow C \cup (c, w_{time})$
7. **end for**
8. **for all** $c \in dohvati\_svojstvene\_karakteristike(G_T)$ **do**
9. $\quad C \leftarrow C \cup (c, w_{prop})$
10. **end for**
11. **return** C

Pretpostavi li se postojanje triju funkcija koje pronalaze topološke, vremenske, odnosno svojstvene opisnice potpuno vremenski određenog grafa G_T, onda se funkcija $\phi(G_T)$ može jednostavno opisati algoritmom 15.

Primjer 13. Graf sa slike 7.1 moguće je opisati s ukupno 46 karakteristika, i to:

- 12 topoloških karakteristika, 6 koje opisuju vrhove i 6 koje opisuju lukove,
- 16 vremenskih karakteristika, od kojih se 8 odnosi na vremenske odrednice vrhova i 8 na vremenske odrednice lukova; 4 vremenske karakteristike se odnose na završetak aktivnosti elementa, a ostale na početak aktivnosti
- 18 svojstvenih karakteristika, od kojih se 14 odnosi na svojstva vrhova, a ostale na svojstva lukova.

Primjeri nekih karakteristika korištenjem navedene konvencije:

- "node_Osoba",
- "edge_Osoba_Tvrtka_Radi",
- "edge_Radi_t_start_1.1.2015.",
- "node_Osoba_ime_Ana",
- ...

Za daljnje razmatranje je zanimljivo primijetiti postojanje više identičnih karakteristika, npr. tri luka s oznakom Stanuje imaju početak aktivnosti na isti datum:
"edge_Stanuje_t_start_5.6.2015.". ■

10.1.2 Težine karakteristika potpuno vremenski određenog grafa

Kategorizacija karakteristika potpuno vremenski određenog grafa omogućava pridjeljivanje jedinstvenih težina svakoj od tih kategorija, pa će topološka karakteristika c_{top} imati pridjeljenu težinu w_{top}, vremenska karakteristika c_{time} težinu w_{time}, a svojstvena karakteristika c_{prop} težinu w_{prop}.

U specifičnim primjenama moguće je uvesti dodatne kategorije karakteristika te im pridijeliti pripadne težine. Na primjer, moguće je dodijeliti različite težine vrhovima i lukovima, tako da postojanje vrha u jednom grafu a nedostatak u drugom više utječe na razliku, odnosno sličnost među grafovima, nego postojanje luka u jednom grafu a nedostatak u drugom. Drugi primjer se odnosi na aktivnost elementa - ukoliko je važan završetak aktivnosti elementa, moguće je dodijeliti veću težinu činjenici da neki element više nije aktivan i od kada, nego činjenici kada je postao aktivan te na taj način povećati razliku među grafovima koji se uspoređuju. U sklopu ovog rada, primjena kategorija karakteristika je ograničena samo na navedene tri.

U procesu usporedbe potpuno vremenski određenih grafova, težine pojedinih karakteristika se zbrajaju ili oduzimaju od težina drugih karakteristika. Stoga je zanimljivo promatrati težine kategorija karakteristika relativno jedne u odnosu na druge i odrediti njihove međusobne omjere.

Pri određivanju omjera težina promatra se utjecaj odnosa težine jedne kategorije i težine druge kategorije na mjeru sličnosti grafova. S tim u vezi, mogu se postaviti sljedeća pravila, odnosno smjernice:

1. Utjecaj razlike u topologiji grafa na sličnost dva grafa treba biti jednak utjecaju razlike u vremenskim aktivnostima elemenata grafa. Odnosno, ukoliko postoji razlika u (ne)postojanju jednog elementa grafa, utjecaj te razlike treba biti jednak utjecaju razlike u vremenu aktivnosti jednog elementa grafa.

   Na ovaj način se vremenskoj komponenti potpuno vremenski određenog grafa daje jednako značenje kao i topološkoj, što i jest ispravno, ako se uzme u obzir da npr. u vremenskom grafu dva vrha nisu povezana uvijek, nego samo tijekom aktivnosti luka među

njima.

2. Utjecaj razlike u svojstvima elemenata grafa na sličnost dva grafa treba biti manji od utjecaja razlike u topologiji tih grafova.

Temeljem rečenoga, može se zaključiti da težine topološke i vremenske kategorije mogu biti međusobno jednake, a težina svojstvene kategorije manja, odnosno:

$$w_{top} = w_{time} > w_{prop}. \tag{10.1}$$

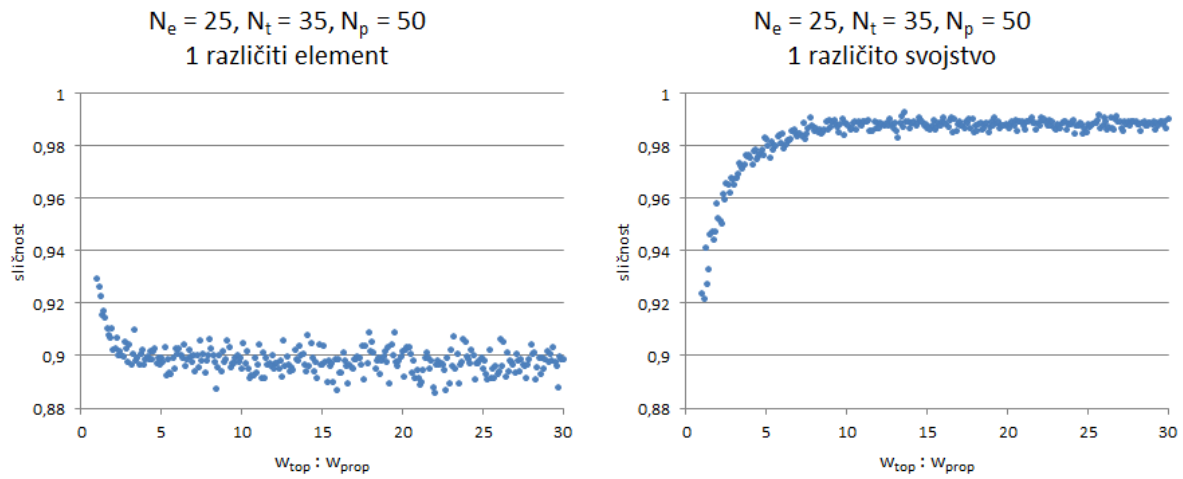
Utjecaj međusobnog omjera težina kategorija se može najjednostavnije promatrati kroz izračune sličnosti dva vrlo slična grafa, onima u kojima je samo jedna karakteristika različita. Ti su izračuni pokazani na slikama 10.1 do 10.4, gdje su na svakom grafikonu navedeni opisi svakog od grafova koji se uspoređuju:

- ukupan broj elemenata pojedinog grafa (N_e), što odgovara ukupnom broju topoloških karakteristika,
- ukupan broj vremenskih odrednica pojedinog grafa (N_t), što odgovara ukupnom broju vremenskih karakteristika,
- ukupan broj svojstava pojedinog grafa (N_p), što odgovara ukupnom broju svojstvenih karakteristika.

Također je na svakom grafikonu navedena razlika između grafova koji se uspoređuju, odnosno radi li se o razlici u topološkoj karakteristici ili razlici u svojstvenoj karakteristici. U ovim promatranjima se smatra da su težine topoloških i vremenskih kategorija jednake, te se promatraju samo topološke i svojstvene razlike, kao i omjeri težina topoloških i svojstvenih karakteristika. Sličnosti su u svim slučajevima bliske potpunoj podudarnosti (odnosno jedinici), s obzirom da se radi o malim razlikama među grafovima. Vrijednosti izračunate sličnosti prikazane su na ordinatama grafikona, a omjeri težina topoloških karakteristika w_{top} i svojstvenih karakteristika w_{prop} na apscisama grafikona.

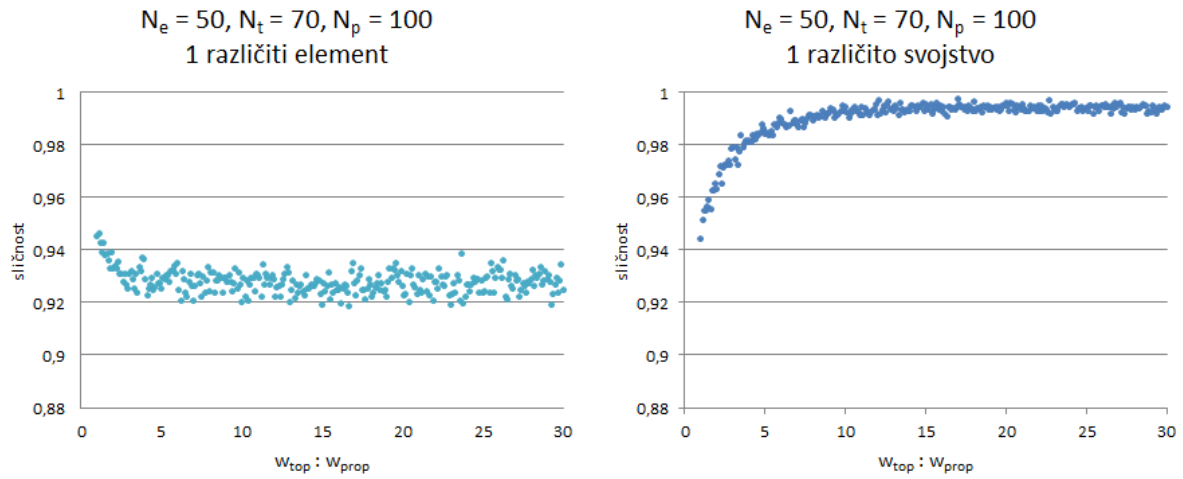
Na pojedinom grafikonu, izračuni su napravljeni za omjere $w_{top} : w_{prop}$ od 1 do 30, u koracima od 0,1. Varijacija iznosa sličnosti u bliskim koracima je posljedica korištenja slučajne funkcije u algoritmu sličnosti potpisa, a smanjuje se povećanjem veličine bit-vektora koji se za pojedine karakteristike koristi. U svim navedenim grafikonima, veličina bit-vektora je $b = 512$.

Slika 10.1 prikazuje usporedbu sličnosti dva grafa koji imaju 25 elemenata, 35 vremenskih odrednica i 50 svojstava. Lijevi grafikon prikazuje situaciju u kojoj su 24 elementa ista, a jedan različit. S povećanjem omjera $w_{top} : w_{prop}$ do otprilike 4:1 izračunata sličnost opada do prosječne vrijednosti 0,897 oko koje daljnjim povećanjem omjera varira. Desni grafikon prikazuje situaciju u kojoj su 49 svojstava i njihovih vrijednosti identična, a jedno različito. U ovom slučaju, s povećanjem omjera $w_{top} : w_{prop}$ do otprilike 7:1 izračunata sličnost raste do prosječne vrijednosti 0,988 oko koje daljnjim povećanjem omjera varira. U drugom slučaju



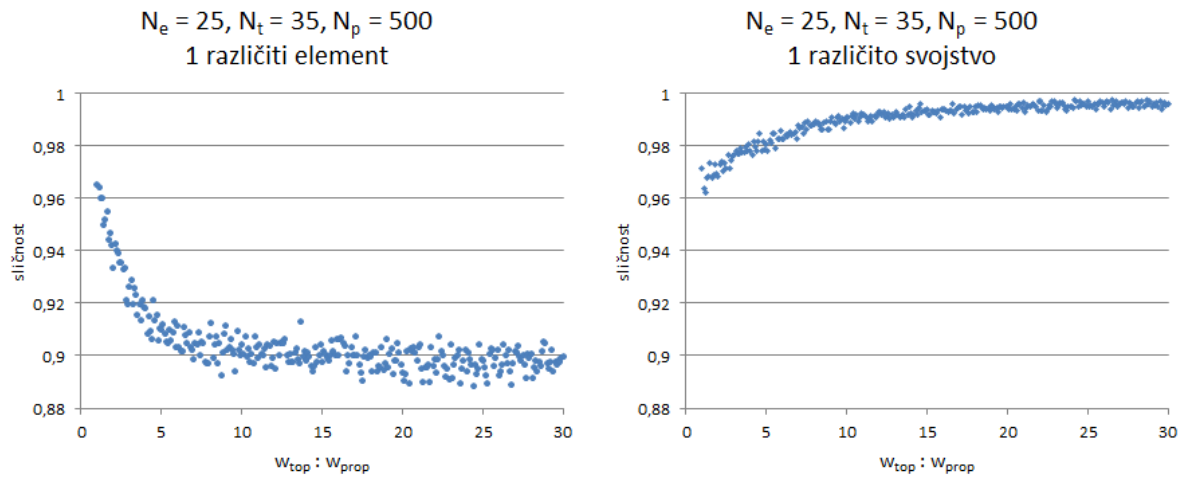
Slika 10.1: Izračuni sličnosti približno jednakih grafova s po 25 elemenata, 35 vremenskih odrednica i 50 svojstava

potrebno je postići nešto veći omjer težina da bi se dostiglo jednaku vrijednost sličnosti, jer je ukupan broj svojstava nešto manji u odnosu na ukupan broj ostalih karakteristika druge težine (elemenata i vremenskih odrednica).



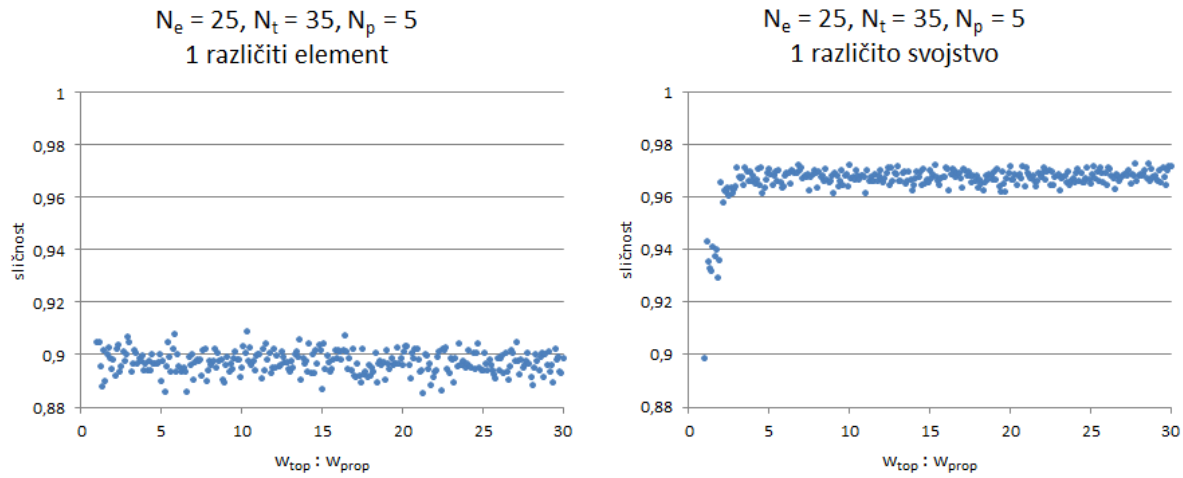
Slika 10.2: Izračuni sličnosti približno jednakih grafova s po 50 elemenata, 70 vremenskih odrednica i 100 svojstava

Na slici 10.2 su prikazane usporedbe izračuna dva grafa koji imaju dvostruko veći broj svih karakteristika nego u prethodnom primjeru: 50 elemenata, 70 vremenskih odrednica i 100 svojstava te su različiti po jedan element grafa (lijevi grafikon) odnosno jedno svojstvo (desni grafikon). Distribucije izračuna sličnosti su vrlo slične onima sa slike 10.1, osim što se do stalnih vrijednosti dolazi s nešto većim omjerom težina, s obzirom na veći broj karakteristika, i što su te stalne vrijednosti nešto veće, jer su razlike među grafovima sumjerljivo manje. U slučaju različitih elemenata, izračuni sličnosti vrijednosti variraju oko 0,927, a u slučaju različitih svojstava oko 0,994.



Slika 10.3: Izračuni sličnosti približno jednakih grafova s većim brojem svojstava po elementu

Na slici 10.3 su prikazane usporedbe izračuna dva grafa koji imaju znatno veći broj svojstava po elementu nego u prethodnim primjerima: 25 elemenata, 35 vremenskih odrednica i 500 svojstava. Također se razlikuju samo po jednom elementu grafa (lijevi grafikon) odnosno jednom svojstvu (desni grafikon). Distribucije izračuna sličnosti su također slične onima iz prethodnih primjera, uz činjenicu da se do stalnih vrijednosti dolazi s još većim omjerom težina. U slučaju različitih elemenata, izračuni sličnosti vrijednosti variraju oko 0,898, a u slučaju različitih svojstava oko 0,996.



Slika 10.4: Izračuni sličnosti približno jednakih grafova s malim brojem svojstava po elementu

Na slici 10.4 su prikazane usporedbe izračuna dva grafa koji imaju mali broj svojstava po elementu: 25 elemenata, 35 vremenskih odrednica i ukupno 5 svojstava. U slučaju razlike samo u jednom elementu grafa (lijevi grafikon) ne primjećuju se varijacije s povećanjem omjera težina, što je i očekivano s obzirom na mali broj svojstava. Također, u slučaju razlike grafova samo u jednom svojstvu (desni grafikon) se puno brže dolazi do vrijednosti oko koje izraču-

nate sličnosti variraju. U slučaju različitih elemenata, izračuni sličnosti vrijednosti variraju oko 0,897, a u slučaju različitih svojstava oko 0,968.

Temeljem prikazanih izračuna sličnosti može se zaključiti kako omjer težina različitih kategorija ima utjecaja do određene mjere, što naravno ovisi o ukupnom broju karakteristika i njihovom međusobnom omjeru. U daljnjim izračunima sličnosti u ovom radu, uzet će se u obzir očekivane veličine grafova koji se uspoređuju te odrediti omjer $w_{top} : w_{prop}$ kao 10:1, uz $w_{time} = w_{top}$ u usporedbi potpuno vremenski određenih grafova, dok će u usporedbi čestih vremenskih podgrafova ta jednakost biti prilagođena.

10.1.3 Usporedba potpuno vremenski određenih grafova

Pri usporedbi dva potpuno vremenski određena grafa $G_{T,1}$ i $G_{T,2}$, najprije je potrebno, kako je rečeno u prethodnim poglavljima, odrediti njihove težinske karakteristike. Svaka karakteristika pri tom dobiva težinu ovisno o kategoriji kojoj pripada. U konačnici, stvaraju se dva pripadna skupa težinskih karakteristika: $C_1 = \{(c_{1,i}, w_{1,i})\}$ i $C_2 = \{(c_{2,j}, w_{2,j})\}$. Ukoliko je neka karakteristika u pojedinom skupu prisutna više puta, skup se reducira tako da sadrži tu karakteristiku samo jednom, ali s težinom pomnoženom s njezinom originalnom pojavnosti. Karakteristika "edge_Stanuje_t_start_5.6.2015." iz primjera 13 bi se pojavila jednom i imala bi težinu $3w_{time}$.

Sljedeći korak u pripremi podataka za usporedbu jest osiguravanje da skupovi težinskih karakteristika C_1 i C_2 sadrže iste karakteristike. Ukoliko se neka karakteristika nalazi u jednom skupu, a ne nalazi u drugom, ona se dodaje u taj drugi skup, ali s negativnom težinom. Ovaj se postupak provodi na svim karakteristikama u oba skupa. Nakon njega skupovi C_1 i C_2 sadrže iste karakteristike s odgovarajućim težinama, i dalje se nad njima primjenjuje algoritam sličnosti potpisa - za sve jedinstvene karakteristike se stvaraju nasumični bit-vektori, te se ovisno o vrijednosti pozicije bit vektora zbrajaju ili oduzimaju težine pojedinih karakteristika, dok se na kraju ne dobije ukupan broj svih težina, koji se opet pretvara u završni bit-vektor. Dva dobivena bit-vektora se uspoređuju spram razlika na pojedinim pozicijama (funkcija *Hamming*), i na kraju temeljem toga izračunava sličnost dvaju grafova, u rasponu od 0 do 1, uključivo.

Rezultat usporedbe ovako pripremljenih skupova težinskih karakteristika C_1 i C_2 je ujedno i rezultat usporedbe potpuno vremenski određenih grafova $G_{T,1}$ i $G_{T,2}$ algoritmom sličnosti potpisa.

Funkcija *compare_pvog*, prezentirana algoritmom 16, obavlja opisani postupak. Za njezin se opis koristi pomoćna programska struktura, razred *SigSim*, koja objedinjuje implementaciju algoritma sličnosti potpisa (funkcija *get_similarity*), deduplikaciju i zbrajanje težina karakteristika koje se ponavljaju višekratno unutar jednog skupa (procedura *deduplicate_chars*), i dodavanje nedostajućih karakteristika jednog skupa u drugi s negativnim težinama (funkcija *add_missing_chars*).

Sve operacije unutar funkcije *compare_pvog* su linearne složenosti te se složenost ovog

Algoritam 16 Usporedba potpuno vremenski određenih grafova: **compare_pvog**($G_{T,1}, G_{T,2}$)

Input: $G_{T,1}$ prvi PVOG za usporedbu
Input: $G_{T,2}$ drugi PVOG za usporedbu
Output: mjera sličnosti $G_{T,1}$ i $G_{T,1}$
1. $C_1 \leftarrow \phi(G_{T,1})$
2. $C_2 \leftarrow \phi(G_{T,2})$
3. $SigSim.deduplicate\_chars(C_1)$
4. $SigSim.deduplicate\_chars(C_2)$
5. $SigSim.add\_missing\_chars(C_1, C_2)$
6. **return** $SigSim.get\_similarity(C_1, C_2)$

algoritma može promatrati kroz ukupan broj karakteristika za usporedbu n i može se reći da je ona $O(n)$. Uz ovo, vrijedi i diskusija o složenosti algoritma sličnosti potpisa (poglavlje 5.2).

10.2 Odstupanja od potpuno vremenski određenog grafa

Algoritam sličnosti potpisa daje mjeru sličnosti dvaju grafova, na temelju koje je moguće donositi daljnje odluke. Jedna od tih je smatra li se graf sličnim drugome grafu ili ne, odnosno, do koje mjere su dva grafa slična, da bi ih se moglo smatrati približno jednakima, ali ne potpuno jednakima.

Definicija 36. Grafovi G_1 i G_2 su *približno jednaki potpuno vremenski određeni grafovi* ako je sličnost njihovih potpisa veća ili jednaka minimalnoj zadanoj sličnosti i manja od 1, odnosno:

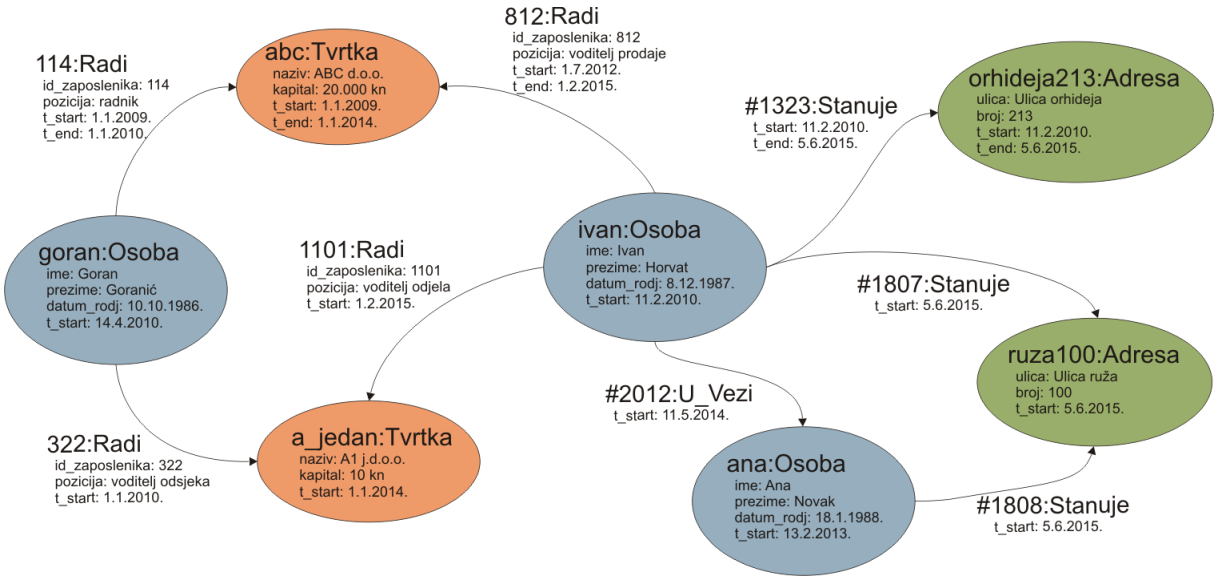
$$sim_{SS}(G_1, G_2) \in [\sigma, 1) \;\rightarrow\; G_1 \simeq G_2 \tag{10.2}$$

pri čemu je σ parametar minimalne sličnosti, odnosno maksimalnog odstupanja. ■

S obzirom na to da različite razlike u grafovima različito djeluju na mjeru sličnosti, da je mjera sličnosti u određenom stupnju neprecizna jer se oslanja na vjerojatnost (koristi funkcije slučajne varijable), parametar maksimalnog odstupanja je potrebno odrediti empirijski, ovisno o pojedinom potpuno vremenski određenom grafu i njegovoj veličini.

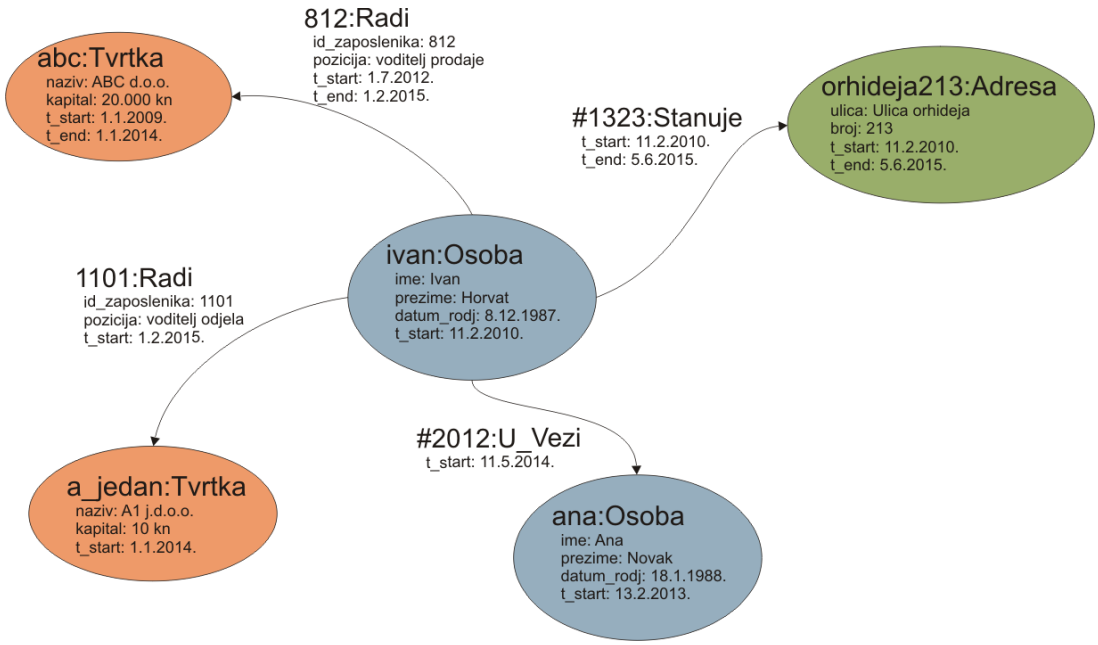
Tipične razlike između podgrafova primarno zanimljivih u ovom istraživanju odnose se na vremenske i topološke varijacije između dva uspoređivana podgrafa. Stoga će neke od takvih varijacija biti prikazane nešto detaljnije:

- jedan od grafova sadrži dodatni vrh i incidentne lukove u odnosu na drugi, pri čemu svaki od ovih elemenata sadrži nekoliko svojstava
- oba grafa su topološki jednaka, ali se dio elemenata u jednom grafu vremenski razlikuje od drugoga
- grafovi sadrže jednake vrhove, ali su lukovi drugačije raspoređeni.



Slika 10.5: Graf sa slike 7.1 uz dodani vrh i dva luka sa svojstvima

Ove će varijacije i njihova odstupanja biti demonstrirana u odnosu na graf prikazan na slici 7.1 (str. 60). Na slici 10.5 je graf koji u odnosu na taj posjeduje dodatne elemente: vrh koji predstavlja još jednu osobu koja je također radila u istim tvrtkama te su mu incidentna dva luka. U usporedbi ova dva grafa stvara se ukupno 50 težinskih karakteristika, 39 za graf sa slike 7.1 i dodatnih 11 za graf sa slike 10.5. Tih dodatnih 11 se dodaje u prvi graf s negativnim težinama, te se temeljem toga izračunava sličnost. Prosječna sličnost ova dva grafa u deset ponovljenih izračuna je 0,803.

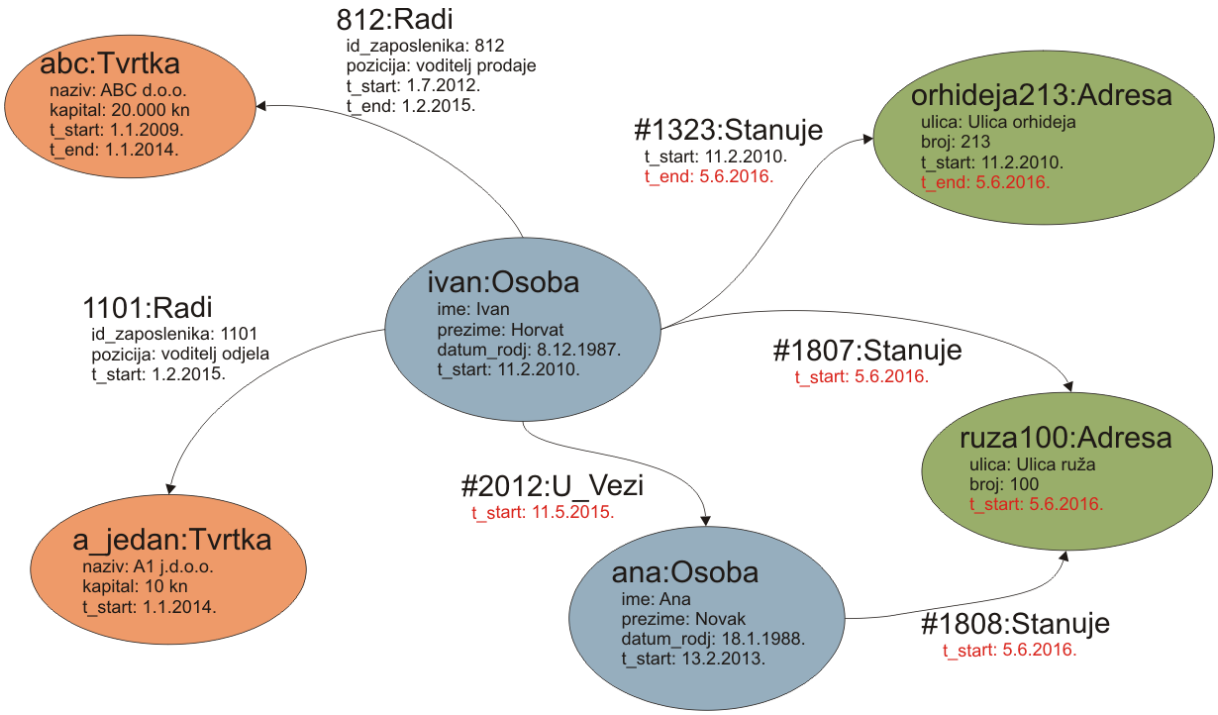


Slika 10.6: Graf sa slike 7.1 bez jednog vrha i dva luka sa svojstvima

Na slici 10.6 je graf koji u odnosu na graf prikazan na slici 7.1 posjeduje manjak elemenata: vrh koji predstavlja adresu i dva incidentna luka. U usporedbi ova dva grafa stvara se ukupno

39 težinskih karakteristika, 39 za graf sa slike 7.1, od kojih 4 nedostaju grafu sa slike 10.6 pa sudjeluju s negativnim težinama, a dvije i dalje postoje, ali u drugom grafu sudjeluju sa smanjenim težinama. Prosječna sličnost ova dva grafa u deset ponovljenih izračuna je 0,730.

Oba prethodna primjera se odnose na isti slučaj - jedan od grafova sadrži dodatni vrh i incidentne lukove u odnosu na drugi, pri čemu svaki od ovih elemenata sadrži nekoliko svojstava. No u drugom slučaju zbog manjeg broja elemenata za usporedbu razlika više dolazi do izražaja. Može se zaključiti da je prosječna veličina grafova koji se uspoređuju jedan od kriterija za određivanje parametra maksimalnog odstupanja σ.

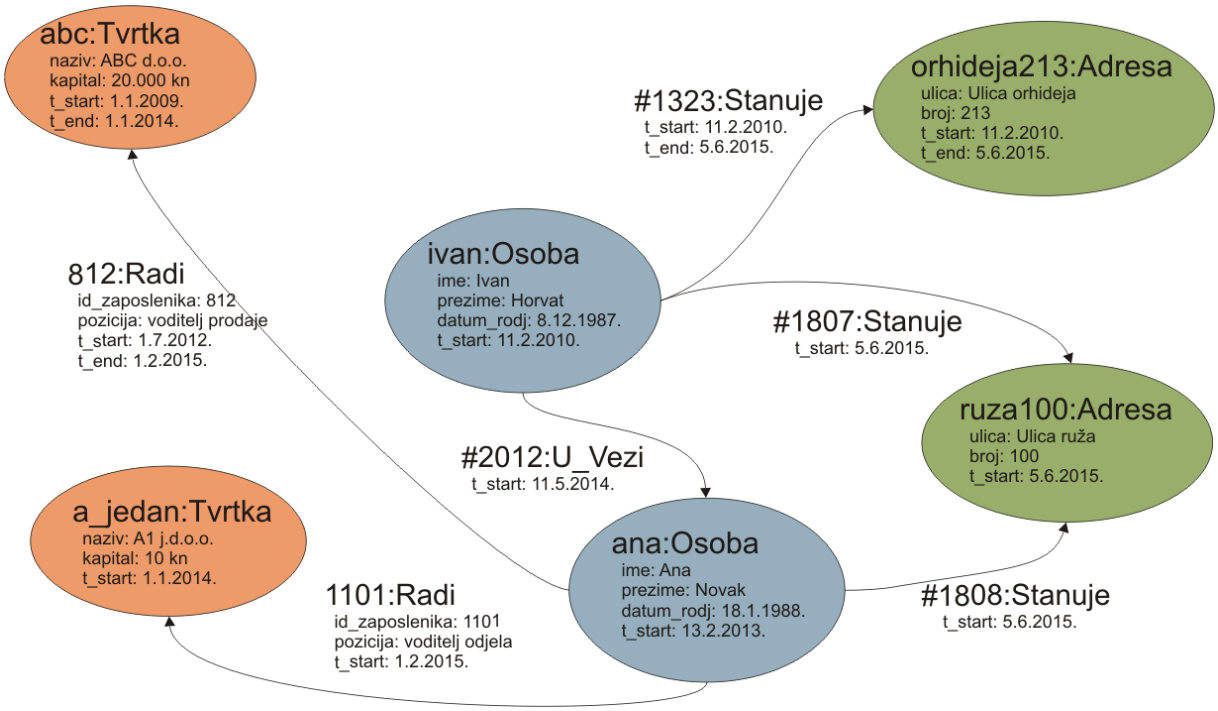


Slika 10.7: Graf sa slike 7.1 uz šest različitih vremenskih odrednica (označene crvenom bojom)

Na slici 10.7 je graf koji je topološki jednak grafu sa slike 7.1, no razlikuje se u 6 vremenskih svojstava (datumi od kad su osobe u vezi i počinju stanovati na istoj adresi su godinu dana kasniji; na grafu radi lakšeg razlikovanja označeni crvenom bojom). U usporedbi ova dva grafa stvaraju se ukupno 44 težinske karakteristike, po 39 za svaki graf, s time da svaki ima i 5 razlikovnih koje u njegovim karakteristikama sudjeluju s negativnim težinama (radi se o 5 karakteristika, a ne 6, zato što su dvije karakteristike jednake, pa su njihove težine zbrojene). Prosječna sličnost ova dva grafa u deset ponovljenih izračuna je 0,638. Iako se ova sličnost može tumačiti premalom za dva topološki jednaka grafa, treba uzeti u obzir razlikovanje u skoro 40% vremenskih odrednica. Nameće se zaključak da su pri usporedbi vremenskih grafova važniji relativni vremenski odnosi od apsolutnih, a ta će se činjenica primijeniti pri usporedbi vremenskih podgrafova s normiranim vremenskim susjedstvima elemenata.

I na kraju, na slici 10.8 je graf koji sadrži jednake vrhove, lukove, njihove vremenske odrednice i svojstva, no topološki se malo razlikuje od grafa sa slike 7.1, i to u činjenici da su dva

luka incidentna drugome vrhu koji predstavlja osobu. Uzevši u obzir način stvaranja karakteristika, ova će dva grafa biti identična, jer je u karakteristikama grafa sadržana informacija o incidentnosti lukova vrhovima spram njihovih oznaka, a sva ostala svojstva se odnose samo na pojedini element grafa. U tom smislu je ovaj algoritam određivanja sličnosti grafova sličan opisanom algoritmu određivanja sličnosti dokumenata (poglavlje 5.2), jer zanemaruje dio lokacijskih informacija. Ipak, dio tih informacija se i dalje uzima u obzir, navedenom činjenicom o incidentnosti lukova vrhovima spram oznaka, a još više topoloških informacija se može uzeti u obzir usporedbom grafova s normiranim vremenskim susjedstvima elemenata, jer oni mogu sadržavati vremenske karakteristike koje se odnose na sve (relevantne) susjedne elemente grafa i tako ga pobliže i topološki opisati. No, i dalje ostaje činjenica da se radi o algoritmu sličnosti potpisa grafa, pa je taj dio informacija namjerno zanemaren.



Slika 10.8: Graf sa slike 7.1 uz različitu incidentnost lukova

Navedenim je primjerima usporedbe vremenskih grafova dan naglasak na razumijevanje algoritma sličnosti potpisa vremenskih grafova i izračuna sličnosti koji daje.

10.3 Osvrt

U ovom poglavlju prikazan je originalni algoritam za izračun sličnosti, odnosno odstupanja jednog potpuno vremenski određenog grafa od drugoga. Algoritam daje mogućnost kvantificiranja sličnosti između dva vremenska grafa u kojima svi elementi imaju vremenske odrednice početka aktivnosti i mogućeg završetka aktivnosti. Ovaj se algoritam može primijeniti na vremenske podgrafove, od kojih neki ranije mogu biti identificirani kao česti vremenski podgrafovi.

Ako se uz to definira i mjera sličnosti, onda je riječ o originalnom algoritmu za pronalazak odstupanja od čestih vremenskih podgrafova potpuno vremenski određenog grafa.

Prikazani algoritam za izračun sličnosti grafova je primjenjiv na sve vrste grafova. U sklopu metode za pronalazak mogućih složenih zlouporaba informacijskih sustava, ovaj će se algoritam koristiti za izračun sličnosti vremenskih grafova s normiranim vremenskim susjedstvima elemenata te kao takav, za pronalazak odstupanja od čestih vremenskih podgrafova.

Poglavlje 11

Metoda za pronalazak mogućih zlouporaba u informacijskim sustavima

Metoda za pronalazak mogućih zlouporaba u informacijskim sustava se temelji na algoritmima i tehnikama prikazanim u prethodnim poglavljima. To su konverzija snimljenog traga relacijskih baza podataka informacijskog sustava u potpuno vremenski određeni graf (poglavlje 8), pronalazak čestih podgrafova tog grafa (poglavlje 9) te pronalazak razine odstupanja, odnosno sličnosti između potpuno vremenski određenih grafova (poglavlje 10). U ovom poglavlju će se ti segmenti ujediniti u zajednički postupak te će biti pokazan i način identifikacije mogućih zlouporaba informacijskih sustava.

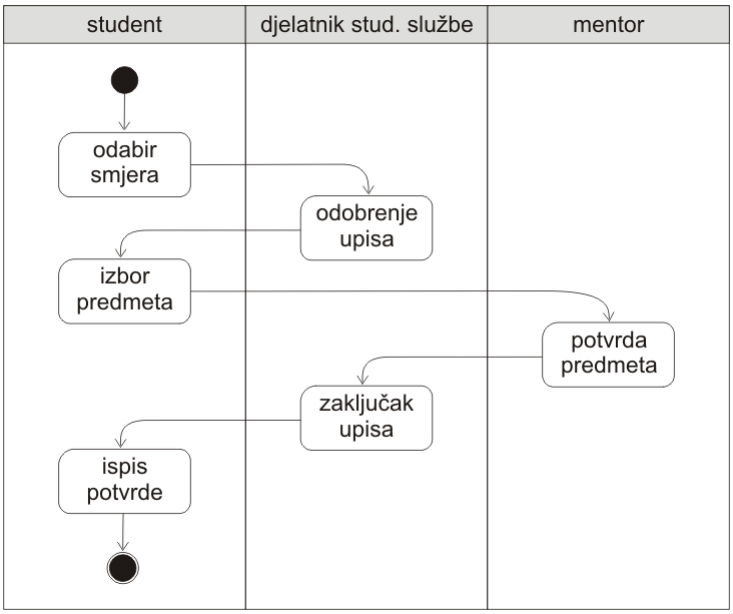
11.1 Pronalazak obrazaca ponašanja korisnika informacijskog sustava

Osnovna pretpostavka na kojoj se temelji metoda pronalaska mogućih zlouporaba informacijskog sustava jest da se taj informacijski sustav temelji na relacijskim bazama podataka čiji je trag, odnosno povijest podataka, snimljen i dostupan za analizu. Iz ovog se traga pronalaze određeni obrasci ponašanja skupina korisnika informacijskog sustava i na taj način ustanovljuje ono što predstavlja uobičajeno korištenje sustava, odnosno uobičajene načine kako neke grupe podataka nastaju u sustavu i bivaju ažurirane tijekom vremena. Iako je te tijekove moguće poznavanjem sustava koji je informatiziran predvidjeti, često u praksi mogu postojati različiti načini korištenja istih dijelova sustava, koji ovise o lokalnim pravilima koje određena skupina korisnika primjenjuje. Također, treba naglasiti da je bit ove metode primjena na višekorisničkom, suradničkom okruženju te da se pri analizi stvaraju zajednički obrasci ponašanja više korisnika, iako je ona primjenjiva i na sustave s malim brojem korisnika koji međusobno ne surađuju obavljajući radnje nad istim skupovima podataka.

Primjer 14. Na primjeru višekorisničkog informacijskog sustava namijenjenoga podršci procesima visokoškolskog obrazovanja, mogu se izdvojiti razni obrasci ponašanja korisnika. Ukoliko se, na primjer, promatra proces upisa studenata u akademsku godinu, tada bi mogući obrazac ponašanja bio:

1. student pristupa sustavu i odabire smjer koji će upisati,
2. djelatnik studentske službe odobrava upis smjera studentu,
3. student bira izborne predmete s odobrenog smjera,
4. mentor studenta potvrđuje izborne predmete koje je student upisao,
5. djelatnik studentske službe zaključuje upis studenta,
6. student iz sustava ispisuje potvrdu o upisu.

Sve navedene točke se obavljaju u točno određenom redoslijedu te na taj način oblikuju ponašanje ove skupine korisnika. Ovaj se uzorak može prikazati dijagramom aktivnosti, pri čemu treba imati na umu da se svi ovi događaji obavljaju tijekom jednoga do najviše nekoliko dana (slika 11.1).

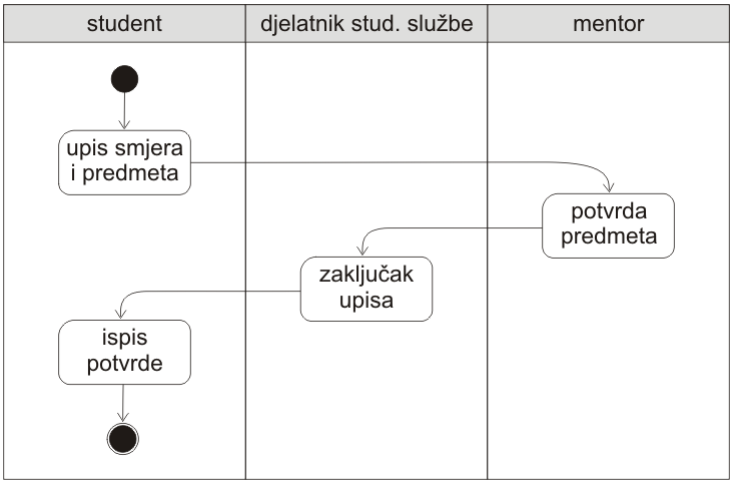


Slika 11.1: Primjer obrasca upisa godine studenta na nekoj visokoškolskoj ustanovi

Navedeni obrazac ponašanja korisnika pak ne mora biti jedinstven da bi se nad istim podacima obavile iste akcije. Na primjer, na nekoj drugoj ustanovi se upisi viših godina mogu obavljati po drugačijim unaprijed određenim pravilima, pa bi mogući obrazac ponašanja mogao biti:

1. student pristupa sustavu i odabire smjer koji će upisati te upisuje izborne predmete,
2. mentor studenta potvrđuje izborne predmete koje je student upisao,
3. djelatnik studentske službe zaključuje upis studenta,
4. student iz sustava ispisuje potvrdu o upisu.

Ovaj se postupak također može prikazati dijagramom aktivnosti (slika 11.2).



Slika 11.2: Primjer obrasca upisa godine studenta na drugoj visokoškolskoj ustanovi

Iako postoji odstupanje od ova dva obrasca ponašanja različitih grupa, a radi se o istoj skupini podataka, ne može se zaključiti da je jedan od tih obrazaca upitan, što govori u prilog tvrdnji da je potrebno ispitati broj odstupanja i ograničiti ga na mjeru u kojoj se odstupanja mogu smatrati izuzecima. ■

11.1.1 Od snimljenog traga do obrazaca ponašanja korisnika

Kako bi se iz snimljenog traga relacijskih baza podataka dobili obrasci ponašanja korisnika, najprije je potrebno iz tog traga stvoriti potpuno vremenski određeni graf. Iako je moguće koristiti sav snimljeni trag koji je na raspolaganju, radi brzine i jednostavnijeg analiziranja, moguće je i odrediti dio snimljenog traga koji će se analizirati, odnosno koji će biti početna točka za stvaranje potpuno vremenski određenog grafa. Ključni dio procesa stvaranja potpuno vremenski određenog grafa iz snimljenog traga opisan je funkcijom *populate_pvog* opisanom u algoritmu 11 i kao takav je integralni dio ove metode.

Obrasci ponašanja korisnika koje je iz potpuno vremenski određenog grafa moguće dobiti su zapravo pronađeni česti vremenski podgrafovi tog grafa, kako je to opisano u poglavlju 9.3. Pri provođenju ovog postupka važno je ispravno odrediti širinu vremenskog susjedstva δ ali i koja će sve svojstva elemenata koristiti prilikom postupka pronalaska čestih podgrafova. Ovisno o broju i raspršenosti vrijednosti svojstava elemenata grafa, uključivanje svih svojstava u postupak može implicirati previše zamjenskih oznaka, jer se elementi mogu razlikovati u velikom broju (možda i nevažnih, ili manje važnih) svojstava. Stoga može biti oportuno takva svojstva zanemariti. Također, kako bi se pronašli obrasci ponašanja korisnika općenito, a ne pojedinih grupa korisnika, potrebno je eliminirati i korisničko ime koje je vezano uz element potpuno vremenski određenog grafa. U cilju pronalaska strožih obrazaca ponašanja, moguće je zamijeniti stvarna korisnička imena s ulogom koju korisnik ima u sustavu (npr. ako je Ivan Ivić nastavnik, onda se korisničko ime "iivic" koje je vezano elemente grafa može zamijeniti ulogom

"nastavnik", slično kao što je prezentirano u primjeru 14) te tu ulogu dalje koristiti kao svojstvo elementa. Bez toga, pronađeni će obrasci opisivati operacije nad podacima i njihove vremenske odnose što, ovisno o primjeni, može biti dovoljno. Funkcija $find\_FTSGs$, opisana algoritmom 14, je također integralni dio ove metode, a prilagodbe navedene u ovom odjeljku je moguće implementirati u pripremnim funkcijama koje obrađuju pojedine elemente grafa *process_node* i *process_edge*.

Nakon obavljanja funkcije $find\_FTSGs$ nad potpuno vremenski određenim grafom popunjenim iz snimljenog traga, pronađeni česti vremenski podgrafovi predstavljaju obrasce ponašanja korisnika informacijskog sustava čiji je trag snimljen.

11.2 Pronalazak dovoljno čestih odstupanja od obrazaca ponašanja korisnika informacijskog sustava

Postupak identifikacije anomalija (odstupanja) također se temelji na potpuno vremenski određenom grafu nastalom iz snimljenog traga relacijske baze podataka. Podaci koji se obrađuju u ovom koraku su također normirana vremenska susjedstva elemenata, pri čemu se najprije normiraju susjedstva svakog pojedinog elementa i njemu se kao svojstva dodaju vremenske odrednice susjedstva. Širina vremenskog susjedstva je u ovom slučaju jednaka širini vremenskog susjedstva temeljem koje je napravljen pronalazak čestih vremenskih podgrafova. Zatim se elementi koji su u susjedstvu promatraju kao podgrafovi, odnosno, iz ovako normiranog potpuno vremenski određenog grafa se uklanjaju oni elementi koji više nisu povezani s drugima unutar ijednog vremenskog susjedstva. Ovako nastali podgrafovi se promatraju kao uzorci ponašanja skupine korisnika u promatranom vremenu (širini susjedstva) i kandidati su za anomalije. Kandidati za anomalije se uspoređuju s pronađenim obrascima ponašanja korisnika općenito, odnosno pronađenim čestim vremenskim podgrafovima i to tako da se svaki kandidat uspoređuje sa svakim čestim vremenskim podgrafom. Ako je kandidat približno jednak makar jednom čestom vremenskom podgrafu, bilježi se kao anomalija. Anomalija pri tom može biti približno jednaka većem broju čestih vremenskih podgrafova. Pri usporedbi se ignoriraju korisnička imena, jer ona nisu sadržana niti u čestim vremenskim podgrafovima, ali se, u slučaju sličnosti, zapisuje skup svih različitih korisničkih imena koja su sudjelovala u akcijama unutar promatranog kandidata za anomaliju. Temeljem rečenoga, moguće je definirati anomaliju.

Definicija 37. *Anomalija A* je vremenski podgraf potpuno vremenski određenog grafa približno jednak (definicija 36) makar jednom čestom vremenskom podgrafu tog grafa i opisuje se kao:

$$A = (v, FTSG, U) \tag{11.1}$$

gdje je:

- v referentni vrh anomalije,
- $FTSG$ skup čestih vremenskih podgrafova kojima je anomalija približno jednaka,
- U skup svih različitih korisničkih imena koja su sudjelovala u akcijama nad elementima anomalije. ■

Nakon što su prethodnom analizom utvrđeni obrasci ponašanja grupa korisnika u zajedničkim poslovima nad informacijskim sustavom i pronađene anomalije od tih obrazaca, moguće je pronaći one grupe korisnika čiji načini korištenja istih dijelova sustava odudaraju od onoga što se smatra uobičajenim obrascem. Ukoliko je broj tih odudaranja značajniji, moguće je da se radi organiziranoj zlouporabi sustava. S druge strane, prevelik broj identičnih odudaranja može značiti da se jednostavno radi o drugačijem načinu korištenja sustava. Kako bi se ovakvi slučajevi unaprijed uklonili iz promatranja, potrebno je odrediti i donju i gornju granicu pojavljivanja pojedinog odstupanja za istu grupu korisnika, da bi se to odstupanje promatralo kao moguća zlouporaba.

Definicija 38. Broj pojavljivanja odstupanja od istog obrasca koje izvodi ista grupa korisnika naziva se *frekvencija anomalije* i označava s f_A, a unaprijed definirane granice te frekvencije da bi se odstupanja promatrala kao zlouporaba se nazivaju *donja i gornja granica frekvencije anomalije* i označavaju s α_l i α_u. ■

Definicija 39. *Moguća zlouporaba sustava* **A** je skup anomalija A_i od istog obrasca ponašanja korisnika (odnosno, čestog vremenskog podgrafa $FTSG_M$) koje ista grupa korisnika U_M ponavlja frekvencijom f_A koja je unutar unaprijed definiranih granica α_l i α_u. Dakle, sve anomalije u skupu se odnose na istu grupu korisnika U_M i u skupu čestih vremenskih podgrafova kojima je anomalija približno jednaka sadrže isti obrazac ponašanja korisnika $FTSG_M$. ■

Informacija o referentnom vrhu svake anomalije, v_i, olakšava naknadne analize anomalija. To je pristupna točka za pregled anomalija unutar potpuno vremenski određenog grafa, a također omogućava i povratnu identifikaciju dijela snimljenog traga koji sadržava informacije koje je potrebno analizirati.

11.2.1 Opis postupka

U svrhu prezentacije postupka pronalaska dovoljno čestih odstupanja od obrazaca ponašanja korisnika koristi se pomoćni razred *AnomalyFinder*. Ovaj razred sadrži skup svih anomalija, pri čemu se za svaku od njih evidentiraju informacije o referentnom vrhu, podgrafovima s kojima su približno jednake i skup korisničkih imena korisnika koji su sudjelovali u akcijama nad elementima anomalije. Osim toga, pretpostavlja se da ovaj razred sadrži i sljedeće funkcije i procedure:

- *process_pvog_for_candidates*($\mathcal{G}, \delta$), koja obavlja normiranje elemenata potpuno vremenski određenog grafa $\mathcal{G}$ na vremensko susjedstvo širine δ i priprema elemente za nastavak postupka pronalaska anomalija (opisana detaljnije u nastavku)
- *purge_edges*(), koja iz normiranog potpuno vremenski određenog grafa uklanja lukove čija su vremenska susjedstva prazna
- *purge_nodes*(), koja iz normiranog potpuno vremenski određenog grafa uklanja vrhove čija su vremenska susjedstva prazna i koji nemaju incidentnih lukova
- *add_anomaly*($Candidate, FTSG$), dodaje informacije o referentnom vrhu i skupu korisničkih imena koja sudjeluju u akcijama nad elementima anomalije iz grafa *Candidate* te podgrafu $FTSG$ kojem je anomalija približno jednaka u skup svih anomalija,
- *get_FTSGs*(), koja daje skup svih različitih vremenskih podgrafova kojima su anomalije približno jednake,
- *get_users*(), koja daje skup svih različitih skupova korisničkih imena koja sudjeluju u akcijama nad elementima anomalija
- *get_anomalies*($FTSG, users$), koja daje sve različite anomalije koje su približno jednake vremenskom podgrafu $FTSG$ pri čemu su korisnici koji sudjeluju u akcijama na elementima anomalije upravo svi iz skupa *users*.

U pripremnoj fazi analize potpuno vremenski određenog grafa, kako bi se pronašli mogući kandidati za anomalije, provodi se normiranje na vremensko susjedstvo za sve elemente grafa. To se normiranje svodi na uporabu pomoćnih postupaka opisanih u razredu *TemporalAdjacency* (poglavlje 9.3.1, str. 87) kako je već opisano u funkcijama *process_node* (algoritam 12, str. 90) i *process_edge* (algoritam 13, str. 91). No razlika ovog postupka u odnosu na te dvije funkcije je ta da se, nakon što su svi relevantni vremenski događaji susjedstva indeksirani, za njih stvaraju dodatna svojstva elementa te oni na taj način sudjeluju u daljnjoj analizi. Rezultat ovog procesa su dodatna svojstva elemenata u kandidatima za anomalije, koji se promatraju kao vremenska svojstva. Radi jednostavnijeg opisa cijelog postupka pronalaska anomalija, ovaj dio postupka je objedinjen u spomenutu funkciju *process_pvog_for_candidates*($\mathcal{G}, \delta$) pomoćnog razreda *AnomalyFinder*.

Nakon što su elementi grafa ovako pripremljeni za pronalazak kandidata za anomalije, potrebno je iz grafa ukloniti one čija su vremenska susjedstva prazna. Najprije se obavlja uklanjanje lukova, a nakon toga i vrhova, s time da se tada uklanjaju i oni vrhovi koji više nemaju incidentnih lukova. Ovaj dio obavljaju spomenute procedure *purge_edges* i *purge_nodes* pomoćnog razreda *AnomalyFinder*.

Vremenska susjedstva koja su ostala u grafu predstavljaju kandidate za anomalije koje treba uspoređivati s čestim vremenskim podgrafovima. Moguće su razne strategije određivanja veličine kandidata za anomalije koji će se uspoređivati. Na primjer, moguće je postaviti određeni minimum broja elemenata koje kandidat mora imati, pa uspoređivati sve varijacije tolikog broja

elemenata unutar cijelog vremenskog susjedstva. Drugi pristup je uspoređivati cijelo vremensko susjedstvo određene širine. Uz pretpostavku da je odabrana širina vremenskog susjedstva dovoljno mala da obuhvati ograničeni broj elemenata (uz činjenicu da mora biti i dovoljno velika da samo susjedstvo ima značenje), korištenje ovog pristupa djeluje opravdano s obzirom da je jednostavniji, pa se on primjenjuje u sklopu ovog istraživanja. U ovom kontekstu, vremensko susjedstvo koje obuhvaća sve incidentne elemente grafa koji su unutar zadane širine susjedstva δ se naziva maksimalno vremensko susjedstvo.

Pronalazak svih maksimalnih vremenskih susjedstava započinje uzimanjem jednog vrha grafa slučajnim odabirom te prolazak njegovim susjedstvom po dubini ili širini, dok se ne dohvati cijeli podgraf koji predstavlja jedno maksimalno vremensko susjedstvo. Taj se podgraf zapisuje kao kandidat za anomaliju, svi njegovi vrhovi se eliminiraju iz daljnjeg odabira za ostale kandidate i postupak se ponavlja dok god je moguć pronalazak novih maksimalnih vremenskih susjedstava.

Postupak pronalaska kandidata za anomalije opisan je funkcijom $find\_candidates$ u algoritmu 17. Funkcija koristi rekurzivnu funkciju $dfs\_add(v, G)$ za prolazak po elementima grafa po dubini počevši od vrha v, koja vrh, njemu incidentne lukove i susjedne vrhove dopunjava u graf G. Nakon izvršavanja funkcije $find\_candidates$, skup $Candidates$ sadržava sve moguće kandidate za anomalije širine vremenskog susjedstva δ unutar potpuno vremenski određenog grafa.

Algoritam 17 Pronalazak kandidata za anomalije: **find_candidates**($\mathcal{G}, \delta$)

Input: grafovska baza podataka $\mathcal{G}$ koja sadrži potpuno vremenski određeni graf
Input: širina vremenskog susjedstva δ
Output: kandidati za anomalije

```
1.  G ← AnomalyFinder.process_pvog_for_candidates(G, δ)
2.  G ← AnomalyFinder.purge_edges()
3.  G ← AnomalyFinder.purge_nodes()
4.  Candidates ← ∅
5.  for all v ∈ G.get_nodes() do
6.     if v ∉ UsedNodes then
7.        Candidate ← ∅
8.        Candidate ← dfs_add(v, Candidate)
9.        if Candidate.get_edges() ≠ ∅ then
10.          Candidates ← Candidates ∪ Candidate
11.       end if
12.       UsedNodes ← Candidate.get_nodes()
13.    end if
14. end for
15. return Candidates
```

Jednom kad su kandidati za anomalije pronađeni, za svaki je moguće izdvojiti težinske karakteristike, kako bi se mogla računati sličnost. Pri tom, u funkciji ϕ treba uzeti u obzir da se

svojstva koja su dodana za opis vremenskog susjedstva elementa treba promatrati kao vremenske, a ne svojstvene karakteristike elementa. Osim toga, s obzirom da vremenska odrednica elementa koja je dio vremenskog susjedstva u tom susjedstvu zapravo sudjeluje dva puta (jednom kao dio samog elementa na koji se odnosi, drugi put kao dio susjedstva incidentnog luka ili susjedstva vrha kojem je luk incidentan), težine vremenskih karakteristika bi trebale biti upola manje od težina topoloških karakteristika, odnosno:

$$w_{time} = 0,5 \times w_{top} \tag{11.2}$$

Postupak usporedbe kandidata za anomalije (sadržanih u skupu $Candidates$) s čestim vremenskim podgrafovima (sadržanih u skupu $FTSGs$) s maksimalnim odstupanjem σ obavlja procedura $find\_anomalies(Candidates, FTSGs)$ opisana algoritmom 18. Svaki se kandidat uspoređuje sa svakim čestim vremenskim podgrafom i, ukoliko su približno jednaki, taj se kandidat evidentira kao anomalija u razredu *AnomalyFinder*. Usporedba se obavlja funkcijom *compare_pvog*, opisanom algoritmom 16. Ovu funkciju za usporedbu potpuno vremenski određenih grafova je moguće bez prilagodbi primijeniti na usporedbu normiranih vremenskih susjedstava (što i kandidati za anomaliju i česti vremenski podgrafovi jesu), jer su ona oblik potpuno vremenski određenih grafova (uz jedinu razliku što mogu sadržavati nešto veći broj vremenskih odrednica) te se njihove karakteristike stvaraju i sudjeluju u usporedbi grafova na identičan način.

Algoritam 18 Pronalazak anomalija: **find_anomalies**($Candidates, FTSGs, \sigma$)

Input: podgrafovi kandidati za anomalije $Candidates$
Input: česti vremenski podgrafovi $FTSGs$
Input: maksimalno odstupanje sličnosti σ
Output: kreiran skup anomalija

```
1. for all Candidate ∈ Candidates do
2.    for all FTSG ∈ FTSGs do
3.       sim ← compare_pvog(Candidate, FTSG)
4.       if sim ≥ σ ∧ sim < 1 then
5.          AnomalyFinder.add_anomaly(Candidate, FTSG)
6.       end if
7.    end for
8. end for
```

Nakon izvršavanja funkcije $find\_anomalies$ razred *AnomalyFinder* sadrži informacije o svim pronađenim anomalijama unutar potpuno vremenski određenog grafa te ih je potrebno prebrojati sumarno po čestom vremenskom grafu od kojeg odstupaju i skupini korisnika angažiranih na elementima anomalije i zaključiti koje od tih kombinacija, sukladno unaprijed određenim parametrima donje i gornje granice frekvencije anomalije, predstavljaju moguće zlo-

uporabe sustava. Na kraju, moguće je i objedinjeno opisati cijeli proces koji počinje analizom snimljenog traga a završava dojavom mogućih zlouporaba, funkcijom $find\_misuses$ (algoritam 19). Ulazni i izlazni parametri ove funkcije, koja predstavlja metodu pronalaska mogućih zlouporaba, najbolje opisuju ulaz i parametriziranje potrebno za izvođenje cijele metode, kao i što je moguće iz nje dobiti. Ulazni parametri su:

- povijesna baza podataka **_r** sa snimljenim tragom radne baze podataka **r**,
- širina vremenskog susjedstva δ koja će se koristiti u pronalasku obrazaca ponašanja korisnika i anomalija,
- minimalna frekvencija čestog podgrafa f, odnosno pojavnost istovjetnog ponašanja korisnika da bi se ono ustanovilo kao obrazac,
- maksimalno odstupanje vremenskih podgrafova σ, da bi se moglo odrediti moguće anomalije
- donja granica frekvencije anomalije α_l, iznad koje mora biti pojavnost slične anomalije određenom obrascu ponašanja korisnika iste skupine korisnika da bi se smatralo mogućom zlouporabom,
- gornja granica frekvencije anomalije α_u, ispod koje mora biti pojavnost slične anomalije određenom obrascu ponašanja korisnika iste skupine korisnika da bi se smatralo mogućom zlouporabom,

dok je izlazni parametar skup mogućih zlouporaba sustava.

Algoritam 19 Pronalazak mogućih zlouporaba: **find_misuses**($\_\mathbf{r}, \delta, f, \sigma, \alpha_l, \alpha_u$)

Input: povijesna baza podataka **_r**
Input: širina vremenskog susjedstva δ
Input: minimalna frekvencija čestog podgrafa f
Input: maksimalno odstupanje približno jednakih vremenskih podgrafova σ
Input: donja granica frekvencije anomalije α_l
Input: gornja granica frekvencije anomalije α_u
Output: skupovi mogućih zlouporaba sustava

1. $\mathcal{G} \leftarrow populate\_pvog(\_\mathbf{r})$
2. $FTSGs \leftarrow find\_FTSGs(\mathcal{G}, \delta, f)$
3. $Candidates \leftarrow find\_candidates(\mathcal{G}, \delta)$
4. $find\_anomalies(Candidates, FTSGs, \sigma)$
5. **for all** $FTSG \in AnomalyFinder.get\_FTSGs()$ **do**
6. **for all** $UserSet \in AnomalyFinder.get\_users()$ **do**
7. $count \leftarrow |AnomalyFinder.get\_anomalies(FTSG, UserSet)|$
8. **if** $count \geq \alpha_l \wedge count \leq \alpha_u$ **then**
9. **return** $AnomalyFinder.get\_anomalies(FTSG, UserSet)$
10. **end if**
11. **end for**
12. **end for**

Nakon obavljanja funkcije *find_misuses* i pronalaska obrazaca ponašanja korisnika analizom potpuno vremenski određenog grafa dobivenog iz snimljenog traga relacijske baze informacijskog sustava i pronalaska anomalija od njih, izdvajaju se oni skupovi anomalija nad istim obrascem i s istim skupom korisnika koji se pojavljuju frekvencijom koja je unutar zadanih granica. Ova povratna informacija predstavlja moguće zlouporabe sustava.

11.3 Analiza mogućih zlouporaba sustava

Nakon obavljanja cijele opisane metode, tek započinje rad na analizi mogućih zlouporaba sustava $\mathbf{A}_i$. S obzirom da se može raditi o složenim višekorisničkim akcijama nad većom količinom podataka, a bez prekoračenja ovlasti ijednog korisnika, ovaj dio postupka treba provesti osoba ili tim koji su zaduženi za održavanje i sigurnost sustava, i koji poznaju podatke i procese u sustavu.

Svaku moguću zlouporabu je potrebno analizirati, anomaliju po anomaliju, uzevši u obzir razlike u odnosu na pronađene uobičajene obrasce ponašanja korisnika te ih cjelovito prihvatiti ili odbaciti. Dok jedna anomalija ne mora ukazivati na zlouporabu, više njih može. Pojedine anomalije se u potpuno vremenski određenom grafu mogu pregledati pomoću referentnog vrha anomalije. Iz tog je vrha moguće dobiti podatke pomoću kojih se može doći do snimljenog traga, ukoliko je potrebno analizirati pojedini događaj na razini relacijskih baza podataka. Također, taj se događaj može povezati i s drugim zapisima koji nisu bili dio ove analize, kao što su na primjer dnevnički zapisi (eng. *logs*) pojedinih dijelova informacijskog sustava i sl.

Iako analiza ovakvih mogućih zlouporaba nije jednostavan niti vremenski zanemariv postupak, potencijalni pronalasci stvarnih složenih zlouporaba sustava opravdavaju uložene resurse.

11.4 Osvrt

U ovom poglavlju su algoritmi ranije prikazani u poglavljima 8, 9 i 10 upotrijebljeni za specifičnu namjenu te objedinjeni u originalnu metodu za otkrivanje mogućih sigurnosnih prijetnji na osnovu odstupanja od čestih vremenskih podgrafova potpuno vremenski određenog grafa. Metoda omogućava pronalazak mogućih složenih unutarnjih zlouporaba u višekorisničkim sustavima, u kojima organizirano sudjeluje više korisnika a niti jedan od njih ne prekoračuje dodijeljene mu ovlasti, što je područje unutarnjih prijetnji sustavima koje trenutno nije dovoljno istraženo.

Ako se u obzir uzme drugačiji izvor podataka koji se koristi u analizi (u ovom slučaju snimljeni trag relacijske baze podataka) i drugačija definicija moguće zlouporabe (u ovom slučaju su to anomalije s ponavljajućim skupom korisnika sustava), tada metoda može biti primjenjiva i u drugim raznim slučajevima. Na primjer, izvor podataka temeljem kojeg se stvara potpuno vre-

menski određeni graf može biti društvena mreža, pojedina anomalija veliki broj pratitelja neke stranice ili osobe u kratkom vremenu, a moguća zlouporaba skup takvih anomalija s obzirom na ishodišta pratitelja (npr. IP adrese ili države iz kojih pratitelji dolaze). Algoritmi za detekciju takvih prijevara na društvenim mrežama su također predmet nekih aktualnih istraživanja (npr. [143]).

Poglavlje 12

Implementacija i testiranje

U ovom poglavlju su prikazani načini implementacije i testiranja algoritama i metoda opisanih u poglavljima 6 do 11, odnosno od samog stvaranja grafovske baze podataka na temelju sadržaja iz relacijske baze podataka pa do metode za pronalazak mogućih prijetnji u informacijskim sustavima temeljem snimljenog traga.

Implementacijska i testna okolina sastoje se od sustava za upravljanje relacijskim bazama podataka IBM Informix [144] te sustava za upravljanje grafovskim bazama podataka Neo4j [42]. Algoritam za pronalazak čestih podgrafova GraMi je izvorno implementiran u programskom jeziku Java [117], pa su i njegova proširenja za rad s potpuno vremenski određenim grafovima kao i pronalazak anomalija također implementirani u programskom jeziku Java. Konačno, pronađene moguće zlouporabe se prikazuju u obliku upita nad Neo4j bazom podataka, kako bi se detaljnije mogle proučiti u vizualnom okruženju tog sustava.

Kako se stvaranje potpuno vremenski određenog grafa iz snimljenog traga relacijske baze podataka temelji na postupku stvaranja grafovske baze podataka iz relacijske baze podataka, tako je najprije prikazana implementacija tog osnovnog postupka (potpoglavlje 12.1), za čije testiranje je bilo dostupno nekoliko relacijskih baza podataka različitih veličina, ukupnog zauzeća diskovnog prostora od 100 Mb do 30 Gb. Prikazana je i ranije objavljena usporedba rezultata ovog algoritma s drugim pristupima slične namjene.

Nakon toga su prikazani postupci koji zajedno čine metodu pronalaska mogućih zlouporaba informacijskih sustava: stvaranje potpuno vremenski određenog grafa temeljem snimljenog traga (potpoglavlje 12.2), pronalazak čestih vremenskih podgrafova (potpoglavlje 12.3) i pronalazak mogućih zlouporaba temeljem ispitivanja sličnosti podgrafova (potpoglavlje 12.4). Na osnovu uvida u snimljeni trag relacijske baze podataka Informacijskog sustava visokih učilišta (ISVU) [145] izgenerirani su testni podaci koji služe kao osnova za analizu. Ovi podaci dakle zamjenjuju rezultat redovitog korištenja sustava, a odnose se samo na jedno visoko učilište i na dio relacija vezanih uz evidenciju ispita. Prilikom generiranja testnih podataka je, u cilju izbjegavanja svake slučajnosti, korišten skup osobnih identifikatora koji u stvarnom sustavu ne

mogu postojati. Zatim su u taj testni skup podataka dodani i iskonstruirani podaci koji se ponavljaju više puta, a predstavljaju uzorak složene zlouporabe sustava koja je svojedobno dobila medijsku pozornost [146, 147, 148]. U toj zlouporabi sudjeluju student, nastavnik, djelatnik studentske referade i posrednik, koji nije korisnik sustava. Uzorak se može opisati na sljedeći način:

1. Student stupa u kontakt s posrednikom od kojeg dobiva instrukcije o ispitnom roku iz predmeta na koji se treba prijaviti, i s kojim dogovara nagradu
2. Student prijavljuje ispit za dogovoreni ispitni rok
3. Administrator ispita na zavodu/katedri na kojem se ispit održava raspoređuje studenta kod jednog od predmetnih nastavnika po slučajnom odabiru
4. Ukoliko student nije raspoređen kod dogovorenog nastavnika, djelatnik studentske referade koji ima ovlasti mijenjati rasporede za ispite, to i čini, kako bi student došao na ispit kod tog nastavnika
5. student izlazi na ispit i dobiva pozitivnu ocjenu koju nastavnik unosi u sustav

Prvi se korak odvija izvan sustava, što i jest odlika ovakvog načina zlouporabe, dok su ostali koraci vidljivi u sustavu. Uspoređujući te korake s postupkom prijave i izlaska na ispit koji je uobičajen na tom visokom učilištu, zapravo je jedini neuobičajeni korak onaj u kojem sudjeluje djelatnik studentske referade. Iako to može značiti da ovaj djelatnik samo popravlja podatke ili pomaže u rješavanju nesporazuma, ukoliko se uzorak u kojem se uvijek promijeni nastavnik za usmeni ispit na točno određenog nastavnika, to uvijek obavi isti djelatnik studentske referade i student uvijek dobije prolaznu ocjenu, ponovi dovoljno česti broj puta (frekvencija anomalije), to može značiti da se radi o mogućoj zlouporabi koju je potrebno istražiti.

U snimljenom tragu, vrijeme obavljanja operacija je zapisano kao (realni) broj sekundi od određenom datuma u povijesti (točnije, 1.1.1970) što će biti vidljivo i u idućim primjerima.

12.1 Popunjavanje grafovske baze podataka sadržajem relacijske baze podataka

Prilikom implementacije programa za stvaranje sadržaja grafovske baze podataka iz sadržaja relacijske baze podataka, uzete su u obzir činjenice da se može raditi o velikoj količini podataka i da su za izvoz i uvoz podataka zasigurno najučinkovitiji za to namijenjeni alati samih proizvođača sustava za upravljanje (relacijskim, odnosno grafovskim) bazama podataka. Stoga je ovaj dio sustava implementiran unutar pohranjenih procedura relacijske baze podataka, koje generiraju naredbe u SQL jeziku za izvoz podataka iz relacijske baze podataka i naredbe u Cypher jeziku za uvoz podataka u grafovsku bazu podataka. Naredbe koje se prikazuju kao primjeri unutar ovog potpoglavlja se odnose na model baze podataka za studentsku službu koji je korišten u primjeru 3 na str. 52.

Svaka naredba za izvoz podataka stvara datoteku s vrijednostima odvojenima zarezima (eng. *comma separated values - CSV*) koja sadrži n-torke relacije. Naredbe za uvoz podataka su znatno zanimljivije, jer svaka sadrži dio za učitavanje podataka i pronalazak drugih referentnih vrhova u bazi podataka prema kojima se stvaraju novi lukovi.

```
UNLOAD TO 'ispit.csv'
SELECT student_id || '_' || predmet_id || '_' || nastavnik_id,
ocjena, student_id, predmet_id, nastavnik_id
FROM ispit;
```

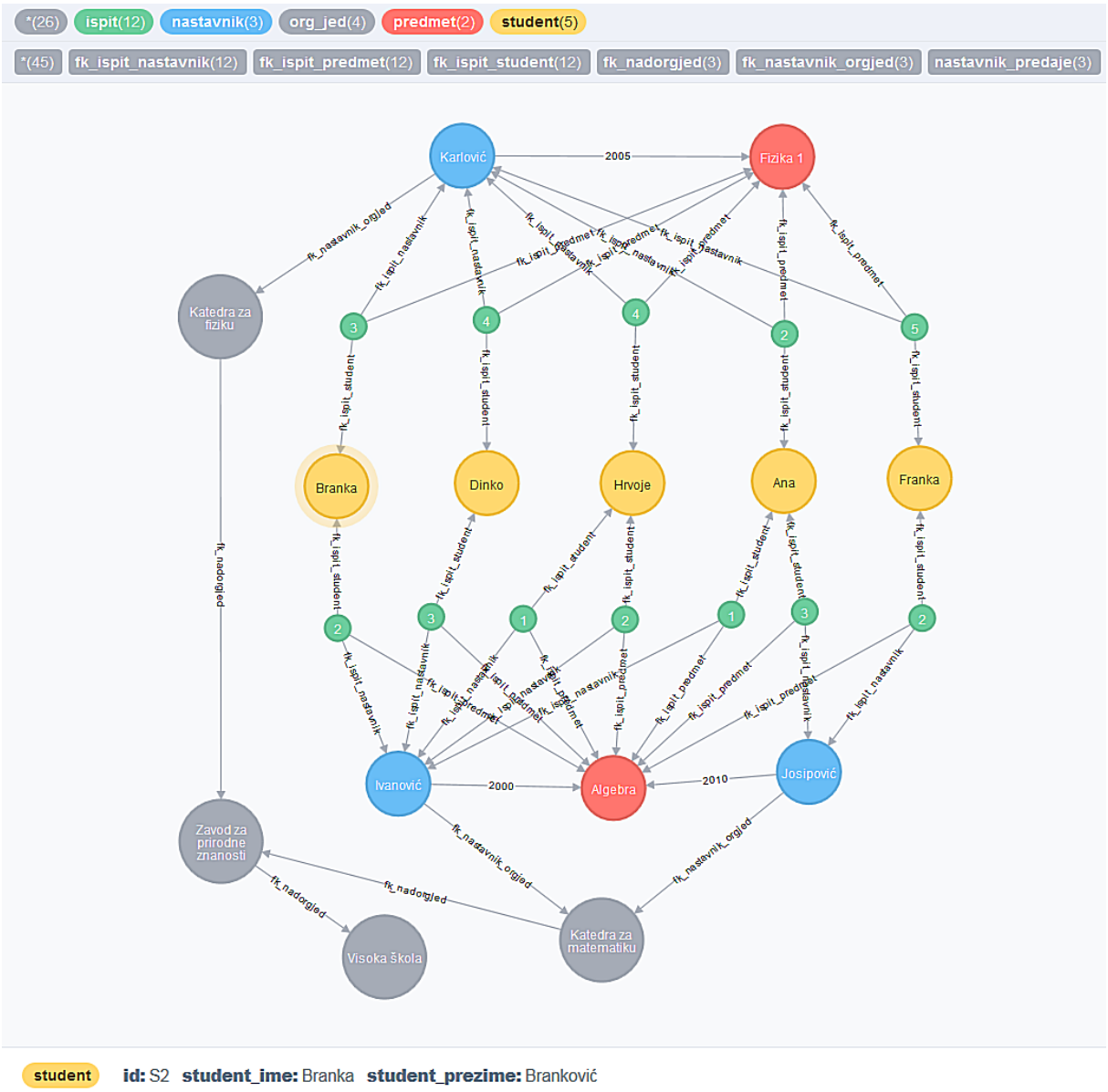
U gornjem odsječku koda, || je operator konkatenacije, odnosno spajanja dvaju tekstualnih podataka u jedan. Programski kod za uvoz podataka u Cypher programskom jeziku učitava podatke u vrhove, pronalazi postojeće vrhove s oznakama koje su kao nazivi triju referenciranih relacija i svojstvima *id* koje odgovaraju vrijednostima stranih ključeva i stvara lukove prema njima:

```
USING PERIODIC COMMIT 1000
LOAD CSV FROM 'file:ispit.csv' AS ln
CREATE (newVertex:ispit {id: ln[0], ocjena: ln[1]})
WITH newVertex, ln
MATCH (a:student) WHERE a.id = ln[2]
CREATE (newVertex)-[:fk_ispit_student]->(a)
WITH newVertex, ln
MATCH (a:predmet) WHERE a.id = ln[3]
CREATE (newVertex)-[:fk_ispit_predmet]->(a)
WITH newVertex, ln
MATCH (a:nastavnik) WHERE a.id = ln[4]
CREATE (newVertex)-[:fk_ispit_nastavnik]->(a);
```

Relacija *nastavnik_predaje* ima dva strana ključa i nije referencirana od drugih relacija, pa se iz njezinih n-torki stvaraju lukovi. Generirani programski kod za uvoz podataka pronalazi postojeće vrhove s oznakama koje su jednake nazivima referenciranih relacija i vrijednostima svojstva *id* koje odgovaraju vrijednostima stranih ključeva i stvara luk sa svojstvima među njima:

```
USING PERIODIC COMMIT 1000
LOAD CSV FROM 'file:nastavnik_predaje.csv' AS ln
MATCH (a1:nastavnik) WHERE a1.id = ln[1]
MATCH (a2:predmet) WHERE a2.id = ln[2]
CREATE
(a1)-[:nastavnik_predaje {id:ln[0],
                          predaje_od:ln[3],
                          predaje_do:ln[4]}]->(a2);
```

Temeljem sadržaja navedenih relacija (tablica 6.1, str. 55) se vidi kreirani sadržaj grafovske



Slika 12.1: Podaci iz primjera 3 popunjeni u Neo4j bazi podataka

baze u pregledu Neo4j sustava za upravljanje grafovskim bazama podataka (slika 12.1). U tom pregledu nije moguće istodobno vidjeti i sadržaj svojstava, nego su na donjem dijelu prikaza vidljiva svojstva trenutno označenog elementa grafa (u ovom slučaju, vrha s oznakom student, vrijednosti *id* svojstva "S2", na slici prikazanog s imenom studenta "Branka"). Umjesto pretpostavljenih vrijednosti koje se ispisuju u prikazu vrhova i lukova (za vrhove i lukove to je vrijednost svojstva *id*, a ako ga nemaju, onda naziv oznake), pregled je izmijenjen tako da se za vrhove s oznakom *predmet* pokazuje naziv predmeta, za vrhove s oznakom *nastavnik* prezime nastavnika, za vrhove s oznakom *ispit* ocjena, za vrhove s oznakom *student* ime studenta a za vrhove s oznakom *org_jed* naziv organizacijske jedinice. Za lukove s oznakom *nastavnik_predaje* se u pregledu umjesto naziva oznake pokazuje godina od koje nastavnik predaje predmet.

Za testiranje je provedeno nekoliko postupaka popunjavanja grafovske baze podataka iz relacijske. Na primjeru jedne relacijske baze podataka namijenjene transakcijskom okruženju, može se vidjeti broj generiranih elemenata grafovske baze.

Relacijska baza podataka u prosjeku ima oko 5,9 atributa po relaciji, i oko 1,6 stranih ključeva po relaciji. Na slici 12.2 su prikazani brojevi objekata u relacijskoj bazi podataka, brojevi generiranih elemenata u grafovskoj bazi podataka i njihove usporedne veličine.

Slika 12.2: Veličine baza podataka u postupku

Od 154 milijuna n-torki, očekivano, od oko 80% su bili stvoreni vrhovi (124 milijuna), dok je petina njih mogla sudjelovati u stvaranju lukova (oko 30 milijuna). Ostali lukovi, 317 milijuna, predstavljaju strane ključeve među n-torkama. Vidljiva je korist uporabe modela sa svojstvima. Svaki vrh i luk koji je nastao temeljem sadržaja n-torke ima svojstvo *id*, može imati i dodatna svojstva (prosječno 2,5 po elementu grafa) koji predstavljaju njihove zavisne atribute. Svi atributi koji predstavljaju strane ključeve, uključujući i kompozitne, su predstavljeni isključivo kao lukovi koji spajaju referencirane vrhove.

Za usporedbu ovog pristupa s onima spomenutim u poglavlju 2.3, izvedeno je popunjavanje grafovskih baza podataka koristeći manju, javno dostupnu relacijsku bazu podataka. Radi se o bazi podataka internetskog portala dba.stackexchange.com namijenjenog postavljanju pitanja vezanih za administraciju baza podataka i dobivanju odgovora na njih. Ovaj je portal jedan od više portala iste namjene objedinjenih pod nazivom Stack Exchange [149]. Ovaj primjer usporedbe je objavljen u [141], zajedno s usporedbom izvršavanja složenih upita nad grafovskim bazama podataka popunjenim različitim algoritmima.

Relacijska baza podataka o kojoj je riječ sastoji se od osam relacija, od kojih je zbog jednostavnosti jedna izuzeta iz postupka (PostHistory), a ostalih sedam je sudjelovalo u postupku (Posts, Users, Badges, Comments, PostLinks, Tags i Votes). Ukupno, ove relacije su sadržavale 903.298 n-torki, 57 atributa i prosječno 1,57 stranih ključeva po relaciji. Svi ključevi su se sastojali od samo jednog atributa.

Izvedeno je popunjavanje triju grafovskih baza podataka (sve u Neo4j sustavu za upravljanje grafovskim bazama podataka) koristeći različite modele podataka:

- model bez svojstava: svaka n-torka postaje vrh, svaki atribut postaje vrh koji je povezan s onim koji predstavlja n-torku, svaki strani ključ postaje luk. Ovaj model koriste algoritmi kao što je RDB2Graph [83].
- model sa svojstvima vrhova: svaka n-torka postaje vrh, svaki atribut postaje svojstvo vrha, svaki strani ključ postaje luk. Ovaj model koriste algoritmi kao što je R2G [86].
- model sa svim svojstvima: model grafa sa svojstvima vrhova i lukova koji koriste algoritmi prezentirani u ovoj disertaciji.

Podaci koji opisuju stvorene grafove za navedena tri modela su navedeni u tablici 12.1.

Tablica 12.1: Broj objekata u popunjenim grafovskim bazama podataka

model	bez svojstava	sa svojstvima vrhova	sa svojstvima vrhova i lukova
vrhovi	6.632.525	903.298	716.825
lukovi	4.631.988	1.092.647	928.240
svojstva	0	2.206.509	2.206.509

U usporedbi s pristupima koji modeliraju sve atribute kao vrhove, kao što je RDB2Graph, ovaj pristup rezultira u znatno manjem broju stvorenih vrhova (oko 90% manjem) i lukova (oko 80% manjem) ali s puno svojstava (svima). U usporedbi s pristupima koji modeliraju sve n-torke kao vrhove, a njihove atribute kao svojstva, kao što je R2G, ovaj pristup rezultira s oko 20% manjim brojem vrhova, oko 15% manjim brojem lukova i istim brojem svojstava. Ako bi model relacijske baze podataka bio takav da je određeni broj ključeva (bilo primarnih bilo stranih) kompozitan, onda bi i broj svojstava bio nešto manji u korist pristupa prezentiranom u ovom radu.

12.2 Stvaranje potpuno vremenski određenog grafa temeljem snimljenog traga

Postupak stvaranja potpuno vremenski određenog grafa temeljem snimljenog traga se u velikoj mjeri oslanja na postupak popunjavanja grafovske baze podataka podacima iz relacijske baze podataka, pa je i implementacija ovog dijela metode obavljena proširenjem implementacije prikazane u poglavlju 12.1.

Naredbe za izvoz podataka iz povijesne relacijske baze podataka koja sadržava snimljeni trag se generiraju tako da sadrže i druge podatke o samom zapisu koji predstavljaju, odnosno atribute o korisniku, obavljenoj operaciji, vremenu operacije i slično. U donjem odsječku koda prikazana je naredba za izvoz povijesnih podataka iz relacije *_ispit*:

```
UNLOAD TO "_ispit.csv"
SELECT
--dodatni atributi za opis operacije:
```

```
Operation, OpUser, OpTimestamp,

--kompozitni primarni kljuc, za svojstvo id
student_id || "_" || predmet_id || "_" || datum_roka,


--zavisni atributi:
ocjena_pismeni,
reklamacija,
ocjena_usmeni,
datum_ispita,
ispit_vrijedi,
broj_izlaska,

--kompozitni strani kljucevi:
student_id || "_" || uciliste_id,
predmet_id || "_" || datum_roka || "_" ||  vrsta_roka,
ozndjelpismeni || "_" || uciliste_id,
ozndjelusmeni || "_" || uciliste_id

FROM _ispit;
```

Programski kod u programskom jeziku Cypher sadrži naredbe koje se odnose na operacija unosa i izmjene (u kojima se stvaraju novi vrhovi) i naredbu koja se odnosi na operacije izmjene i brisanje (u kojima se postavlja vrijeme završetka aktivnosti vrha i pripadnih lukova). Prilikom obrade naredbi za operacije unosa i izmjene, naredba sadrži dio za stvaranje novog vrha, dijelove za pronalazak odgovarajućih vrhova (prema oznaci, vrijednosti svojstva *id* i prethodnom najbližem vremenu aktivnosti):

```
USING PERIODIC COMMIT 1000
LOAD CSV FROM 'file:_ispit.csv' AS ln
WITH ln WHERE ln[0] IN ['i', 'u']
//stvaranje novog vrha
CREATE (newVertex:_ispit { id: ln[3], t_start: ln[2], user: ln[1],
                           ocjena_pismeni: ln[4], reklamacija: ln[5],
                           ocjena_usmeni: ln[6], datum_ispita: ln[7],
                           ispit_vrijedi: ln[8], broj_izlaska: ln[9]})

// stvaranje luka prema najmladjem starijem vrhu s oznakom _studvu
WITH newVertex, ln
MATCH (a:_studvu) WHERE a.id = ln[10] AND a.t_start < ln[2]
WITH MAX(a.t_start) AS found_max, newVertex, ln
MATCH (b:_studvu) WHERE b.id = ln[10] AND b.t_start = found_max
CREATE (newVertex)-[:fkispitstudvu {t_start:ln[2], user: ln[1]} ]->(b)
```

```
// stvaranje luka prema najmladjem starijem vrhu s oznakom _rok
WITH newVertex, ln
MATCH (a:_rok) WHERE a.id = ln[11] AND a.t_start < ln[2]
WITH MAX(a.t_start) AS found_max, newVertex, ln
MATCH (b:_rok) WHERE b.id = ln[11] AND b.t_start = found_max
CREATE (newVertex)-[:fkispitrok {t_start:ln[2], user: ln[1]} ]->(b)

// stvaranje luka prema najmladjem starijem vrhu
// za djelatnika na pismenom ispitu
WITH newVertex, ln
MATCH (a:_osobaust) WHERE a.id = ln[12] AND a.t_start < ln[2]
WITH MAX(a.t_start) AS found_max, newVertex, ln
MATCH (b:_osobaust) WHERE b.id = ln[12] AND b.t_start = found_max
CREATE (newVertex)-[:fkispitdjelatpismeni {t_start:ln[2],
                                           user: ln[1]} ]->(b)

// stvaranje luka prema najmladjem starijem vrhu
// za djelatnika na usmenom ispitu
WITH newVertex, ln
MATCH (a:_osobaust) WHERE a.id = ln[13] AND a.t_start < ln[2]
WITH MAX(a.t_start) AS found_max, newVertex, ln
MATCH (b:_osobaust) WHERE b.id = ln[13] AND b.t_start = found_max
CREATE (newVertex)-[:fkispitdjelatusmeni {t_start:ln[2],
                                          user: ln[1]} ]->(b);
```

Također se generira i naredba za stvaranje indeksa nad pojedinim svojstvom vrha s određenom oznakom, kako bi se u budućim obradama brže pronalazili vrhovi nad kojima je potrebno obaviti druge akcije:

```
CREATE INDEX ON :_ispit(id);
```

Prilikom obrade naredbi za operacije izmjene i brisanja, naredba sadrži dio za postavljanje svojstva *t_end* za prethodnu instancu vrha s istom oznakom i vrijednosti svojstva *id*, kao i za pripadne lukove tog vrha:

```
USING PERIODIC COMMIT 1000
LOAD CSV FROM 'file:_ispit.csv' AS ln
WITH ln WHERE ln[0] IN ['d', 'u']

// postaviti t_end za prethodnu instancu ovog vrha
MATCH (a:_ispit) WHERE a.id = ln[3] AND a.t_start < ln[2]
WITH MAX(a.t_start) AS found_max, ln
```

```
MATCH (b:_ispit) WHERE b.id = ln[3] AND b.t_start = found_max
SET b.t_end = ln[2]

// postaviti t_end za sve pripadne lukove tog vrha
WITH found_max, ln
MATCH (b:_ispit)-[c]-() WHERE b.id = ln[3] AND b.t_start = found_max
SET c.t_end = ln[2];
```

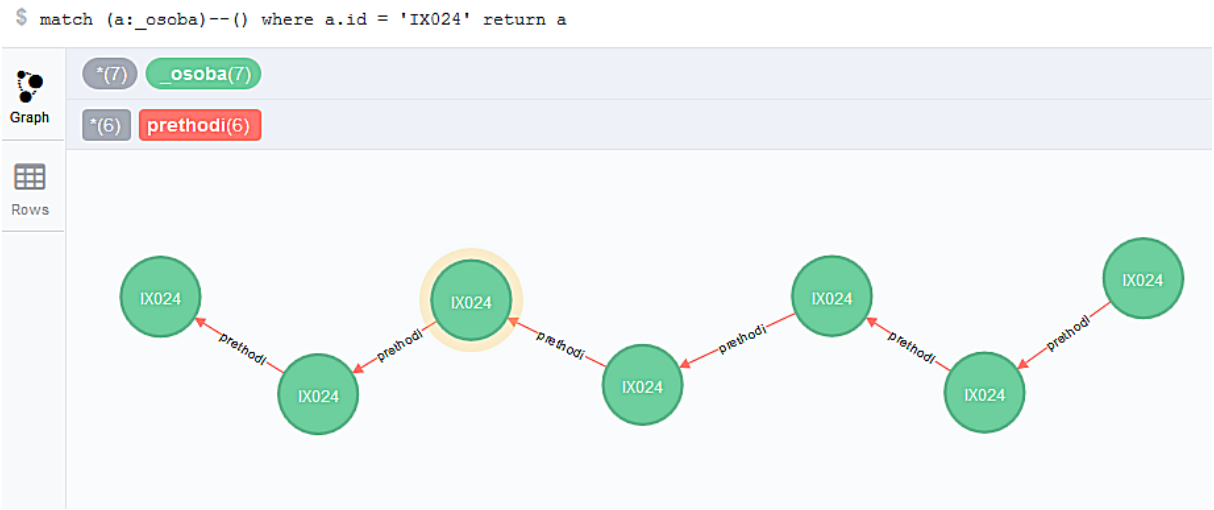
Obavljanjem naredbi za izvoz podataka nad povijesnom relacijskom bazom podataka i naredbi za uvoz podataka u grafovsku bazu podataka se stvara potpuno vremenski određeni graf. Naredbe za uvoz podataka u grafovsku bazu podataka moraju se obavljati u ispravnom redoslijedu, kako bi se elementi grafa mogli povezati jedni s drugima.

Konstruirani snimljeni trag koji se koristi za testiranje metode sadrži podatke o operacijama nad 21 relacijom koje se odnose na središnji dio ispitne evidencije. Ove relacije ukupno sadrže podatke o 3.927.622 obavljene operacije nad odgovarajućim radnim relacijama u vremenskom razmaku od 3.528 dana, odnosno nešto više od 9,5 godina. Ukupno, relacije sadrže 314 atributa, od čega su 105 dodatni atributi za opis obavljenih operacija, a 209 atributa su atributi radnih relacija. Također, odgovarajuće radne relacije u tom dijelu sheme baze podataka koji se odnosi na ispite sadrže 63 strana ključa, od kojih se u postupku ne koriste svi, nego njih 32, jer nisu uzete u obzir sve relacije iz radne baze podataka, nego je od ostatka samo izuzet dio sheme koji se odnosi na ispitnu evidenciju.

Slika 12.3: Konverzija snimljenog traga u potpuno vremenski određeni graf

U postupku stvaranja potpuno vremenski određenog grafa, u grafovskoj bazi podataka koja ga sadrži se stvara 3.835.004 vrha s 18 oznaka, 17.766.945 lukova s 36 različitih oznaka (18 povijesnih relacija je predstavljeno vrhovima, ostale 3 lukovima, 32 strana ključa su prikazana lukovima, a tu je također i luk koji povezuje prethodnu verziju vrha s novijom s oznakom *prethodi*). Vrhovi i lukovi sadrže ukupno 30.770.007 svojstava. Omjeri količine elemenata u snimljenom tragu i potpuno vremenski određenom grafu su prikazani na slici 12.3.

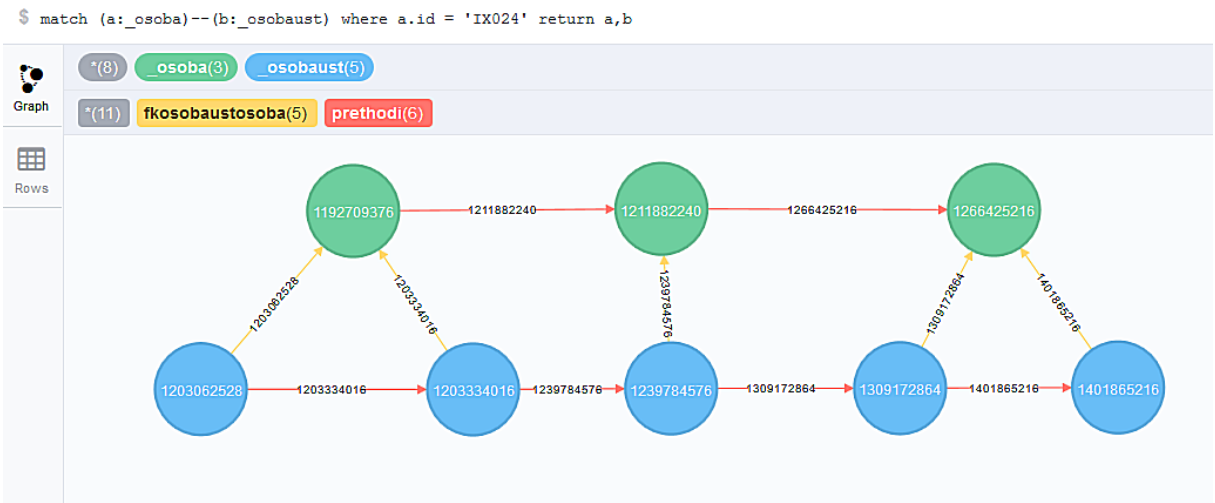
Na slici 12.4 je prikazan dio potpuno vremenski određenog grafa u pregledniku Neo4j baze podataka. Na vrhu slike je vidljiva naredba u programskom jeziku Cypher a ispod je prikazana legenda s bojama i oznakama vrhova i lukova te graf. Slika prikazuje sedam instanci jednog vrha



Slika 12.4: Sedam instanci vrha koji predstavljaju povijest podataka o osobi u potpuno vremenski određenom grafu

s vrijednosti svojstva *id* "IX024" koji predstavlja osobu s istom takvom šifrom u snimljenom tragu. Ovih sedam instanci su posljedica jedne operacije unosa i šest operacija izmjena zapisa o toj osobi. Vrhovi su povezani lukom s oznakom *prethodi*.

Na slici 12.5 je prikazan dio potpuno vremenski određenog grafa u pregledniku Neo4j baze podataka koji predstavlja dio povijesti podataka o istoj osobi kao i na prethodnoj slici (zeleni vrhovi s oznakom *_osoba*), ali u kombinaciji s dijelom povijesti pripadnog zapisa o toj osobi u svojstvu djelatnika u ustanovi (plavi vrhovi s oznakom *_osobaust*). Između ove dvije vrste vrhova postoje lukovi koji predstavljaju strani ključ između odgovarajućih radnih relacija (žuti lukovi s oznakom *fkosobaustosoba*), a između vrhova s istim oznakama su crveni lukovi s oznakom *prethodi* koji povezuju prethodne instance vrha s novijima. S obzirom da je u prikazu

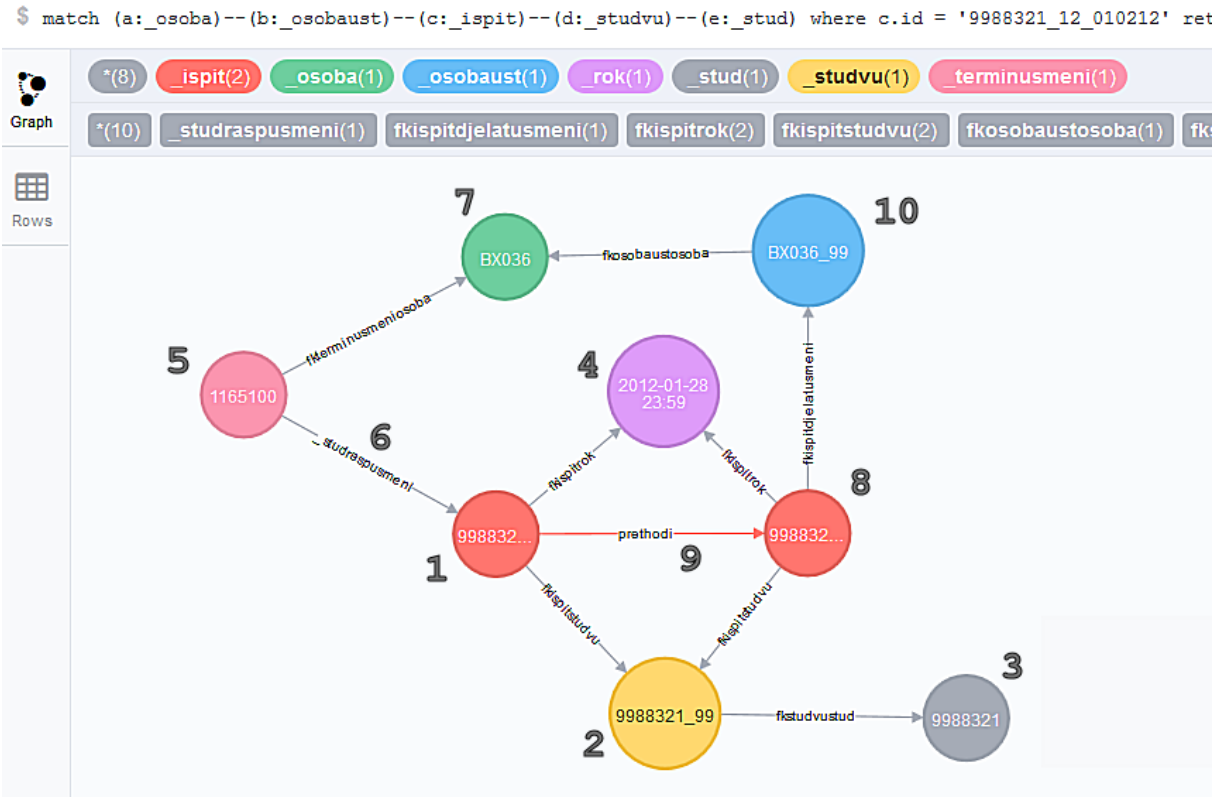


Slika 12.5: Instance dva povezana vrha kroz vrijeme

podataka iz Neo4j baze podataka moguće podesiti tekst koji se prikazuje na pojedinim vrhovima

i lukovima, ovaj je prikaz tako podešen da za sve elemente pokazuje vrijeme početka aktivnosti (svojstvo *t_start* u bazi podataka), tako da je moguće vidjeti kada su pojedini vrhovi nastajali i da oni koji nastaju naknadno imaju lukove koji predstavljaju strane ključeve povezane na najnovije instance referentnih vrhova.

Na slici 12.6 je prikazan uobičajeni postupak provođenja ispita koji se sastoji samo od usmenog dijela. Vidljivi su vrhovi sa sedam različitih oznaka i lukovi s isto toliko različitih oznaka. Tijek podataka u evidenciji ispita je takav da student prijavljuje ispit čime nastaje novi zapis u pripadnoj relaciji, a time i prva instanca vrha koji predstavlja taj ispit (na slici je to lijevi od dva crvena vrha s oznakom *_ispit*, dodatno naznačen brojem 1). Zapis je u relacijskoj bazi podataka povezan na zapis o studentu na visokom učilištu (u grafu žuti vrh naznačen brojem 2, koji je povezan na vrh naznačen brojem 3 koji predstavlja studenta) i zapis o ispitnom roku (vrh s oznakom *_rok* naznačen brojem 4).



Slika 12.6: Evidencija podataka o ispitu

Nakon što je izrađen raspored za usmeni ispit za ovog studenta, stvaraju se vrh koji predstavlja termin za usmeni ispit (naznačen brojem 5, s oznakom *_terminusmeni*) i luk koji predstavlja raspored za usmeni ispit (nastao iz relacije, ne iz stranog ključa, s oznakom *_studraspusmeni* i naznačen brojem 6 na slici). Ovaj se vrh također spaja s vrhom koji predstavlja osobu koja treba biti ispitivač (naznačen brojem 7). Evidencija ocjene studentu znači izmjenu zapisa o ispitu, što znači da je za tu izmjenu stvoren novi vrh, u grafu naznačen brojem 8. Osim toga stvara se i njegova veza na prethodnu instancu (luk naznačen s 9) te isti strani ključevi koji se odnose

na studenta na visokom učilištu, ispitni rok i ispitivača na ustanovi (vrh s oznakom *_osobaust*, naznačen brojem 10, koji je povezan na vrh koji predstavlja tu osobu).

Naslovi svih vrhova na ovom prikazu su vrijednosti njihovih *id* svojstava, a svih lukova njihove oznake.

12.3 Pronalazak čestih vremenskih podgrafova

Proširenja GraMi algoritma u smislu omogućavanja pronalaska čestih vremenskih podgrafova unutar potpuno vremenski određenog grafa opisana u poglavlju 9 su implementirana u programskom jeziku Java, naslanjajući se na izvorni programski kôd GraMi algoritma. U ova proširenja pripadaju i svi pomoćni razredi na osnovu kojih su opisani algoritmi koji su dio postupka pronalaska čestih podgrafova.

S obzirom na veliki ukupni interval kroz koji se protežu svi događaji unutar potpuno vremenski određenog grafa, moguć je jako veliki broj različitih vrijednosti širine vremenskog susjedstva δ. S druge strane, taj broj treba biti takav da omogućava pronalazak dovoljno čestih vremenskih podgrafova. Najvažnije od svega, taj broj treba procijeniti uzevši u obzir narav podataka koji se proučavaju. Kako se radi o evidenciji podataka uz ispite, tako se može očekivati da je svaki događaj udaljen jedan od drugoga u razmaku nekoliko dana. S time na umu, učinjeno je nekoliko različitih izračuna čestih vremenskih podgrafova s različitim širinama vremenskog susjedstva.

Također, uzevši u obzir relativno veliki broj elemenata grafa u odnosu na relativno mali broj različitih oznaka, minimalnu frekvenciju pojavljivanja podgrafa f je potrebno postaviti razmjerno visoko.

U nastavku je prikazan slučaj s vrijednostima parametara $\delta = 86.400$ sekundi (odnosno jedan dan) i $f = 10.000$.

Prilikom normiranja vremenskih susjedstava uz širinu vremenskog susjedstva $\delta = 86.400$ sekundi (jedan dan) i preslikavanja svojstava kako bi se dobila zamjenska oznaka, popunjavaju se mape $M_{Properties}$, M_{Values} i M_{Labels} te nakon ovog postupka mapa $M_{Properties}$ sadrži 99 svojstava, koja zajedno imaju ukupno 180.613 vrijednosti, pohranjenih u mapi M_{Values}. Mapa koja sadržava zamjenske oznake, M_{Labels}, sadrži 10.477 ključeva s ukupno 1.564.727 različitih kombiniranih indeksa svojstva i njegove vrijednosti. Od navedenih 10.477 zamjenskih oznaka, njih se 21 pojavljuje više od f puta, te čine osnovu za česte podgrafove. Od tih 21, dvije s najvećom frekvencijom se pojavljuju 1.076.912, odnosno 848.865 puta, što implicira veliki broj različitih podgrafova koje GraMi algoritam treba istražiti da bi utvrdio jesu li dovoljno česti.

Pronađeni česti podgrafovi uz zadanu frekvenciju $f = 10.000$ se pohranjuju u datoteku. U prilagođenom ispisu iz GraMi algoritma, umjesto zamjenskih oznaka se ispisuju svojstva i njihove vrijednosti, iz čega je moguće rekonstruirati česte vremenske podgrafove s normiranim

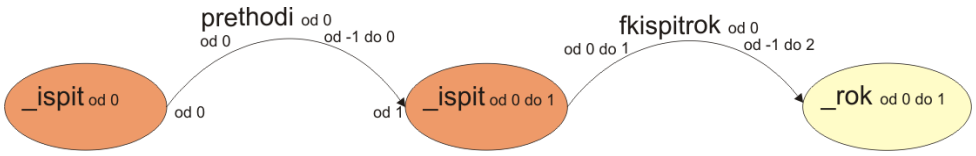
vremenskim susjedstvima. U navedenom slučaju, GraMi pronalazi 245 različitih vremenskih podgrafova. Jedan je prikazan u sljedećem ispisu:

```
...
11:

 v 0
        node_label: _ispit
        prethodi_start: 0
 v 1
        node_label: _ispit
        node_end: 1
        prethodi_start: 1
 v 2
        node_label: _rok
        node_end: 1
 e 0 1
        edge_label: prethodi
        _ispit_start: 0
        _ispit_start: -1
        _ispit_end: 0
 e 1 2
        edge_label: fkispitrok
        _rok_start: -1
        _rok_end: 2
        _ispit_start: 0
        _ispit_end: 1
12:
...
```

Brojka 11 na početku ovog dijela ispisa označava da se radi o jedanaestom pronađenom čestom podgrafu. On se sastoji od tri vrha (oznake "v 0", "v 1", "v 2"), lukova između vrhova 0 i 1 (oznaka "e 0 1") te 1 i 2 (oznaka "e 1 2"). Uz svaki element su ispisana njegova svojstva. Najprije, tu je oznaka vrha ili luka, koja je predstavljena svojstvom s nazivom *node_label* odnosno *edge_label*, a nakon toga tu su svojstva koja opisuju normirano vremensko susjedstvo svakog od elemenata. Nazivi tih svojstava su kombinirani od oznake susjednog elementa i teksta "start" ili "end" što označava početak ili kraj aktivnosti elementa. Prikazani česti vremenski podgraf čine dva vrha s oznakama *_ispit* od kojih jedan prethodi drugome, a drugi je vezan na vrh s oznakom *_rok* incidentnim im lukom s oznakom $fkispitrok$. Uzevši u obzir i normirana vremenska susjedstva ovih elemenata, ovaj bi česti podgraf izgledao kao na slici 12.7.

Uzevši u obzir da se događaji vezani uz ispite mogu proširiti na nekoliko dana (npr. od



Slika 12.7: Grafički prikaz jednog čestog vremenskog podgrafa

prijave ispita do samog ispita može proći i mjesec dana), obavljeni su izračuni i s puno većom širinom vremenskog susjedstva. U sljedećem ispisu je prikazan česti podgraf koji opisuje dio uobičajenog procesa ispita prikazanog na slici 12.6, koji se dobiva uz širinu vremenskog susjedstva $\delta = 1.296.000$ sekundi (15 dana).

```
v 0
        node_label: _osoba
v 1
        node_label: _osobaust
        fkosobaustosoba_start: 0
v 2
        node_label: _ispit
        fkispitdjelatusmeni_start: 0
        fkispitrok_start: 0
        fkispitstudvu_start: 0
        prethodi_start: 0
v 3
        node_label: _rok
        fkispitrok_start: 1000002
        fkispitrok_start: 1000001
        fkispitrok_end: 1000002
v 4
        node_label: _ispit
        fkispitrok_start: 0
        fkispitstudvu_start: 0
        prethodi_start: 2
        fkispitrok_end: 2
        fkispitstudvu_end: 2
        node_end: 2
        _studraspusmeni_start: 1
v 5
        node_label: _studvu
        fkispitstudvu_start: 1000002
        fkispitstudvu_start: 1000001
        fkstudvustud_start: 0
        fkispitstudvu_end: 1000002
v 6
```

```
        node_label: _stud

v 7
        node_label: _terminusmeni
        _studraspusmeni_start: 1
        fkterminusmeniosoba_start: 0

e 0 1
        edge_label: fkosobaustosoba
        _osobaust_start: 0
e 1 2
        edge_label: fkispitdjelatusmeni
        _ispit_start: 0
e 2 3
        edge_label: fkispitrok
        _ispit_start: 0
e 3 4
        edge_label: fkispitrok
        _ispit_start: 0
        edge_end: 1
        _ispit_end: 1
e 4 5
        edge_label: fkispitstudvu
        _ispit_start: 0
        edge_end: 1
        _ispit_end: 1
e 5 6
        edge_label: fkstudvustud
        _studvu_start: 0
e 4 7
        edge_label: _studraspusmeni
        _ispit_start: -2
        _ispit_end: 1
        _terminusmeni_start: -1
```

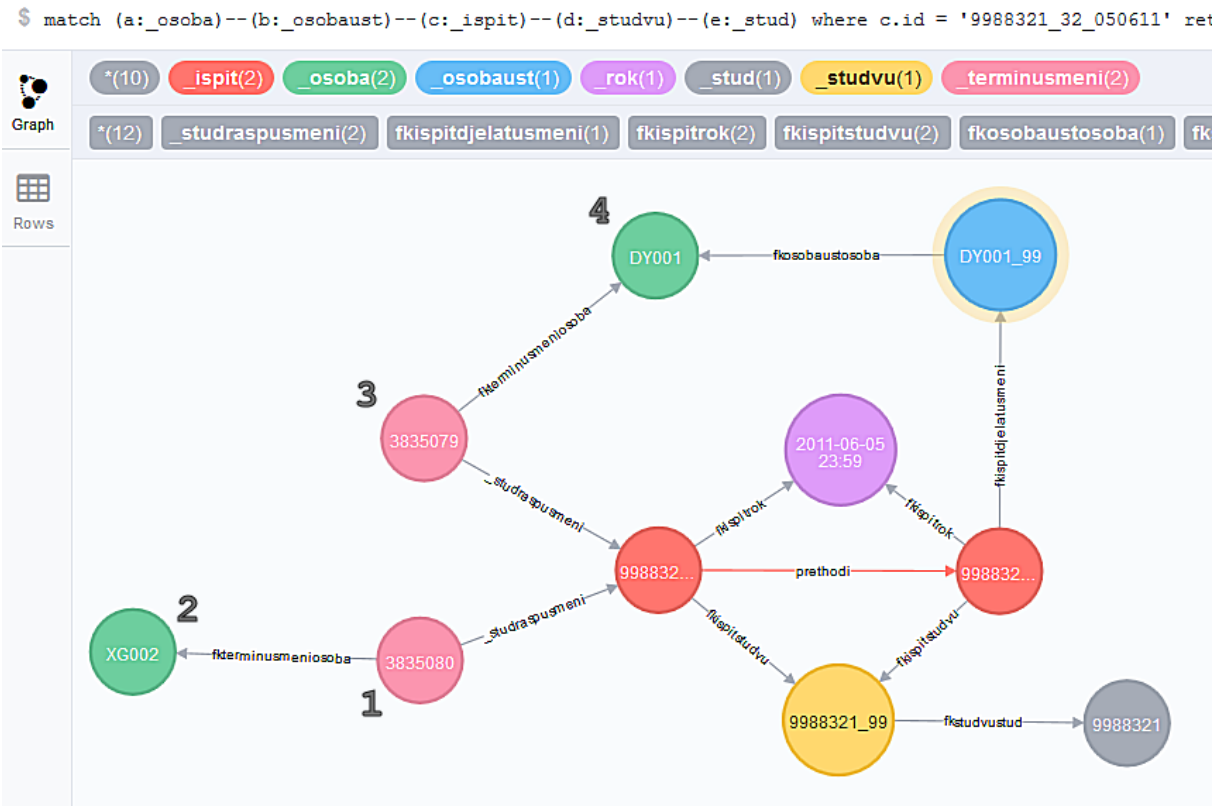
U gornjem izračunu vrijednost ξ je 10^6.

12.4 Pronalazak mogućih zlouporaba sustava

Implementacija algoritama za pronalazak mogućih zlouporaba sustava također koristi isti dio programskog koda za normiranje vremenskih susjedstava koji se koristi pri pronalasku čestih vremenskih podgrafova. Ovaj se pak dio kombinira s algoritmom za određivanje sličnosti potpisa grafova, pa su on, varijante funkcije ϕ i pripadnih pomoćnih razreda, kao što je opisano

u poglavlju 10, također implementirani kao jedinstveni programski kod u programskom jeziku Java.

Kako je rečeno u uvodu ovog poglavlja, u testni snimljeni trag su dodane i iskonstruirane anomalije koje se odnose na drugačiji tijek evidencije ispita. Jedna takva anomalija, prikazana u pregledu podataka iz Neo4j baze podataka je na slici 12.8.



Slika 12.8: Odstupanje od uobičajenog procesa evidencije ispita

Na slici se, u odnosu na graf sa slike 12.6, nalaze dva dodatna vrha (naznačeni brojevima 1 i 2), koji se odnose na početni dio rasporeda za usmeni ispit, gdje se vidi da je student prvobitno bio raspoređen za ispit kod jednog nastavnika (s vrijednosti *id* svojstva "XG002"). Nakon toga je student raspoređen kod drugog nastavnika (vrhovi naznačeni brojevima 3 i 4) koji je taj ispit i dovršio.

U odnosu na pronađene česte vremenske podgrafove, ovaj će se podgraf razlikovati topološki (već spram uobičajenog postupka evidencije ispita ovaj podgraf ima 22% više elemenata) i vremenski (svaki od tih elemenata ima svoje vremenske odrednice, koje se odnose ne samo na početak aktivnosti, nego i na završetak aktivnosti elemenata, pa tako srazmjerno više utječu na razliku spram grafa gdje većina elemenata ima samo jednu vremensku odrednicu). Razlike u svojstvima elemenata grafa se prilikom ovog testiranja ignoriraju, odnosno, dodatna svojstva koja proizlaze iz zavisnih atributa radi jednostavnijeg testiranja nisu uzeta u obzir pri stvaranju potpuno vremenski određenog grafa.

Zbog navedenih razlika, a uzevši u obzir rečeni omjer težina karakteristika ($w_{top} = 2 \times w_{time}$),

u cilju identifikacije ove anomalije, parametar maksimalnog odstupanja između dva navedena grafa se postavlja relativno nisko, na otprilike 60%, odnosno $\sigma = 0,6$. Testiranje utjecaja granica frekvencije anomalije α_l i α_u se može provjeriti pomjeranjem tih granica i promatranjem pojavnosti dojavljivanja moguće zlouporabe. Za potrebe testiranja je iskonstruirano 25 anomaličnih situacija sa slike 12.8 u kojima sudjeluje isti skup korisnika, pa je za detekciju te anomalije kao moguće prijetnje potrebno postaviti α_l i α_u tako da obuhvate ovaj broj. U stvarnoj situaciji istraživanja mogućih zlouporaba na uzorku iz snimljenog traga bi valjalo pretpostaviti manji broj anomalija koje čine zlouporabu, između 10 i 20, pa bi bilo uputno postaviti $\alpha_l = 10$, a gornju granicu nešto više, $\alpha_u = 30$.

Nakon izvršavanja glavne funkcije za pronalazak mogućih zlouporaba, one su kao skup anomalija pohranjene u datoteku iz koje je moguće iščitati podgrafove koje čine te anomalije i proučiti ih. U donjem ispisu je prikazan pronađeni skup anomalija s njih tri (od spomenutih 25), zajedno s naredbama za pregled tih podgrafova u Neo4j sučelju.

```
Skup anomalija 1:
        Anomalija:
                Pripadni FTSG: 184
                Korisnici: korisnik_a, korisnik_b, korisnik_c, korisnik_d
                Referentni vrh: 1231029

        Anomalija:
                Pripadni FTSG: 184
                Korisnici: korisnik_a, korisnik_b, korisnik_c, korisnik_d
                Referentni vrh: 321090

        Anomalija:
                Pripadni FTSG: 184
                Korisnici: korisnik_a, korisnik_b, korisnik_c, korisnik_d
                Referentni vrh: 989922

Pregled ovih anomalija u Neo4j:
        MATCH(a) WHERE id(a) = 1231029 RETURN a;
        MATCH(a) WHERE id(a) = 321090 RETURN a;
        MATCH(a) WHERE id(a) = 989922 RETURN a;
```

Pri izvršenju upita u Neo4j pregledniku, prvi od njih će pokazati jedan od vrhova čijim proširenjem (dvostruki klik na vrh u pregledniku) je moguće dobiti graf prikazan na slici 12.8, a drugi će pokazati slične podgrafove. U slučaju potrebe, iz prikazanih elemenata grafa je moguće vidjeti vrijednosti primarnog ključa (konkatenirana vrijednost svojstva *id*) i vremena nastanka elemenata te temeljem tih podataka proučavati i snimljeni trag ali i samu radnu relacijsku bazu podataka. U svakom slučaju, kako je već napomenuto, osoba ili tim zadužen za sigurnost

sustava treba poznavati podatke i procese u sustavu, kako bi mogla zaključiti radi li se o mogućoj zlouporabi i detaljnije ju istražiti.

12.5 Osvrt

U ovom poglavlju su prikazane implementacije originalnih algoritama opisanih u sklopu disertacije, kao i cijele metode za pronalazak mogućih složenih zlouporaba informacijskih sustava. Također su prikazani rezultati testiranja. Dok je testiranje prvog koraka, konverzije relacijskih podataka u graf, bilo moguće obaviti koristeći razne relacijske baze podataka, izvori testnih podataka za ostale algoritme su bili ograničeni.

Osnovni razlog za nemogućnost testiranja algoritama i metode za pronalazak mogućih zlouporaba nad stvarnim podacima jest sama osjetljivost teme kojom se ova disertacija bavi. I uz činjenicu da postoji više informacijskih sustava za čije se relacijske baze podataka snima trag u opisanom formatu te uz pretpostavku da bi bilo moguće od vlasnika sustava ishoditi dozvolu za korištenje tih podataka u testne svrhe, i dalje ostaje upitno kako provjeriti rezultate testiranja (eventualno pronađene moguće zlouporabe) pa i kako objaviti te rezultate. Zbog toga su u testiranju i verifikaciji korišteni iskonstruirani podaci pomoću kojih je moguće potvrditi ispravnost metode. Kako bi test bio što vjerniji, podaci su konstruirani sukladno korištenju stvarnog informacijskog sustava uz dodavanje ponavljajućih anomalija koje prate uzorak otprije objavljen u javnosti.

Poglavlje 13

Zaključak

Disertacija donosi pregled mogućnosti identifikacije složenih zlouporaba sustava koje se dijelom temelje na pripadnim informacijskim sustavima. Pri tom se *složenost* odnosi na one zlouporabe u kojima sudjeluje više dionika sustava, zlouporaba počinje dogovorom tih dionika izvan informacijskog sustava, a unutar samog informacijskog sustava nitko od njih ne prekoračuje svoje ovlasti.

Predloženo je rješenje za otkrivanje mogućih zlouporaba, koje se temelji na otkrivanju skupina korisnika čije zajedničko djelovanje odstupa od uobičajenih obrazaca ponašanja korisnika. Osnova predloženog rješenja je snimljeni trag relacijske baze podataka na kojoj se informacijski sustav zasniva. Da bi se pronašli uobičajeni obrasci ponašanja, koristi se algoritam za pronalazak čestih podgrafova unutar grafa, a u tu svrhu se iz snimljenoga traga stvara potpuno vremenski određeni graf. To je graf u kojem su svi elementi takvi da imaju svoj početak i eventualni završetak aktivnosti. S obzirom da ovaj graf sadrži vremenske odrednice koje predstavljaju točke u kontinuiranom vremenu, potrebno je odrediti vremensku toleranciju u međusobnom odnosu elemenata grafa koja također postaje dio vremenskih čestih podgrafova i obrazaca ponašanja. Od tako pronađenih čestih podgrafova se pronalaze anomalije - vremenski podgrafovi s jednakom tolerancijom koji u određenoj mjeri odstupaju od istog čestog vremenskog podgrafa, akcije nad čijim elementima uvijek provodi ista skupina korisnika, a koje se pojavljuju frekvencijom koja je unutar zadanih granica. Ove anomalije predstavljaju moguću zlouporabu sustava koju je potrebno detaljnije istražiti.

Uzevši u obzir činjenicu da je snimljeni trag relacijske baze podataka također u obliku relacijske baze podataka, osnovni elementi predloženog rješenja, a ujedno i glavni doprinosi disertacije su:

- algoritam za transformaciju podataka iz relacijskih baza podataka u grafovske baze podataka, s posebnim naglaskom na transformaciju vremenskih relacijskih podataka u potpuno vremenski određene grafove,
- algoritam za pronalazak čestih vremenskih podgrafova potpuno vremenski određenog

grafa,

- algoritam za pronalazak odstupanja od čestih vremenskih podgrafova potpuno vremenski određenog grafa,
- metoda za otkrivanje mogućih sigurnosnih prijetnji na osnovu odstupanja od čestih vremenskih podgrafova potpuno vremenski određenog grafa, koja objedinjuje navedene algoritme i prilagođava njihovu uporabu ovoj svrsi.

U okviru disertacije prikazana je implementacija predloženog rješenja u programskom kôdu te je obavljeno testiranje na dijelu snimljenog traga relacijske baze podataka informacijskog sustava namijenjenog visokim učilištima koji se odnosi na evidenciju ispita. Ovaj je snimljeni trag potpuno anonimiziran te su u njega ručno dodani uzorci složene prijevare. Temeljem implementacije i provedenog testiranja mogu se donijeti sljedeći zaključci.

Predložene algoritme je moguće koristiti u zasebne svrhe, kao i objedinjene u metodu za otkrivanje mogućih sigurnosnih prijetnji.

Algoritam za konverziju relacijskih baza podataka u grafovske baze podataka ostvaruje usporedive ili nešto bolje rezultate od drugih algoritama iste namjene.

Potpuno vremenski određeni graf, koji sadrži vremenske odrednice svih elemenata grafa, je moguće iskoristiti i u drugim analizama koje se odnose na vremenske grafove. U usporedbi s vremenskim nizom grafova, potpuno vremenski određeni graf je prikladniji za rad s podacima koji se promatraju u kontinuiranom poimanju vremena. Također, bilježi samo promjene grafu, pa je prikladniji za trajnu pohranu. S druge strane, nije najprikladniji za analizu grafa u pojedinom trenutku, nego je adekvatnije područje njegove primjene ono koje promatra međusobne odnose elemenata grafa i podgrafova. U usporedbi s dinamičkim grafovima, potpuno vremenski određeni graf ima prednost jer čuva povijest grafa koja u običnom dinamičkom grafu ne postoji.

Normirana vremenska susjedstva unutar potpuno vremenski određenog grafa su prikladna osnova za pronalazak čestih vremenskih podgrafova, jer je iz njih moguće rekonstruirati tijek događaja kojim je podgraf nastao, što je dobra osnova za daljnje usporedbe.

Usporedba vremenskih grafova algoritmom sličnosti potpisa je prikladna osnova za dobivanje mjere sličnosti ovakvih grafova.

Metoda za pronalazak mogućih zlouporaba je iskoristiva za detekciju slučajeva koje je potrebno dalje istražiti. Osoba ili tim zadužen za sigurnost sustava mora dobro poznavati podatke i procese kako bi kompetentno mogao prosuditi radi li se u nekim slučajevima o stvarnim zlouporabama ili ne.

Iako je prikazana metoda za pronalazak mogućih zlouporaba uspješna, za primjenu nad stvarnim sustavima su potrebna dodatna testiranja i prilagodbe. Kao osnovni preduvjet, potrebno je omogućiti snimanje traga za relacijske baze podataka i posjedovati trag u određenom

vremenskom periodu koji bi se mogao analizirati. Nadalje, osobi koja se brine za sigurnost sustava i provodi ovu metodu treba omogućiti grafičko korisničko sučelje za upravljanje procesom stvaranja potpuno vremenski određenog grafa iz snimljenog traga, procesom pronalaska čestih vremenskih podgrafova i njihovu vizualizaciju te upravljanje procesom pronalaska anomalija. Ovisno o sustavu čiji je trag snimljen, načinu njegove uporabe i samoj veličini grafa, potrebno je testirati različite širine vremenskog susjedstva da bi se dobili smisleni obrasci ponašanja korisnika. Također je potrebno odrediti i koje će težine biti korištene pri stvaranju težinskih karakteristika grafa, kao i parametre za određivanje sličnosti i granice anomalija. Dobro korisničko sučelje bi sve ove parametre trebalo ponuditi, a uz njih i jednostavne i brze analize utjecaja pojedinih parametara na trajanje pojedinih postupaka i količinu rezultata koje bi oni mogli isporučiti. Takve analize tek valja osmisliti.

Dodatna poboljšanja odnose na performanse algoritama. Iako je metoda testirana na prilično velikom uzorku (milijuni elemenata grafova), bilo bi ju potrebno dodatno optimirati kako bi se rezultati i za znatno veće grafove dobivali u još kraćem vremenu.

Na kraju, tek uporaba u stvarnim analizama može pokazati potrebe za dodatnim parametriziranjem, dodatnim finim prilagođavanjima ili poboljšanjima samih algoritama koja su nužan dio života svakog programskog rješenja.

Literatura

[1] Ted, E., Goldberg, H. G., Memory, A., Young, W. T., Rees, B., Pierce, R., Huang, D., Reardon, M., Bader, D. A., Chow, E. *et al.*, "Detecting insider threats in a real corporate database of computer usage activity", in Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2013, str. 1393–1401.

[2] Chen, Y., Nyemba, S., Zhang, W., Malin, B., "Leveraging social networks to detect anomalous insider actions in collaborative environments", in Intelligence and Security Informatics (ISI), 2011 IEEE International Conference on. IEEE, 2011, str. 119–124.

[3] Spalka, A., Lehnhardt, J., "A comprehensive approach to anomaly detection in relational databases", in Data and Applications Security XIX. Springer, 2005, str. 207–221.

[4] Chung, C. Y., Gertz, M., Levitt, K., "Demids: A misuse detection system for database systems", in Integrity and Internal Control in Information Systems. Springer, 2000, str. 159–178.

[5] Beutel, A., "User behavior modeling with large-scale graph analysis", 2016.

[6] Bertino, E., Terzi, E., Kamra, A., Vakali, A., "Intrusion detection in RBAC-administered databases", in Computer security applications conference, 21st annual. IEEE, 2005, str. 10–pp.

[7] Mathew, S., Petropoulos, M., Ngo, H. Q., Upadhyaya, S., "A data-centric approach to insider attack detection in database systems", in Recent advances in intrusion detection. Springer, 2010, str. 382–401.

[8] Orel, O., "Nadzor nad radom korisnika u relacijskim bazama podataka", magistarski rad. Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, Zagreb, 2008.

[9] Genga, L., Potena, D., Martino, O., Alizadeh, M., Diamantini, C., Zannone, N., "Subgraph mining for anomalous pattern discovery in event logs", International Workshop on New Frontiers in Mining Complex Patterns, 2016.

[10] Castelltort, A., Laurent, A., "Rogue behavior detection in nosql graph databases", Journal of Innovation in Digital Ecosystems, Vol. 3, No. 2, 2016, str. 70–82.

[11] Codd, E. F., "A relational model of data for large shared data banks", Communications of the ACM, Vol. 13, No. 6, 1970, str. 377–387.

[12] Pavčević, M.-O., Nakić, A., Uvod u teoriju grafova. Zagreb, Hrvatska: Element, 2014.

[13] Bringmann, B., Nijssen, S., "What is frequent in a single graph?", in Advances in Knowledge Discovery and Data Mining. Springer, 2008, str. 858–863.

[14] Angles, R., Gutierrez, C., "Survey of graph database models", ACM Computing Surveys (CSUR), Vol. 40, No. 1, 2008, str. 1.

[15] Roussopoulos, N., Mylopoulos, J., "Using semantic networks for data base management", in Proceedings of the 1st International Conference on Very Large Data Bases. ACM, 1975, str. 144–172.

[16] Shipman, D. W., "The functional data model and the data languages daplex", ACM Transactions on Database Systems (TODS), Vol. 6, No. 1, 1981, str. 140–173.

[17] Kuper, G. M., Vardi, M. Y., "A new approach to database logic", in Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems. ACM, 1984, str. 86–96.

[18] Kunii, H. S., "DBMS with graph data model for knowledge handling", in Proceedings of the 1987 Fall Joint Computer Conference on Exploring technology: today and tomorrow. IEEE Computer Society Press, 1987, str. 138–142.

[19] Lecluse, C., Richard, P., Velez, F., "O2, an object-oriented data model", in ACM SIGMOD Record, Vol. 17, No. 3. ACM, 1988, str. 424–433.

[20] Gyssens, M., Paredaens, J., Gucht, D. V., "A graph-oriented object model for database end-user interfaces", ACM SIGMOD Record, Vol. 19, No. 2, 1990, str. 24–33.

[21] Andries, M., Gemis, M., Paredaens, J., Thyssens, I., Van den Bussche, J., "Concepts for graph-oriented object manipulation", in Advances in Database Technology—EDBT'92. Springer, 1992, str. 21–38.

[22] M., A. B. S., "Gram: A graph data model and query language", in European Conference on Hypertext Technology (ECHT), Milano, Italija, 1992.

[23] Gemis, M., Paredaens, J., "An object-oriented pattern matching language", in Object Technologies for Advanced Software. Springer, 1993, str. 339–355.

[24] Hidders, J., Paredaens, J., GOAL, A graph-based object and association language. Springer, 1994.

[25] Paredaens, J., Peelman, P., Tanca, L., "G-Log: A graph-based query language", Knowledge and Data Engineering, IEEE Transactions on, Vol. 7, No. 3, 1995, str. 436–453.

[26] Hidders, J., "Typing graph-manipulation operations", in Database Theory—ICDT 2003. Springer, 2003, str. 394–409.

[27] Levene, M., Poulovassilis, A., "The hypernode model and its associated query language", in Information Technology, 1990.'Next Decade in Information Technology', Proceedings of the 5th Jerusalem Conference on (Cat. No. 90TH0326-9). IEEE, 1990, str. 520–530.

[28] Graves, M., Bergeman, E. R., Lawrence, C. B., "A graph-theoretic data model for genome mapping databases", in System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on, Vol. 5. IEEE, 1995, str. 32–41.

[29] Levene, M., Poulovassilis, A., "An object-oriented data model formalised through hypergraphs", Data & Knowledge Engineering, Vol. 6, No. 3, 1991, str. 205–224.

[30] Soussi, R., Aufaure, M.-A., Baazaoui, H., "Graph database for collaborative communities", in Community-Built Databases. Springer, 2011, str. 205–234.

[31] Soussi, R., "Querying and extracting heterogeneous graphs from structured data and unstrutured content", Doktorski rad, Ecole Centrale Paris, 2012.

[32] Codd, E. F., "Data models in database management", ACM Sigmod Record, Vol. 11, No. 2, 1981, str. 112–114.

[33] Cruz, I. F., Mendelzon, A. O., Wood, P. T., "A graphical query language supporting recursion", in ACM SIGMOD Record, Vol. 16, No. 3. ACM, 1987, str. 323–330.

[34] Cruz, I. F., Mendelzon, A. O., Wood, P. T., "G+: Recursive queries without recursion.", in Expert Database Conf., 1988, str. 645–666.

[35] Consens, M. P., Mendelzon, A. O., "Expressing structural hypertext queries in graphlog", in Proceedings of the second annual ACM conference on Hypertext. ACM, 1989, str. 269–292.

[36] Blau, H., Immerman, N., Jensen, D., "A visual language for querying and updating graphs", University of Massachusetts Amherst Computer Science Technical Report, Vol. 37, 2002, str. 2002.

[37] Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wiener, J. L., "The Lorel query language for semistructured data", International journal on digital libraries, Vol. 1, No. 1, 1997, str. 68–88.

[38] Sheng, L., Özsoyoğlu, Z., Özsoyoğlu, G., "A graph query language and its query processing", in Data Engineering, 1999. Proceedings., 15th International Conference on. IEEE, 1999, str. 572–581.

[39] Ronen, R., Shmueli, O., "SoQL: A language for querying and creating data in social networks", in Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on. IEEE, 2009, str. 1595–1602.

[40] He, H., Singh, A. K., "Graphs-at-a-time: query language and access methods for graph databases", in Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, 2008, str. 405–418.

[41] "Cypher query language", http://neo4j.com/developer/cypher-query-language/, dohvat: 2016-04-26.

[42] "Neo4j graph database", http://www.neo4j.com, dohvat: 2016-04-26.

[43] Kuper, G. M., Vardi, M. Y., "The logical data model", ACM Transactions on Database Systems (TODS), Vol. 18, No. 3, 1993, str. 379–413.

[44] Levene, M., Loizou, G., "A graph-based data model and its ramifications", Knowledge and Data Engineering, IEEE Transactions on, Vol. 7, No. 5, 1995, str. 809–823.

[45] Kaplan, I., "A semantic graph query language. technical repport, Lawrence Livermore National Laboratory, October 17", 2006.

[46] "Sparql 1.1 query language, w3c recommendation", https://www.w3.org/TR/sparql11-query/, dohvat: 2017-05-10.

[47] "Resource description framework", https://www.w3.org/RDF/, dohvat: 2017-05-10.

[48] Wood, P. T., "Query languages for graph databases", ACM SIGMOD Record, Vol. 41, No. 1, 2012, str. 50–60.

[49] "The gremlin graph traversal machine and language", http://tinkerpop.apache.org/gremlin.html, dohvat: 2017-05-10.

[50] "Apache tinkerpop", http://tinkerpop.apache.org/, dohvat: 2017-05-10.

[51] "NoSQL databases", http://nosql-databases.org, dohvat: 2016-05-07.

[52] Sadalage, P. J., Fowler, M., NoSQL distilled: a brief guide to the emerging world of polyglot persistence. Pearson Education, 2012.

[53] "Hibernate – relational persistence for Java and .NET", http://www.hibernate.org, dohvat: 2016-05-07.

[54] "MyBatis", http://www.mybatis.org, dohvat: 2016-05-07.

[55] Angles, R., "A comparison of current graph database models", in Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on. IEEE, 2012, str. 171–177.

[56] Tauro, C. J., Aravindh, S., Shreeharsha, A., "Comparative study of the new generation, agile, scalable, high performance NoSQL databases", International Journal of Computer Applications, Vol. 48, No. 20, 2012, str. 1–4.

[57] "Basho Riak database", http://basho.com/products/riak-overview, dohvat: 2016-05-07.

[58] "Redis", http://redis.io, dohvat: 2016-05-07.

[59] "Oracle Berkeley DB", http://www.oracle.com/technetwork/products/berkeleydb, dohvat: 2016-05-07.

[60] "upscaledb", https://upscaledb.com/, dohvat: 2016-05-07.

[61] "mongoDB database", http://www.mongodb.org, dohvat: 2016-05-07.

[62] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., Gruber, R. E., "Bigtable: A distributed storage system for structured data", ACM Transactions on Computer Systems (TOCS), Vol. 26, No. 2, 2008, str. 4.

[63] "Apache Cassandra database", http://cassandra.apache.org, dohvat: 2016-05-07.

[64] "Apache Hbase database", http://hbase.apache.org, dohvat: 2016-05-07.

[65] "Amazon SimpleDB", http://aws.amazon.com/simpledb, dohvat: 2016-05-07.

[66] "AllegroGraph database", http://www.franz.com/agraph/allegrograph, dohvat: 2016-04-26.

[67] "DEX graph database", http://sparsity-technologies.com/{#}sparksee, dohvat: 2016-04-26.

[68] Martínez-Bazan, N., Muntés-Mulero, V., Gómez-Villamor, S., Nin, J., Sánchez-Martínez, M.-A., Larriba-Pey, J.-L., "Dex: high-performance exploration on large graphs for information retrieval", in Proceedings of the sixteenth ACM conference on Conference on information and knowledge management. ACM, 2007, str. 573–582.

[69] Iordanov, B., "HyperGraphDB: a generalized graph database", in Web-Age information management. Springer, 2010, str. 25–36.

[70] "HyperGraphDB", http://www.hypergraphdb.org, dohvat: 2016-04-26.

[71] "Objectivity Infinite graph", http://www.objectivity.com/products/infinitegraph/, dohvat: 2016-04-26.

[72] "Titan", https://github.com/thinkaurelius/titan/wiki, dohvat: 2017-01-21.

[73] Dubey, A., Hill, G. D., Escriva, R., Sirer, E. G., "Weaver: a high-performance, transactional graph database based on refinable timestamps", Proceedings of the VLDB Endowment, Vol. 9, No. 11, 2016, str. 852–863.

[74] Pokorny, J., "NoSQL databases: a step to database scalability in web environment", International Journal of Web Information Systems, Vol. 9, No. 1, 2013, str. 69–82.

[75] Padhy, R. P., Patra, M. R., Satapathy, S. C., "RDBMS to NoSQL: Reviewing some next-generation non-relational databases", International Journal of Advanced Engineering Science and Technologies, Vol. 11, No. 1, 2011, str. 15–30.

[76] Park, J., Lee, S.-g., "Keyword search in relational databases", Knowledge and Information Systems, Vol. 26, No. 2, 2011, str. 175–193.

[77] Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., Karambelkar, H., "Bidirectional expansion for keyword search on graph databases", in Proceedings of the 31st international conference on Very large data bases. VLDB Endowment, 2005, str. 505–516.

[78] He, H., Wang, H., Yang, J., Yu, P. S., "BLINKS: ranked keyword searches on graphs", in Proceedings of the 2007 ACM SIGMOD international conference on Management of data. ACM, 2007, str. 305–316.

[79] Agrawal, S., Chaudhuri, S., Das, G., "DBXplorer: enabling keyword search over relational databases", in Proceedings of the 2002 ACM SIGMOD international conference on Management of data. ACM, 2002, str. 627–627.

[80] Hristidis, V., Papakonstantinou, Y., “Discover: Keyword search in relational databases”, in Proceedings of the 28th international conference on Very Large Data Bases. VLDB Endowment, 2002, str. 670–681.

[81] Xirogiannopoulos, K., Khurana, U., Deshpande, A., “GraphGen: Exploring interesting graphs in relational data”, Proceedings of the VLDB Endowment, Vol. 8, No. 12, 2015.

[82] Simmen, D., Schnaitter, K., Davis, J., He, Y., Lohariwala, S., Mysore, A., Shenoi, V., Tan, M., Xiao, Y., “Large-scale graph analytics in Aster 6: bringing context to big data discovery”, Proceedings of the VLDB Endowment, Vol. 7, No. 13, 2014, str. 1405–1416.

[83] Palod, S., “Transformation of relational database domain into graphs based domain for graph based data mining”, Department of Computer Science and Engineering, University of Texas at Arlington, Arlington, SAD, 2004.

[84] Sowa, J. F. Conceptual graph summary, dostupno na: http://www.jfsowa.com/cg/cgif.htm

[85] Pradhan, S., Chakravarthy, S., Telang, A., “Modeling relational data as graphs for mining.”, in COMAD. Citeseer, 2009.

[86] De Virgilio, R., Maccioni, A., Torlone, R., “Converting relational to graph databases”, in First International Workshop on Graph Data Management Experiences and Systems. ACM, 2013, str. 1.

[87] Jindal, A., Rawlani, P., Wu, E., Madden, S., Deshpande, A., Stonebraker, M., “Vertexica: your relational friend for graph analytics!”, Proceedings of the VLDB Endowment, Vol. 7, No. 13, 2014, str. 1669–1672.

[88] Fan, J., Gerald, A., Raj, S., Patel, J. M., “The case against specialized graph analytics engines”.

[89] Holme, P., “Modern temporal network theory: a colloquium”, The European Physical Journal B, Vol. 88, No. 9, 2015, str. 1–30.

[90] Huang, S., Cheng, J., Wu, H., “Temporal graph traversals: Definitions, algorithms, and applications”, arXiv preprint arXiv:1401.1919, 2014.

[91] Tang, J., Scellato, S., Musolesi, M., Mascolo, C., Latora, V., “Small-world behavior in time-varying graphs”, Physical Review E, Vol. 81, No. 5, 2010, str. 055101.

[92] Acer, U. G., Drineas, P., Abouzeid, A. A., “Random walks in time-graphs”, in Proceedings of the Second International Workshop on Mobile Opportunistic Networking. ACM, 2010, str. 93–100.

[93] Semertzidis, K., Pitoura, E., "Time traveling in graphs using a graph database", in Proceedings of the Workshops of the (EDBT/ICDT), 2016.

[94] Wackersreuther, B., Wackersreuther, P., Oswald, A., Böhm, C., Borgwardt, K. M., "Frequent subgraph discovery in dynamic networks", in Proceedings of the Eighth Workshop on Mining and Learning with Graphs. ACM, 2010, str. 155–162.

[95] Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N., "Time-varying graphs and dynamic networks", International Journal of Parallel, Emergent and Distributed Systems, Vol. 27, No. 5, 2012, str. 387–408.

[96] Kovanen, L., Karsai, M., Kaski, K., Kertész, J., Saramäki, J., "Temporal motifs in time-dependent networks", Journal of Statistical Mechanics: Theory and Experiment, Vol. 2011, No. 11, 2011, str. P11005.

[97] Hulovatyy, Y., Chen, H., Milenković, T., "Exploring the structure and function of temporal networks with dynamic graphlets", Bioinformatics, Vol. 31, No. 12, 2015, str. i171–i180.

[98] Bader, D. A., Berry, J., Amos-Binks, A., Chavarría-Miranda, D., Hastings, C., Madduri, K., Poulos, S. C., "Stinger: Spatio-temporal interaction networks and graphs (sting) extensible representation", Georgia Institute of Technology, Tech. Rep, 2009.

[99] Jiang, C., Coenen, F., Zito, M., "A survey of frequent subgraph mining algorithms", The Knowledge Engineering Review, Vol. 28, No. 01, 2013, str. 75–105.

[100] Anastasiu, D. C., Iverson, J., Smith, S., Karypis, G., "Big data frequent pattern mining", in Frequent Pattern Mining. Springer, 2014, str. 225–259.

[101] Lakshmi, K., Meyyappan, T., "A comparative study of frequent subgraph mining algorithms", International Journal of Information Technology Convergence and Services, Vol. 2, No. 2, 2012, str. 23.

[102] Holder, L. B., Cook, D. J., Djoko, S. *et al.*, "Substucture discovery in the SUBDUE system.", in KDD workshop, 1994, str. 169–180.

[103] Inokuchi, A., Washio, T., Motoda, H., "An apriori-based algorithm for mining frequent substructures from graph data", in Principles of Data Mining and Knowledge Discovery. Springer, 2000, str. 13–23.

[104] Kuramochi, M., Karypis, G., "Frequent subgraph discovery", in Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on. IEEE, 2001, str. 313–320.

[105] Borgelt, C., Berthold, M. R., "Mining molecular fragments: Finding relevant substructures of molecules", in Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on. IEEE, 2002, str. 51–58.

[106] Yan, X., Han, J., "gSpan: Graph-based substructure pattern mining", in Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on. IEEE, 2002, str. 721–724.

[107] Kuramochi, M., Karypis, G., "Finding frequent patterns in a large sparse graph*", Data mining and knowledge discovery, Vol. 11, No. 3, 2005, str. 243–271.

[108] Huan, J., Wang, W., Prins, J., "Efficient mining of frequent subgraphs in the presence of isomorphism", in Data Mining, 2003. ICDM 2003. Third IEEE International Conference on. IEEE, 2003, str. 549–552.

[109] Krishna, V., Suri, N., Athithan, G., "A comparative survey of algorithms for frequent subgraph discovery", Current Science(Bangalore), Vol. 100, No. 2, 2011, str. 190–198.

[110] Elseidy, M., Abdelhamid, E., Skiadopoulos, S., Kalnis, P., "GraMi: Frequent subgraph and pattern mining in a single large graph", Proceedings of the VLDB Endowment, Vol. 7, No. 7, 2014, str. 517–528.

[111] Moussaoui, M., Zaghdoud, M., Akaichi, J., "Posgrami: Possibilistic frequent subgraph mining in a single large graph", in International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems. Springer, 2016, str. 549–561.

[112] Abdelhamid, E., Abdelaziz, I., Kalnis, P., Khayyat, Z., Jamour, F., "Scalemine: scalable parallel frequent subgraph mining in a single large graph", in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press, 2016, str. 61.

[113] Gurukar, S., Ranu, S., Ravindran, B., "COMMIT: A scalable approach to mining communication motifs from dynamic networks", in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015, str. 475–489.

[114] Zong, B., Xiao, X., Li, Z., Wu, Z., Qian, Z., Yan, X., Singh, A. K., Jiang, G., "Behavior query discovery in system-generated temporal graphs", Proceedings of the VLDB Endowment, Vol. 9, No. 4, 2015, str. 240–251.

[115] Han, W., Miao, Y., Li, K., Wu, M., Yang, F., Zhou, L., Prabhakaran, V., Chen, W., Chen, E., "Chronos: a graph engine for temporal graph analysis", in Proceedings of the Ninth European Conference on Computer Systems. ACM, 2014, str. 1.

[116] Yang, Y., Yan, D., Wu, H., Cheng, J., Zhou, S., Lui, J., "Diversified temporal subgraph pattern mining", in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016, str. 1965–1974.

[117] "GraMi GitHub projekt", https://github.com/ehab-abdelhamid/GraMi, dohvat: 2016-05-13.

[118] Dhiman, A., Jain, S., "Optimizing frequent subgraph mining for single large graph", Procedia Computer Science, Vol. 89, 2016, str. 378–385.

[119] Akoglu, L., Tong, H., Koutra, D., "Graph based anomaly detection and description: a survey", Data Mining and Knowledge Discovery, Vol. 29, No. 3, 2015, str. 626–688.

[120] Akoglu, L., Faloutsos, C., "Anomaly, event, and fraud detection in large network datasets", in Proceedings of the sixth ACM international conference on Web search and data mining. ACM, 2013, str. 773–774.

[121] "Anomaly, event, and fraud detection in large network datasets - prezentacije", http://www3.cs.stonybrook.edu/~leman/wsdm13/, dohvat: 2016-05-15.

[122] Aggarwal, C., Subbian, K., "Evolutionary network analysis: A survey", ACM Computing Surveys (CSUR), Vol. 47, No. 1, 2014, str. 10.

[123] Henderson, K., Gallagher, B., Li, L., Akoglu, L., Eliassi-Rad, T., Tong, H., Faloutsos, C., "It's who you know: graph mining using recursive structural features", in Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2011, str. 663–671.

[124] Chakrabarti, D., "Autopart: Parameter-free graph partitioning and outlier detection", in Knowledge Discovery in Databases: PKDD 2004. Springer, 2004, str. 112–124.

[125] Noble, C. C., Cook, D. J., "Graph-based anomaly detection", in Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2003, str. 631–636.

[126] Gao, J., Liang, F., Fan, W., Wang, C., Sun, Y., Han, J., "On community outliers and their efficient detection in information networks", in Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2010, str. 813–822.

[127] Muller, E., Sánchez, P. I., Mulle, Y., Bohm, K., "Ranking outlier nodes in subspaces of attributed graphs", in Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on. IEEE, 2013, str. 216–222.

[128] Bunke, H., Dickinson, P. J., Kraetzl, M., Wallis, W. D., A graph-theoretic approach to enterprise network dynamics. Springer Science & Business Media, 2007, Vol. 24.

[129] Berlingerio, M., Koutra, D., Eliassi-Rad, T., Faloutsos, C., "NetSimile: a scalable approach to size-independent network similarity", arXiv preprint arXiv:1209.2684, 2012.

[130] Koutra, D., Vogelstein, J. T., Faloutsos, C., "DeltaCon: A principled massive-graph similarity function". SIAM, 2013.

[131] Papadimitriou, P., Dasdan, A., Garcia-Molina, H., "Web graph similarity for anomaly detection", Journal of Internet Services and Applications, Vol. 1, No. 1, 2010, str. 19–30.

[132] Akoglu, L., Faloutsos, C., "Event detection in time series of mobile communication graphs", in Army science conference, 2010, str. 77–79.

[133] Araujo, M., Papadimitriou, S., Günnemann, S., Faloutsos, C., Basu, P., Swami, A., Papalexakis, E. E., Koutra, D., "Com2: fast automatic discovery of temporal ('comet') communities", in Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, 2014, str. 271–283.

[134] Aggarwal, C. C., Zhao, Y., Philip, S. Y., "Outlier detection in graph streams", in Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011, str. 399–409.

[135] Heard, N. A., Weston, D. J., Platanioti, K., Hand, D. J. *et al.*, "Bayesian anomaly detection methods for social networks", The Annals of Applied Statistics, Vol. 4, No. 2, 2010, str. 645–662.

[136] Mongiovi, M., Bogdanov, P., Ranca, R., Papalexakis, E. E., Faloutsos, C., Singh, A. K., "Netspot: Spotting significant anomalous regions on dynamic networks", in Proceedings of the 2013 SIAM International Conference on Data Mining. SIAM, 2013, str. 28–36.

[137] Rossi, R. A., Gallagher, B., Neville, J., Henderson, K., "Modeling dynamic behavior in large evolving graphs", in Proceedings of the sixth ACM international conference on Web search and data mining. ACM, 2013, str. 667–676.

[138] Page, L., Brin, S., Motwani, R., Winograd, T., "The PageRank citation ranking: bringing order to the web.", 1999.

[139] Charikar, M. S., "Similarity estimation techniques from rounding algorithms", in Proceedings of the thiry-fourth annual ACM symposium on Theory of computing. ACM, 2002, str. 380–388.

[140] Henzinger, M., “Finding near-duplicate web pages: a large-scale evaluation of algorithms”, in Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2006, str. 284–291.

[141] Orel, O., Zakošek, S., Baranović, M., “Property oriented relational-to-graph database conversion”, Automatika–Journal for Control, Measurement, Electronics, Computing and Communications, Vol. 60, No. 1, 2017.

[142] Redmond, U., Cunningham, P., “Subgraph isomorphism in temporal networks”, arXiv preprint arXiv:1605.02174, 2016.

[143] Hooi, B., Song, H. A., Beutel, A., Shah, N., Shin, K., Faloutsos, C., “Fraudar: Bounding graph fraud in the face of camouflage”, in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016, str. 895–904.

[144] “IBM Informix database”, https://www-01.ibm.com/software/data/informix/, dohvat: 2016-05-18.

[145] “Informacijski sustav visokih učilišta”, http://www.isvu.hr, dohvat: 2016-05-18.

[146] “Počinje Indeks 3 - suđenje za korupciju na Ekonomskom fakultetu (Internet monitor)”, http://www.monitor.hr/clanci/pocinje-indeks-3-sudenje-za-korupciju-na-ekonomskom-fakultetu/21617/, dohvat: 2017-02-21.

[147] “Indeks 3: Glavni posrednik priznao krivnju (RTL vijesti)”, http://www.vijesti.rtl.hr/novosti/8271/indeks-3-glavni-posrednik-priznao-krivnju/, dohvat: 2017-02-21.

[148] “Indeks 3: Glavni posrednik Ćevid dobio dvije i pol god. (24 sata)”, http://www.24sata.hr/news/indeks-3-glavni-posrednik-cevid-dobio-dvije-i-pol-god-156728, dohvat: 2017-02-21.

[149] “Stack exchange”, http://stackexchange.com/, dohvat: 2016-08-03.

Popis oznaka

$\mathbf{r}$	relacijska baza podataka
r	relacija
R	relacijska shema
A	atribut relacije
t	n-torka
$t[A]$	vrijednost atributa A u n-torki
G	graf
v	vrh u grafu
e	brid u grafu
$\mathcal{G}$	grafovska baza podataka
$\_r$	relacija koja sadrži povijest podataka relacije r
$\_\mathbf{r}$	relacijska baza podataka koja sadrži povijest relacija relacijske baze podataka $\mathbf{r}$
G_T	potpuno vremenski određeni graf
δ	širina vremenskog susjedstva
ξ	odmak indeksa rubnih događaja vremenskog susjedstva
f	minimalna frekvencija čestog podgrafa
FSG	skup čestih podgrafova
$FTSG$	skup čestih vremenskih podgrafova
$\{(c_i, w_i)\}$	skup karakteristika c potpuno vremenski određenog grafa i njihovih težina w
φ	funkcija preslikavanja G_T u skup karakteristika

Hamming funkcija prebrojavanja razlika na istim pozicijama unutar dva bit-vektora

$sim_{SS}(G_{T,1}, G_{T,2})$ sličnost dva potpuno vremenski određena grafa

σ parametar maksimalnog odstupanja približno jednakih potpuno vremenski određenih grafova

A anomalija potpuno vremenski određenog grafa

f_A frekvencija anomalije potpuno vremenski određenog grafa

α_l donja granica frekvencije anomalije

α_u gornja granica frekvencije anomalije

A moguća zlouporaba sustava

Životopis

Ognjen Orel rođen je 13. studenog 1976. godine u Sarajevu, gdje živi do 1992. godine. Diplomirao je 2001. godine na studiju Računarstva na Fakultetu elektrotehnike i računarstva u Zagrebu. Na istom fakultetu je obranio magistarski rad pod nazivom "Nadzor nad radom korisnika u relacijskim bazama podataka" 2008. godine.

Od listopada 2001. godine je zaposlen u Sveučilišnom računskom centru Sveučilišta u Zagrebu, gdje je trenutno voditelj Službe za akademske informacijske sustave.

U okviru nastavnih djelatnosti na Fakultetu elektrotehnike i računarstva sudjelovao je u nastavi iz predmeta "Programiranje", "Uvod u baze podataka" i "Baze podataka". Na Tehničkom veleučilištu u Zagrebu je sudjelovao u nastavi iz predmeta "Programiranje".

Koautor je projekata "Informacijski sustav visokih učilišta", "Informacijski sustav Mozvag", a koautor i voditelj projekta "Informacijski sustav Registra Hrvatskog kvalifikacijskoga okvira".

Autor je ili koautor znanstvenih i stručnih radova, nastavnih materijala i projekata otvorenog koda. Znanstveno djelovanje je usmjerio u unaprjeđivanje sigurnosti informacijskih sustava i baza podataka. Također je zainteresiran za područja arhitekture, projektiranja i održavanja informacijskih sustava i baza podataka na kojima se oni temelje.

Popis objavljenih djela

Rad u časopisima

1. Orel, O., Zakošek, S., Baranović, M., "Property oriented relational-to-graph database conversion", Automatika–Journal for Control, Measurement, Electronics, Computing and Communications, Vol. 60, No. 1, 2017.

Biography

Ognjen Orel was born on November 13th, 1976 in Sarajevo, where he lived until 1992. He graduated at the Faculty of Electrical Engineering and Computing, University of Zagreb in 2001. At the same faculty, he defended his Master Thesis titled "Supervision of users in relational databases" in 2008.

Since October 2001 he has been employed at the University Computing Centre, University of Zagreb, where he currently works as the head of Academic Information Systems Department.

At the Faculty of Electrical Engineering and Computing he participated as teaching assistant in courses "Programming", "Introduction to Databases" and "Databases". Also, he participated in the course "Programming" at the Polytechnic of Zagreb.

He is co-author of "Information System for Higher Education", "Mozvag Information System" and also co-author and the lead of the "Croatian Qualifications Framework Registry Information System".

He is author or co-author of scientific, professional and educational publications in his field, as well as open-source projects. His scientific work is aimed towards improvement of information systems and database safety. He is also interested in topics covering information systems' and underlying databases' architecture, design and maintenance.