Tampereen yliopisto

Erkki Keränen

ALIGRAAFIEN LOUHINTA TIETÄMYSVERKOSTOSTA

Ilmiöiden paikantaminen epätäydellisillä tiedoilla

Diplomityö
Johtamisen ja talouden tiedekunta
Tarkastajat: Kari Systä
Timo Mäkinen
Helmikuu 2024

TIIVISTELMÄ

Erkki Keränen: Aligraafien louhinta tietämysverkostosta
Diplomityö
Tampereen yliopisto
Johtamisen ja tietotekniikan maisterikoulutus
Helmikuu 2024

Tässä diplomityössä tutkitaan olemassa olevan PINGS-algoritmin (Procedures for INvestigative Graph Search) soveltuvuutta kuvattujen ilmiöiden paikantamiseen geneerisestä tietämysverkostosta. Tietämysverkosto on graafi, joka kuvastaa kaikkia asioita ja asioiden välisiä yhteyksiä mitä tiedetään tarkasteltavasta kokonaisuudesta. Löydöksiä paikannetaan kuvaamalla kyselygraafi algoritmille. Kyselygraafi esittää ilmiön tietämysverkoston pienenä aligraafina, jolle halutaan löytää vastineita tietämysverkostosta. Kun tietoaineistosta tehdään haku, saadaan vastauksena samankaltaisia mutta ei välttämättä täydellisesti hakua vastaavia osumia. Alkuperäisen PINGS-algoritmin on kehittänyt Muramudalige et al.

Aluksi perehdytään alkuperäiseen PINGS-algoritmin toteutukseen sekä todetaan algoritmin toimivuus saatavilla olevien tietojen perusteella. Algoritmia kokeillaan alkuperäisten tekijöiden kehittämällä tietoaineistoilla.

Työn valmisteluvaiheen edetessä huomataan, että julkisesti saatavilla oleva ohjelmakoodi ei sellaisenaan toimi, joten tutkimustyön tekemistä varten toteutetaan algoritmin kuvausten perusteella PINGS-algoritmia noudattava lähdeaineistoriippumattomampi algoritmi. Geneerisen käsittelykyvyn arviointia varten luodaan keinotekoinen tietämysverkosto, joka koostuu tietokoneista, verkoista sekä haavoittuvuuksista. Kehitetty verkosto on rakenteeltaan kompleksisempi ja monipuolisempi kuin alkuperäisten tutkimusartikkelien testiaineisto.

Työn tuloksena todetaan, että PINGS-algoritmi voi karkeasti soveltua käyttäjän kuvaamien ilmiöiden louhimiseen geneerisestä epätäydellisestä tietämysverkostosta, mikäli toteutuksessa otetaan huomioon tässä tutkimuksessa kokeiltuja ja esitettyjä korjauksia.

Lopputuloksena algoritmin avulla voidaan paikantaa haavoittuneet, yhteenliitetyt tietokoneet sekä niiden väliset yhteydet. Lopuksi esitetään ehdotuksia jatkotutkimukselle sekä parannusehdotuksia algoritmin jatkokehitykselle.

Avainsanat: graafi, verkostoanalyysi, aligraafi, isomorfisuus, tietoturva

SAMMANDRAG

Erkki Keränen: Minering av subgrafer ur kunskapgsgrafer
Diplomarbete
Tammerfors universitet
Magistersutbildning i förvaltning och informationsteknik
Februari 2024

I detta diplomarbete undersöks hur den existerande PINGS-algoritmen (Procedures for INvestigative Graph Search) tillämpar sig för minering av beskrivna fenomen ur en generisk kunskapsgraf. En kunskapsgraf är ett nätverk av information som beskriver alla saker och förbindelser vad man vet om någon helhet. Fynden finner man med att beskriva det eftersökta fenomet med hjälp av ett frågediagram. Frågediagrammet representerar en subgraf vi vill finna i kunskapsgrafen. När man frågar kunskapsgrafen med ett frågediagram får man som resultat likadana subgrafer men inte nödvändigtvis exakta representationer av frågan. Den ursprungliga PINGS-algoritmen är utvecklad av Muramudalige et al.

Först studerar vi den ursprungliga PINGS-algoritmens implementation och konstaterar dess funktionering med hjälp av tillgänglig information. Vi provkör algoritmen med den allmänt tillgängliga datauppsättning som de ursprungliga utckeclarna producerat.

Under arbetets preparationsfas märker vi att den allmänt tillgängliga algoritmens källkod fungerar inte som sådan med generisk material. Vi besluter oss för att implementera en mer generiskt kapabel PINGS-algoritm med att följa informationen i källmaterialet. För att testa algoritmens funktion i generisk problemlösning, skapar vi ett syntetiskt kunskapsnätverk som består av datorer, nätverk och sårbarheter. Det syntetiska nätverket är mer kompleks och heterogen till struktur jämfört med källmaterialets exempelstudie.

Som resultat av detta arbete konstaterar vi att PINGS-algoritmen kan på grov nivå tillämpa sig till att minera beskrivna fenomen ur ett icke perfekt kunskapsnätverk. För en bättre prestation krävs dock att man tar i beaktande förbättringar som provas och presenteras i denna studie. Slutresultatet är att vi kan finna sårbara datasystem och förbindelser i ett nätverk. I sammandraget beskrivs också förslag till vidare studier och förbättring samt vidareutveckling av algoritmen.

Nyckelord: graf, nätverksanalys, subgraf, isomorfism, datasäkerhet

ABSTRACT

Erkki Keränen: Mining subgraphs in knowledge graphs
Master of Science Thesis
Tampere University
Master's degree education in management and information technology
February 2024

In this thesis we study how the existing PINGS-algorithm (Procedures for INvestigative Graph Search) can be used for finding described phenomena in a generic knowledge graph. The knowledge graph represents all identified entities (nodes) and connections (edges) that we assume to know about something as a network. We identify candidates, subgraphs, by describing the phenomena as a query graph. The query graph is a small representation of known entities and connections we assume to at least partially match. As we query the knowledge graph using the query graph, the result can contain similar but not necessary exact matches. The original PINGS-algorithm is developed by Muramudalige et al.

First, we study the original PINGS-algorithm and test the functionality on the basis of available information. We test the algorithm with a dataset that is created by the original developers and presented in the cited articles.

During the study, we realize that the publiced source code of the PINGS-algorithm doesn't work as such for the intended purpose. We decide to re-write the algorithm following the published articles and documentation to make the algorithm work for our knowledge graph. For evaluating the algorithm suitability for our case, we develop a synthetic knowledge graph consisting of computers, networks and vulnerabilities. This synthetized netowrk is more heterogenous of nature compared to the original material.

As a result of this work, we manage to find vulnerable computers and connections in the synthetic network. We discover that the PINGS-algorithm can roughly fit the purpose of mining described phenomena from an generic incomplete knowledge graph. We describe identified and proposed improvements and proposals of continued study that can improve the algorithm and similar solutions.

Keywords: graph, network analysis, subgraph, isomorphism, information security

ALKUSANAT

Tahdon kiittää teitä kaikkia, ketkä ovat jaksaneet kulkea mukanani tämän matkan. Tämän diplomityön tekemisestä muodostoi hiukan pidempi prosessi kuin mitä alunperin ajattelin. Työn ensiaskeleet otettiin syksyllä 2021.

Erityisesti haluan kiittää teitä näkymättömiä tukijoita, kenen kanssa keskustelin tästä työstä enemmän tai vähemmän satunnaisissa kohtaamisissa – sain paljon arvokkaita näkemyksiä aihepiirin potentiaalista ja uskoa omalle ajatukselleni – graafialgoritmeihin liittyvä tutkimustyö on arvokasta ja releventtia.

Yksi voimakas ajuri valmistumisen kannalta on ollut kannustus, mitä aikuisopiskelija voi saada muilta ihmisiltä. Monen tuntemattoman ja satunnaisen ihmisen kanssa käyty keskustelu kouluttautumisesta on osoittanut, että itsensä haastaminen ja kehittäminen on monen mielestä kunnioitettavaa. Näille kaikille ihmisille, joita en varsinaisesti tunne taikka tapaa – en tuottanut teille pettymystä tai pelkästään puhunut tekeväni asioita. Ilman teidän kannustusta tämä työ olisi ollut todella vaikea saattaa loppuun.

Erityiskiitokset haluan esittää professori Kari Syställe, joka jaksoi tukea, neuvoa ja haastaa minua. Ilman erinomaista palautetta ja muistuttelua työnteko olisi pysähtynyt. Kiitokset kuuluu myös Timo Mäkiselle, joka osallistui myös tämän työn tarkastamiseen.

Vantaalla, 5. helmikuuta 2024

Erkki Keränen

SISÄLLYSLUETTELO

KUVALUETTELO

OHJELMA- JA ALGORITMILUETTELO

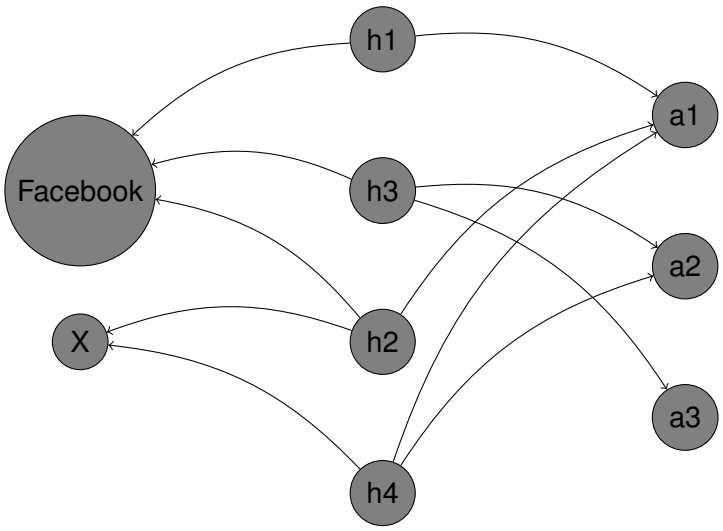
LYHENTEET JA MERKINNÄT

Cypher	Neo4J graafitietokannan käyttämä kyselykieli vrt. SQL
edge	Graafin viiva. kts. viiva
graafitietokanta	Graafitietokannassa tieto kuvataan pisteinä ja pisteiden välisinä yhteyksinä
Java	Ohjelmointikieli, jonka käännettyjä ohjelmia suoritetaan JVM-virtuaalikoneessa
JVM	Java Virtual Machine, joka suorittaa Java-ohjelmia
kaari	kts. viiva
Key-Value	Avain-arvo pari. Tällä voidaan tarkoittaa tietokantaa, johon tallennetaan tietoa yksinkertaisella avain-arvo pari -tietomallilla.
knowledge graph	kts. tietämysverkosto
line	Graafin viiva. kts. viiva
node	Graafin piste. kts. piste
NoSQL	Tietokantateknologia, jolla tarkoitetaan esim. dokumenttikantaa tai muuta ratkaisua jossa tietoa ei kuvata pääasiassa relaationa vrt. relaatiokanta
painoarvo	kts. weight
path	polku
PINGS	Procedures for INvestigative Graph Search -algoritmi
piste	Verkoston solmu, kts. vertex
point	Graafin piste. kts. piste
property	Graafin viivan tai pisteen ominaisuus, esim. "väri: sininen"
relaatiokanta	Tietokanta, jossa tietoa voidaan kuvata tietojen välisenä relaatioina
solmu	Verkoston piste, kts. piste
SQL	Structured Query Language, relaatioitietokantojen kyselykieli. kts relaatiokanta
tietämysverkosto	Verkoston muodostova graafi, joka esittää kaikkea saatavailla olevaa tietämystä jostain kokonaisuudesta

trail	Graafin reitti
vertex	Graafin piste. kts. piste, node, point
viiva	Graafin viiva eli kahden pisteen välinen yhteys. kts. edge, line
walk	Graafin kulku
weight	Graafin viivan painoarvo tai hinta, jonka avulla voidaan laskea esim. pisteiden välillä lyhyin, halvin tai tärkein reitti

1. JOHDANTO

Graafien avulla asioita ja niiden välisiä yhteyksiä voidaan esittää verkostona ja käsitellä graafia algoritmillisesti sekä visualisoida tietoa. Esitettävällä verkostolla voidaan asioina kuvata esimerkiksi henkilöitä, sosiaalisen median alustoja, keskusteluja ja keskustelujen aiheita. Vastaavasti verkoston yhteyksinä voidaan kuvata henkilöiden käyttämiä sosiaalisen median alustoja sekä henkilöiden osallistuminen keskusteluihin. Kuvassa 1.1 on esitetty esimerkki edellä mainitusta verkostosta, jolla kuvataan henkilöiden keskusteluaktiviteettiä eri alustoilla.



Kuva 1.1. *Henkilöiden (h) aktiviteetti sosiaalisissa medioissa eri keskusteluaiheissa (a)*

Kuvan 1.1 tietoa ei kokonsa ja rajallisen tiedon takia voida kuitenkaan pitää kattavana, mutta suuremmalla määrällä tietoa sekä monipuolisimmalla tiedoilla kyseinen graafi voisi pitää sisällään tiedon ketkä tuottavat tietoa jostakin aiheesta ja ketkä sitä kuluttavat.

Graafista ilmenevän tiedon perusteella voidaan muodostaa johtopäätöksiä asioiden välisestä vuorovaikutuksesta – esim. mistä aiheista henkilöt ovat kiinnostuneita tai missä henkilöt ovat vuorovaikutuksessa samasta aiheesta muiden ihmisten kanssa. Täydentämällä graafia lisätiedoilla, kuten henkilöiden välisillä yhteyksillä, kiinnostuksen kohteilla

tai tykkäyksillä, saamme rikastettua graafia. Saamme esiin uutta löydettävää tietoa, joka aikaisemmin oli piilossa.

Jos graafin tiedot ovat vääriä tarkasteltavien osien kohdalta, saattaa se vinoutuneilta osin vaikuttaa johtopäätösten todenperäisyyteen. Oikean tiedon suhde virheelliseen tietoon ja analyysin kohteena olevaan asiakokonaisuuteen määrittelee sen, mikä on virheellisen tiedon vaikutus. Sama pätee puuttuvaan tietoon – voi olla että ei voida löytää jotain ilmiselvää tietoa, jos graafi ei kuvasta puuttuvaa tekijää.

Graafien käytännön sovellutuksia ovat esim. verkostoanalyysi ja erilaiset optimointitehtävät logistiikan ja navigoinnin alalla. Yksinkertaisia graafeja voidaan tutkia graafisesti riittävän yksinkertaisella kysymyksellä, kun taas suuria graafeja voi tutkia tietokoneen suorittamilla algoritmeillä. Graafin kannalta ei ole merkityksellistä mitä tietoa tutkitaan kunhan tutkittavan tiedon voi kuvata graafina. Suurempi merkitys on käyttää ratkaistavaan kysymykseen soveltuvaa algoritmia.

Konkreettisesti graafeja ja niiden ominaisuuksia tarkastellessa esitetään kysymyksiä, joihin tarkasteltavasta graafista toivotaan löytyvän vastaus. Esimerkiksi:

> Esiintykö aligraafi H graafissa G?
>
> Onko pisteestä A kulkua pisteeseen D?
>
> Kuinka lyhyt on lyhin kulku pisteestä C pisteeseen B?
>
> Onko piste N pisteen O vierekkäinen piste?

Toisin sanoen tyypillisesti vastaus on yleensä yksiselitteisesti jokin (mahdollisesti tyhjä) joukko, piste, viiva, lukuarvo, tosi tai epätosi.

Todellisuudessa ratkaistavat ongelmat ja niistä syntyvät kysymykset saattavat kuitenkin olla epämääräisesti määriteltyjä. Voi hyvinkin olla, että sinänsä relevanttin kysymyksen asettelija ei tiedä kaikkea tai vallitsevan tiedon valossa ei voi tarkasti esittää oikeaa kysymystä. Olemme mahdollisesti taltioineet suuren määrän tietoa, mahdollisesti jopa mallinnettu johonkin tietokantaan loogisiin relaatioihin. Voi myös olla että kysymyksen asettaja ei tarkalleen tiedä odotettavan vastauksen tarkkaa muotoa tai laatua. Erittäin todennäköistä on myös, että taltioidussa tiedossa on puutteita – oleellisia entiteettejä puuttuu, entiteettien tiedot on väärin tai oleellisia relaatioita ei ole.

Esimerkiksi yleisimmät relatiokantamoottorit ovat hyviä ja nopeita vastaamaan seuraaviin kysymyksiin:

> Kuuluuko alkio A joukkoon S kun otetaan huomioon x, y ja z?
>
> Mikä on yrityksen G osoite?

Tämä pätee myös yleisellä tasolla NoSQL-, Key-value- sekä graafikantoihin yksinkertaisessa tiedonhaussa. Tietokantajärjestelmän on huomattavasti vaikeampaa vastata seu-

raaviin kysymyksiin, riippumatta siitä onko tietoa paljon vai vähän – huomioiden tiedossa mahdollisesti olevia virheitä tai puutteita:

> Voisiko myös alkiot B ja D potentiaalisesti kuulua edellä mainittun joukkoon S?
>
> Mitä kanavia on olemassa, jolla voimme mahdollisesti epäsuorasti saavuttaa yrityksen G?
>
> Mitä samankaltaisia asioita kuin C?

Nämä huomattavasti haastavammat kysymykset ovat sellaisia, joiden vastaukset kertovat asioita joita me emme välttämättä voi suoraan nähdä tai osaa kysyä oikein. Triviaalissa ja korrektissa tietomassassa nämä voidaan päätellä – mutta voitaisiinko nämä ehdokkaat louhia esiin suuresta massasta tiekoneavusteisesti?

1.1 Tutkimuksen tavoitteet ja tutkimuskysymykset

Tässä työssä tarkastellaan samankaltaisen tiedon etsimistä graafista käyttäen pieneen aligraafiin pohjautuvia kyselyjä eli kyselygraafeja. Tämän työn kontekstissa samankaltaisella tiedolla tarkoitetaan toistuvaa kuviota, joka graafissa ilmenee tietynlaisten solmujen välisenä yhteytenä. Kyselygraafilla kuvataan tunnistettava ilmiö tarkasteltavan graafin entiteettien ja relaatioiden välisinä yhteyksinä. Graafeista löydetään kyselyä muistuttavia, samankaltaisia vastineita.

Tämän työn tavoitteena on kokeilla, miten tietämysgraafina kuvatun yrityksen tietokoneiden muodostaman verkkoympäristön konfiguraation haavoittuvuuksia voidaan paikantaa soveltamalla Muramudalige et al kehittämää PINGS-algoritmia[1] (Procedures for INvestigative Graph Search) ja kuinka hyvin algoritmi soveltuu johonkin muuhun käyttökohteeseen kuin alkuperäisissä tutkimustyössä käytettyyn materiaaliin. Tämän työn tutkimustyyppi on exploratiivinen tapaustutkimus, jonka tavoitteena on luoda aiheeseen liittyviä uusia ideoita ja löytää mahdollisia selityksiä tutkittavalle ilmiölle[2, s. 13–14].

Tässä työssä vastataan seuraaviin tutkimuskysymyksiin:

1. Mitä PINGS-algoritmin käyttöönotto vaatii ts. miten se otetaan käyttöön?
2. Miten PINGS-algoritmia voitaisiin käyttää työkaluna kuvattujen kokonaisuuksien paikantamisessa tietoaineistosta?
3. Miten PINGS-algoritmia voitaisiin jatkokehittää yleishyödyllisemmäksi työkaluksi?

Tutkimuskysymysten vastaukset muodostavat työn tavoitteiden ohjaaman lopputuleman, joka esitellään luvussa 5. Tutkimustyö kokonaisuutena simuloi tilannetta, miten jokin tutkimusartikkelissa esitetty graafialgoritmi voitaisiin valjastaa liiketoimintakäyttöön ja millaisella menestyksellä. Edellä kuvattu simuloitu tilanne on tutkimuksen kannalta mielenkiintoinen siksi että sillä voidaan arvioida tutkimuksen alla olevien menetelmien valmiutta ja

potentiaalia suhteessa valmiisiin ratkaisuihin.

Tässä työssä tehtävä graafialgoritmejä käsittelevä tutkimustyö on merkityksellistä ja ajankohtaista tietojen käsittelyssä, vaikka koneoppimis- ja tekoälysovellukset ovat saaneet paljon huomiota ja ovat omissa kategorioissaan tärkeitä. Yleisellä tasolla suurilla tietomassaoilla ennustaminen tai kyky nähdä metsä puilta on keinoja, joiden merkitystä ei tulisi aliarvioida.

1.2 Aikaisempi tutkimus

Tämän työn keskiössä oleviin teemoihin, eli tietämysgraafien ja kyselygraafien käyttöön, on tehty aikaisempaa tutkimusta Muramudalige et al lisäksi. Tietämysgraafien louhintaa kyselygraafien avulla on tutkittu toisenlaisella algoritmilla Sun et al artikkelissa A subgraph matching algorithm based on subgraph index for knowledge graph[3]. Kyselygraafien muodostamisesta ja käytöstä on käsitelty artikkelissa Multi-Example Search in Rich Information Graphs[4]. Aligraafien isomorfisuusprobleemaa, johon Muramudalige et al tutkimuskohteensa luokittelevat, on tutkittu mm. Kowaluk et al artikkeleissa Are unique subgraphs not easier to find?[5].

2. TEOREETTINEN TAUSTA

Tämän työn teoreettisena taustana tuodaan esille tehtävän tutkimuksen ja menetelmän ymmärtämisen kannalta oleellisia määritelmiä graafeista, käsitellään algoritmien laskennallisen kompleksisuuden arviointia, esitellään tutkittava PINGS-algoritmi ja sen toiminta sekä arvioidaan graafien tallentamista erityyppisiin tietokantaratkaisuihin. Graafien persistointi on oleellinen asia kun huomioidaan käytännöllistä tietokoneohjelmaa, samalla tavalla kuin minkä tahansa muun tärkeän tiedon luotettavaa tallentamista tehokasta hakua varten. Luvussa 2.6 määritellään keskeisiä käsitteitä koskien tutkimusmenetelmää ja aineistoa.

2.1 Graafi

Yksinkertaisuudessaan voidaan sanoa että graafi on kokoelma pisteitä, jotka mahdollisesti yhdistyvät toisiinsa viivoilla.

Olkoon $G = (V, E)$, ts. graafi G koostuu joukosta solmuja, tai pisteitä, V sekä joukosta kaaria, tai viivoja, V. Kaaret kytkevät solmut toisiinsa. Graafin G pistejoukko on $V(G)$ ja vastaavasti viivajoukko on $E(G)$. Graafi voidaan esittää piirtämällä kuva pisteistä ja niiden välisistä viivoista. [6, s. 2]

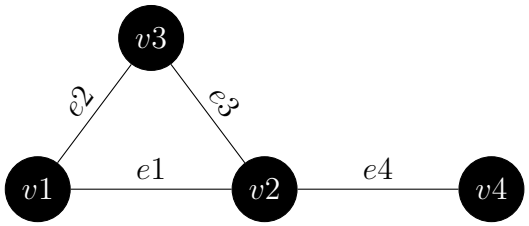
Viivalla on kaksi pistettä, mitkä määrittelevät minkä pisteiden välille viiva piirretään.

2.1.1 Graafin kuvaaminen viivoina ja pisteinä

Graafit voidaan piirrosteknisesti kuvata pisteinä ja viivoina. [6, s. 2] Kuvassa 2.1 esitetään yksinkertainen suuntaamaton graafi. Kuvan graafista voidaan todeta että $V(G) = \{v1, v2, v3, v4\}$ ja $E(G) = \{\{v1, v2\}, \{v1, v3\}, \{v2, v3\}, \{v2, v4\}\}$.

2.1.2 Graafin matriisiesitys

Graafit voidaan myös esittää matriiseina. Yksi esitystapa on insidenssimatriisi (engl. incidency matrix) $A(G)$. Matriisin rivit vastaavat pisteitä ja sarakkeet viivoja. Arvo 1 merkitsee että kyseiseseen pisteeseen yhdistyy jokin viiva kun vastaavasti 0 kertoo että yhteyttä ei ole. [7, s. 96–98]



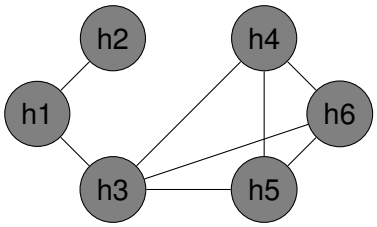
Kuva 2.1. *Yksinkertainen suuntaamaton esimerkkigraafi* G*.*

Kun G on kuvassa 2.1 esitetty graafi, määritellään sen insidenssimatriisi seuraavanlaisesti:

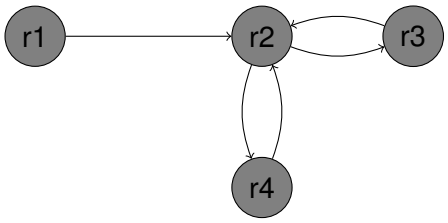
$$A(G) = \begin{array}{c} \\ v1 \\ v2 \\ v3 \\ v4 \end{array} \begin{array}{c} \begin{array}{cccc} e1 & e2 & e3 & e4 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Toinen esitystapa on vierekkäisyysmatriisi $X(G)$. Matriisissa rivit ja sarakkeet vastaavat solmuja, arvot 1 (on) ja 0 (ei ole) vastaavasti onko pisteiden välillä viivaa ts. ovatko pisteet vierekkäisiä. [7, s. 101–102] Vastaavalla tavalla kuvassa 2.1 esitetyn graafin G vierekkäisyysmatriisi on:

$$X(G) = \begin{array}{c} \\ v1 \\ v2 \\ v3 \\ v4 \end{array} \begin{array}{c} \begin{array}{cccc} v1 & v2 & v3 & v4 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$



Kuva 2.2. *Suuntaamaton graafi. Verkosto, joka koostuu henkilöistä ja miten henkilöt tuntevat toisensa.*



Kuva 2.3. *Suunnattu graafi. Pisteillä kuvataan risteyksiä ja viivoilla ajo-yhteyksiä pienessä katuverkostossa.*

2.1.3 Suuntamattomat ja suunnatut graafit

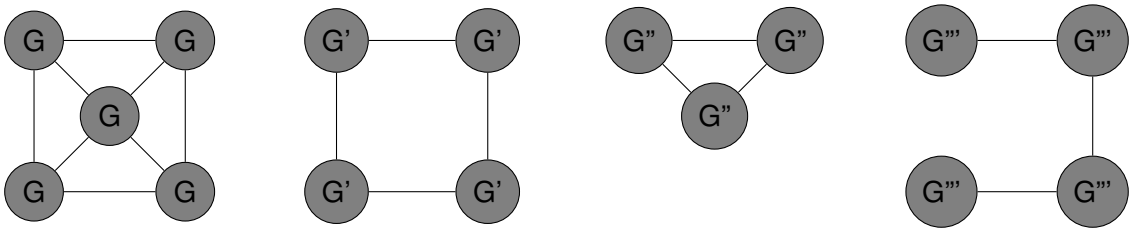
Graafi voi olla suuntaamaton [7, s. 1–2][6, s. 2] tai suunnattu [7, s. 9–10][6, s. 27–28]. Suuntaamattomissa graafeissa ei ole merkitystä miten päin pisteiden välistä yhteyttä kuljetaan [8, s. 196] eli miten päin pisteiden välistä relaatiota tulkitaan. Esimerkkinä kuvassa 2.2 esitetyt henkilöt $h1$ ja $h2$ tuntevat toisensa, ts. henkilö $h1$ tuntee henkilön $h2$ ja vastaavasti henkilö $h2$ tuntee henkilön $h1$.

Suunnatuissa graafeissa taas viivojen suunnilla on merkitystä. Esimerkkinä [8, s. 196] voisi olla kaupungin katuverkostoa kuvaava graafi. Kuvassa 2.3 on esitetty neljän risteyksen (pisteet $r1$–$r4$) väliset ajo-yhteydet (suunnatut viivat). Risteyksestä $r1$ on yksisuuntainen ajo-yhteys risteykseen $r2$ ts. risteyksestä $r2$ ei ole ajo-yhteyttä risteykseen $r1$ vrt risteysten $r2$ ja $r3$ välillä on kaksisuuntainen ajo-yhteys. Risteyksistä $r3$ ja $r4$ voi myös ajaa takaisinpäin risteykseen $r2$.

Suunnattuja graafeja voidaan käyttää sellaisissa sovellutuksissa, jossa yksisuuntaisuudella on merkitystä. Suunnatuun graafiin voi tuoda suuntaamattomia ominaisuuksia lisäämällä puuttuvan yhteyden myös toiseen suuntaan kahden pisten väliltä. Graafialgoritmi voi olla tehty joko sunnattua tai suuntamattomaa graafia varten, tai vaihtoehdot on otettava huomioon algoritmiä tehtäessä. Esimerkiksi tämän työn keskiössä oleva PINGS-algoritmi on kirjoitettu suunnattua graafia varten [9].

2.1.4 Aligraafi

Graafi G' on aligraafi graafille G jos $G' \subseteq G$ ja $G' \neq G$ [6, s. 4]. Yksinkertaistettuna, jos graafin G' pisteet ja viivat löytyvät graafista G, on G' aligraafi graafille G. Kuvassa 2.4 on esitetty muutama aligraafi, jotka voidaan muodostaa esimerkkigraafille.



Kuva 2.4. *Graafi G ja aligraafit $G' \subseteq G$, $G'' \subseteq G$ sekä $G''' \subseteq G$*

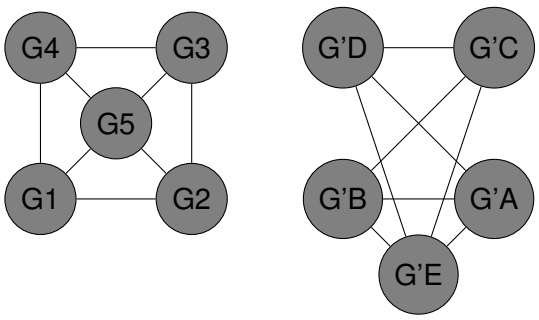
Koska yksikin piste voi muodostaa triviaalin graafin[6, s. 2], voidaan kuvan 2.4 esimerkin perusteella päätellä että suuressa graafissa on suuri määrä aligraafeja. Aligraafien lukumäärällä ja potentiaalisilla koolla voi täten olla suuri laskennallinen merkitys algoritmeissä, jotka käsittelevät aligraafeja suuressa tietoaineistossa.

2.1.5 Graafien isomorfisuus

Isomorfisuudella voidaan tarkoittaa vertailtavien asioiden samankaltaisuutta. Graafien välistä isomorfisuutta voidaan yksinkertaistettuna arvioida siten, ovatko ne saman muotoisia.

Tarkastellaan esimerkkinä ovatko graafit G ja G' isomorfisia. Mikäli jokaista paria pisteitä kohden on vastaavat viivat molemmissa graafeissa, ovat graafit isomorfisia. Samanmuotoisella ei ole merkitystä miten ne kuvataan eli piirrettään, vaan onko vastaavilla pisteillä vastaavia yhteyksiä. [7, s. 5–6][8, s. 199–200][6, s. 3, 99–100]

Mikäli pisteet olisi nimetty, voitaisiin graafejä myös tarkastella nimeämättöminä ja todeta



Kuva 2.5. *Graafi G ja graafi ovat isomorfisia, ts. $G \simeq G$*

niiden keskinäinen samanmuotoisuus [8, s. 200]. Kuvassa 2.5 on esitetty kahden graafin välinen isomorfisuus – pisteiden nimillä, piirtotavalla tai keskinäisellä sijainnilla ei ole merkitystä.

2.1.6 Kulku, reitti ja polku

Navigaatiossa, kustannusoptimoinnissa ja verkostoanalyysissä saatetaan olla kiinnostuneita graafin kahden pisteen välisestä lyhyimmästä tai halvimmasta matkasta. Graafin pisteiden välillä kuljetut matkat (käytyjen pisteiden joukko sekä kuljettujen viivojen joukko) muodostavat aligraafeja tarkastetulle graafille. Kulkuja voi olla useita ja eripituisia. Kulun pituus mitataan siirtymien määrässä pisteestä seuraavaan yhdistävää viivaa pitkin. $W \equiv v$ ts. kulku pisteestä v pisteeseen v – kulun pituus on triviaali eli 0.[7, s. 14]

Riippumatta kulkua hyödyntävän graafialgoritmin toimintatavasta, arvioidaanko kahden eri pisteen välisen kulun, reitin tai polun pituutta – pituuden ollessa 1, on kyseessä vierekkäinen piste.

Kulku

Kulku (engl. walk) on matka kahden graafin pisteen välillä, jossa jokaisessa matkan pisteessä saa käydä useammin kuin kerran ja jokaista yhdistävää viivaa pitkin voi kulkea useasti.[7, s. 14]

Reitti

Reitti (engl. trail) on matka kahden graafin pisteen välillä, jossa matkan jokaisessa pisteessä saa käydä useammin kuin kerran ja jokaista yhdistävää viivaa pitkin voi kulkea

korkeintaan kerran. Jokainen reitti on myös tämän määritelmän mukaan kulku.[7, s. 15]

Polku

Polku (engl. path) on matka kahden graafin pisteen välillä, jossa matkan jokaisessa pisteessä saa käydä vain kerran ja jokaista yhdistävää viivaa pitkin voi kulkea korkeintaan kerran. Jokainen polku on myös tämän määritelmän mukaan reitti.[7, s. 15]

2.2 Graafialgoritmit ja probleemat

Graafialgoritmien avulla etsitään vastauksia kysymyksiin, esimerkiksi mikä on lyhyin polku kahden pisteen välillä. Teknisesti graafialgoritmit käsittelevät pisteitä, niiden välisiä yhteyksiä ja ominaisuuksia kuten esimerkiksi yhteyden hintaa.

Probleemat ovat haasteita, joihin ei välttämättä ole tarjolla täydellisiä ratkaisuja mutta voivat toimia ajurina tietojenkäsittelytieteen tutkimukselle ja innovaatioille. Esimerkiksi yksi tunnettu graafialgoritmeihin liittyvä probleema on Traveling Salesman problem. Ratkaistava probleema on miten myyntimies kiertää n kaupunkia mahdollisimman tehokkaasti, käyden jokaisessa kaupungissa kerran ja päätyen samaan kaupunkiin josta lähti.[8, s. 530, 535] Tämä työ ei käsittele probleemoja, mutta huomioidaan että aligraafien louhiminen suurista graafeista voi täyttää probleeman tunnusmerkit.

Tämän työn kontekstissa algoritmilla tai hypoteettisella probleemalla on kompleksisuus, joka voi johtua joko toteutuksesta tai ratkaistavan ongelman laadusta.

2.2.1 Algoritmien kompleksisuus

Tässä työssä algoritmin kompleksisuudella tarkoitetaan yleisesti kuinka paljon tietokoneohjelma joutuisi laskemaan ja mahdollisesti uudelleenkäsittelemään jo laskettuja osia graafista. Kompleksisuutta lisäävillä seikoilla voi olla merkittävä vaikutus algoritmin vaatimaan suoritusaikaan ja laskentatehoon sekä muistintarpeeseen.

Tämä työ ei käsittele suoranaisesti algoritmien tehokkuutta tai suorituskykyä mutta algoritmien kompleksisuuden arviointi tulisi olla oleellinen osa algoritmien suunnitteluprosessia ja arviointia. Olettama voisi yksinkertaistettuna olla, että jokin tarkoitusta varten kehitettävä algoritmi kykenee ratkaisemaan annetun ongelman kohtuullisessa ajassa suhteessa käytettävissä oleviin resursseihin. Toisin sanoen, investoitu aika (laskenta) ja raha (koneet) tuottaa riittävän (odotettavan) määrän arvoa (tietoa). Esimerkiksi Muramudalige et al tuottamissa julkaisuissa[1, 10] yksi tulosten arviointikriteeri on PINGS-Algoritmin suorituskyky Neo4J-graafitietokannassa erisuuruisille tietomäärille ja kyselyparametreille.

Algoritmien kompleksisuuden arviointi ja merkitys

Yhden määritelmän mukaan tietokoneiden suorittamien algoritmien kompleksisuutta voidaan mitata kuinka tehokkaasti Turingin kone suorittaa niitä[8, s. 513–516]. Bonan määritelmän perusteella voitaisiin tulkita että kompleksisuus liittyy jollain tavalla tehokkaaseen suorittamiseen. Käytännönläheisempi ja arkisempi tapa arvioida algoritmien kompleksisuutta tehokkuusnäkökulmasta olisi algoritmin suoritusajan arviointi vakiosyötteillä sekä vakioaineistolla määritetyllä määrällä laskentatehoa, esim. yleisesti saatavilla oleva testiaineisto ja tietynlainen yleisesti saatavilla oleva palvelin.

Kompleksisuudella ei tämän työn kontekstissa viitata algoritmin kompleksisuuteen monimutkaisuuden kannalta ts. ymmärrettävyyden, vaan kompleksisuudella tarkoitetaan kuinka monipuolisen syvällisiä laskutoimituksia tai päättelyä algoritmi kykenee tekemään, joka itsessään tarvitsisi jonkinlaista kompleksisuutta. Kompleksisuus voi joko lisätä tai vähentää suoritusaikaa, riippuen siitä onko kompleksisuus laskentaa lisäävää vai optimoivaa.

Jokin valittu tapa mitata algoritmin kompleksisuutta auttaa algoritmi-implementaatioiden ja eri ongelmaratkaisumenetelmien keskinäistä vertailua. Näin voidaan algoritmeja vertailla objektiivisesti keskenään resurssintarpeen ja ajankäytön näkökulmasta, jolloin tiedetään onko tarkastelun kohteena olevan ratkaisun käyttäminen taloudellista ja realistista suhteessa käsiteltävän syötteen tietomäärään, käytettävissä olevaan aikaan, saatavilla oleviin resursseihin ja saavutettavaan hyötyyn.

Kun kompleksisuuden arvioimisella mitataan tehokkuutta, voi jokin tehottomampi algoritmi olla parempi saavutettavan hyödyn näkökulmasta jos sen avulla saadaan odotusarvojen kannalta parempia vastauksia mitä tehokkaampi algoritmi ei löydä. Näin ollen algoritmien keskinäinen vertailu voidaan ainoastaan tehdä yhteneväisten ominaisuuksien osilta, jos esim. keskenään vertailtavat algoritmit löytävät yksiselitteisesti samasta tietomassasta saman vastauksen. Voi myös olla, että vastausten kannalta parempi algoritmi on liian kompleksinen, jolloin parempia vastauksia antavaa algoritmia on parannettava käyttökelpoiseksi tai lisättävä laskentatehoa.

Big O-notaatio

Olkoon $n > 0$ algoritmin syötteen alkioiden lukumäärä. Mikä on algoritmin kompleksisuus suhteessa käsiteltävän syötteen kokoon?

Asymptoottisiin notaatioihin kuuluvat mm. O, o, Θ ja Ω[11, s. 8]. Edellä mainituista notaatioista tässä työssä käytetään Big O-notaatiota O, koska kyseistä notaatiota esiintyy alan julkaisuissa ja on pääasiassa käytetty tämän työn keskeisimmissä lähteissä[1, 7, 8] kuvaamaan algoritmin kompleksisuutta. Seuraavaksi esitetään karkea yleistys jonka pitäisi

olla riittävä alustus tämän työn ymmärtämisen kannalta.

$$f(n) = O(g(n)) \tag{2.1}$$

Benoitin kirjassa[11, s. 8] esitetyssä kaavassa 2.1 on esimerkkinä kuvattu funktio $f(n)$, joka algoritmina palauttaa vastauksen annetulle syötteelle jossa n on käsiteltävien alkioiden lukumäärä ja $g(n)$ funktio, joka palauttaa funktion suorituksen kompleksisuuden, josta voimme tulkita funktion maksimisuoritusajan. Huomio tulkinnasta O-notaation liittymisestä funktion maksimisuoritusaikaan on aiemmin tuotu esiin asymptoottisia notaatioita käsittelevässä Knuthin artikkelissa Big Omicron and big Omega and big Theta[12, s. 18].

Esimerkiksi yksinkertainen funktio, joka aina palauttaa kokonaisluvun 1 syötteestä riippumatta, olisi kompleksisuudeltaan $O(1)$ riippumatta syötteen koosta. Voidaan sanoa, että näin yksinkertaisen funktion syötteen koolla ei ole merkitystä kompleksisuuden kannalta, kompleksisuuden ollessa vakio.

Vastaavasti funktio joka laskee yhteen syötetyt alkiot edustaisi lineaarista (tasaisesti kasvavaa) kompleksisuutta ja kuvattaisiin kompleksisuudella $O(n)$, koska kaikki alkiot on käsiteltävä samalla metodilla. Eli summaavan funktion $f(\{1, 2, 3, 4, 5\}) = 1 + 2 + 3 + 4 + 5$ kompleksisuus olisi $O(n) = O(5)$ koska $n = 5$ ts. kaikki alkiot on käsiteltävä. Kompleksisuus voi myös olla muuta esim. $O(n^2)$ tai $O(1 + n/2)$ mikä kuvaisi funktion tiedetyn kompleksisuuden kasvua suhteessa syötteen määrään.

Tämän työn kontekstissa riittää, että Big O-notaatiolla kuvataan algoritmin syötteen koon vaikutusta käsittelyaikaan. Huomioidaan, että asymptoottinen notaatio ei edellä esitetyn määritelmän mukaan huomioi laskutoimitustuen säikeistämistä tai hajauttamista eri tietokoneille joten esitetyt määritelmät eivät välttämättä suoranaisesti sovellu käytännönläheiseen tarkasteluun.

Probleemat ja kompleksisuusluokat P ja NP

Graafiteoriaasta mainittakoot kompleksisuusluokat P ja NP, koska kyseiset käsitteet saattavat esiintyä aihepiirin julkaisuissa.

Tutkittavan algoritmin kompleksisuus kuuluu luokkaan P jos on olemassa Turingin kone T mikäli algoritmin tulos muodostuu ajassa $O(n^k)$ ts. vastaus löytyy n^k Turingin koneen askeleella. Täten P on ongelma, joka on ratkaistavissa polynomisessa ajassa. Näin ollen triviaalein kompleksisuus on n^1, koska pelkästään syötteen lukemiseen kuluu n askelta. [8, s. 516–517]

NP-täydellisellä ongelmalla tarkoitetaan sitä, että algoritmien vastaukset voidaan varmistaa polynomisessa ajassa $O(n^k)$ tai jopa nopeammin, mutta tiedossa ei ole löytääkö algoritmi vastauksen polynomisessa ajassa[8, s. 518]. Näin ollen NP-luokkaan edustavat

probleemaa, eli haastetta johon ei ole selkeää ratkaisua tai tietoa sen tarkasta kompleksisuudesta syötteen koolle $n \in Z^+$.

2.3 Graafit verkostoanalyysin työkaluna

Graafilla voidaan kuvata tietoa verkostona eli asioiden ja asioiden välisten yhteyksien muodossa[13, s. 4]. Verkostossa asiat vastaavat graafin pisteitä ja asioiden yhteydet graafin viivoja. Näin käsiteltävää tietoa on mahdollista hyödyntää sekä analysoida visuaalisesti että laskennallisesti. Kuvassa 2.2 on graafina kuvattu kuuden henkilön muodostama verkosto pisteinä ja viivoina.

Näin ollen graafien pisteillä voidaan esittää erillaisia asioita kuten esim. henkilöitä, paikkoja ja esineitä. Viivat taas muodostavat yhteyksiä pisteiden välille ja voivat kuvata erilaisia relaatioita kuten esimerkiksi henkilöt $h1$ ja $h2$ tuntevat toisensa, henkilöllä $h1$ on auto $a1$, osoitteesta $o1$ on kulkuyhteys osoitteeseen $o2$ tai autossa $a1$ on pysäköintitutka $p1$.

Sosiaalisen median kasvaneen merkityksen myötä verkostoanalyysi käsittelee sosiaalisten verkostojen analyysiä mutta menetelmät soveltuvat muunkinlaisen yhteyksellisten rakenteiden tarkasteluun kuten esimerkiksi yritysten väliseen maksuliikenteeseen. Scott esittää kirjassa Social Network Analysis, että verkostoanalyysillä kyetään analysoimaan kohteena olevaa verkostoa ja täten nähdä rakenteita ja seurauksia toiminnalle, joka ei tarkastelussa muuten tulisi ilmi[14, s. 2]. Voidan sanoa, että Scottin kuvaama rakenteiden ja seurauksien huomioiminen verkostosta on lähestymiskulma, jonka avulla PINGS-algoritmin soveltuvuutta tässä työssä tutkitaan.

2.4 PINGS-algoritmi

Tämän työn keskeisenä osana on PINGS-algoritmin hyödyntäminen. Kyseisen algoritmin on kehittänyt Muramudalige et al ratkaisuna epätarkkojen hakujen suorittamiseen verkostoista. Algoritmi kuuluu aligraafin isomorfisuutta tutkivien menetelmien luokkaan, joka edustaa NP-täydellisiä ongelmia. [10, s. 60–61].

PINGS-algoritmin avulla kohdistetaan parametrisoituja kyselyjä kyselygraafin muodossa tietämysgraafille. Kyselygraafilla kuvataan jokinnäköinen löydös, esim. henkilö joka omistaa auton sekä tuntee kesämökin omistavan henkilön. Tietämysgraafi voisi koostua aineistosta jossa on tietoa henkilöistä, autoista, kesämökeistä ja niiden välisistä suhteista. Algoritmilla voitaisiin löytää tietämysgraafista vastauksena pieniä aligraafeja, jotka koostuvat kaikista autollisista henkilöistä ketkä tuntevat kesämökin omistavat henkilöt. Sallivammalla parametrisoinnilla voitaisiin sallia pieniä poikkeamia, jolloin vastaukseksi saataisiin myös autottomat henkilöt jotka tuntevat kesämökin omistavia henkilöitä tai henkilöitä jotka omistavat sekä auton että kesämökin. Kyselygraafin periaate esitellään tarkemmin luvussa 2.6.3 ja vastaavasti tietämysgraafi kuvataan luvussa 2.6.1.

PINGS-algoritmikokonaisuus tarjoaa seuraavat suoritettavat alialgoritmit:

- $individualSimilarity$ jolla etsitään kyselyä vastaavia aligraafeja jotka ovat yksilöllisesti samankaltaisia,
- $neighborhoodSimilarity$ jolla etsitään kyselyä vastaavia aligraafien verkostoja
- ja $implicitIndividualSimilarity$ jolla voidaan parametrein tarkentaa hakua jolla haetaan yksilöllisesti samankaltaisia tuloksia.

PINGS-algoritmin alialgoritmien toiminta käsitellään tarkemmin luvussa 3, jossa todetaan alkuperäisen algoritmikoodin olevan rajoittunut geneeristä aineistoa käsiteltäessä, huomioiden tämän työn tutkimuskysymykset. Esiselvitysvaihe on osa tämän työn tutkimusmenetelmää, ja algoritmin uudelleenkirjoitus ja tarkastelu muodostuu oleelliseksi osaksi menetelmää.

Yksilöllinen samankaltaisuus on suora käännös alkuperäisen algoritmin alialgoritmin englanninkielisestä nimestä, individual similarity. Nimi saattaa viitata siihen, että Muramudalige et al tutkivat verkostoja henkilö-entiteettien näkökulmasta, jolloin löydöksistä voidaan puhua samankaltaisista yksilöistä.

Algoritmien $individualSimilarity$ ja $neighborhoodSimilarity$ käyttöä valmistellaan seuraavilla parametreilla:

- $0 \leq similarityTreshold \leq 1$, jolla määritellään pienin samankaltaisuus jonka perusteella löydetty graafi tulkitaan samankaltaiseksi.
- $1 \leq redFlagMultiple \leq \infty$, jolla voidaan kasvattaa tiettyjen löydettyjen graafien samankaltaisuus pisteytystä tietyn solmu- tai yhteystyypin esiintyessä.
- $queryLabel$ lipuke, jolla kyselygraafin noodit on merkitty. Alkuperäisen algoritmikoodin ja Neo4J-tietokantaan liittyvä implementaatioyksityiskohta joka ei suoraan myötävaikuta algoritmin toimintaan.
- $queryFocusLabel$ lipuke, jolla kyselygraafin alkupiste voidaan yksiselitteisesti yksilöidä. Määrittelee solmutyypin, josta tietämysgraafin tutkiminen alkaa.

Poikkeavista ominaisuuksista johtuen algoritmi $implicitIndividualSimilarity$ ei saa kyselygraafia parametriksi, vaan parametrit ovat:

- $entity$ – lipuke, jolla noodin tyyppi valitaan,
- $identifier$ – $entity$-parametrilla viitatun noodin ominaisuuden (property) nimi,
- $identifierValue$ – $identifier$ -parametrilla viitatun ominaisuuden arvo,
- $relationshipToFocus$ – oleellinen relaatio minkä oletetaan liittyvän $entity$ -noodiin tai -noodista,
- $redFlagMultiple$ (kts. $individualSimilarity$ ja $neighborhoodSimilarity$),

- ja $similarityThreshold$ (kts. $individualSimilarity$ ja $neighborhoodSimilarity$).

2.5 Graafin mallintaminen tietovarantoon

Tämä osio käsittelee yleisellä tasolla graafien persistointia ja miten eri tietokantaratkaisut hyödyttävät hakemista ja louhimista. Jotta graafeja voitaisiin tutkia, versioida ja jakaa, tulee ne kyetä tallentamaan jollain menetelmällä. Graafit tulisi voida tallentaa johonkin tietovarantoratkaisuun, joka tukee tiedon tehokasta hakemista ja päivittämistä. Lisäksi käytännön syistä liiketoimintakäytössä varmuuskopiointi sekä yleisesti saatavilla olevat teknologiat helpottavat valitun ratkaisun elinkaaren hallintaa.

2.5.1 Relaatiokannat

Relaatiotietokantoja käytetään tietoteknisissä sovelluksissa tiedon tallentamiseen ja käsittelyyn. Yleisesti tuettu ja hallittu SQL-kieli (Structured Query Language) on laajalti tuettu ja ymmärretty. SQL-kieli tukee mm. rekursiivisia funktioita useassa tietokantamoottorissa, joten graafin tallentamiseen ja monipuoliseen käsittelyyn relaatiokannassa ei ole teknisiä esteitä, eikä rekursiivisten funktioiden puute olisi myöskään este graafien käsittelyyn.

Tämän työn tutkimusosassa mallinnetaan graafikanta SQL-kantaan, käyttäen pohjana yksinkertaista tietomallia. Syy relaatiokannan valinnalle tässä työssä oli yleisesti saatavilla oleva dokumentaatio, olemassa oleva osaaminen ja monipuoliset ohjelmointimahdollisuudet.

Graafin tallennus tietokantaa varten tietomallin vähimmäismääritys pitää sisällään tiedot kaavasta $G = (V, E)$, eli graafi G on joukko pisteitä V sekä joukko viivoja $E(vx, vy)$ pisteiden vx ja vy välillä. Kuvassa 2.6 on esitetty yksinkertainen relaatiomalli yhden suunnatun graafin tallentamista varten.

Joissain relaatiokannoissa saattaa olla ominaisuuksia, optimointia tai lisäosia graafien käsittelyä varten, mutta kohtuullisen pienissä tapauksissa kuten tässä työssä, voi muistin ja suorituskyvyn riittäessä graafialgoritmit kirjoittaa SQL-kyselyinä tai ohjelmakoodina esim. osana Java-sovelluksen toteutusta. Relaatiokannat tukevat yleensä myös keinoja kirjoittaa aliohjelmia, stored procedure, joita tietokanta voi suorittaa. Nämä relaatiokannassa käytetyt tallennetut proseduurit kirjoitetaan SQL-kielellä ja niitä voidaan syöttää kyselyrajapinnasta tietokannan ollessa käynnissä. Proseduurien syntaksi tai saatavilla oleva toiminnallisuus saattaa poiketa tietokantajärjestelmän valmistajasta riippuen, mutta periaate on sama.

Kuva 2.6. *Yksinkertaistettu relaatiomalli suunnatun graafin tallentamista varten, UML-notaatiolla kuvattu*

2.5.2 Graafitietokannat

Lissandrini et al mukaan[15] graafitietokantojen suosio on kasvanut sitä myötä kun mm. verkostoanalysointi ja tietämysgraafien käsittely on yleistynyt. Koska graafikannat itsessään eivät ole olleet olemassa niin kauan kuin esimerkiksi relaatiokannat, ei graafikannoilla ole standardoitua toteutustapaa jolloin eri ratkaisut voivat poiketa merkittävästi suorituskyvyltään tai ominaisuuksiltaan. Artikkelissa Beyond macrobenchmarks: microbenchmark-based graph database evaluation Lissandrini et al suorittavat suorituskykyvertailua eri graafikantatoteutuksien välillä. Artikkelista käy ilmi graafikantatoteutuksien tekniset erot niin toteutuksen[15, s. 392] kuin suorituskyvyn kannalta[15, s. 397–399]. Lisäksi vertailussa on graafitoteutus PostgreSQL-tietokantaan, joka suoriutuu vertailukelpoisesti[15, s. 401] sopivalla kuormalla.

Graafitietokannat ovat nimensä mukaan optimoituja graafimuotoisen tiedon tallentamiseen ja käsittelemiseen. Graafikannat sisältävät myös luonnostaan algoritmejä graafien käsittelyä varten. Muramudalige et al [1, 10] ovat alunperin toteuttaneet PINGS-algoritmin Java-kielellä Neo4J-graafikantaa varten, ja tutkimukset tutkivat Neo4J-tietokannan suorituskykyä tutkimusmateriaalilla ja algoritmin kyselyillä.

Koska graafikannoilla ei ole samankaltaisia standardeja tai historiallisia perinteitä kuten relaatiokannoilla, voi tallennettujen proseduurien tai mukautettujen toimintojen tuottaminen poiketa merkittävästi eri tietokantavalmistajien välillä. Esimerkiksi tarkastellessa PINGS-algoritmin alkuperäistä toteutusta[9] Neo4J-tietokantaan huomataan, että algoritmi on toteutettu Java-sovelluksena jota tietokantamoottoria suorittava JVM (Java Virtual Machine) ajaa. Neo4J-tietokanta tuo käyttäjän luomat tallennetut proseduurit saataville tietokannan käynnistysvaiheessa luetuista soveltuvien Java-binäärien metodeista. Esimerkki tallennetun proseduurin kehittämisestä Neo4J-tietokannalle on esitetty Matt Holfordin Neo4J Developer Blogille kirjoitetusta artikkelista[16] Let's Write a Stored Procedure in Neo4j – Part I. Verrattuna relaatiokantojen tallennettuihin proseduureihin, joiden käyttöönotto on verraten nopeaa, vaatii Neo4J tietokannan uudelleenkäynnistämisen saadakseen uuden tai muutetun Java-aliohjelman ajoon.

2.5.3 Key-value store, dokumenttikannat ja muut tietovarannot

Mainittakoon, että on myös olemassa muunlaisia tallennusjärjestelmiä, joihin voisi teoriassa tallentaa graafeja. Key-value store, eli avain–arvo-pari tallennusjärjestelmä voidaan nähdä kaksisarakkeiseina relaatiokannan tauluna, jossa avaimen arvona voi olla käytännössä mitä tahansa. Esimerkiksi Redis[17] ja etcd[18] ovat esimerkkejä avain- arvoparitallennusjärjestelmistä. Avain- arvoparitietomalli ei kuitenkaan ole tehokas monimutkaisiin hakuihin jossa viitataan ristiin aineistossa. Lisäksi arvojen käsitteleminen muuna kuin yhtenä arvona vaatii ylimääräisiä luku- ja käsittelyoperaatioita. Sen sijaan avain- arvoparitallennusta voidaan hyödyntää tehokkaana välimuistina, josta saadaan nopeasti suuria määriä tietoa käsittelyyn, kun esimerkiksi tallennettu arvo voidaan suoraan lukea muistiin tietoa hyödyntävässä sovelluksessa. [19, 20]

Vastaavalla tavalla dokumenttikannat ratkaisevat skeemattoman dokumentin tallentamisen avain- arvopari menetelmällä, mutta tarjoavat lisäominaisuuksia tiedon indeksointiin ja hakemiseen dokumenteista.[21] Yleensä tallennusmuoto on JSON (JavaScript Object Notation), esimerkiksi Amazon DocumentDB:ssä[22] jonka rajapinta on yhteensopiva MongoDB dokumenttikannan kanssa. Graafien tallentamisen näkökulmasta dokumenttikannassa JSON-rakenteeseen tallennettuja graafeja voitaisiin käyttää välimuistina, kuten avain- arvoparikannoissa mutta eivät tarjoa mitään merkittävää etua graafien persistointiin.

2.5.4 Graafien persistointiin liittyvät tila- ja muistitarpeet

Riippumatta tietovarannosta ja tallennusmuodosta, ei graafeissa itsessään ole mitään mikä muodostaisi suurta painetta tallenustila- tai muistitarpeiden suhteen. Resurssivaatimuksiin voi vaikuttaa tallennusjärjestelmän käyttämät avaimet, esim. onko kyseessä 32-bittinen kokonaisluku vai 128-bittinen UUID (Universal Unique Identifier), saattaa tallennuskapasiteetin tarve nelinkertaistua pelkästään avaimien ja vierasavaimien osalta. Lisäksi graafissa oleva metatiedon määrä vaikuttaa myös merkittävästi tallennuskapasiteetin tarpeeseen.

Yksinkertaisuudessaan graafien koostuessa ainoastaan pisteistä ja viivoista ja pienestä määrästä metatietoa, voidaan olettaa graafien vaatiman tallennustilatarpeen tai muistitarpeet käsittelyssä olevan melko maltillisia. Esimerkiksi tätä työtä varten käytettyjen graafien tallennukseen käytetyn PostgreSQL-tietokannan tallennuskoko levyllä on n. 10 megatavua. Tietokantaan on tallennettu n. 1500 solmua, 1500 kaarta ja 1500 solmun metatietoarvoa. Avaimet ja vierasavaimet ovat tutkimustoteutuksessa kooltaan 32-bittisiä kokonaislukuja.

Riippuen taustalla olevasta teknisestä ratkaisusta ja miten hakuja voi tehostaa, voidaan luoda hakuindeksejä tai näkymiä, joiden avulla suuresta tietojoukosta saadaan nopeam-

min tietoa ulos. Esimerkiksi graafissa jossa on paljon kaaria solmujen välillä, hakua voidaan tehostaa käytettävän muistin kustannuksella luomalla vierekkäisten solmujen näkymä. Indeksit ovat hakuluetteloita, jotka mahdollistavat jonkin asian tehokkaan löytämisen muun kuin relaation pääavaimen perusteella.

Tutkimustietokannan koko tässä työssä on melko pieni, eikä sisällä indeksejä tai välimuistiin tallennettuja näkymiä. Karkeasti voidaan arvioida että n. 350000 solmusta koostuvaa ei-triviaalia testiaineistoa vastaava tietomassa tarvitsisi tallennustilaa muutama sata megatavua riippuen metatiedon ja yhteyksien suhteellisesta määrästä. Tässä tapauksessa tietomassa mahtuisi nykyaikaisen tietokoneen keskusmuistiin kokonsa puolesta, ottamatta kantaan tietokantajärjestelmän tallennustekniikkaan, indeksointiratkaisuihin tai itse tiedon käsittelyyn eli miten paljon algoritmit, tietokantaoperaatiot tai tulosjoukot tarvitsevat muistia.

2.6 Määritelmiä

Tässä osassa käsitellään keskeisiä määritelmiä tämän työn ymmärtämisen kannalta. Osa määritelmistä saattaa muistuttaa olemassa olevia vastaavia määritelmiä. Esitettävät tai johdetut määritelmät pätevät tämän työn konteksissa.

2.6.1 Tietämysgraafi

Tietämysgraafilla, knowledge graph[1, s. 1][4], tarkoitetaan verkostona mallinnettua tietoa jostain asiasta. Tämän työn kontekstissa tietämysgraafi muodostetaan saatavilla olevasta tiedosta, joka koostuu solmuista ja viivoista. Solmut ovat substantiivejä, esimerkiksi henkilö, auto, viesti tai tietokone. Viivat ovat verbejä, jotka kuvaavat tavan miten solmut yhdistyvät toisiinsa, kuten tuntee, omistaa, lähettää tai käyttää. Solmuilla voi olla relevantteja attribuutteja kuten esimerkiksi nimi, osoite, rekisterinumero tai sarjanumero. Vastaavasti viivoilla voi olla attribuutteja, jotka kuvaavat toimintaa ja sen ulottuvuutta esim. tapahtuman aikaleima, kesto tai menetelmä.

Tietämysgraafin tulee olla riittävän tarkka tutkittavaan asiaan, eli graafin tulee olla tietosisällöltään riittävä asetettuun kymykseen. Ideaalitilanteessa graafin on ladattu kaikki tieto mitä on saatavilla. Riittävän tarkalla tarkoitetaan esitettävän kysymyksen kannalta relevantit yhteydet ja asiat, joiden perusteella asiaa voidaan tutkia. Attribuuttien avulla voidaan kyselyn avulla tarkentaa, korostaa tai hylätä tietyntyyppisiä löydöksiä.

Jos tietämysgraafilla kuvattaisiin esimerkiksi tutkimuksen kohteena olevien tietokoneiden välistä verkkoa, voisi solmut kuvata

palvelimia,

työasemia,

ohjelmistoja,

käyttäjiä,

haavoittuvuuksia tai

lähiverkkoja.

Graafin viivat edellä mainitulle solmuille voitaisiin kuvata verbeillä

kytkee (työasema kytkeytyy lähiverkkoon),

suorittaa (palvelin suorittaa ohjelmaa),

käyttää (käyttäjä tai palvelin käyttää ohjelmaa)

liikennöi (työasema liikennöi lähiverkon kautta palvelimelle) ja

sisältää (ohjelma sisältää haavoittuvuuden).

2.6.2 Epätarkka tai täydentyvä tietämysgraafi

Kuten oikeassa elämässä, tietoa voi olla aluksi vähän tai tieto voi olla väärää. Tietoa tulee rikastaa kunnes sitä on riittävästi ja löydetty väärä tieto joko poistaa, korjata tai merkitä. Täydentyvällä tietämysgraafilla on mahdollista saada oikeampaa ja relevantimpaa tietoa, mitä paremmaksi se muuttuu, vaikka se ei koskaan saavuttaisi täydellisyyttä. Epätarkan vääristyneet tai vääristymiin viittaavat vastaukset voivat antaa vihjettä siitä, mitä aspektia tietämysgraafista tulisi parantaa tai korjata.

Epätarkka graafi voitaisiin verrata suureen toleranssisallimaan, jolloin esitetylle kysymykselle saattaisi tulla paljon vääriä vastausehdotuksia. Tarkemmalla tiedolla toleranssia voitaisiin parantaa. Tarkentuvalla tietämysgraafilla tulosjoukkojen pisteyttämisen ja järjestäminen pisteiden mukaan voi helpottaa vastausten arviointa, kun taas epätarkalla tietämysgraafilla tulosjoukosta on jälkikäteen arvioitava useampi mahdollisten vääristymien takia.

Täydentyvää tietämysgraafia tulisi tutkia samoilla kysymyksillä aina tarkennusten jälkeen ja verrata aikaisempiin vastauksiin ja vallitsevaan käsitykseen asioiden oikeasta tilasta jolla täydennys on tehty. Tällä voidaan varmistaa että graafi ja sen käyttö (esimerkiksi algoritmi) toimii oikein.

Jos esimerkkinä luvussa 2.6.1 esitettyä tietämysgraafia haluttaisiin tarkentaa, voisi haavoittuvuuksia käsitellä ohjelmiston versioiden mukaan lisäämällä solmutyypin versio ja muuttamalla graafia siten, että suoritettavat ohjelmat ovat versioissa joilla taas on ohjelmaversiosta tiedossa olevia haavoittuvuuksia.

2.6.3 Kyselygraafi

Kyselygraafi, query graph[1, 10], on graafi, joka kuvaa graafikannasta tehtävää kyselyä. Kyselygraafin tulee sisältää samantyyppisiä solmuja ja viivoja kuten tietämysgraafi. Tämän myötä kyselygraafin esittämän kysymyksen vastauksen tulee myös toteutua ainakin osittain tietämysgraafissa. Kuten tietämysgraafia, kyselygraafia voi täydentää attribuuteilla. Kyselygraafin attribuuteilla voidaan rajata tai korostaa tiettyjen attribuuttien arvojen esiintyvyyttä. Lisäksi attribuutille tulisi voida asettaa painoarvo, mikä voisi korostaa tietyn attribuutin esiintymistä tulosjoukossa osassa 2.6.4 kuvattavassa löydöksessä. Painoarvoilla voidaan määritellä pienempien löydösten merkittävyyttä suhteessa suuriin löydöksiin, joissa ei ole korkeasti painotettuja arvoja.

Lissandrini et al käsittelevät artikkelissa Multi-Example Search in Rich Information Graphs [4] haun kuvaamista joukolla esimerkkejä jotka löytyvät tietämysgraafista, joiden avulla algoritmi voisi löytää samankaltaisia ilmentymiä. Haettavien asioiden kuvaaminen konkreettisella esimerkillä on sama, mitä kyselygraafilla tavoitellaan, epätarkkuus sallien.

2.6.4 Löydös

Löydös, match graph, on tietämysgraafin aligraafi joka vastaa joiltain osin kyselygraafia. Löydös sisältää vähintään yhden kyselygraafissa mainitun solmun ja attribuutit ja siitä kyselygraafia noudattavaa verkostoa solmujen, attribuuttien ja viivojen suhteen. Suuresta graafista voidaan rajata tarkasteltavia löydöksiä tulosjoukosta laskemalla niiden laatu suhteessa kykyyn muistuttaa kyselygraafia. Tietämysgraafista voidaan löytää nolla tai useampi kyselyä vastaava löydös. Laskennassa otetaan huomioon löydöksessä kyselygraafia vastaavien solmujen ja viivojen muodostoma rakenne, sekä kyselygraafissa erikseen mainittujen attribuuttien painotukset.

2.6.5 Hyökkäysgraafi

Terminä hyökkäysgraafi, attack graph, on visuaalinen tapa kuvata keinoa, jolla verkkoympäristöön voidan hyökätä. Esimerkkinä hyökkäysgraafien muodostamiselle voidaan käyttää mallia artikkelista Attack graph generation and analysis techniques [23]. Hyökkäysgraafeista ja niiden hyödyntämisestä on tehty aikaisempaa tutkimusta[24][25][26][27].

Tässä työssä on tarkoitus soveltaa työn kontekstissa hyökkäysgraafin käsitettä kyselygraafeja muodostoaessa. Toisin sanoen, hyökkäysgraafi on kyselygraafi, jolla selvitetään keinoa käyttää hyödykseen verkostossa kuvattua kokonaisuutta. Hyökkäysgraafi terminä voi myös olla käänteinen puolustusmielessä, eli tutkitaan toteutuuko verkostossa muutostoimenpiteitä aiheuttava uhka. Tämän työn kontekstissa hyökkäysgraafi voisi etsiä vastauksia kysymyksiin jotka voisivat laajennetussa tutkimuksessa liittyä luvussa 4 tutkitun

yrityksen tietämysverkoston tietoturvallisuuteen:

> Mitkä tilat on saavutettavissa kulkuoikeuksilla X jos minulla on pääsy huoneeseen Y josta saan yleisavaimen T_n?
>
> Mitkä yrityksen palvelimet ovat haavoittuvaisia käyttäjän välittämälle liitetiedostotyypille R?
>
> Onko yrityksen tietoliikenneverkossa haavoittuvuuksia, jotka altistavat tietojärjestelmät haavoittuvuudelle?
>
> Liittyykö joidenkin toimipaikkojen tietokoneisiin huomioon otettavia riskejä?

Hyökkäysgraafeja muodostaessa muodostetaan sopivia kyselyitä, joiden avulla voidaan löytää potentiaalisia puutteita (riskejä) olemassa olevan tiedon avulla. Luvussa 4 esitetyt haavoittuvuuden sisältävät kyselygraafit muodostavat yksinkertaistettuja hyökkäysgraafeja, joilla etsitään yhdessä haavoittuneita verkoston osia, vaikka hyökkäysgraafi-termiä ei tässä työssä aktiivisesti käytetä.

3. TUTKIMUSMENETELMÄ JA AINEISTO

Tämän työn tutkimusmenetelmän pohjana käytetään Muramudalige et al [1, s. 3] artikkelissa esitettyä PINGS-algoritmiä (Procedures for INvestigative Graph Search). Artikkelissa[1] esitetty tutkimistyö on jatkoa aikaisemmalle tutkimukselle [10]. Tämän työn tutkimusmenetelmänä on PINGS-algoritmin käyttöönotto ja sen kokeilu tätä työtä varten keinotekoisesti luodulle aineistolle ja aineiston aiheeseen sopiviin kyselygraafeihin, kuten Muramudalige et al ovat vastaavasti tehneet aikaisemmissa töissään[1][10]. Keinotekoisesti luotu aineisto kuvataan yleisellä tasolla luvussa 3.2 ja sitä aineistoa käsitellään kokonaisuudessaan luvussa 4.

Muramudalige et al käyttävät esimerkeissään generoitua tietoaineistoa[28], joka käsittää radikalisoituneiden henkilöiden verkostoa. Tämä muodostaa tietämysgraafin (knowledge graph) joka sisältää verkoston koostuen henkilöistä ja radikalisoitumisesta indikoivia tekoja jotka liittyvät toisiinsa. Algoritmin avulla tietämysgraafista voidaan etsiä kyselygraafilla (query graph) Q:

> Etsimme henkilöä joka on saanut taistelijakoulutusta ja osallistuu Twitterissä (nyk. X) ja Facebookissa radikalisoituneeseen keskusteluun ($similarityMeasure$).
>
> Etsimme henkilöitä ja mahdollisia samalla tavalla toimivaa verkoston osaa eli joukkoa potentiaalisia yhteenliittyviä henkilöitä ($neighborhoodSimilarity$).

Tietämysgraafista etsitään potentiaalisia löydöksiä kyselygraafin Q avulla. Q kuvaa karkeasti etsittävän aligraafin. Esimerkki Muramudalige et al mukaan on esitetty kuvassa 3.1. Tutkimuksissa[1][10] esitetään että toteutulla algoritmilla ja kyselygraafeilla voidaan tehokkaasti löytää potentiaaliset löydökset tietämysverkostosta.

3.1 Tekninen menetelmä

Tässä tutkimustyössä hyödynnetään Muramudalige et al artikkeleita varten kehitettyä, avoimena lähdekoodina GitHubissa julkaistua Neo4J graafikantaan toteutettua PINGS-algoritmia[9]. Algoritmi kirjoitettiin uudestaan lähdemateriaalien[10][1][9] perusteella. Alkuperäinen tavoite oli toteuttaa tämä työ alkuperäisillä teknologiavalinnoilla seuraten lähdeartikkeleita, mutta aineistoon perehdyttäessä huomattiin haasteita käyttää olemassa olevaa toteutusta. Seuraavaksi käsitellään alkuperäisen saatavilla olevan PINGS-algoritmin

Kuva 3.1. *Esimerkkikyselygraafi Q jonka avulla halutaan löytää tietyntyyppisiä henkilöitä. Toteutettu [10, s. 63] mukaan.*

kokeilua, josta seurasi PINGS-algoritmin uudelleen implementointi mahdollisimman tarkasti lähdemateriaalia noudattaen. Lopuksi kuvataan mahdollisimman tarkasti tässä tutkimustyössä implementoidun teknisen menetelmän lopullisesta toteutuksesta ja käyttöönotosta.

Pings-algoritmin toimivuus oli tarkoitus todentaa ennen käyttöönottoa ja tämän työn sovellutusta olemassa olevalla synteettisellä aineistolla [28], jonka Muramudalige et al kehittivät artikkeleitaan varten [1, 10]. Tietoaineisto sisältää 100, 1000, 10000 ja 100000 solmun verkoston esimerkit, joita on käytetty algoritmin toiminnan testaamiseen ja suorituskyvyn mittaamiseen[10, s. 66–67][1, s. 11–13].

Alunperin graafit ja kyselyt oli tarkoitus toteuttaa Neo4J-tietokannassa käyttäen alkuperäistä avoimen lähdekoodin PINGS-algoritmia, jotta voitaisiin varmistaa kokeen toistettavuus ja algoritmin toiminta alkuperäisten artikkelien mukaan. Tällä perustalla voitaisiin tutkia, voidaanko samalla menetelmällä louhia potentiaalisia haavoittuvuuksia graafina kuvatusta yritysverkko- ja palvelinkonfiguraatiosta.

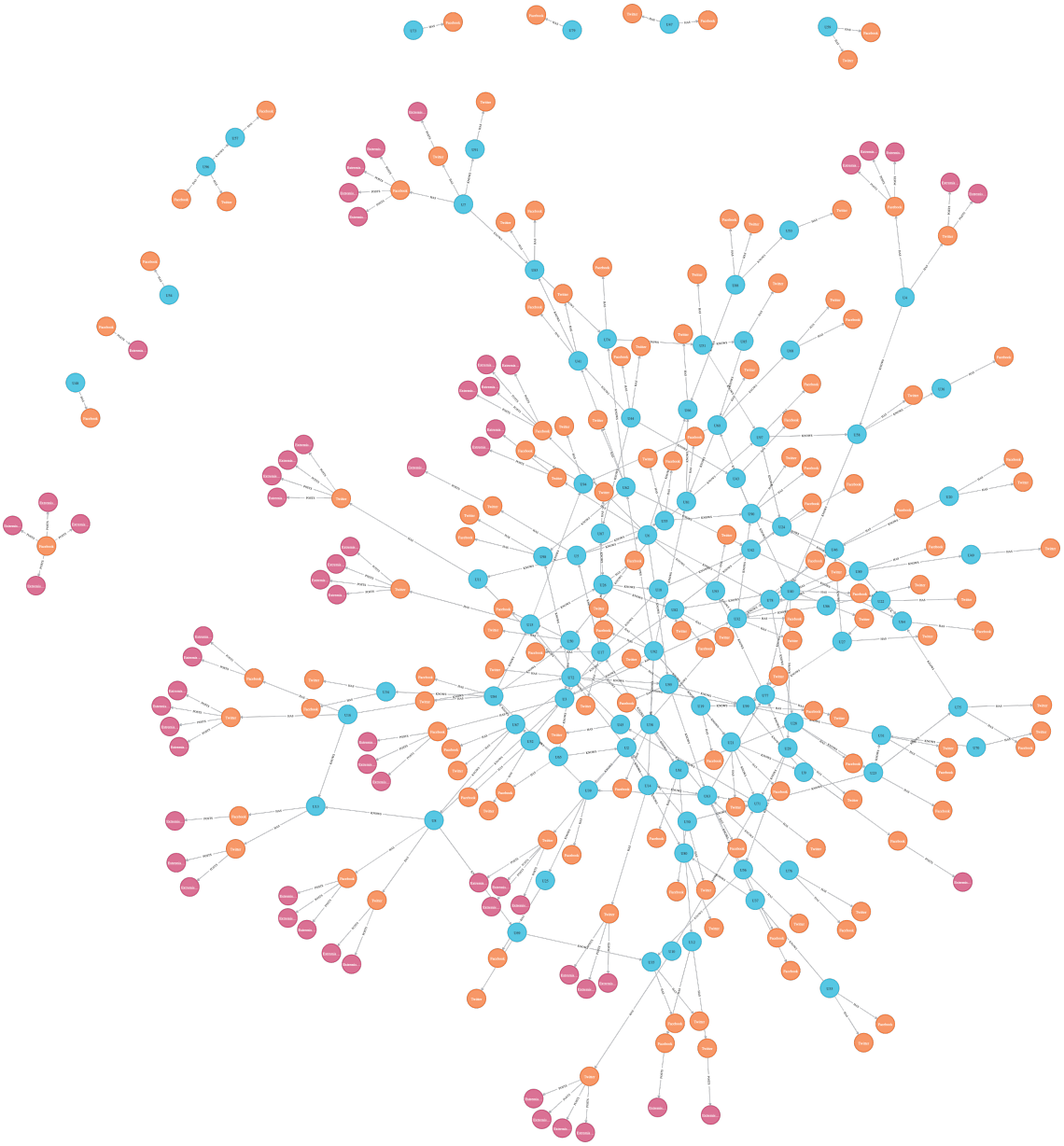
3.1.1 Esimerkkiaineiston valmistelu Pings-algoritmin toiminnan todentamista varten

Koska Muramudalige et al tai Colorado State University eivät ole tarkemmin kuvanneet miten CSV-muodossa oleva esimerkkiaineiston[28] otetaan käyttöön [1, 9, 10, 28], päätettiin muodostaa Neo4J tuontikomennot soveltamalla tietokannan kehittäjän ohjeistusta[29]. Graafin pisteet ja viivat muodostettiin CSV-tiedostoista olettamalla seuraavin olettamuksin:

- Tiedostot, joiden nimet sisältävät substantiivin ja tietosisältö ei sisällä ymmärrettäviä viitteitä muihin pisteisiin, muodostettiin solmuja.
- Tiedostot, joiden nimet sisältävän verbin ja tietosisältö sisältää ymmärrettäviä viitteitä muihin pisteisiin, muodostettiin kaaria.

Cypher-koodilla toteutettu ohjelma csv-tiedostojen sisäänajo Neo4J-tietokantaan on kokonaisuudessa esitetty ohjelmalistauksessa A.1.

Edellä mainittujen olettamusten pohjalta luotiin Neo4J-tuontikoodi soveltaen tietokannan valmistajan ohjeita[29]. Muodostettu graafi Neo4J tietokannan käyttöliittymästä on esitetty kuvassa 3.2. Verkosto kuvastaa radikalisoituneiden henkilöiden kontaktiverkostoa, toimintaa ja sosiaalisen median aktiviteettia.



Kuva 3.2. *PINGS-esimerkkiaineiston 100 solmun kokoinen verkosto tuonnin jälkeen*

Muodostetun graafin ja artikkelien [1, 10] perusteella voitiin olettaa että testiaineisto oli ymmärretty oikein.

3.1.2 Alkuperäisen PINGS-algoritmin toiminnan esiselvitys

Kun käytännössä kokeiltiin yleisessä jakelussa olevaa algoritmia se ei toiminut tai sen toteutus ei ollut sillä tasolla että alkuperäisen projektin ulkopuolinen taho voisi sitä käyttää. Lähdekoodia tarkemmin tarkasteltuna kävi ilmi että kyseessä on ehkä prototyyppi jonka koodissa on kovakoodattu konfiguraatio, jota on mahdollisesti mukautettu artikkelia tehdessä. Tämä päätelmä tehtiin siksi, että kovakoodatussa konfiguraatiossa viitattiin pisteiden nimiin joita ei ollut käytössä kaikissa aineistoissa joten kyseisellä julkisessa jakelussa olevalla algoritmilla ei parametreja muuttamalla ja graafeja vaihtamalla voitu toteuttaa artikkeleissa[1, 10] kuvattuja hakuja ja löydöksiä.

Julkisesti saatavilla olevassa ohjelmakoodissa[9] mm. oletetaan että tietoaineisto on tallennettu tietyssä järjestyksessä ja että se on tietyn tyyppistä ts. ohjelmakoodi ei tämän tulkinnan mukaan kykene käsittelemään kuin yhtä aineistoa ja aineistotyyppiä.

Lähdekoodia[9] analysoitaessa löydettiin mm. seuraavaa:

- Java-projektitiedostossa pom.xml algoritmin kääntäminen ja asentaminen olettaa että käytössä on alkuperäisen kirjoittajan kehityshetkellä oleva ohjelmistokehitysympäristö,
- paketissa edu.colostate.cnrl.sim olevissa luokissa SimilarityMeasure ja NeighborhoodMeasure ohjelmakoodi käyttää konfiguraationa kiinteästi määriteltynä Java-resurssia conf_rad.json,
- tiedostossa src/main/resources/conf_rad.json on kiinteästi määritelty konfiguraatio johonkin käyttötarkoitukseen,
- konfiguraatiotiedoston käytölle ei löytynyt koodista vahvoja perusteita, koska sitä ei voi käyttäjä dynaamisesti tarjota,
- luokassa edu.colostate.cnrl.sim.Common ilmenee että konfiguraatiotiedostossa käytettävistä listoista ei käytetä kuin ensimmäistä alkiota,
- ja tiedostossa src/main/java/edu/colostate/cnrl/sim/Common.java yleisesti käytössä oleva getNodeLabel-metodi olettaa että solmulla on vain yksi lipuke ja ensimmäinen on aina oikea.

Kokeilujen edetessä tehtiin pieniä kokeita lähdekoodin muutoksella, mm. konfiguraatiotiedoston dynaaminen osoittaminen. Konfiguraatiotiedoston toimintojen dokumentoimattomuus osoittautui suurimmaksi haasteeksi, mistä voitiin todeta että kyseisen tiedoston merkitys on melko pieni ja toteutus on mahdollisesti jäänyt kesken. Samalla löytyi myös selitys [1, s. 7, 9] algoritmien pseudokoodeissa viitatulle ConfigurationList-muuttajlle, jo-

ka on lähdekoodin implementaatioykstyiskohta – staattisen konfiguraatiotiedoston sisältö muutetaan hajautustauluksi.

Muutosten kokeilu oli hidasta ja hankalaa. Syitä olivat vähemmän käytetyt teknologiat, SQL-yhteensopiviin relaatiokantoihin verrattuna marginaalinen kyselykieli (Cypher) ja tietokanta (Neo4J), Neo4J-algoritmin (PINGS) muutoksen aktivointi vaati käännetyn Java-tiedoston asentamista tietokantaan ja tietokannan uudelleenkäynnistystä sekä ohjelmakoodin kirjoitustyyli (globaalien muuttujien käsittelyä useista paikoista) sekä testien ja esimerkkien puute.

Koska alkuperäisen koodin käyttöönottaminen osoittautui vaikeaksi, päädyttiin kirjoittamaan uudestaan vastaava algoritmi artikkelien [1, 10] sekä lähdekoodin[9] esittämien periaatteiden pohjalta.

3.1.3 Sovelletun PINGS-algoritmin kehityksen lähtöasetelma ja rajaus

Tässä tutkimustyössä päädyttiin uudelleenkirjoittamaan PINGS-algoritmit alkuperäisten pseudo-koodi esitysten[1, 10] pohjalta. Lisäksi algoritmejä parannettiin havaittujen huomioiden perusteella niin että niistä tulisi yleiskäyttöisempiä ja vältettiin alkuperäisessä toteutuksessa havaittua sisäänrakennettua olettamusta aineistosta ja sen järjestelystä. Käytännössä PINGS-algoritmista kirjoitettiin uudestaan osat $individualSimilarity$ ja $neighborhoodSimilarity$ sekä edellämainittujen aliohjelmat.

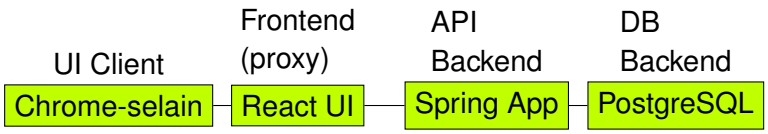
$implicitIndividualSimilarity$-algoritmin uudelleenkirjoitus ja kokeilu päätettiin rajata tutkimuksen ulkopuolelle, koska alkuperäistä toteutusta tarkastellessa ja artikkelin[1] perusteella tulkittiin algoritmin olevan pieni variaatio tai kokeilu algoritmista, jonka suorituksen voi parametrisoida ylimääräisellä tarkasti määritetyllä ehdolla. Kyselygraafin relaatioiden painotusarvojen huomioivalla toteutuksella voitaisiin mahdollisesti toteuttaa dynaaminen vaihtoehto $implicitIndividualSimilarity$-algoritmin toiminnalle jatkokehityksenä.

3.1.4 Menetelmää varten kehitetty järjestelmä ja suoritusympäristö

Järjestlemäkaaviossa 3.3 on kuvattu sovellusympäristö, jossa kokeilu tehtiin. Tätä työtä varten implementoidun sovelluksen lähdekoodi[30] on julkaistu GitHubiin. Sovellettu PINGS-algoritmi ja sen sovellusympäristö toteutettiin seuraavilla teknologioilla ja teknologioiden avulla:

- PostgreSQL 14 Relaatiokanta (graafin persistenssi),
- Java Spring Framework Web application (Algoritmi ja API),
- React 18 käyttöliittymä graafien visualisointiin tarkoitetulla ReGraph-kirjastolla,

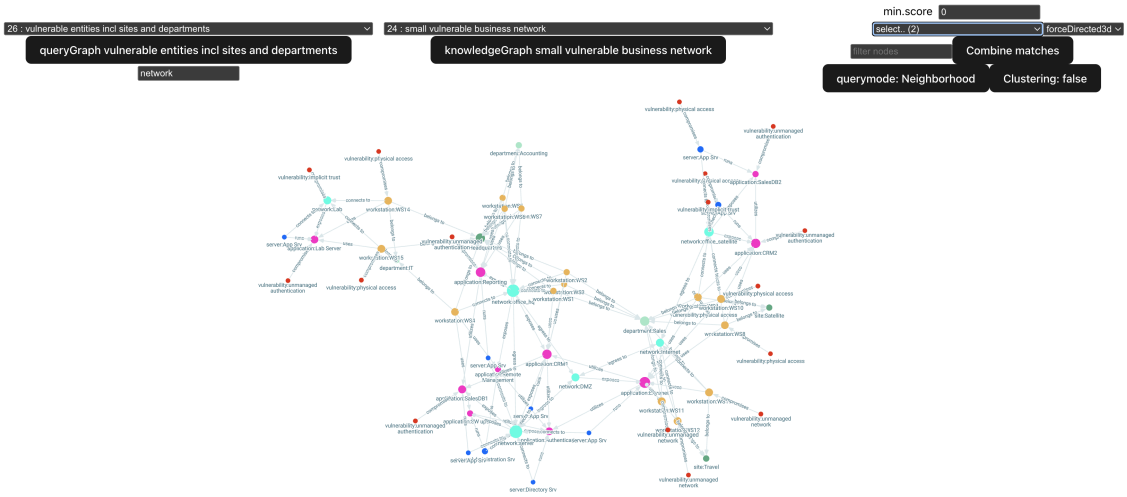
- Google Chrome selain ja
- Docker (Linux-binäärisen tietokannan ajo Mac-tietokoneella)



Kuva 3.3. *PINGS-toteutuksen sovellusympäristön järjestelmäkaavio*

Sovellusta ja sen koko ympäristöä suoritettiin ja kehitettiin Mac-työasemalla (vm. 2017, Intel Core i7). Selaimella (Chrome) voitiin välittömästi kokeilla käyttöliittymän kautta algoritmin toimivuutta visualisoituna. Kehitys- ja kokeiluympäristö pidettiin mahdollisimman yksinkertaisena nopean palautteen näkökulmasta, esim. käyttöliittymän päivitetty ohjelmakoodi kääntyi uudestaan reaaliaikaisesti vite-kehityspalvelinkomponentilla. Käyttöliittymän projektitiedosto joka kuvaa käytetyt ohjelmistokehityskirjastot on listattu ohjelmalistauksessa B.1. Kehitetty käyttöliittymä on esitetty kuvassa 3.4.

Taustapalvelua ja algoritmia (API ja backend) kehitettiin niin ikään nopean palautteen näkökulmasta, jolloin tietokone käänsi uuden sovellusversion muutetun ohjelmakoodin perusteella käyttöliittymän välitettäväksi. Taustapalvelujen projektitiedosto C.1 esittää taustapalvelun ohjelmistokehityskirjastoriippuvuudet.



Kuva 3.4. *Ruutukaappaus kehitetystä käyttöliittymästä*

Tavoite ei ollut keskittyä yhtä tehokkaan algoritmin ja tietokantamoottorin optimointiin, vaan saada PINGS-algoritmi käyttöön kokeilua varten. Merkittävimmät muutokset alkuperäiseen toteutukseen olivat seuraavat:

- ohjelmakoodi helpommin ymmärrettävää, muutettavaa ja testattavaa koska globaalien muuttujien käsittelemistä minimoitu,

- nopeutettiin kehityksen tuottavuutta käyttämällä yleisesti käytetyillä ja hyvin dokumentoiduilla teknologioilla joissa kehitystyölle nopea palaute-silmukka,
- prototyyppikäyttöliittymä, joka soveltuu paremmin PINGS-algoritmin tulosten tarkasteluun kuin Neo4J-tietokannan yleiskäyttöliittymä,
- ei otettu käyttöön logiikkaa joka alkuperäisestä aineistosta on olettamalla lukenut $RedFlag$-tietoa sekä $RedFlagMultiple$-parametri ja korvattiin toiminnallisuus kyselygraafin relaatioiden painotuksella,
- mahdollistettiin myös sisääntulevien yhteyksien käsittely toisin kuin alkuperäinen saatavilla oleva toteutus kuten artikkelin[1, s. 1] johdannossa mainittiin,
- ja muutettiin pisteitä siten että jokainen yhdistetty solmu ja yhdistetyn solmun kyselyssä esiintynyt attribuutti (property) nostaa löydöksen pisteitä.

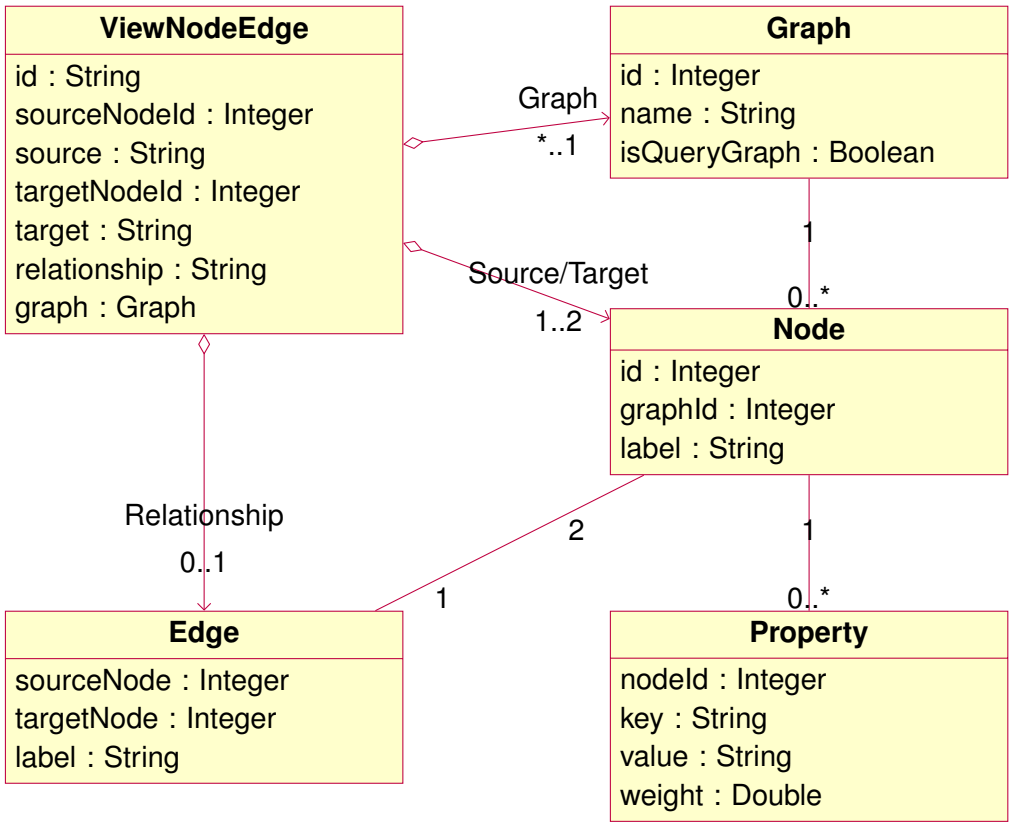
Tehdyn kehitystyön myötä tavoiteltiin tietoaineisto-agnostisempaa algoritmia.

3.1.5 Graafien persistointi uudelleenimplementaatiossa

Koska uudelleen implementoinnissa tietovarantona käytettiin relaatiotietokantaa (PostgreSQL), tuli algoritmin persistenssiä käsittelevät koodit kirjoittaa uudestaan. Relaatiokantaan päädyttiin, koska relaatiotietokannasta oli aikaisempaa kokemusta, ohjelmointiin liittyvää dokumentaatiota on olemassa paljon ja kokonaisuudessaan yleisesti käytössä olevat tutut teknologiat mahdollistivat nopeamman kehitystyön ja joustavamman visualisoinnin. Tietokannaksi olisi tämän työn kontekstissa voitu valita mikä tahansa SQL-yhteensopiva, ilmaisesti käytettävissä oleva ratkaisu. Suorituskyvyn tai erityisten graafienkäsittelyominaisuuksien takia ei myöskään ollut tarveta aidon graafikannan ominaisuuksille.

Kuva 3.5 esittää implementaatiossa käytetyn tietomallin, johon tallennettiin suunnattuja graafeja (solmuja, viivoja ja solmujen attribuutteja). Kehitetyn sovelluksen tietokantaoperaatiot käyttivät pääasiallisesti tietokantanäkymää $ViewNodesEdges$ josta sovellus sai tehokkaasti kysyttyä graafiin liittyvät pisteet ja viivat.

Relaatiomallissa $property$-relaatiolla solmulle voi antaa avain-arvo pareina attribuutteina sekä painoarvon $weight$. Painoarvoa voidaan käyttää kyselygraafissa solmujen attribuuteissa, mikä vaikuttaa tietämysgraafeista löydettyjen aligraafien pisteytykseen. Tällä painotettavalla attribuutilla pyrittiin toteuttaa dynaamisesti käytettävä ratkaisu kyselyiden parametrisointiin sekä toteuttamaan $Redflag$ ominaisuutta vastaava helpommin parametrisoitava toiminnallisuus. Tietomallissa $node.label$ kertoo solmun tyypin (esim. Person) ja vastaavasti $edge.label$ viivan suunnan mukaisen suhteen kahden pisteen välillä, esimerkiksi Knows. Kyselygraafit voidaan merkitä boolean-arvolla $queryGraph$ ja täten erottaa tietämysgraafeista, esimerkiksi jotta voidaan palvella käyttöliittymätarpeita listaamalla tiedossa olevat kyselygraafit ja tietämysgraafit.



Kuva 3.5. *Graafien tallentamiseen käytetty tietomalli, UML-kuvauskielellä esitettynä*

3.1.6 PINGS-algoritmin uudelleenimplementointi

Muutokset algoritmien syötteiden käsittelyihin

Merkittävänä erona uudelleenkäytettävyyden kannalta on tapa käsitellä algoritmien syötteitä. Alkuperäinen $getConfigurationList$-metodi lukee staattisesta Java-resurssitiedostosta konfiguraatioparametreja jonka muuttamisesta seuraa ohjelmakoodin uudelleenkääntäminen, algoritmin asennus Neo4J-tietokantapalvelimelle ja tietokannan uudelleenkäynnistys. Kyseisen konfiguraation alustuksen tarkoitus jäi myös hieman epäselväksi oliko sen tarkoitus konfiguroida tietämysgraafia vai kyselygraafin käyttöä. Lisäksi $Redflagmultiple$ kerrointa ei sovelletussa toteutuksessa ole. $Redflag$-indikaattorit on myös alkuperäisessä toteutuksessa toteutettu luettavaksi konfiguraatiotiedostosta, mutta niiden kertoimen voi antaa algoritmin parametrina.

Alkuperäisessä toteutuksessa käytettävät kyselygraafit on tallennettava graafin komponenttina osaksi tietämysgraafia Neo4J tietokantaan. Jotta kyselygraafia voidaan käyttää, tulee lisäksi jokainen kyselygraafin solmu merkitä samalla lipukkeella (label) jotta siihen voidaan viitata. Kyselygraafi etsitään silmukoiden Neo4J tietokannasta. Kyselyjen parametrisointi ja asioiden painottaminen vrt $Redflag$-ominaisuus ratkaistiin toteutuksen relaatiokannan tietomalliin kirjoitetuilla ominaisuudella.

Sovellettu individualSimilarity eli similarityMeasure-algoritmi

Ohjelmalistauksessa D.1 on kuvattu uudelleenkirjoitettu $individualSimilarity$-algoritmi alkuperäistä toteutusta mukaillen[1, s. 7]. Päädyimme nimeämään kyseisen algoritmin hieman alkuperäisestä esimerkistä poikkeavasti, korostaen uuteen toteutukseen syntyvää eroa. Uusi nimi on $similarityMeasure$. Uudistettu algoritmi nimensä mukaisesti suorittaa joustavammin arviointia samankaltaisuudesta (similarity), kun taas alkuperäinen nimi ymmärrettiin yksilöllisenä samanakaltaisuutena mikä kuvaisi rajaavammin operaatiota, jossa tulosjoukkoa tarkastellaan jonkinlaisina yksilöinä, esimerkiksi yksittäisinä henkilöinä. Merkittävin ero koko algoritmin toiminnallisuuteen on syötteiden käsittely, kyselyjen parametrisointi sekä tietomalliratkaisu jotka kuvattiin tässä luvussa.

Lyhyesti algoritmi D.1 laskee kyselygraafin vertailupisteet, etsii algoritmin $searchSimilarGraphs$ avulla vastaavia löydöksiä, pisteyttää löydökset suhteessa kyselygraafin vertailupisteisiin ja suodattaa pois annetun pisteiden $treshold$-kynnysarvon alittavat löydökset. $qflabel$ parametrilla asetetaan jokin kyselygraafissa Q ja tietämysgraafissa KG sijaitseva solmutyyppi, joista tietämysgraafin tutkiminen aloitetaan.

Sovellettu searchSimilarGraphs-algoritmi

Uudelleentoteutettu $searchSimilarGraphs$-algoritmi D.2 on implementoitu lähes identtisesti kuin alkuperäinen vastine. Algoritmi aloittaa tutkimisen iteroimalla tietämysgraafista $qflabel$ parametria vastaavia solmuja. Poiketen saatavilla olevasta toteutuksesta, jossa graafin tutkimista jatkettiin ulospäin suuntautuneilla viivoilla, tutkii sovellettu versio tietämysgraafia myös solmujen sisääntulevien viivojen näkökulmasta. Graafia tutkitaan rekursiivisessa algoritmissa $matchQueryGraphNodes$. Kun aligraafit on kerätty, sisällytetään ne graafien tulosjoukkoon ennen pisteytystä jos niissä on vähintään yksi solmu. Tällä tavalla saadaan laajennettua tulosjoukkoja pienellä toivotulla epätarkkuudella heterogeenisessa graafissa, ja tulosjoukko jossa yksikin solmu voisi jossain tapauksessa olla merkki tarkemman tarkastelun tai täsmennyksen tarpeesta.

Sovellettu matchQueryGraphNodes-algoritmi

Kehitetty $matchQueryGraphNodes$-algoritmi, ohjelmakoodi D.8, on käytännössä vastine alkuperäisen algoritmin[1, s. 7] $searchSimilarGraphs$ sisäiselle silmukalle. Algoritmi suoritetaan jokaiselle $qflabel$ parametria vastaavalle solmulle joka löydetään tietämysgraafista. Funktio on rekursiivinen, joka päättyessä palauttaa yhden kerätyn graafin joka saadaan kulkemalla tietämysgraafia. Jokaisella kutsulla iteroidaan kyselygraafin solmu ja tarkastetaan tietämysgraafin käsittelyssä olevan löydetyn solmun vierekkäiset solmut. Jos löydetty solmu ja viiva vastaavat vähintään kyselygraafin määrittelemällä tavoilla, lisätään solmu ja viiva tulosjoukkoon. Toisin sanoen tulosjoukon solmuilla voi olla

useampi attribuutti kuin kyselyssä, edellyttäen että yhteiset attribuutit ovat samat kuten $matchNode$-algoritmissa D.9 on kuvattu. Viivojen suuntien ja lipukkeiden tulee olla yhteneväiset, kuten $matchEdge$-algoritmissa D.10 ilmenee. Tulosjoukon solmujen attribuutteihin lisätään kyselygraafin attribuuttien painoarvot myöhempänä tapahtuvaa pisteytystä varten $addEdgeNode$ -algoritmissä D.11.

Sovellettu neighborhoodSimilarity-algoritmi

Ohjelmalistauksessa D.5 kuvattu uusi $neighborhoodSimilarity$-algoritmi on toteutettu alkuperäistä [1, s. 8–9] algoritmia mukaillen. Ensin tietämysgraafista louhitaan $searchSimilarGraphs$ algoritmilla kyselyä vastaavat graafit. Alkuperäistä toteutusta noudattaen, tietämysgraafista kerätään äskeisessä vaiheessa löydettyjen graafien solmujen vierekkäiset solmut sovelletulla algoritmilla $searchNeighborNodes$ joka on kuvattu listauksessa D.7. Algoritmi kerää löydettyjen pisteiden vierekkäiset pisteet, joita ei vielä ole tulosjoukossa. Poiketen alkuperäisestä saatavilla olevasta toteutuksesta [9], vierekkäisiä pisteitä poimitaan myös sisäänpäin tulevien kaarien päistä vrt. alkuperäiseen, jossa vierekkäisiä pisteitä etsittiin ainoastaan ulosmenevistä yhteyksistä. Huomiotava on, että artikkelissa[1, s. 1] mainitaan että uusi algoritmi on paranneltu mm. siten, että huomioidaan molempiin suuntiin kulkevat viivat. Tätä versiota ei kuitenkaan ollut julkisesti saatavilla.

Sovellettu searchNeighborMatchGraphs-algoritmi

Kun vierekkäiset pisteet on saatavilla, suoritetaan varsinainen sopivien naapurustojen verkostojen haku. Tämä toteutuu sovelletulla $searchNeighborMatchGraphs$-algoritmilla joka on esitetty ohjelmalistauksessa D.6. Jokaisen $searchSimilarGraphs$ löydökseen liittyvälle vierekkäiselle pisteelle etsitään kyselygraafia vastaava verkosto jälleen $searchSimilarGraphs$ algoritmilla, jonka löytämä aligraafi lisätään tulosjoukkoon. Jos löydöksen naapurisolmusta louhittu kyselygraafin muoto toistuu, kuuluu löydös osaksi tulosjoukkoa. Alkuperäisen kuvauksen ehdotuksessa[1, s. 8–9] ja saatavilla olevassa toteutuksessa[9] naapurisolmusta löydetyt aligraafien solmut käyvät läpi sisäisen pistetytysmenetelmän, jossa valituksi tulemisen edellytyksenä on solmujen esiintyminen kovakoodatussa $configurationList$-objektissa. Tämä toteutus jätettiin tietoisesti tekemättä uudessa toteutuksessa, koska sen arvioitiin vaikuttavan epäedullisesti algoritmin yleiskäyttöisyyteen ja saatavilla olevaa toteutuksen toimintaa ei voitu todentaa esimerkiksi olemassa olevalla dokumentaatiolla tai testeillä.

Alkuperäisessä artikkelissa kuvattiin naapuriverkostojen valintaa algoritmissa $checkEligibility$ ja $applyCollectives$ funktioilla. Ohjelmakoodissa [9] ei kuitenkaan löydy yksiselitteisesti suoraa vastinetta joka täysin vastaisi artikkelin kuvausta. $checkEligibility$ arvioi mikäli solmut voivat toimia verkostoista löytyvien ryhmien myötävaikuttajina ja $applyCollectives$ ylläpitää merkkejä, joita myöhemmin käytetään kyse-

lygraafin vastaavuuden tarkistamisessa. [1, s. 8–9]

Pisteytystoteutus

PINGS-algoritmin tarkkaa toimintaa oli vaikea todentaa, koska mitään testejä tai muita testattavia todisteita ei ollut saatavilla ja näin ollen tässä esitetyt olettamukset perustuivat nopeisiin kokeiluihin ja ohjelmakoodin tulkitsemiseen. Alkuperäinen algoritmi[9] pisteyttää kaikki löydetyt graafit suhteessa kyselygraafiin ylätasolla laskukaavalla 3.1. Pisteytysmekanismia ei tarkalleen kuvattu artikkeleissa[1, 10] eikä se myöskään selkeästi käynyt ilmi lähdekoodista, joten tässä työssä tehtiin olettamuksia saatavilla olevan ohjelmakoodin pohjalta. PINGS-algoritmissa kyselygraafin pisteytys, $totalWeight$ laskettaan konfiguraatiossa olevien aktiviteettinoodien määrän sekä $Redflag$-löydösten perusteella. Vastaavalla tavalla löydetyn graafin pisteet $matchedGraphWeight$ laskettiin vastaavalla tavalla huomioiden että silmukassa keskenään vertailtavat kyselygraafin ja tietämysgraafin solmut vastaavat toisiaan. Näin ollen tulkittiin että alkuperäinen pisteenlasku kertoo prosentuaalisesti kuinka suurilta osin löydökset täydellisesti vastaavat kyselyä, jolloin algoritmin käyttäjä voi rajata tulosjoukosta pois osumia antamalla parametrina $treshold$ minimiarvon, joka pisteytyksessä tulee ylittää.

$$matchedGraphScore = \frac{matchedGraphWeight}{totalWeight} \tag{3.1}$$

Uudelleenkirjoituksessa pisteenlaskua muutettiin hieman sillä ajatuksella, että sieltä saisi myös epätarkempia osumia koska alkuperäisissä kokeiluissa näytti siltä että pienemmillä pisteillä olevat löydökset ovat rakenteellisesti puutteellisia ja niistä saattaa puuttua solmuja, jos jonkin solmun attribuutit poikkeaa edes hieman kyselygraafista. Uudelleenkirjoitettu algoritmin kyselygraafin pisteytys on kuvattu ohjelmakoodissa D.3, jonka avulla saadaan eräänlainen $totalWeight$. Pisteenlasku hyödyntää kaavaa 3.2, jolla lasketaan kyselygraafin solmujen attribuuttien painotusarvot, vrt. $Redflag$-toiminnallisuus.

$$propWeights(n) = \sum_{p \in P(n)} weight(prop) \tag{3.2}$$

Kaavassa 3.3 on esitetty uuden algoritmin toteutus kyselygraafin pisteytykselle, eli $totalWeight$ muodostukselle.

$$totalWeight = |Q| + \sum_{node \in Nodes(Q)} propWeights(n) \tag{3.3}$$

Ohjelma D.4 kuvaa sovelletun laskukaavan, jolla saadaan löydetyn aligraafin $matchedGraphWeight$. Koska löydettyjen pisteiden kelvollisuus on arvioitu jo siinä vai-

heessa kun $searchSimilarGraphs$ algoritmi on suoritettu, tehdään tässä vaiheessa enään pisteytys. Kaava 3.4 kuvaa pisteiden laskukaavan, jossa symboli mgw on $matchGraphWeight$. Uudelleen kirjoitetun algoritmin löydöksiä keräävässä vaiheessa ($searchSimilarGraphs$) löydettyyn graafiin on tallennettu kyselygraafin painotuspisteet.

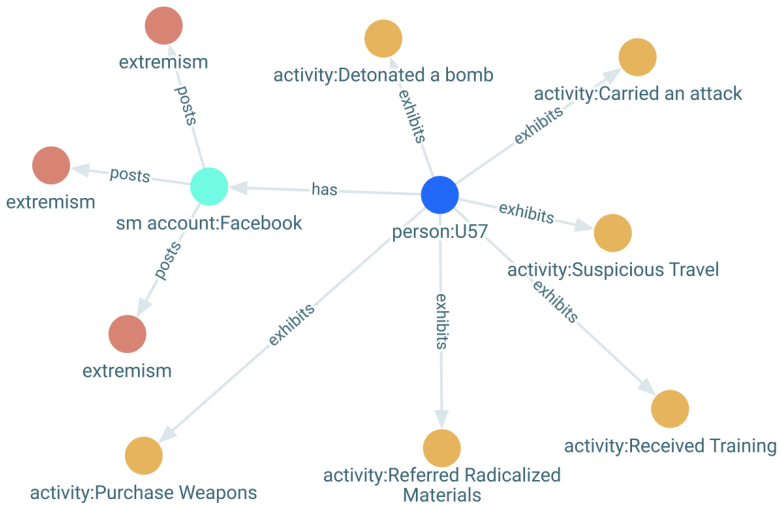
$$mgw = |MatchedNodes| + |MatchedEdges| + \sum_{n \in MatchedNodes} propWeights(n) \quad (3.4)$$

Uusien algoritmiajojen päätteeksi vertailupisteet lasketaan edellä kuvatun kaavan 3.1 mukaisesti. Huomioitavaa on, että pisteet muodostuvat hieman eri tavalla kuin alkuperäisessä toteutuksessa. Tässä työssä on tarkoitus tutkia algoritmin soveltuvuutta heterogeeniseen aineistoon, jolloin riittää että voimme korostaa osumia ja järjestää niitä vastaavuuksien perusteella.

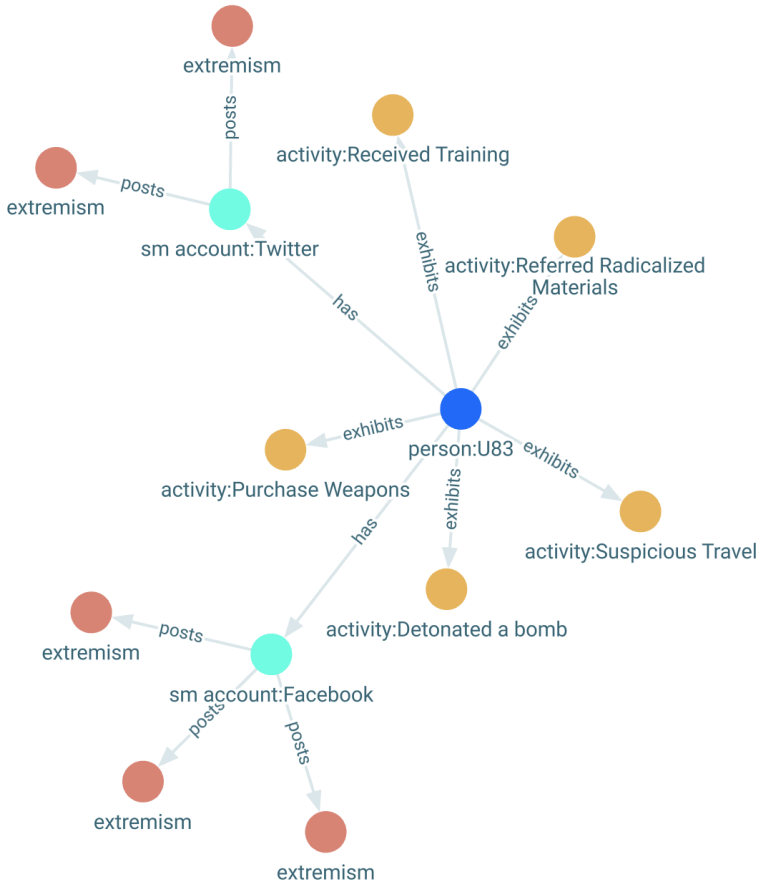
3.1.7 Uudelleenkirjoitetun PINGS-algoritmin toimivuuden toteaminen

Koska mitään testejä tai esimerkkejä ei ollut saatavilla, päätettiin kehitetyn algoritmin toimintaa kokeilla käytännössä alkuperäisen testimateriaalin[28] pohjalta.

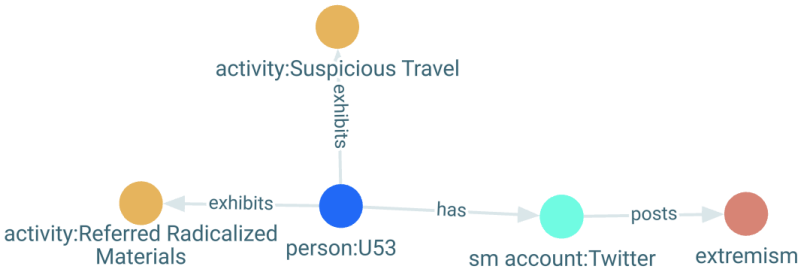
Uudelleen toteutetulla algoritmilla tehtiin esimerkkihakuja $similarityMeasure$-metodilla aineistossa kyselygraafilla Q 3.1. Esimerkiksi tämän avulla paikannettiin radikalisoituneet henkilöt U57 3.6 ja U83 3.7. Algoritmi pisteytti myös vertailun vuoksi huomattavasti pienemmäksi pisteyten löydöksen henkilöstä U53 3.8.



Kuva 3.6. *PINGS-esimerkkiaineistosta uudella algoritmilla löydetty pisteiltään vahva löydös henkilö U57*



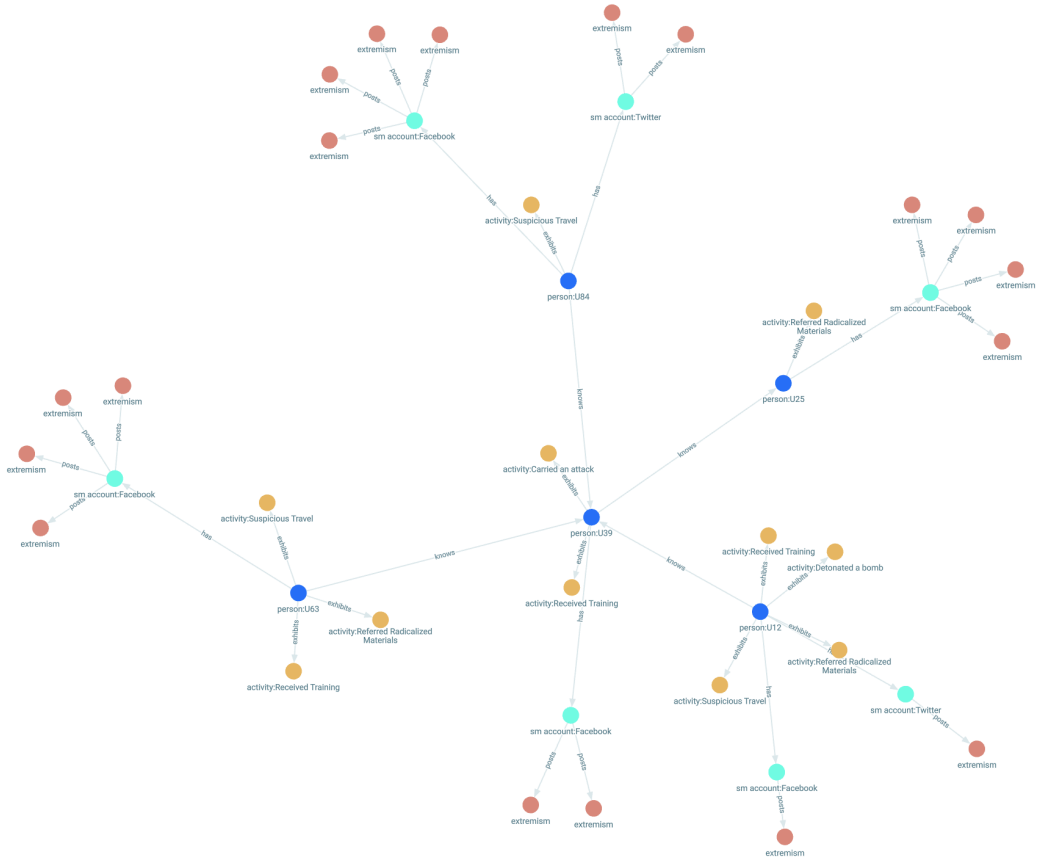
Kuva 3.7. *PINGS-esimerkkiaineistosta uudella algoritmilla löydetty pisteiltään vahva löydös henkilö U83*



Kuva 3.8. *PINGS-esimerkkiaineistosta uudella algoritmilla löydetty pisteiltään heikompi löydös henkilö U83*

Tämän lisäksi todettiin algoritmin löytävän radikalisoituneita verkostoja $neighborhoodSimilarity$-metodilla. Kuvassa 3.9 on esitetty viiden henkilön verkosto henkilön U39 ympärillä, joilla kaikilla verkoston jäsenillä on kyselygraafissa Q mainittuja taipumuksia. Vaikka henkilö U57 oli pisteytykseltään huomattavasti aktiivisempi yksilönä tarkasteltuna, on graafissa pienempi henkilöön liittyvä verkosto joka toistaa kyselygraafin

Q kuvaamaa ilmiötä. Henkilön U57 solmujen naapurusto on kuvattu kuvassa 3.10.



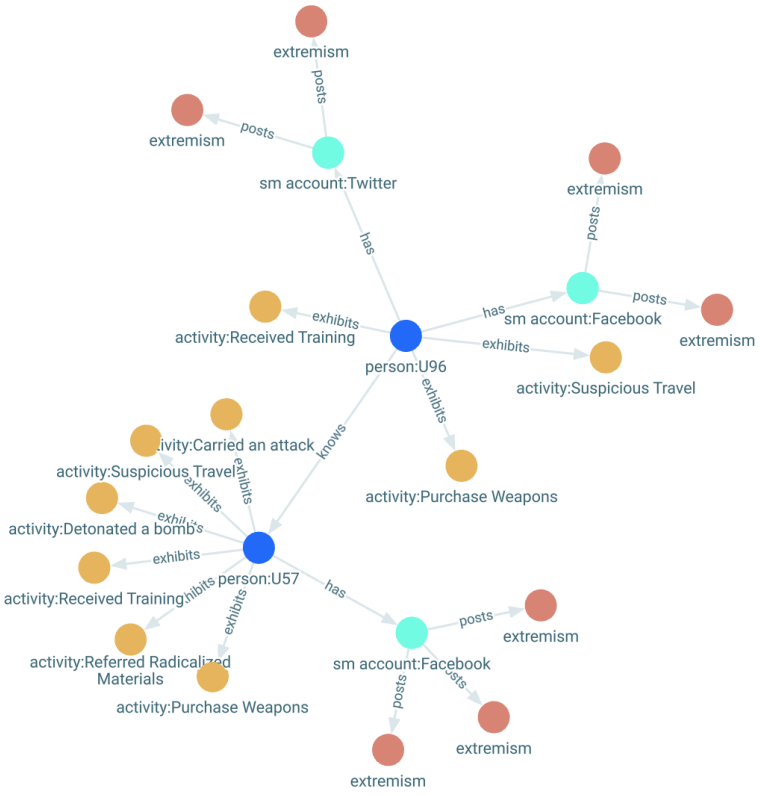
Kuva 3.9. *PINGS-esimerkkiaineistosta uudella algoritmilla löydetty samankaltaisten henkilöiden verkosto*

Tämän perusteella todettiin että tutkimustyötä ja algoritmin soveltuvuuden kokeilua voitaisiin jatkaa omalla tietoaineistolla.

3.2 Tutkimuksessa käytetyt tietämysgraafit

Tätä tutkimustyötä varten toteutettiin synteettiset tietämysgraafit, joista louhittiin tulosjoukkoja tutkimusta varten kehitetyillä kyselygraafeilla. Tietämysgraafit ja kyselygraafit esitellään tarkemmin ja käsitellään luvussa 4 työn tulosten tarkastelun yhteydessä. Kehityt tietämysgraafit esittävät pienen yrityksen tietämysverkostoa sekä hieman suuremman yrityksen tietämysverkostoa. Tietämysverkostot ovat täysin keinotekoisia siten, ettei niiden mallintamiseen ole käytetty mitään olemassa olevaa todellista tietoverkkoa tai yritystä tai sen toimintaa. Kehitetyt tietämysverkostot ovat melko pieniä, jolloin niiden käsittely ei vaadi mittavia määriä suorituskykyä tutkittavalla algoritmilla. On oletettavaa että todellinen algoritmillisesti tutkittava tietämysgraafi olisi merkittävästi suurempi. Pienehköt graafit ovat tutkimustyön kannalta parempia havainnollistettavuuden sekä graafisen analyysin näkökulmasta.

Tietämysverkkojen solmut esittävät esim. tietokoneita, palvelmia, lähiverkkoja, sovelluk-



Kuva 3.10. *PINGS-esimerkkiaineistosta uudella algoritmilla henkilön U57 naapurusto*

sia, haavoittuvuuksia, osastoja, konttoreita ja kaaret solmujen välisiä yhteyksiä kuten esim. tietokone kytkeytyy lähiverkkoon. Tietämysgraafeilla on ensisijaiseti tarkoitus tutkia samojen kyselygraafien toimivuuden todentamista erikokoisella aineistolla. Lisäksi tuloksia voidaan ensiksi tarkastella pienessä asiayhteydessä ja sen jälkeen siirtyä laajempaan tarkastelukontekstiin. Tietämysverkkoja ja niiden variaatioita tutkitaan tässä työssä kuvitteellisessa tilanteessa, jossa yrityksen tietojärjestelmiin ja tietoverkkoihin kohdistuneisiin haavoittuvuuksiin tehdään korjaustoimenpiteitä täydentyvän tietämysgraafin, kyselygraafien ja PINGS-algoritmin avulla.

Tuotettua materiaalia käsitellään luvussa 4 kuvitteellisena simulaationa, miten yritys voisi pienentää tietotekniikkaan liittyviä riskejä tässä työssä käsiteltävien menetelmien avulla.

3.2.1 Pienen yrityksen synteettinen tietämysgraafi

Pienen yrityksen tietämysverkostosta toteutettiin asteittain muuttuvia ja tarkentuvia tietämysgraafeja tulosten esittämistä varten. Koko verkosto on esitetty kuvassa 4.1 jota myöhemmin täydennetään haavoittuvuuksilla sekä haavoittuvuuksiin liittyvillä korjaustoimenpiteillä luvussa 4. Pienen yrityksen tietämysgraafin koko on 46 solmua ja 109 kaarta, laajentuen haavoittuvuuksineen 63 solmuun ja 126 kaareen.

3.2.2 Suuremman yrityksen synteettinen tietämysgraafi

Suuremman yrityksen tietämysverkostosta toteutettiin versio, johon jäi joitain haavoittuvuuksia mutta sisälsi jonkun verran korjauksia. Tämän graafin tarkoitus on simuloida tilanne jossa työkaluja käytetään normaaliolosuhteissa kun yritys tietää joidenkin asioden olevan hyvin ja joidenkin huonosti, ja miten työkalun käyttö istuisi jatkuvan parantamisen malliin. Suuremman yrityksen tietämysgraafi koostuu 124 solmusta ja 326 kaaresta.

4. TULOSTEN TARKASTELU

4.1 Yleiset havainnot PINGS-algoritmista

Koska tätä työtä varten toteutettiin sovellettu PINGS-algoritmi alkuperäistä noudattaen, oletetaan että uudelleenkirjoitetulla algoritmilla saadaan samoja tai vähintään samankaltaisia vastauksia kuin alkuperäisellä PINGS-algoritmilla. Koska alkuperäisestä työstä ei ollut testejä tai konkreettisia esimerkkejä olemassa, ei sovellettua versiota voitu kehittää esimerkkiä todentamalla, vaan pelkästään julkaisujen algoritmien kuvauksia vasten. Uudelleenkirjoitettu algoritmi antoi kuitenkin toistettaessa saman tulosjoukon samalle kyselylle, kuten saatavilla oleva alkuperäinen toteutus.

Tässä luvussa käsitellään huomioita, joilla saattaa olla merkitystä tuloksia muodostaessa. Vääristymiä on pyritty välttämään, mutta niiden vaikutusta ei voida täysin poissulkea. Kun PINGS-algoritmiin perehdyttiin työn valmisteluvaiheessa kävi ilmi seikkoja, joiden perusteella artikkelien kuvaamasta algoritmista[1, 10] ja saatavilla olevasta toteutuksesta[9] voitiin todeta, että kyseinen algoritmi ja sen olemassa oleva toteutus ei sellaisenaan täysin sovellu geneeriseen ongelmanratkaisuun. Muramudalige et al kertovat algoritmin löytävän tarkkoja tai epätarkkoja vastaavuuksia (exact or inexact patterns) sekasortoisista ja puutteellisista (noisy and incomplete) graafeista [1, s. 3].

4.1.1 PINGS-algoritmin ominaisuuksien vaikutus hakutuloksiin

Kuvatut alialgoritmit $individualSimilarity$ ja $searchSimilarGraphs$[1, s. 7] kykenevät louhimaan kyselygraafeja muistuttavia aligraafeja sillä rajoituksella, että löydettyjen aligraafien tulee noudattaa muodoltaan solmujen tyypeiltä ja niiden välisiltä kaarilta täysin kyselygraafia. Näin ollen voidaan löytää ilmentymiä, jotka vastaavat joko hyvin tai heikosti kyselyä ja suodattaa niitä.

Jotta algoritmin arvioima aligraafi ylipäätänsä päätyy tulosjoukkoon, tulee algoritmin pisteyttää aligraafi ja pisteytyksen tulee ylittää käyttäjän antama raja-arvo $similarityThreshold$. Algoritmin suorituksen aikana arvioitavan aligraafin päätyminen pisteytytykseen edellyttää samaa rakenteellista muotoa sekä vastaavia solmu- ja kaarityyppejä kuin kyselygraafi.

PINGS-algoritmi käsittelee kyselygraafissa ei-mainittuja solmutyyppejä ja yhteyksiä siten, että tuloksissa ei huomioida löydettyjen aligraafien muita vierekkäisiä pisteitä. Tämä toiminnallisuus tuo ilmi algoritmin heikkouden tai keskeneräisyyden, koska alialgoritmien $neighborHoodSimilarity$ ja $searchNeighborMatchedGraphs$ kuvattu[1, s. 9] ja toteutettu[9] toiminnallisuus perustuu siihen että aligraafien naapurustot voivat vain liittyä toisiinsa yhdellä tavalla. Tämä yksi ja ainoa sallittu tapa on alkuperäisessä koodissa[9] erikseen mainittu relaatio joka vie aligraafista toiseen.

Jos PINGS-algoritmilla tutkitaan monimuotoista tietämysverkostoa, joka on rikastettu erityyppisillä tiedoilla ja moninaisilla yhteyksillä, tulisi tietämysgraafi yksinkertaistaa palvelemaan PINGS-algoritmia. Yksinkertaistaminen vähentäisi merkittävästi tietämysgraafin tarkkuutta ja todellisuutta vastavuutta. Tietämysgraafin yksinkertaistaminen ei ole tehokas summittaisen ja muuttuvan tietomassan analysoinnissa.

Alkuperäisten $neighborHoodSimilarity$ ja $searchNeighborMatchedGraphs$ alialgoritmien heikkous on kykenemättömyys löytää yhteenliittyviä verkostoja sellaiselle kyselygraafille, joka ei erikseen mainitse kaikkia tuloksessa esiintyviä yhteystyyppejä. Joissakin tapauksissa voi olla että on tarkoitus esittää kysymys kyselygraafin muodossa erittäin yksiselitteisesti (exact pattern) tai toisena vaihtoehtona jos ei tiedetä mitä halutaan tai miltä vastaus voisi näyttää, tulisi olla jokin keino korostaa epätarkkaa löydöstä (inexact pattern). Mikäli kyselygraafissa ei ole mainittuna kaikkia solmutyyppejä mitä tulee vastauksessa palauttaa, ei algoritmi palauta niitä. Vastaavalla tavalla jos tietämysgraafista löytyy muun tyyppisiä pisteitä tai yhteyksiä vastaavan kyselygraafin kuvaamien pisteiden ja yhteyksien väliltä, saattaa sekin vaikuttaa ei-toivotusti tulosjoukon muodostumiseen.

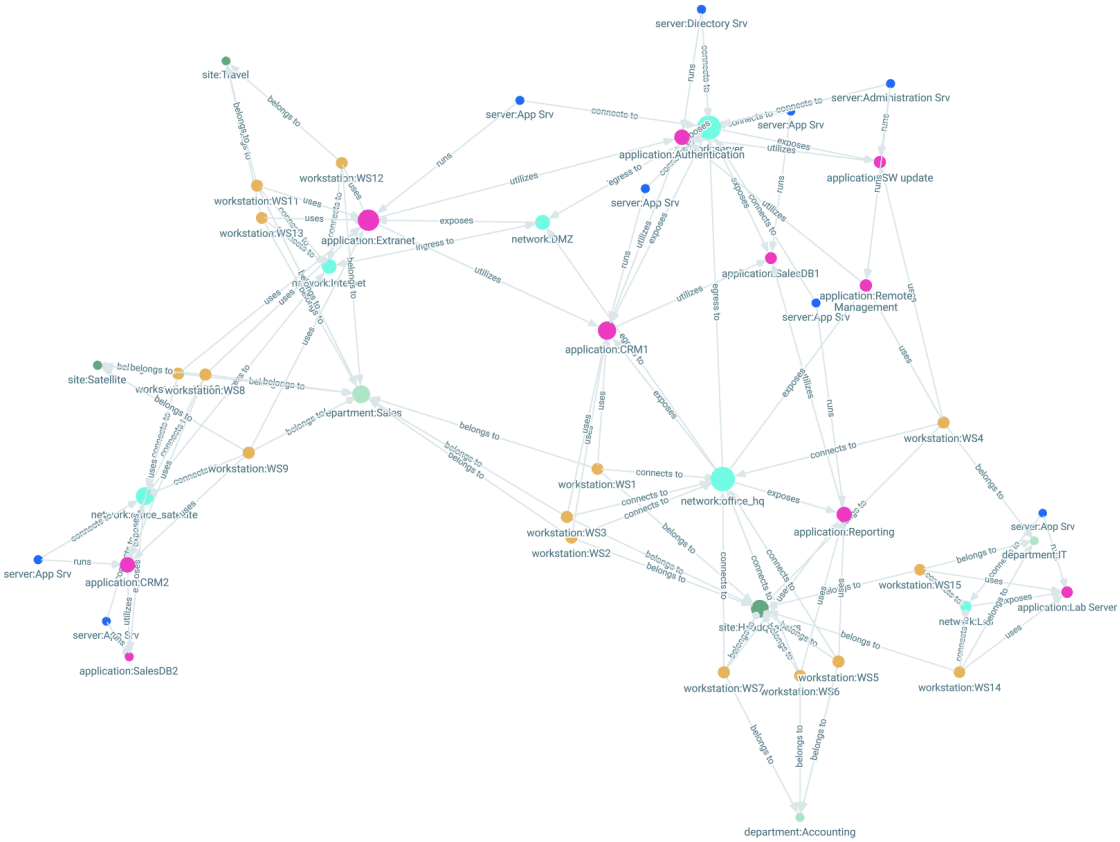
Naapurustoverkostojen poimintaa on $searchNeighborMatchedGraphs$ algoritmissa[1, s. 9] kuvattu $checkEligibility$ ja $updateCollectives$-metodien yhteistoiminnalla. Artikkelissa[1, s. 8] kerrotaan $checkEligibility$-metodin tarkistavan, toimiiko naapuriverkosto myötävaikuttajana (contributor). Algoritmin em. osaa ei ole kuvattu tarkemmin, mutta ohjelmakoodista[9] käy ilmi että kyseisessä osassa lasketaan kyselygraafia vastaavia pisteitä jonka perusteella naapurustot lisätään tai jätetään pois hakutuloksesta. Ongelmaksi voi muodostua odottamaton toiminnallisuus, jos algoritmi jättää pois naapurustoja heikkojen pisteiden takia sillä olettamalla että tietämysverkosto ei ole tai ei koskaan voi olla täydellinen. Näin ollen tietämysverkostosta voitaisiin vaan kysyä ja huomioida asioita joita jo entuudestaan tiedetään.

4.2 Keinotekoisesti luodut tietämysverkostot

Tässä työssä suoritettavaa koetta varten luotiin kaksi keinotekoista verkostoa, josta toinen on ensimmäisen laajennus. Verkostot kuvastavat pienen ja vähän suuremman yrityksen tietoverkkoihin liitettyjä laitteita, yhteyksiä ja tiedossa olevia haavoittuvuuksia. Tietämysverkostot eivät ole täydellisiä, mutta riittäviä yksityiskohdiltaan algoritmin ja sen käytettä-

vyyden kokeilemiseen ja tulosten analysointiin.

4.3 Pienen yritysverkon tietämysverkosto



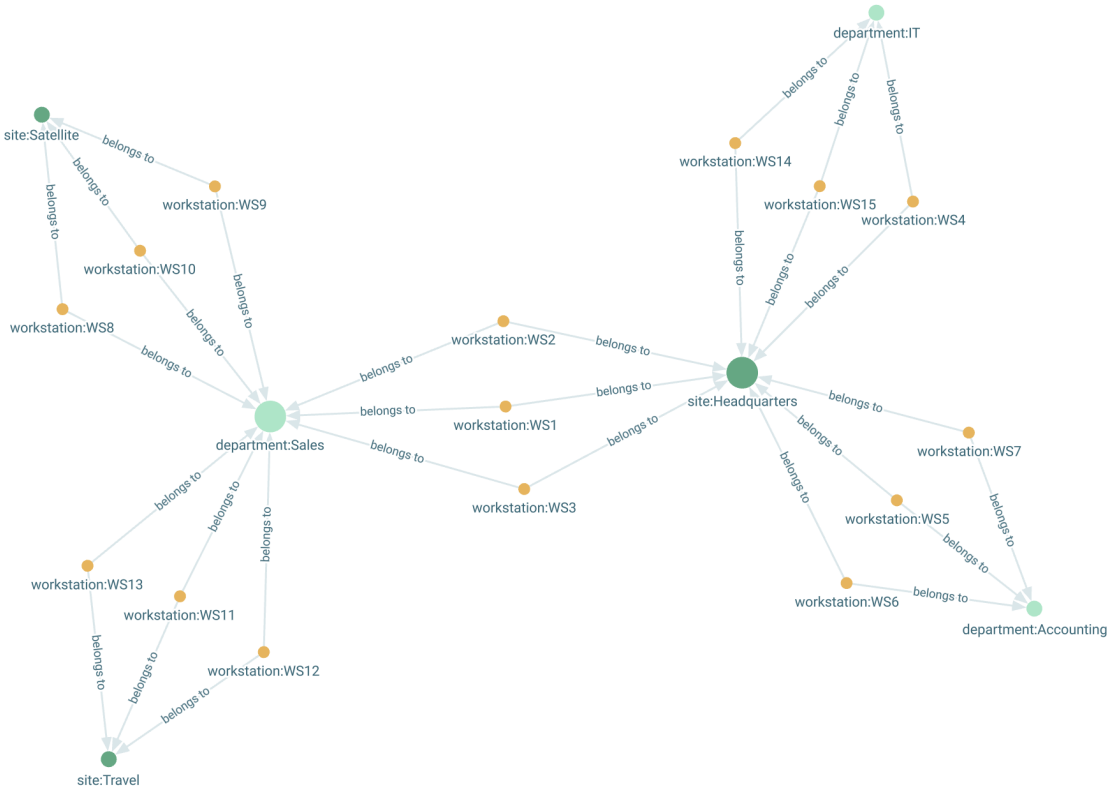
Kuva 4.1. *Kuvitteellisen pienen yrityksen tietämysverkosto*

Tutkimustyötä varten luotiin kuvassa 4.1 kuvattu kuvitteellinen pienen yritysverkon tietämysgraafi. Luodun yrityksen pääkonttorilla työskentelee 7 henkeä, sivukonttorilla kolme ja kenttämyynnissä kolme henkeä. Yrityksen osastot, konttorit ja osastojen henkilöille kuuluvat työasemat on esitetty kuvassa 4.2.

Pääkonttorin kolme myyjää käyttävät CRM-järjestelmää eli Customer Relationship Management, asiakkuudenhallintajärjestelmää. Kolmen hengen muodostoma talous- ja hallinto-osasto seuraavat raportointisovelluksella myynnin edistymistä. Yhden henkilön vahvuinen IT-osasto ylläpitää yrityksen tietojärjestelmiä. Yrityksellä on käytössä pieni opetusluokka, jossa voidaan järjestää koulutuksia asiakkaille.

Hiljattain yritysjärjestelyiden kautta liitetty sivukonttori käyttää omaa CRM-järjestelmää. Kenttämyyjät ja sivukonttorin henkilöstö käyttävät pääkonttorin CRM-järjestelmää Ekstranet-sovelluksen kautta. Ekstranet-sovellus mahdollistaa sisäverkon järjestelmien käyttämisen Internetin välityksellä. Eri sijainneissa tapahtuva sovellusten käyttö ilmenee kuvasta 4.3.

Kuvitteellisen yrityksen yhden henkilön vahvuinen IT-osasto on suunnitellut verkkoyhtey-



Kuva 4.2. *Pienen yritysverkon osastot, konttorit ja työasemat*

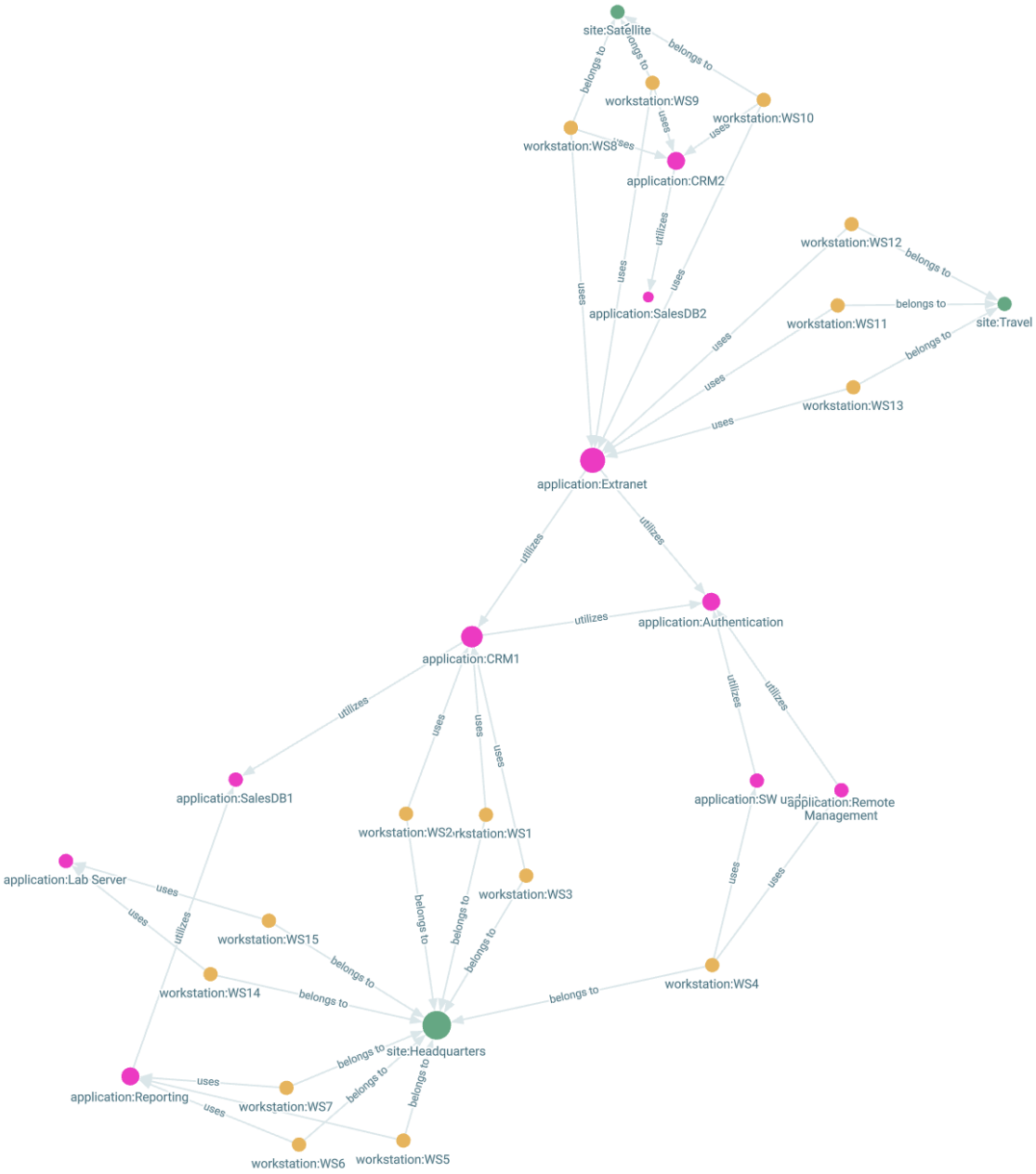
det, jossa palvelimet ja toimistoverkko on erillään. Lisäksi sisäverkkojen ja Internetin välinen liikenne kulkee eteisverkon, eli DMZ-verkon läpi (De-militarized Zone). Yrityksen koulutusluokka ei ole kytketty Internetiin.

Verkkorakenne ilmenee kuvista 4.4 ja 4.5. Huomattavaa on, että kun esitetyistä graafeista on puuttuu yhdistävät osasto- (department) ja konttori- (site) solmut, esiintyvät tietoverkot erillisinä graafeina joiden välillä ei ole yhteyttä vrt. koko tietäysgraaffi 4.1. Voidaan todeta, että näiden kahden verkkosaarekkeen välillä ei ole tiedossa olevaa tietoliikenneyhteyttä, vaan ne yhdistyvät toisiinsa silloin kun huomioidaan ympäröivän kontekstin, eli yrityksen, sanelemat relaatiot. Nämä kaksi esitystapaa ovat mahdollisia ja loogisia siten, että solmun poistuessa poistuu myös solmuun yhdistetyt kaaret. Tässä tapauksessa tulosjoukosta on suodatettu pois osasto- ja konttorisolmut, jolloin verkkojen välinen yhteys on poistunut tulosjoukosta.

4.3.1 Pienen yrityksen tietämysverkon tutkiminen

Esimerkki: olemme kiinnostuneita siitä, millä tasolla sovellukset tukeutuvat tietoturvalliseen keskitettyyn tunnistautumiseen, verrattuna siihen että niissä on paikallisesti hallitut käyttäjätilit. Esitetään kysymys: Mitkä työasemat käyttävät sovelluksia, jotka tukeutuvat keskitettyyn tunnistautumisratkaisuun?

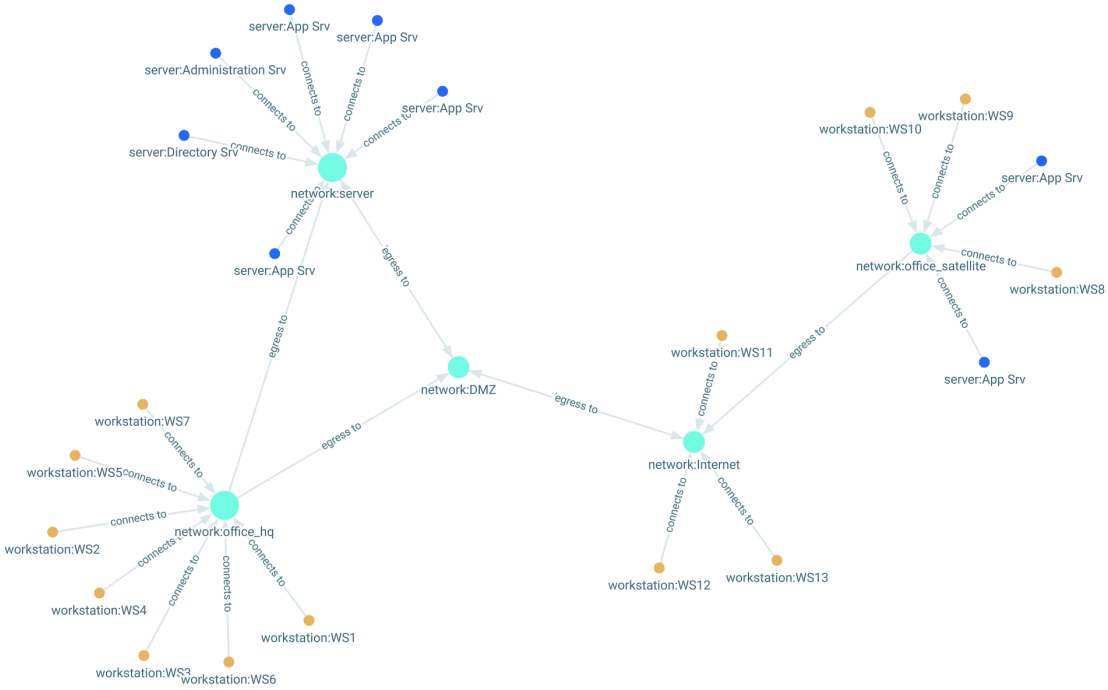
Kysymys kuvataan algoritmille kyselygraafina, joka on esitetty kuvassa 4.6. Koska ha-



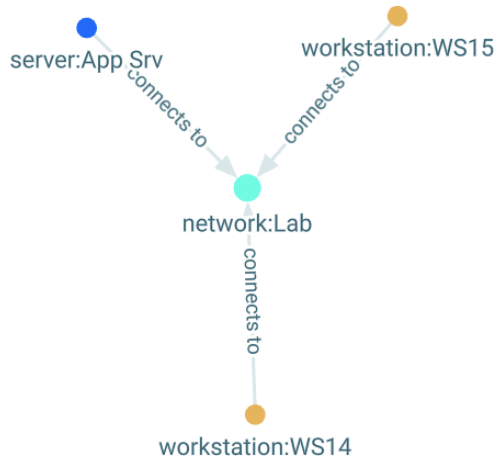
Kuva 4.3. *Pienen yritysverkon konttoreiden käyttämät sovellukset*

luamme nähdä verkostona ympäröivää kontekstia, kyselygraaffi ilmaistaan hieman yksinkertaisemmin – etsimme tietokoneita (workstation) jotka käyttävät muita sovelluksia (application) hyödyntäviä sovelluksia (utilizes application). Tunnistautuminen olisi mahdollisesti kyselyn tulosjoukossa hyödynnettävä sovellus (utilizes application). Kysely parametrisoidaan parametreilla $queryFocusLabel = workstation$, käyttäen algoritmia $searchSimilarGraphs$, minimipisterajalla $similarityThreshold = 0.0$.

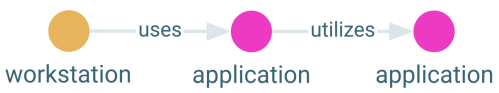
Kysely palauttaa kolme graafia, jotka on esitetty kuvassa 4.7. Laajin, kuvassa vasemmassa reunassa sijaitseva verkosto sisältää 18 solmua ja on saanut 18.167 pistettä. Sovellukset $CRM1$, $RemoteManagement$, $SWupdate$ ja $Extranet$ hyödyntävät keskitettyä tunnistautumisratkaisua. Lisäksi graafiin on löytänyt tiensä $CRM2$-sovellus, vaikka se



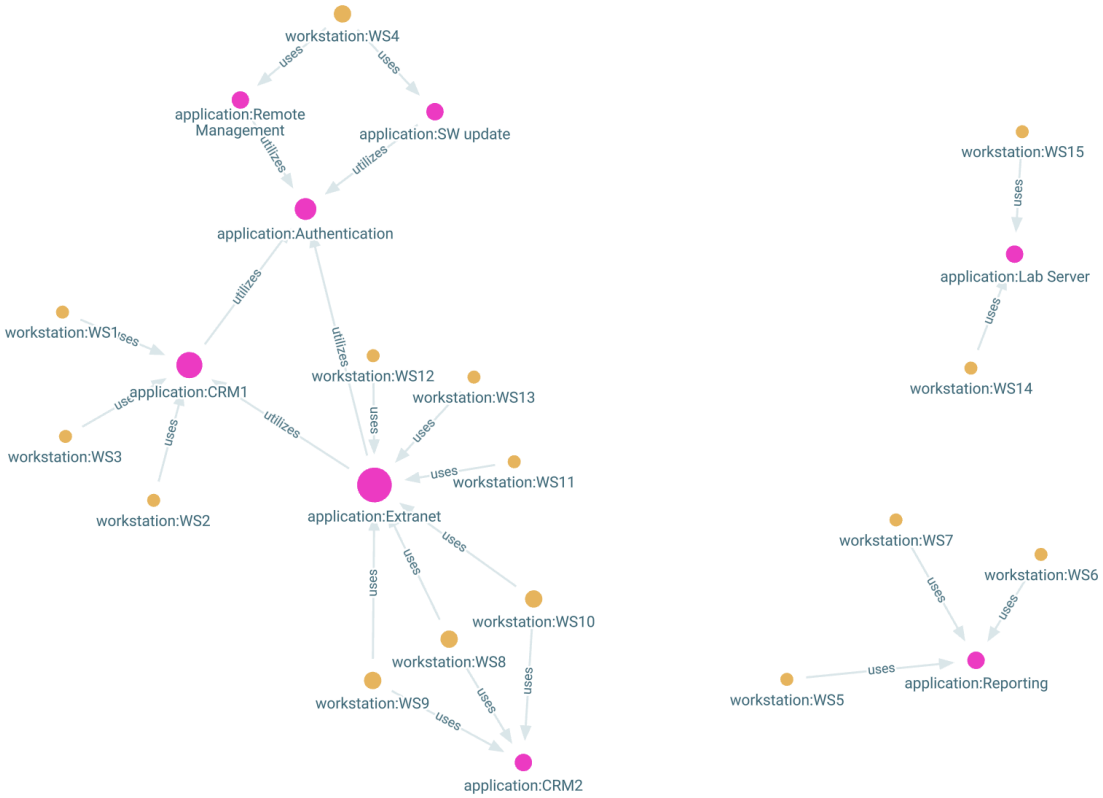
Kuva 4.4. *Pienen yritysverkon verkkoon liitetyt työasemat ja palvelimet*



Kuva 4.5. *Pienen yrityksen opetusluokan verkko, jolla ei ole yhteyttä muihin tietoverkkoihin*



Kuva 4.6. *Kyselygraafi: Työasemat, joilta käytetään muita sovelluksia hyödyntäviä sovelluksia.*



Kuva 4.7. *Kyselyn tulos: Työasemat, joilla käytetään sovelluksia, jotka tukeutuvat keskitettyyn tunnistautumiseen*

ei hyödynnä keskitettyä tunnistautumista. Sovellus on tullut valituksi tulosjoukkoon sen perusteella, että se on tyypiltään oikeanlainen mitä on odotettu $workstation$-tyyppisen solmun jälkeen.

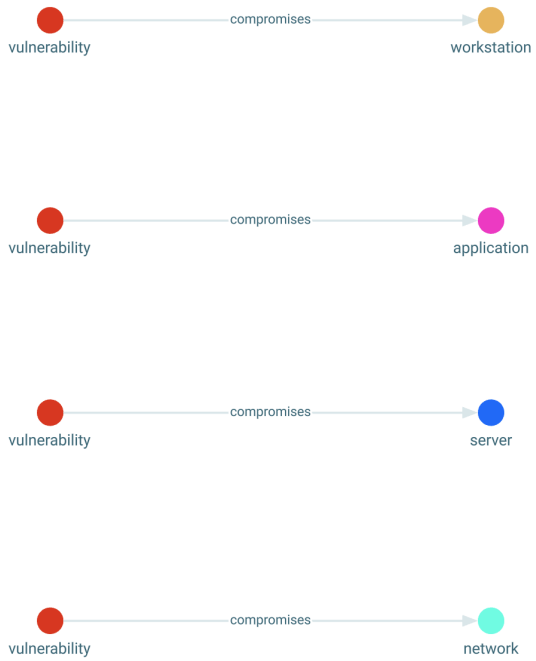
Toinen ja kolmas graafi, jotka sisältävät koulutusluokan sovelluksen työasemineen (3.167 pistettä) ja talousosaston raportointisovelluksen työasemineen (4.167 pistettä),eivät hyödynnä keskitettyä tunnistautumista. Nämä tulokset olisi ollut mahdollista suodattaa pois asettamalla $similarityThreshold = 5.0$. Sinänsä näiden lähelle pääsevien tulosten esiintyminen tulosjoukossa on hyödyllistä visuaalisessa tarkastelussa, koska ne eivät verkotu keskitettyyn tunnistautumiseen.

Erilliset saarekkeet voidaan tulkita myös siten, että tiedossa ei ole, tukeutuvatko sovellukset keskitettyyn tunnistautumiseen.

4.3.2 Pienen yritysverkon tietämysverkosto haavoittuvuuksilla

Jotta haavoittuvuuksien löytämistä algoritmin avulla voitaisiin tutkia, rikastettiin tietämysverkostoa haavoittuvuuksilla. Tieto haavoittuvuuksista voisi oikeassa tilanteessa tulla ilmi esim. auditoinnista tai sattuneista vahingoista, jolloin ne lisättäisiin tietämysverkostoon. Verkostoon lisättyjä haavoittuvuustyyppejä olivat päätelaitteille hallitsematon fyy-

sinen pääsy (physical access), työasemaan liitetty hallitsematon tietoverkko (unmanaged network), sovelluksen käyttämä ei-keskitty tunnistautumisratkaisu (unmanaged authentication) ja puutteellisesti konfiguroitu mutta yrityksen hallitsema tietoverkko (implicit trust).

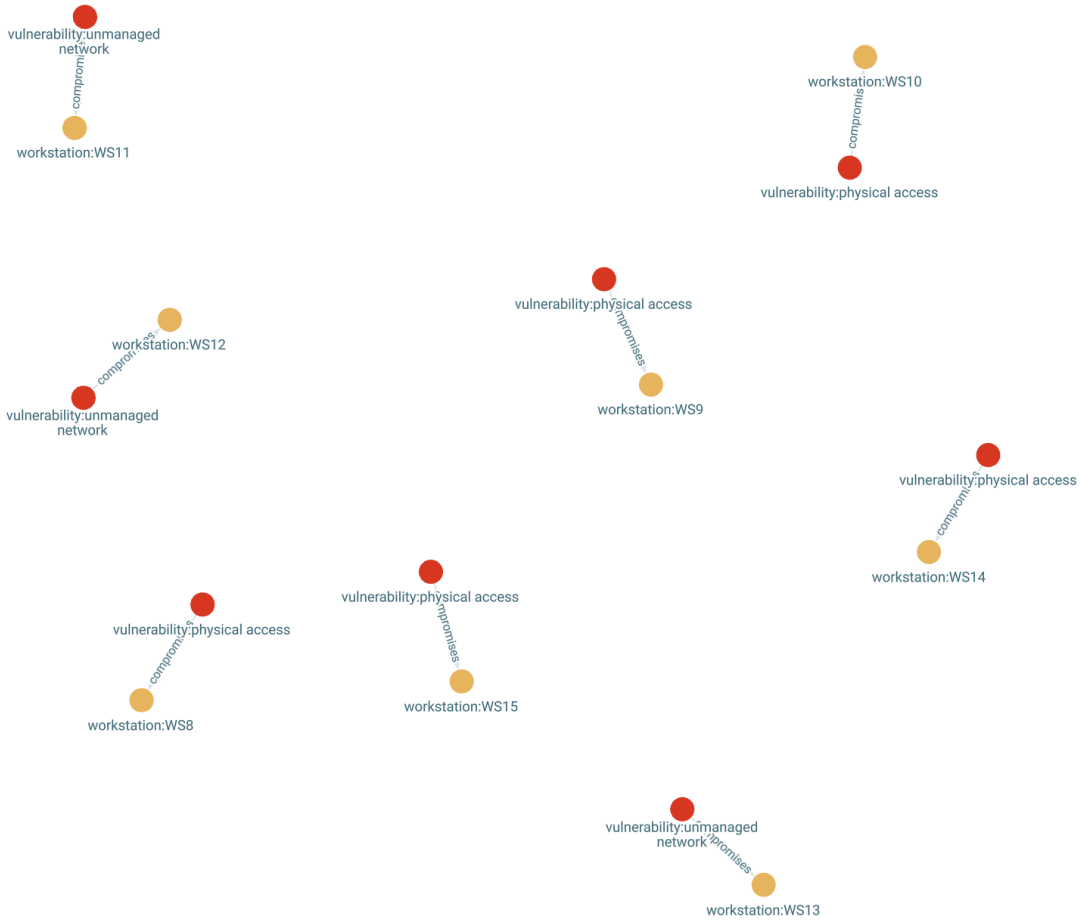


Kuva 4.8. *Kyselygraafi: Mitä haavoittuneita komponentteja tietämysverkostossa on?*

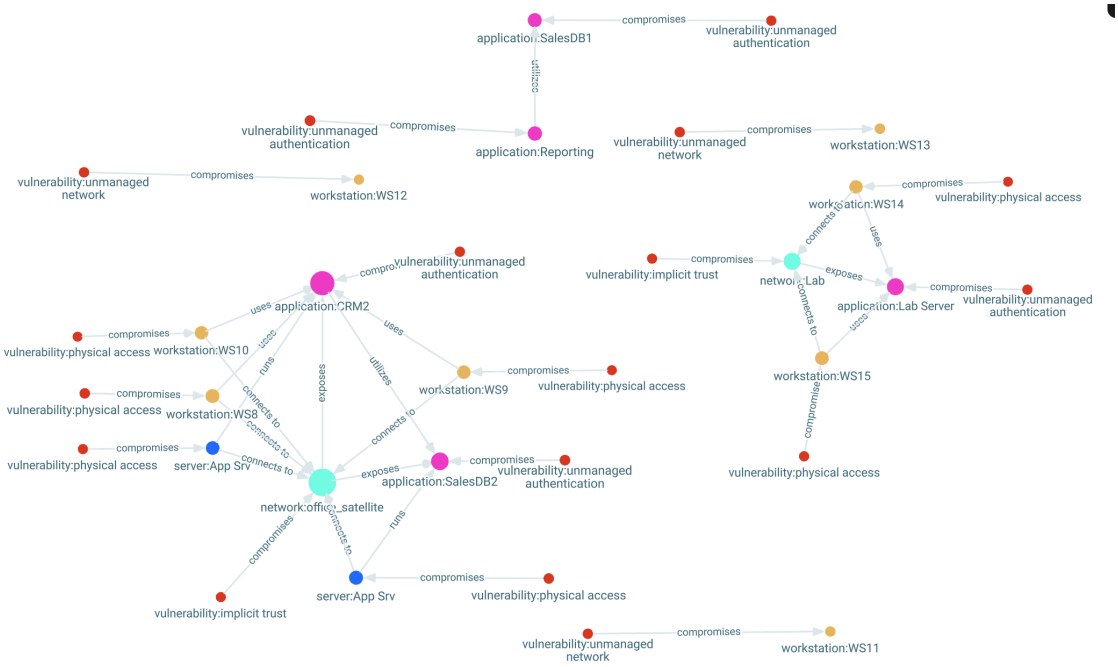
Haluamme tutkia, minkälaisia haavoittuneita sovelluksia, työasemia, palvelimia ja sovelluksia meidän verkossa on. Syötetty kyselygraafi on esitetty kuvassa 4.8. Kyselygraafi on kokeellinen esimerkki Multi-Example Search-kyselystä[4].

Kun $queryFocusLabel$ on $workstation$, $server$, $network$ saadaan algoritmilla $searchSimilarGraphs$ tulokseksi joukko haavoittuvuus -työasema, -palvelin tai -verkko pareja. Esimerkkinä haavoittuneiden työasemien yhdistetty tulosjoukko kuvassa 4.9. Vastaus noudattaa exact match-periaatetta. Laajemmat tulokset saadaan kuvassa 4.8 esitetyllä kyselygraafilla, kun asetetaan $queryFocusLabel = vulnerability$. Kyselyn tulokseksi saadaan verkosto jossa kaikilla entiteeteillä on jokin haavoittuvuus.

Tulosjoukko on esitetty yhdistettynä kuvassa 4.10. Haavoittuvuuksista huomataan että kenttämyyjien koneet kytkeytyvät hallitsemattomiin tietoverkkoihin (esim. lentokentän langaton verkko), talous- ja hallinto-ostaston raportointijärjestelmä ja CRM-järjestelmä tietokanta ei ole keskitetyn tunnistautumisen piirissä, sivukonttorilla tietokoneisiin ja palvelimiin on fyysinen pääsy asiattomilla (sivukonttori jakaa toimistotilat jonkin muun yrityksen kanssa), sivukonttorin CRM-järjestelmä ja tietokanta ei ole keskitetyn tunnistautumisen piirissä, sivukonttorin tietoliikenneverkko ja säännöstö on puutteellisesti konfiguroitu ja koulutusluokassa on melkein kaikki edeltämainitut haavoittuvuudet.



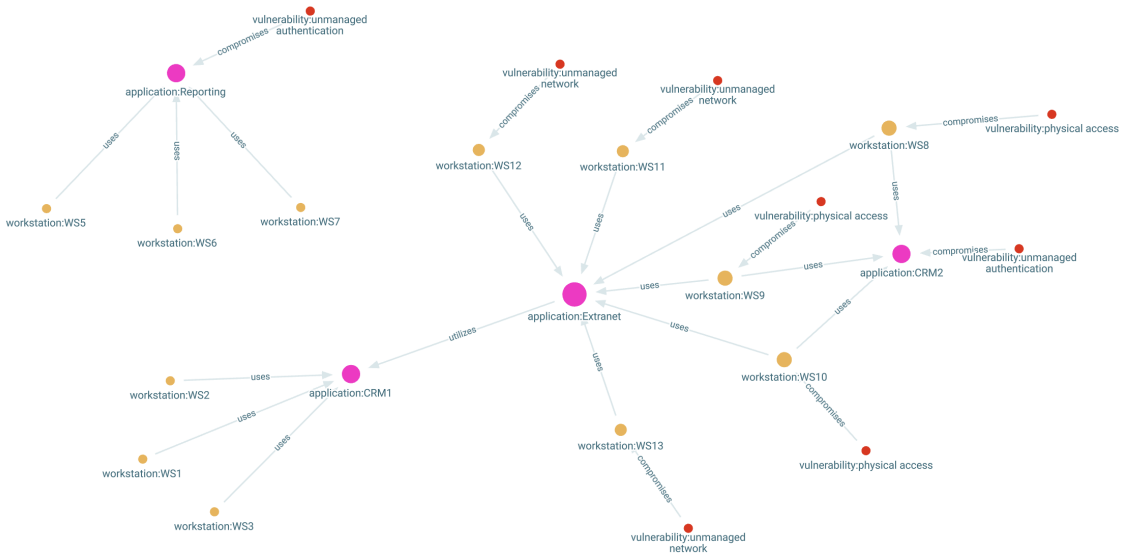
Kuva 4.9. *Tulosjoukko: Työasemia, joilla jokin haavoittuvuus*



Kuva 4.10. *Tulosjoukko: Kaikki haavoittuneet entiteetit*

Vastaavalla tavalla asettamalla $queryFocusLabel = application$ voitaisiin saada haavoittuvuuneiden sovellusten verkosto, joka sisältäisi koulutusluokkien koneilla käytettävän sovelluksen, sivukonttorin CRM-järjestelmän ja tietokannan sekä pääkonttorin raportointijärjestelmän sekä CRM-tietokannan, jotka ilmenevät yhdistetyn kuvan tulosjoukossa 4.10.

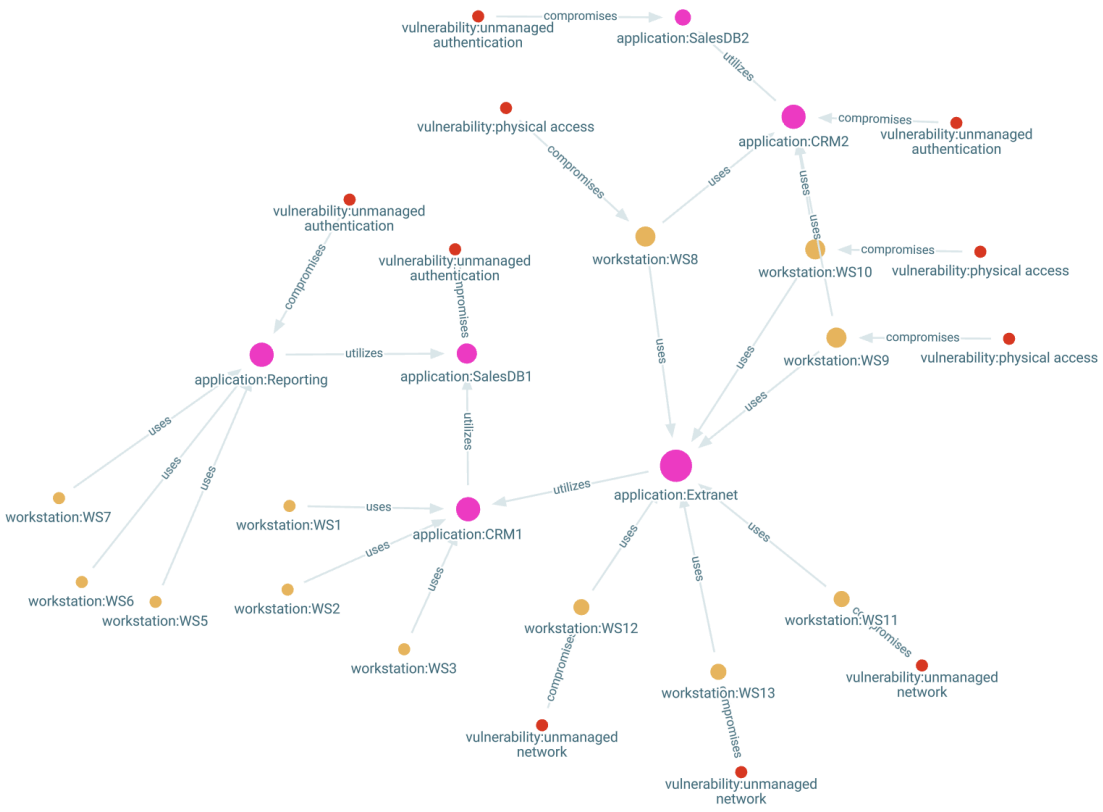
Kuva 4.11. *Kyselygraafi: Haavoittuneet työasemat jotka käyttävät haavoittuneita sovelluksia*



Kuva 4.12. *Tulosjoukko: searchSimilarGraphs algoritmin tulos kyselylle kuvasta 4.11*

Tarkempi kysymys, kuvassa 4.11, olisi, mitkä haavoittuneet koneet käyttävät sovelluksia joissa on haavoittuvuuksia? Parametrisoimalla $queryFocusLabel = workstation$ ja käyttämällä algoritmia $searchSimilarGraphs$ saadaan tulokseksi neljä aligraafia. Tutkimme kahta tulosjoukon graafia, jotka näkyvät kuvassa 4.12. Vertailun vuoksi, teemme saman kyselyn $neighborhoodSimilarity$-algoritmilla, koska yritämme saada esille mahdollisen riippuvuuden näiden kahden erillisen graafin välillä. Tulosjoukossa on nyt kolme graafia, joista oleellinen on esitetty kuvassa 4.13. Toinen algoritmi mahdollisti tutkittavan verkoston venyttämisen samankaltaisuudellaan, jolloin saatiin löydettyä linkki kahden haavoittuneen sovelluksen välillä, eli raportointisovellus ja siihen liitetty tietokanta ei ole keskitetyn tunnistautumisen piirissä.

Lisäksi voidaan huomioida yrityksen tietoturvariskejä yleistasolla – kaikilla Extranet-sovellusta käyttävillä työasemilla on jotain tiedossa olevia haavoittuvuuksia.



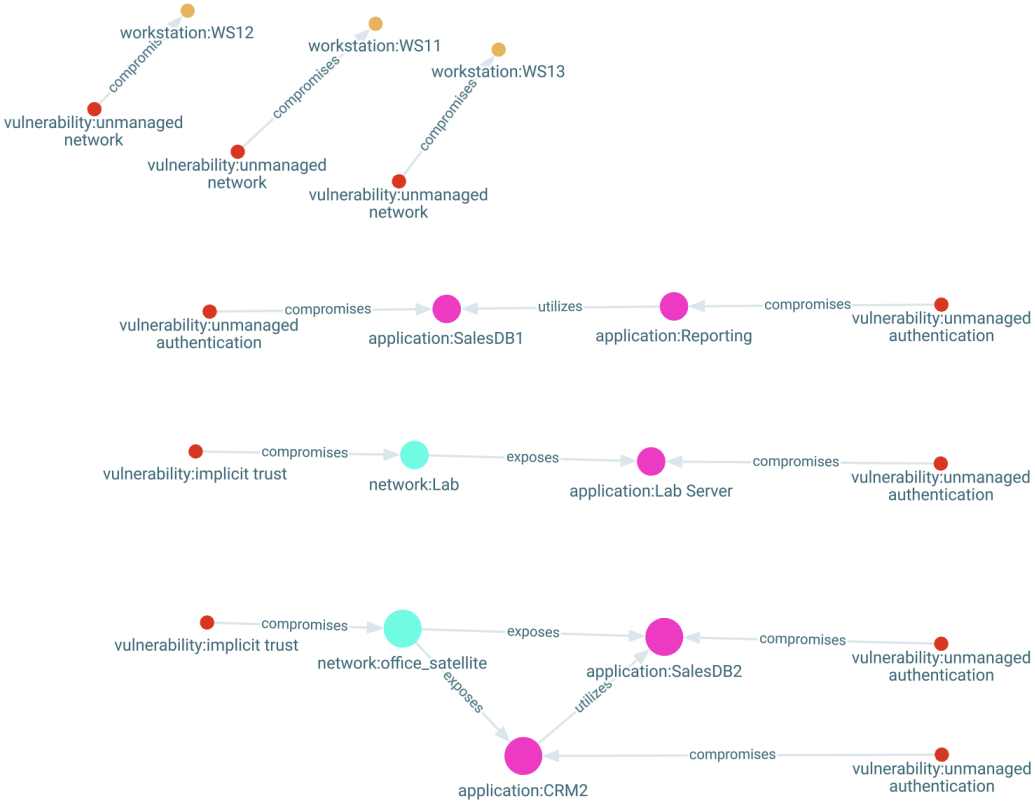
Kuva 4.13. *Tulosjoukko: neighborhoodSimilarity algoritmin tulos kyselylle kuvasta 4.11*

4.3.3 Pienen yritysverkon tietämysverkosto ensimmäisten korjaustoimenpiteiden jälkeen

Ensimmäisenä korjauksena simuloitiin tilannetta, jossa yritys päättää asentaa lukot kaikkiin tiloihin, joissa on havaittu kulunvalvonnan olevan puutteellinen. Haavoittuvuus jonka vaikutusta yritetään pienentää on fyysinen pääsy, physical access. Kun haavoittuneiden entiteettien kyselygraafi 4.8 suoritetaan korjattua tietämysverkostoa vasten, saatiin kuvan 4.14 esittämä verkosto. Kyselyn parametriksi asetettiin $queryFocusLabel = vulnerability$.

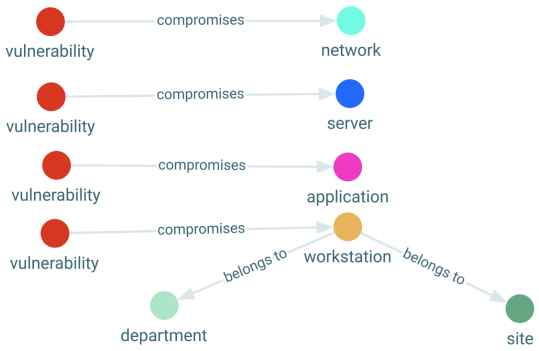
Kuvan 4.14 tulosta tarkastellessa huomataan, että pienen yritysverkon hyökkäyspinta-ala on pienentynyt merkittävästi, koska tietokoneisiin ja palvelimiin on vaikeampi päästä käsiksi, joiden kautta olisi mahdollista väärinkäyttää verkon haavoittuvuuksia. Tulosjoukon kuvaamat haavoittuvuudet sisältävät edelleen huomioitavan matkatyöntekijöiden tietokoneiden hyödyntämät hallitsemattomat verkot, jolloin teoriassa näiltä työasemilta voisi avautua verkostossa haavoittuva polku yrityksen tietojärjestelmiin.

Lisäksi kuvan 4.14 tulosjoukossa esiintyy edelleen keskittämättömät tunnistautumiset yrityksen tietojärjestelmiin, jonka avulla yritys voi tehdä riskiarviota sen suhteen, kuinka kriittisen haavoittuvuuden tieto kuvastaa. Koska korjauksen jälkeen tässä tapauksessa työntekijöiden käyttämät työasemat ovat valvotuissa ja hallituissa tiloissa, voitaisiin riski nähdä



Kuva 4.14. *Tulosjoukko: SimilarityMeasure algoritmin tulos kyselylle 4.8, kun pienessä yrityksessä on hankittu lukot tiloihin.*

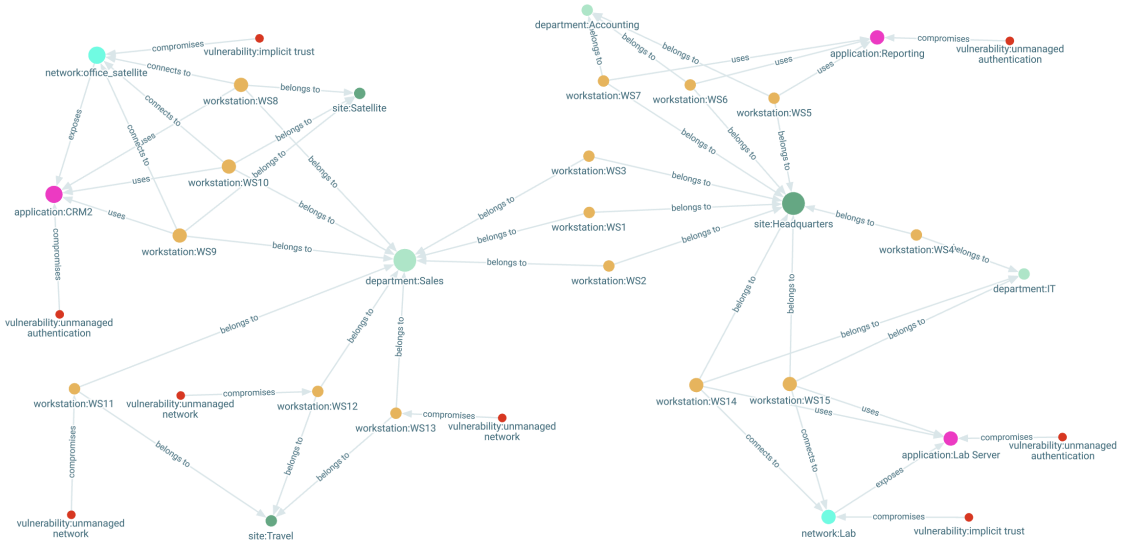
melko pienenä, olettaen että työntekijät ovat luotettavia ja tilojen fyysinen pääsynhallinta toimii odotetulla tavalla.



Kuva 4.15. *Kyselygraafi: Haavoittuneet entiteetit, työasemat osasto- ja sijaintinäkökulmasta*

Yrityksen tietoliikenneverkot opetusluokassa ja sivutoimistolla pitävät sisällään edelleen mahdollisuuden kytkeä laitteita suoraan verkkoon verkolaitteiden ehdottoman luottamuksen periaatteella, eli implicit trust-periaatteella. Koska tilat on nyt lukittu, on riski tuntemattomien laitteiden kytkennän suhteen merkittävästi pienempi. Kuvassa 4.14 kuvatun tiedon avulla yritys voi edelleen tiedostaa, että on olemassa riski ei-toivotun laitteen tahallisen

tai tahattoman kytkennän suhteen.



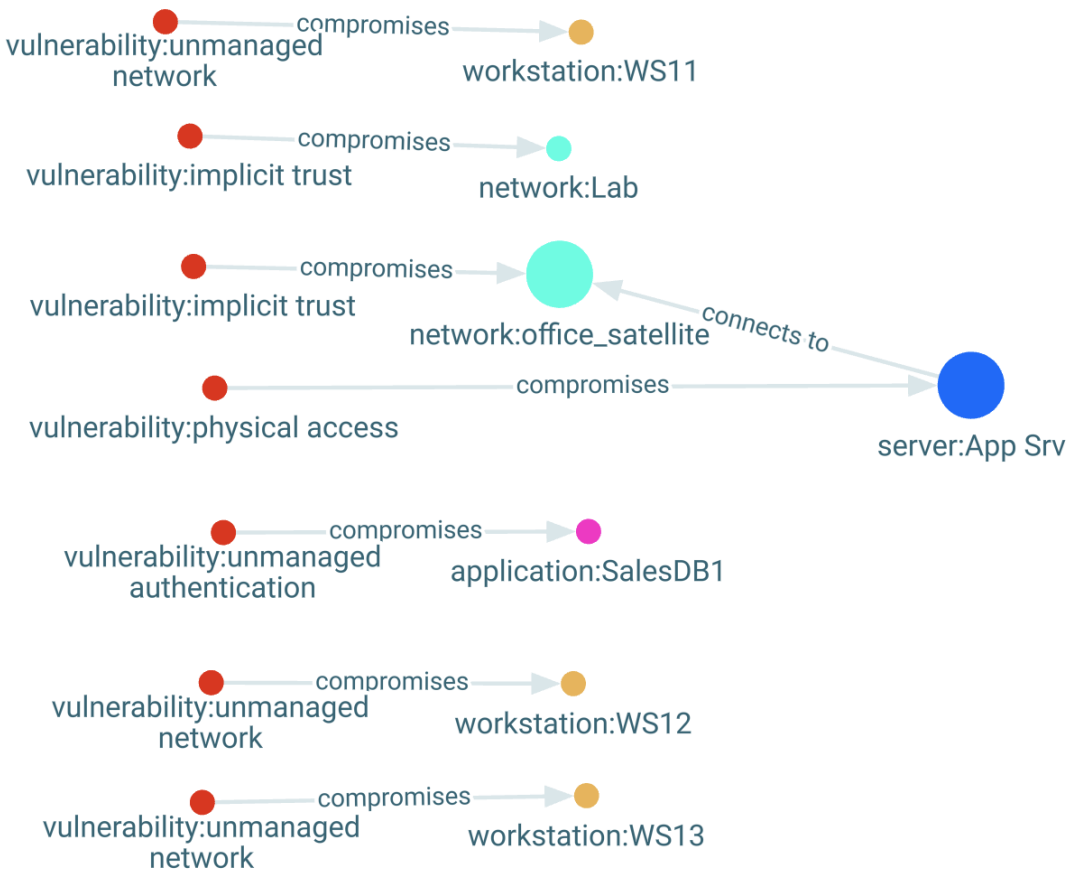
Kuva 4.16. *Tulosjoukko: SimilarityMeasure algoritmin tulos kyselylle 4.15, kun pienessä yrityksessä on lukitut tilat.*

Parannusta voidaan myös tarkastella haavoittuvuuksien ja niihen liittyvien työasemien osastojen ja sijaintien näkökulmasta kyselygraafilla, joka on esitetty kuvassa 4.15. Tulosjoukko on esitetty kuvassa 4.16. Tulosjoukon perusteella voidaan graafisesti todeta ketkä käyttävät työasemia, missä työasemat sijaitsevat ja miten asiat liittyvät työasemalta suoraan saavutettaviin haavoittuvuuksia sisältäviin verkostojen osiin. Kuvasta voidaan tulkita, että myyntiosastolla on toiminnassaan enemmän hallitsemattomia haavoittuvuuksia, samalla kun IT-osastolla ja kirjanpidolla on omansa. Yritys voi näiden päivitettyjen tietojen perusteella päättää tulevista korjaustoimenpiteistä.

4.3.4 Pienen yritysverkon tietämysverkosto vaihtoehtoisen korjaustoimenpiteiden jälkeen

Toisessa tapauksessa toteutettiin keinotekoisesti parannuksia yrityksen haavoittuvuustilanteeseen lukitsemalla tiloja ja parantamalla yrityksen tietojärjestelmiä siten, että ne käyttävät keskitettyä tunnistautumisratkaisua. Korjaustoimenpiteestä jätettiin yksi palvelin lukitsemattomaan tilaan ja yksi sovellus kytkemättä tunnistautmisratkaisuun. Tietoliikenneverkot sivukonttorilla ja opetusluokassa hyväksyy edelleen kaikki kytketyt laitteet. Haavoittuvuudet voidaan nähdä yhteenvetona tulosjoukossa joka on esitetty kuvassa 4.19.

Yksi huomio tilanteen parantumisesta on se, että haavoittuneet entiteetit eivät luo suoranaisesti verkostoja keskenään, mikäli voidaan teoreettisesti olettaa että tietämysgraafi sisältää aukottoman kuvan todellisuudesta ja mahdollisesti realisoituvista riskeistä. Ainoa haavoittuvuuksien verkosto, joka ilmenee kuvan 4.19 tulosjoukosta on sivukonttorin tietoliikenneverkko sekä siihen kytketty palvelin. Tiedon perusteella yritys voisi tulkita esi-

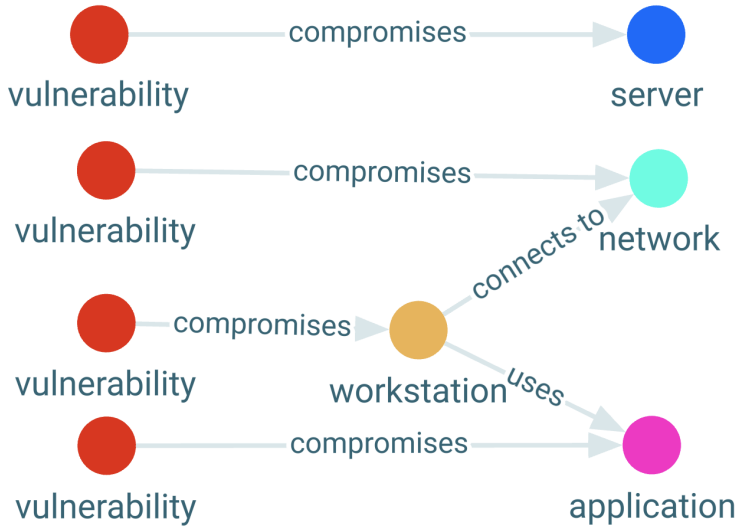


Kuva 4.17. *Tulosjoukko: SimilarityMeasure algoritmin tulos kyselylle 4.8, kun pienessä yrityksessä tehty erinäisiä parannuksia.*

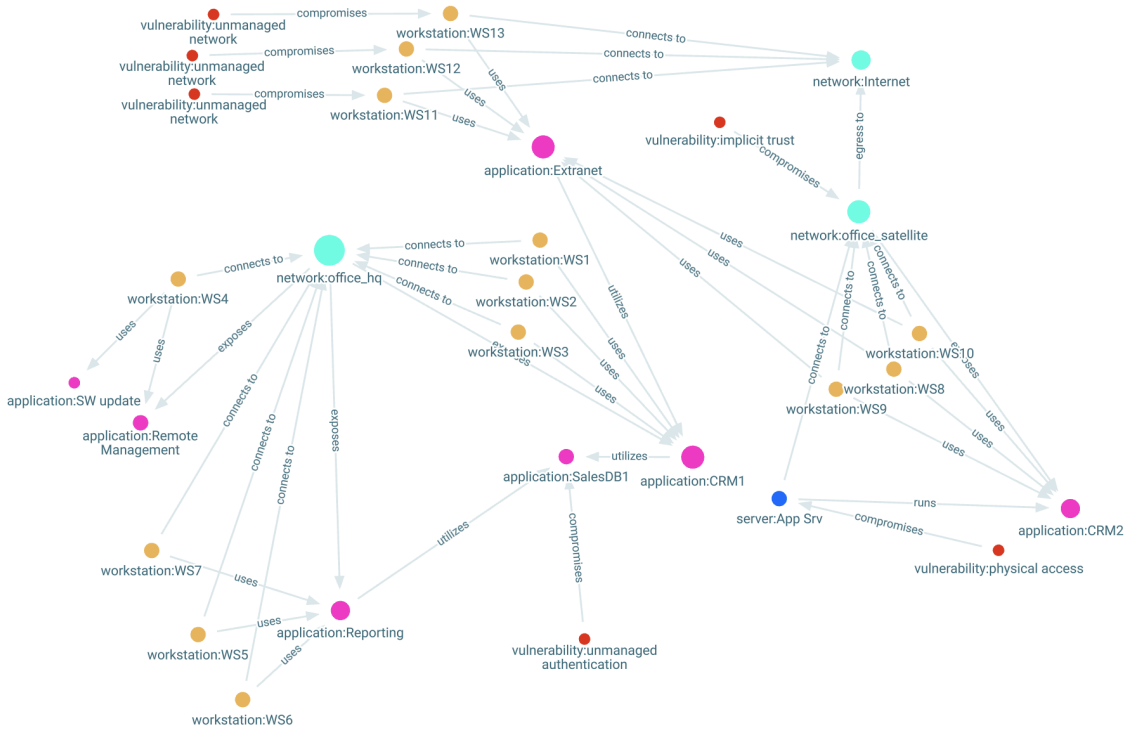
merkiksi: kun palvelin sijaitsee lukitsemattomassa kaapissa ja tietoliikenneverkko toimii ehdottoman luoton periaatteella, voisi asiaton taho kytkeä päätelaitteen palvelimen tilalle tietoliikenneverkkoon.

Toinen johtopäätös mitä yritys voisi osittaisesti korjatun tietämysverkoston analysoimisessa tehdä on, että niin kauan kun on olemassa yksi hyödynnettävä paikallinen haavoittuvuus, on verkosto edelleen haavoittuva. Koska tässä toisessa esimerkissä ei korjauksessa lukittu kaikkia sivukonttorin tiloja, on fyysiseen pääsyyn liittyvällä haavoittuvuudella lähes yhtä suuri mahdollisuus toteutua kuin korjaamattomassa tilanteessa.

Parannettua kokonaistilannetta voidaan tarkastella kuvassa 4.18 esitetyn kyselygraafin avulla, tulosjoukosta joka on esitetty kuvassa 4.19 sekä opetusluokan aligraafi 4.20. Kokonaiskuvista ongelmakohdat ovat pitkälti sitä kuin aikasemminkin, eli matkatyöläisten käyttämät langattomat verkot ja ehdottoman luoton periaattella toimivat tietoliikenneverkot opetusluokassa ja sivukonttorilla. Yksi tietokanta ei vieläkään kuulu keskitetyn tunnistautumisen piiriin, mutta saadun tiedon perusteella yritys voisi tehdä johtopäätöksen, että riskitaso on pienentynyt vaikka ei olisikaan täydellinen, koska työntekijät eivät suoraan



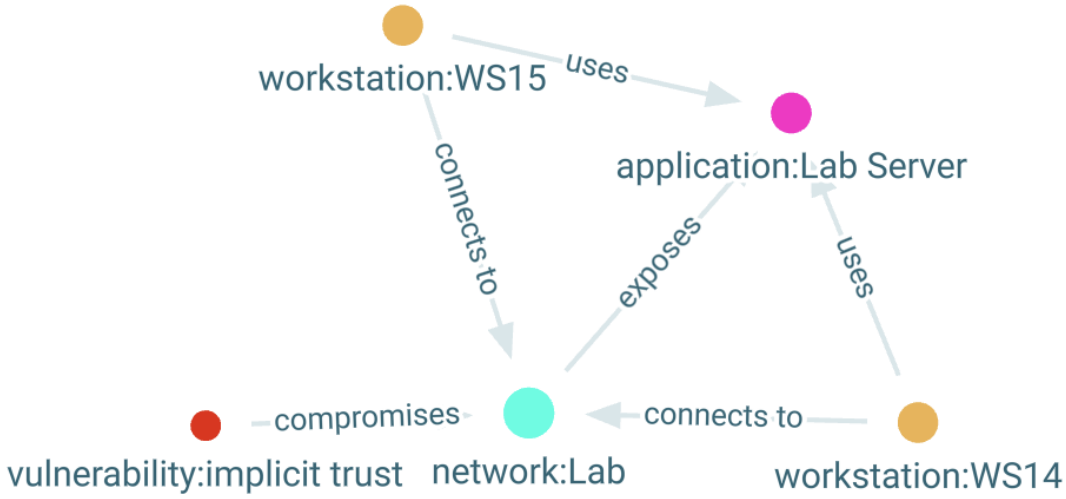
Kuva 4.18. *Kyselygraafi: haavoittuneet palvelimet sekä haavoittuneet tietokoneet, jotka kytkeytyvät haavoittuneisiin verkkoihin tai käyttävät haavoittuneita sovelluksia.*



Kuva 4.19. *Tulosjoukko: neighborhoodSimilarity algoritmin tulos kyselylle 4.18, suuri verkosto, kun pienessä yrityksessä tehty erinäisiä parannuksia.*

käytä sovelluksia jotka eivät ole keskitetyn tunnistautumisen piirissä.

Parantaakseen matkatyöläisten aiheuttamaa riskiä, yritys voisi harkita esimerkiksi VPN-ratkaisujen (Virtual Private Networking) käyttöönottoa yhdessä yrityksen hallitsemien mo-



Kuva 4.20. *Tulosjoukko: neighborhoodSimilarity algoritmin tulos kyselylle 4.18, opetusluokka, kun pienessä yrityksessä tehty erinäisiä parannuksia.*

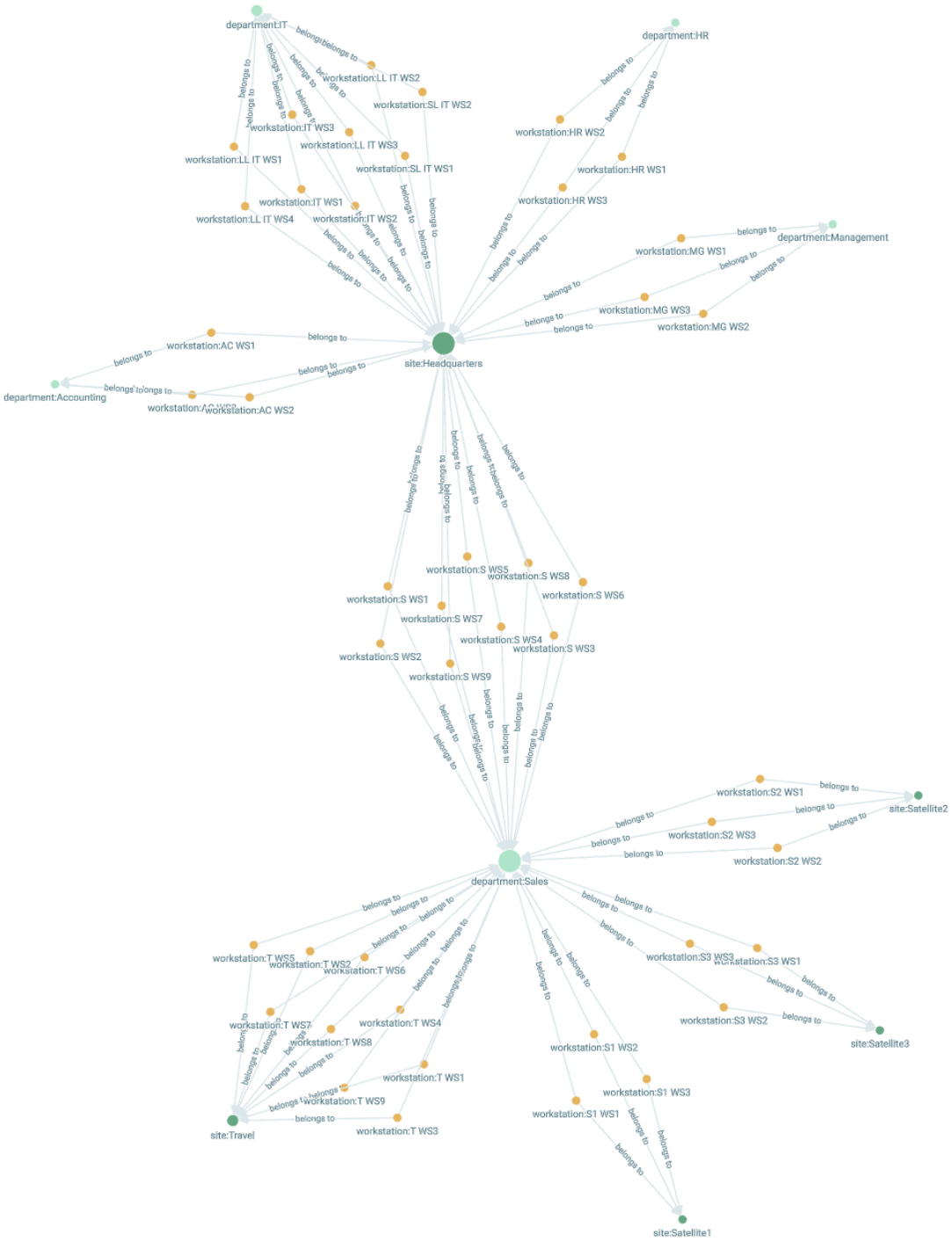
biilireitittimien kanssa, pienentääkseen mahdollisia riskejä, mikä liittyy Extranetin käyttöön. Lisäksi yrityksen tulisi miettiä jotain muuta ratkaisua kuin suoraa yhteydenottoa internetin välityksellä sivukonttorilta Extranetiin, joka voisi myös mahdollistaa keskitetyn CRM-järjestelmän käytön.

4.4 Keskikokoisen yritysverkon tietämysverkoston analysointi

Algoritmeja kokeiltiin myös hieman suuremmalla keinotekoisesti luodulla tietämysgraafilla. Kuvassa 4.21 on esitetty yrityksen rakenne, joka koostuu pääkonttorista, kolmesta sivukonttorista ja kenttämyyntiorganisaatiosta. Lisäksi pääkonttorilla on henkilöstö-, talous- ja hallinto-osasto IT-osaston ja myynnin lisäksi. Tällä suuremmalla yrityksellä on myös kaksi opetusluokkaa, suurempi ja pienempi, jotka ovat kytketty muuhun yrityksen verkkoon. Esimerkkiyrityksen henkilöstövahvuus on n. 50 henkilöä.

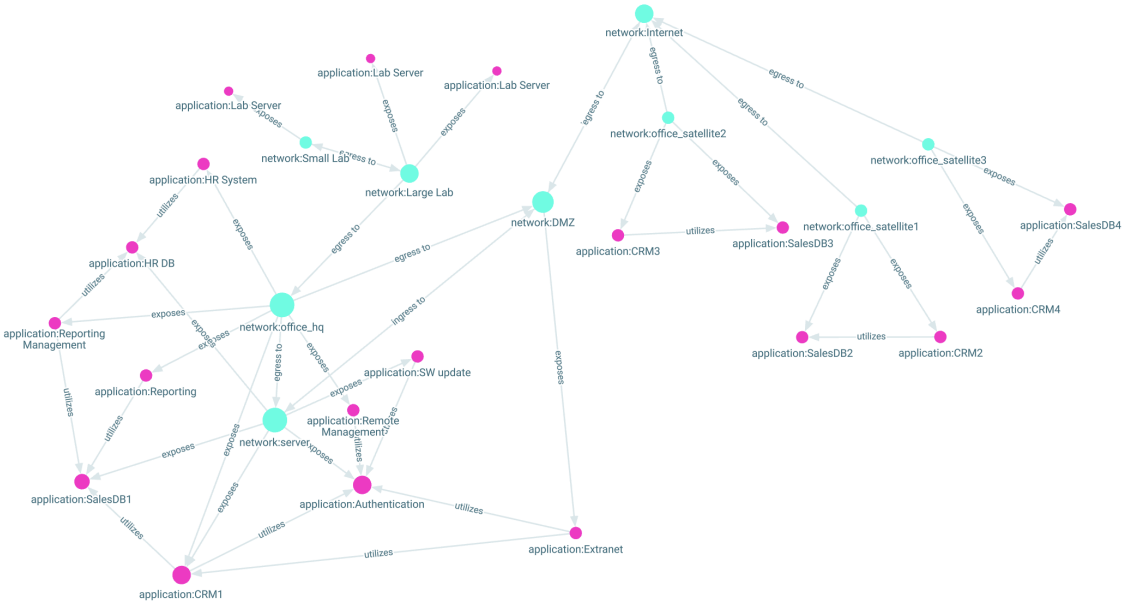
Keskikokoisella yrityksellä on hieman enemmän tietojärjestelmiä, jotka ilmenevät kuvasta 4.22. Jokaisella sivukonttorilla on omat myyntijärjestelmät, jotka ovat erillään pääkonttorin järjestelmästä. Henkilöstöhallinnolla on omiin tarpeisiin räätälöity raportointijärjestelmä, kuten kirjanpidolla. Kirjanpidon raportointijärjestelmä hyödyntää myyntijärjestelmän tietokantaa. Hallinto-osaston raportointijärjestelmä hyödyntää sekä myynnin että henkilöstöhallinnon tietokantoja.

Keskikokoisen yrityksen haavoittuvuudet on esitetty kuvassa 4.23. Ongelmia on samoja kuin aikaisemminkin – sovellukset, jotka eivät tukeudu keskitettyyn tunnistautumiseen, rajoittamaton fyysinen pääsy tiloihin ja koneisiin sekä puuttellisesti hallitut tietoliikenneverkot.

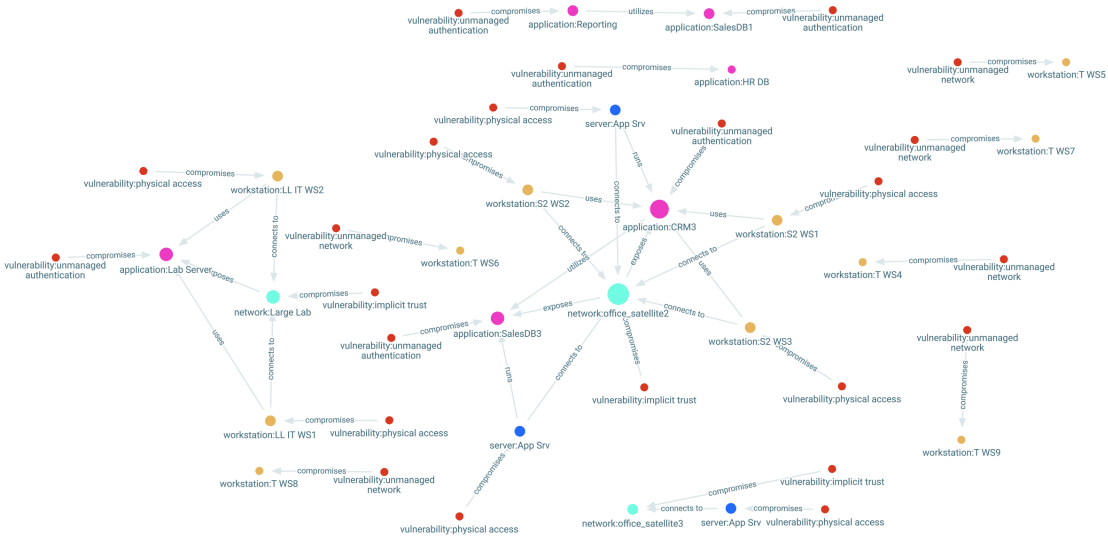


Kuva 4.21. *Keskikokoisen yrityken tietämysverkko: tietokoneet, sijainnit ja osastot*

Kun esitetään kuvassa 4.18 esitetyllä kyselygraafilla ja parametrilla $queryFocusLabel = server$, pyritään selvittämään mitkä verkoston osat muodostavat haavoittuvan kokonaisuuden haavoittuneen palvelimen takia. Tulosjoukko on esitetty kuvassa 4.24. Tämän tiedon avulla yritys voi huomata, että sivukonttoreilla 2 ja 3 on haavoittunut palvelin, jotka on lisäksi kytketty hallitsemattomaan yritysverkkoon. Lisäksi sivukonttori 2:essa on myös työasemia joihin on pääsy asiattomilta. Tiedossa on myös, että sivukonttorin 2 sovellukset eivät tiettävästi käytä keskitettyä tunnistautumisratkaisua.

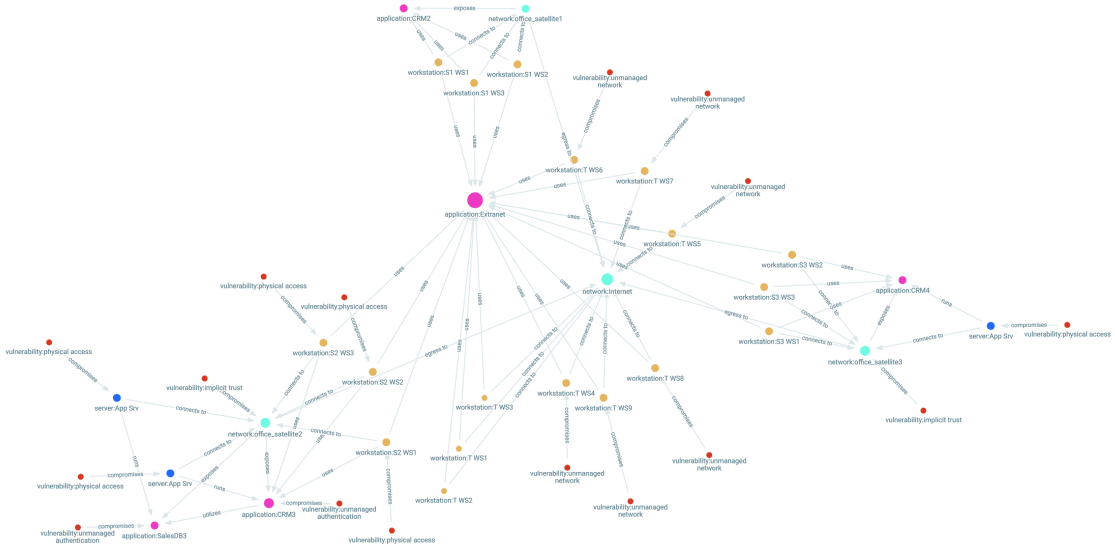


Kuva 4.22. *Keskikokoisen yrityken tietämysverkko: sovellukset ja verkot*



Kuva 4.23. *Tulosjoukko: Keskikokoisen yritysverkon haavoittuvuudet, jotka löydettiin* $neighborHoodSimilarity$*-algoritmillä kuvassa 4.8 esitetyllä kyselygraafilla.*

Kuvitellaan että tässä esimerkkitapauksessa yrityksellä heräisi huoli siitä, että mitä tiedetään keskitetyn tunnistatumisen käytöstä? Jos tietämysgraafilta kysytään sovellusten välisistä yhteyksistä huomioiden haavoittuvuudet haavoittuneiden entiteettien kyselygraafilla 4.8, saadaan vastaukseksi kuvassa 4.25 esitetty tulosjoukko. Tulosjoukosta ilmenee, että tietämysgraafissa on ainoastaan tieto siitä että ohjelmistopäivitys, etähallintapääte, Extranet ja CRM1-myyntijärjestelmä tukeutuvat keskitettyyn tunnistautumiseem. Eksplisiittisesti tietämysgraafista ilmenee, että yksi opetusluokan, henkilöstöhallintojärjestelmän tietokanta, ja myyntijärjestelmän tietokanta sekä taloushallinnon raportointijärjestelmä eivät tukeudu keskitettyyn tunnistautumiseen, kuten ei myöskään yhden sivukonttorin myynti-

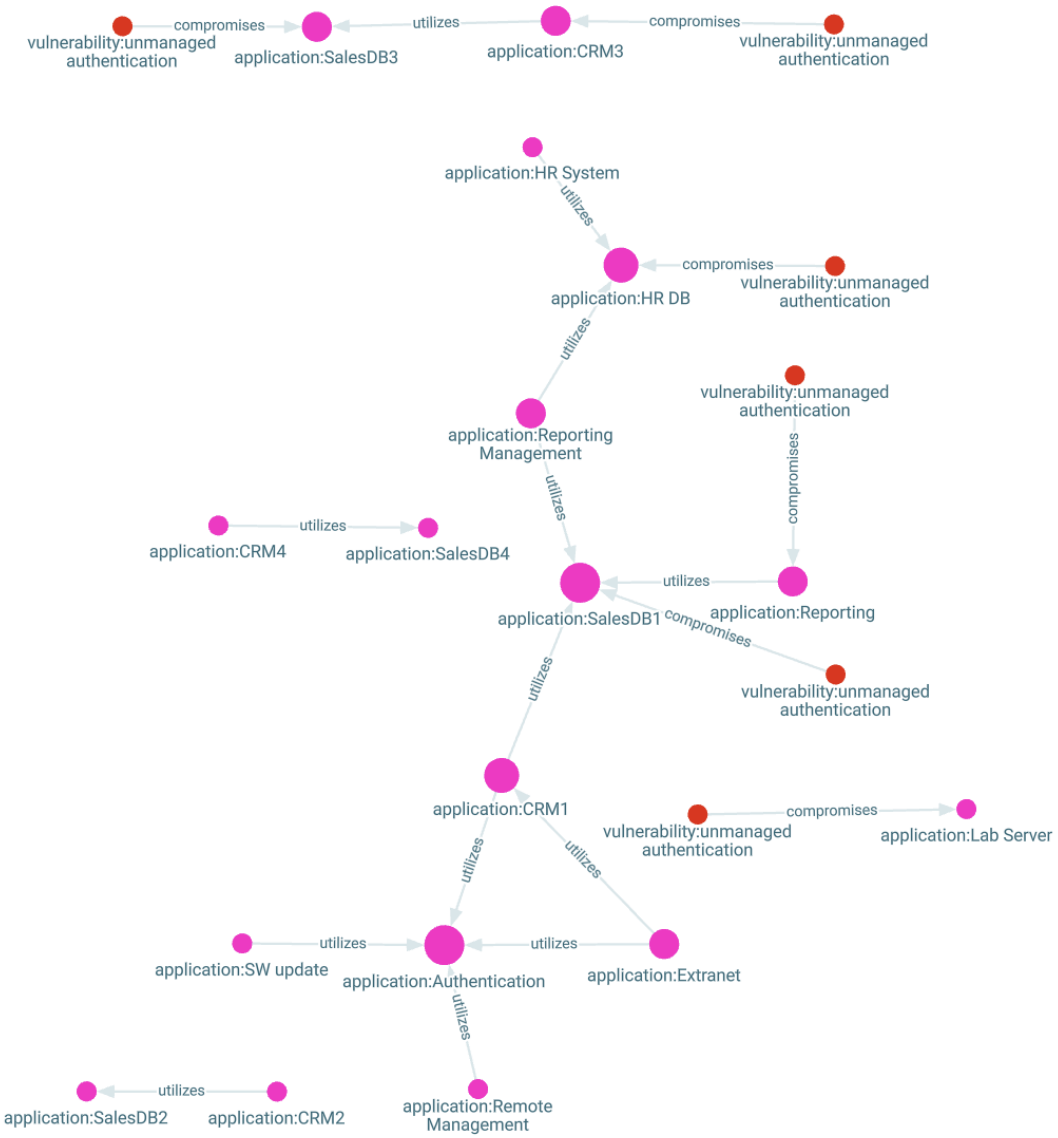


Kuva 4.24. *Tulosjoukko: Keskikokoisen yritysverkon haavoittuneet verkon osat, havoittuneen palvelimen näkökulmasta jotka löydettiin* $neighborHoodSimilarity$*-algoritmillä kuvassa 4.18 esitetyllä kyselygraafilla.*

järjestelmä kokonaisuudessaan.

Tulosjoukon analysoinnin seurauksena yritys voisi toimeksiantaa jatkotoimenpiteet. Selkeät riskit ja haavoittuvuudet voidaan toimeksiantaa korjattavaksi välittömästi. Lisäksi yrityksen tulisi toimeksiantaa selvitys, onko kaikki tieto olemassa tietämysgraafissa – ts. selvittää asioiden todellinen laita ja täydentää tietämysverkostoaan niiltä osin kun se on mahdollista. Analyysissä tulisi ottaa huomioon myös asioita joita ei suoraan nähdä, esim. haavoittuvuuden puuttuminen voi tarkoittaa joko jonkin haavoittuvuuden toteutumisen, haavoittuvuutta ei ole tai ei ole tiedossa päteekö haavoittuvuus. Parantaakseen ymmärrystä yrityksen olisi hyvä keskittyä etenkin niihin osioihin tietämysverkostossa, jossa tietoa on vähän ja tehdä arvioita mahdollisten toteutuvien riskien kautta missä tietämysgraafin osissa(esim. osasto tai sijainti) tai solmutyyppeissä (esim. työasema, palvelin) tai vierekkäisissä pisteissä tiedon rikastamisella olisi suurin vaikutus.

Tässä tilanteessa ja näillä työkaluilla, yritys voisi käynnistää selvitystyön sivukonttoreiden tilanteesta ja täydentää tietämysgraafia kerättyjen tietojen osalta. Olemassa olevilla tiedoilla, korjaustoimenpiteet voisi aloittaa sivukonttori 2:selta. Lisäksi yrityksen johto voisi toimeksiantaa IT-osaston selvittämään sovellusten siirtyminen keskitettyjen tunnistautumisratkaisujen piiriin, tietoliikenneverkkojen hallintaan ja esimerkiksi VPN- ja yhdyskäytäväratkaisujen käyttöönottoon.



Kuva 4.25. *Tulosjoukko: Keskikokoisen yritysverkon haavoittuneet applikaatiot ja niiden välinen yhteistoiminta,* $neighborHoodSimilarity$*-algoritmillä kuvassa 4.8 esitetyllä kyselygraafilla.*

5. YHTEENVETO

Alkuperäisestä suunnitelmasta poiketen päädyttiin toteuttamaan uusi, hieman sovellettu versio PINGS-algoritmista, kuitenkin alkuperäistä määrittelyä mahdollisimman tarkasti noudattaen. Tämän työn tuotoksena syntyi PINGS-algoritmin kontekstissa muodostuneita parannusehdotuksia, joilla voisi tehdä enemmän poikkeuksellisia löydöksiä tietämysverkostoista sekä käytännöllisiä parannuksia ja parannusehdotuksia jotka toisivat algoritmin askeleen lähemmäksi arkista käyttöä. Luvussa 1.1 esitettyyn ensimmäiseen tutkimuskysymykseen eli mitä PINGS-algoritmin käyttöönotto vaatii – PINGS-algoritmin todellinen käyttöönotto vaatii uudelleenimplementoinnin saatavilla olevalla tiedolla, ts. mitään julkisessa tiedossa olevaa valmista tuotetta ei ole saatavilla.

Tämän työn perusteella huomattiin graafien ja graafialgoritmien olevan edelleen käyttökelpoinen menetelmä tiedon louhimiseen suurista tietomassoista. Vaikka työn rajauksen takia ei sen tarkemmin huomioitu algoritmien sisäistä toimintaa ja suorituskykyä, huomattiin teknisessä toteutuksessa että suurella tietomäärällä toteutustavoilla, tallennustavalla ja tietokannan sisäänrakennettujen operaatioiden hyödyntämisellä voi olla merkittäviä vaikutuksia siihen, voiko jokin algoritmi olla skaalautuvasti käyttökelpoinen tai taloudellinen.

Tätä työtä varten algoritmin kokeilua varten kehitetty tietomassa verkostoon liitetyistä koneista ja haavoittuvuuksista auttoi havaitsemaan alkuperäisen algoritmin puutteita ja soveltuvuutta geneeriseen ongelmanratkaisuun. Tutkimusta tehdessä kävi myös ilmi, että ongelma, ilmiöiden paikantaminen epätäydellisillä tiedoilla, ei ole luonteeltaan yksinkertainen ratkaista. PINGS-algoritmi toimii paremmin homogeenisen verkostorakenteen kanssa, ja selkeästi haastavempaa on toimia heterogeenisella aineistolla. Yksinkertainen pisteytys ja suodatus ei välttämättä kerro mitään oleellista kyselijälle vaikka tietokoneella tai ohjelmalle voi olla selvää miten jokin asia lasketaan. Käyttökelpoisuuden näkökulmasta tärkeintä on mitä vastauksella voidaan tehdä ja miten vastausta voidaan parantaa tai mitä vastauksessa tulisi voida korostaa.

Työn edetessä tuloksia tarkastellessa huomattiin, että ongelman ja ratkaisun keskiössä oli enemmän kyse kokonaiskuvan ymmärtämisestä ja tiedon visualisoimisesta, mikä antoi oman syötteen tähän yhteenvetoon – huomattiin että ollakseen toimiva työkalu, on otettava huomioon mitä käyttäjä tarvitsee työnsä tueksi. Luvussa 1.1 esitetty toinen tutkintakysymys, miten PINGS-algoritmia voitaisiin käyttää työkaluna kuvattujen kokonaisuuksien

paikantamisessa tietoaineistosta – vastaukseksena voidaan todeta että PINGS-algoritmi kykenee löytämään kyselygraafilla kuvattuja ilmiötä, korostaen tulosjoukossa lähes täydellisesti vastaavia verkoston osia.

Seuraavissa yhteenvedon luvuissa vastataan käytännöllisen tietokoneohjelman tai menetelmän tavoittelun kannalta luvussa 1.1 esittyyn kolmanteen tutkimuskysymykseen: Miten PINGS-algoritmia voitaisiin jatkokehittää yleishyödyllisemmäksi työkaluksi? Lyhyttä ja ytimekästä vastausta tähän tutkimuskysymykseen ei ole helppo antaa, koska parannettavaa potentiaalia on monella alueella. Lähtökohtaisesti PINGS-algoritmi itsessään on melko raakile, jonka julkisesti saatavilla oleva toteutus palvelee enemmän tutkimuksellisia tarpeita kuin liiketoimminan tarpeita.

5.1 Kyselyn konfiguroitavuuden merkitys lopputulokseen

Tämän työn edetessä, kyselygraafeja ja niitä käyttäessä, huomattiin että on tärkeätä kyetä parametrisoimaan nopeasti kyselyä. Riippuen kyselygraafista, tulosjoukko oli odotuksen mukainen tai ei. Tämän puutteen tueksi kehityssovellukseen tehtiin tiputusvalikko, josta pystyi tarkastelemaan ja valitsemaan kyselyn, sekä nopeasti vaihtamaan $queryFocusLabel$-muuttujan arvon. Lisäksi tehtiin mahdolliseksi vaihtaa nopeasti $SimilarityMeasure$ ja $neighborHoodSimilarity$ algoritmien välillä.

Koska tutkimusmateriaali, eli kehitetyt tietämysverkostot, olivat rakenteeltaan melko heterogeenisiä verrattuna alkuperäisen PINGS-algoritmin käyttämään julkisesti saatavissa olevaan materiaaliin[28], huomattiin että $similarityThreshold$ pisterajaa pidettiin lähes kokoajan nollassa, ts. tarkasteltiin koko tulosjoukkoa ettei oleellisia osia jää pois.

Suuresta verkostosta tuli helposti useampi samanlainen osuma samoilla solmuilla ja pisteillä, mikä johtui siitä että graafia tarkastellessa algoritmi kykeni saamaan kuljettua saman aligraafin kyselygraafin perusteella useasta eri pisteestä. Tämä tuki ajatusta siitä että mielivaltaisesta pisteestä kuljettuna algoritmi toimii odotetulla tavalla, mutta ei saa pienestä tietämysgraafista suurta määrää aligraafeja, ellei graafissa ole jotain mikä katkaisee etenemisen välillä.

Yhtenä oivalluksena kyselyn konfiguroitavuutena oli esimerkiksi kuvassa 4.18 esitetty kyselygraafi joka koostuu kahdesta komponentista. Suurempi komponentti kuvaa mahdollisesti haavoittuneen työaseman, joka kytkeytyy mahdollisesti haavoittuneeseen verkkoon, joka kytkeytyy mahdollisesti haavoittuneeseen sovellukseen. Pienempi komponentti kuvaa taas palvelimen, jolla on haavoittuvuus.

Kun tulosjoukkoon saatiin myös erillisenä löydöksenä haavoittuneet palvelimet, voitiin kaikki tulosjoukon graafit yhdistää toisiinsa esitystä varten, jolloin saatiin eheä graafi, joissa asiat liittyivät toisiinsa mikäli löydettyjen solmujen välillä oli yhdistäviä viivoja.

5.2 Graafinen ratkaisu keinona kokonaisuuden hahmottamiseen

Tulosjoukkoja tarkastellessa huomattiin, että sopivalla rajauksella ja tarkastelulla ne voisivat toimia sellaisenaan analyysin tukena. Tätä menetelmää käytettiin osittain luvussa 4 kun kuvattiin keinotekoisten esimerkkiyritysten toimintaympäristöä ja mahdollisia vastatoimia tulkituille haavoittuvuuksille.

Riippuen kyselygraafista, joka algoritmista johtuen ei sisällyttänyt muuntyyppisiä solmuja kuin kyselyissä esiintyviä, muodosti selkeän tarpeen kyetä nähdä jonkin solmun ympäriltä seuraavat solmut. Riippuen tietämysgraafin rakenteesta, lopetti algoritmit jossain vaiheessa kulkemisen. Huomioiden sen, että tietämysgraafin ja tutkimustyön taustalla olisi sama taho, kuten tässä työssä, tulee objektiivisessa mielessä huomioida sellaisten työkalujen ja ratkaisujen tarve, jotka mahdollistavat tuntemattoman tai ruutupaperia suuremman aineiston tutkimisen.

Jotta tätä työtä varten saatiin esitettyä oleellinen tieto tulosjoukosta, parannettiin kyselygraafeja jotta niiden avulla saataisiin vastaukseksi oikentyyppisiä asioita jossa on riittävä määrä yksityiskohtia. Tämän ratkaisun toteuttamisen teki helpommaksi se, että tutkittavat tietämysgraafit tunnettiin ja toteutettiin itse.

Keskeisten solmujen, eli solmujen joilla on paljon sisääntulevia yhteyksiä, on hyvä voida korostaa. Tällä asetuksella saatiin pientä hyötyä esityksen selkeyteen, kun keskipisteenä olevat solmut voitiin esittää hieman isompina kuin yksittäiset solmut jotka pääasiassa toimitti metatiedon asemaa, kuten esim. haavoittuvuus.

Hyödyllistä oli myös suodatusominaisuus, joilla tiettyjä solmutyyppejä voitiin jättää pois tulosjoukoista. Tätä ominaisuutta käytettiin joidenkin graafien esittämiseen, jotta saatiin tarpeettomat solmut pois kuvasta, jos oli tarkoitus esimerkiksi esittää tietokoneiden, osastojen ja sijaintien välistä verkostoa. Nopean suodatuksen avulla oli myös mahdollista nopeasti tarkistaa kehitettävien tietämysgraafien loogista rakennetta, koska pienetkin graafit alkavat muuttua vaikeaselkoisiksi jos niissä esitetään kaikki tieto.

5.3 Mahdollisia vaihtoehtoja jatkokehitykselle tai -tutkimukselle

Aluksi oletus oli, että tarkkojen löydösten tekeminen on helppoa, ja tämä kävi myös ilmi valjastettua algoritmia soveltaessa. Vaikeampaa on edelleen löytää tietoa epätarkasti joko siten että kysymys on epätarkka tai vastaus on epätarkka. Koska suuressa verkostossa aligraafeja on määrällisesti paljon, ei kaikkien löydösten käsitteleminen tai vastaukseen sisällyttäminen ole käytännöllistä.

Kun käsiteltävää tietoa on paljon, muodostuu haasteeksi saatavilla oleva laskentakapasiteetti sekä käsittelyn lopputuotteena loppukäyttäjälle suuri määrä tietoa analysointia varten. Vaikka erinäiset tietojärjestelmät ja algoritmit voisivat löytää paljonkin tietoa, on myös

joillain keinoin kyettävä rajaamaan esitettäviä tuloksia. PINGS-algoritmissä tulosjoukon rajauksena toimii pisteytytys jolla kuvattaan vastauksen vastaavuutta kyselyyn. Jatkotutkimuksen kannalta aihepiiristä löytyy useita vastaamattomia kysymyksiä, tai kysymyksiä joihin voi pohtia parempia ratkaisuja.

5.3.1 Löydösten välisten polkujen poiminta

Käyttäen jotain lyhin polku-algoritmia, voisi sellaisen poimintametodin kehittää joka poimii valittujen algoritmin löytämien aligraafien väliltä polun ja matkalla olevat vierekkäiset pisteet sekä muita aligraafeja. Mikäli polkuja on useampi, voisi parametrilla olla myös mahdollista palauttaa kaikki polut pisteiden välillä, ja niitä voisi tarkastella kokonaisuutena tai erillisinä löydöksinä. Tällä ratkaisulla voitaisiin vastata kysymykseen mitä ei-vierekkäisten naapurustoverkostojen välillä on tai onko niiden välillä mitään tiedossa olevaa yhteyttä.

Yksi tapa toteuttaa toiminto olisi kahden yhdistämättömän graafin pisteiden valinta tulosjoukosta ja pyytää algoritmia etsimään graafien välinen polku.

5.3.2 Naapurisolmujen esittäminen

Kun kyselygraafilla on saatu tulosjoukko, saattaa herätä kysymys: mitä tämän tai näiden pisteen lähettyvillä on, jota tulosjoukosta on jätetty pois? Eli kysyjä kysyy itseltään, osasinko kysyä oikealla tavalla? Kun PINGS-algoritmin tulosjoukko ei sisällä muuntyyppisiä solmuja kuin mitä kysely esittää, olisi hyödyllistä jos voisi esim. käyttöliittymästä valita solmun tai solmutyypit ja pyytää näyttää seuraavat solmut, riippumatta tyypistä. Tämän toiminnon avulla käyttäjä voisi löytää tietoa tutkittavaksi, mitä ei aluksi osanut kysyä mutta tulosjoukko johdatteli tutkimaan. Tällä ratkaisulla voisi parantaa PINGS-algoritmin toimivuutta ja hyödyntämistä heterogeenisissä tietämysverkoissa.

5.3.3 Jokerisolmut, jokerikaaret ja jokerihypyt

Algoritmista ja kyselygraafeista voitaisiin tehdä sallivampia ennakoimattoman tiedon kannalta. Käyttäjä voisi tällöin kyselygraafissa kuvata jonkin tuntemattoman liitoksen tai asian jokerimerkillä, jonka käyttäjä olettaa olevan olemassa ja merkittävä tekijä tuloksen muodostamisessa. Näin algoritmin suorituksen tulosjoukossa voisi esiintyä laajempi otanta analyysia varten. Jokerimerkein tuettu kyselygraafi ja algoritmi voisi olla avuksi sellaisessa tilanteessa kun käyttäjä etsii suuresta tietämysgraafista tavalliselle kyselygraafille piilossa olevaa löydöstä. Tässä tapauksessa käyttäjä ei tavallisella kyselygraafilla kykenisi kuvaamaan tarkasti löydettävää rakennetta olemassa olevalla tiedolla, jolloin kyselijä tukeutuisi jokerimerkein tuettuun kyselygraafiin.

Kyselygraafissa jokerisolmulla, -kaarella ja -hypyllä tarkoitetaan vastaavankaltaista toi-

minnallisuutta kuin jokerimerkillä tietojenkäsittelytieteessä. Jokerimerkki, esim. $*$, $\%$ tai $?$ sallii hauissa jonkin osan merkkijonosta esittävän tuntematonta tulosta. Esimerkiksi kysely merkkijonolle $tyyppi = tieto*$ voisi löytää tietokannasta rivit, joissa nimi on tietokone tai tietojärjestelmä.

Jokerisolmulla ja jokerikaarella voitaisiin kyselygraafiin vastaavasti määritellä mahdollinen solmu ja kaari, joka voisi toimia kyselygraafin laajennuksena. Jokerisolmulla ja -kaarella voitaisiin jatkaa aligraafin poimintaa siten, että jokerisolmun ja -kaaren tilalle voi poimiintua mikä tahansa yhdistettu solmu ja kaari, ja jokerisolmusta eteenpäin pyritään louhia saman kyselygraafin vastinetta. Käyttäjä voisi halutessaan lisätä haluamansa määrän jokeriyhteyksiä kyselygraafille. Jokeriominaisuuksilla pyrittäisiin laajentamaan hakutuloksia sallivammin, siten että saadaan mahdollisesti laajenettua vastausta graafista joka sisältää muitakin tietotyyppejä mitä kyselygraafilla kuvataan.

Jokerihyppy voisi olla minkälainen yhteys tahansa kyselygraafin kahden osan välillä. Jokerihypyn prototyyppi voisi toimia siten että ensiksi louhitaan kyselygraafit purettuna alikyselygraafeihin, jotka sisältävät hyppyjen eristämät osat. Kahden tulosjoukon välisiä aligraafeja tarkastellaan keskenään, löytyykö graafien välillä polkuja. Jokerihyppy saattaa olla raskas operaatio laskea, jolloin tulee myös vastausten mielekkyyden kannalta miettiä tuleeko jokerihyppyä voida parametrein rajata.

5.3.4 Negatiivinen tietämysgraafi ja negatiiviset löydökset

Jossain tapauksissa kun tietämysgraafista löytyy paljon osumia, voisi olla hyödyllistä kyetä näkemään ne asiat mitkä jäivät poiminnan ulkopuolelle. Jäljelle jäävää negatiivista tietämysgraafia voisi yleisellä tasolla tarkastella. Negatiivinen tietämysgraafi olisi teknisesti yksinkertainen ja graafin koosta riippuen helppo koostaa. Negatiivinen löydös, ns. epälöydös kuvaisi taas $searchSimilarGraphs$ algoritmin tutkimaa graafia, joka ei kuitenkaan päätynyt löydökseksi joko liiallisen erilaisuuden tai pisteytysrajan vuoksi. Negatiivinen löydös voitaisiin louhia puhtaasti isomorfisesta näkökulmasta, eli kyselygraafia muistuttava muoto pelkästään solmujen ja viivojen näkökulmasta pl. solmujen ja viivojen tyypit.

5.3.5 Keinotekoisesti laajennettu tietämysgraafi

Keinotekoisesti laajennettu tietämysgraafi olisi muunnos alkuperäisestä tietämysgraafista jota voitaisiin jollain toistaiseksi tuntemattomalla algoritmilla keinotekoisesti täydentää pyydettäessä. Esitetylle laajennusalgoritmille käyttäjä voisi kuvata tapoja muodostaa graafille puuttuvia yhteyksiä pisteiden ja graafin komponenttien tai toisiinsa liittymättömien aligraafien välillä. Kehittyneempi ratkaisu voisi tutkia graafista miten pisteet liittyvät toisiinsa, ja lisätä puuttuvia yhteyksiä ja mahdollisesti keinotekoisia pisteitä. Keinotekoisesti lisättyjen verkoston osien tulisi ilmetä loppukäyttäjälle yksiselitteisesti.

Vaikka pienessä tietämysverkostossa henkilö voisi tarkastelemalla ja arvailemalla täydentää graafia spekuloiduilla yhteyksillä, voisi keinotekoisesta graafin täydennyksestä olla hyötyä suuren tietoaineiston käsittelemisessä ja aligraafien esille nostamisesta sellaisissa tapauksissa joissa aligraafit ei muuten saisi riittävästi pisteitä. Jos tietämysverkosto tunnetaan englanniksi termillä knowledge graph, voitaisiin keinotekoisesti laajennettu graafi tuntea nimellä augmented knowledge graph, vrt. augmented reality (AR) eli laajennettu todellisuus.

Tietämysverkoston täydentämiselle voisi antaa raja-arvoja jonka perusteella täydennys tehdään, esim. kuinka puutteellisia aligraafeja täydennetään ja täydennysasteet. Mikäli käytettävissä oleva laskentateho, toteutus ja tietämysverkoston suuruus ovat keskenään sopusuhdassa, voisi graafeja täydentää kyselyhetkellä. Idempotentti täydennysalgoritmi toteuttaisi samalla syötteellä ja samoilla parametreilla saman lopputuloksen. Toiminnallisuuden rinnalle olisi hyvä myös tehdä kaoottinen täydennysalgoritmi jolla voitaisiin vastata mitä jos-skenaarioihin. Kaoottinen algoritmi voisi myös muuttaa joitain asioita tietämysverkostosta kyselyä varten, jolloin voitaisiin nähdä asioita jos tietämysgraafin tiedot ovat väärät.

Keinotekoisesti lisättyjen verkoston osien avulla voisi myös arvioida, mitä oikeata tietoa graafista puuttuu, mitä tutkijoiden tulisi hankkia tai varmistaa tiedon olemassaolo. Jollain tavalla voisi myös ehkä merkitä osia graafista siten, että merkittyjen osien ympärille ei muodosteta keinotekoisia liittymiä.

5.3.6 Tulosten poiminta ja pisteytys

PINGS-algoritmi ja tässä työssä sovellettu versio algoritmista suodattaa ja järjestää löydöksiä pisteiden perusteella. Koska kyselygraafi on pelkkä arvaus kuten vastaus, tulisi ratkaisun sisältää jokin mekanismi millä voisi joustavasti pisteyttää kyseltävän ilmiön lisäksi osailmiöitä. Osailmiöllä tarkoitetaan jotain pientä asiaa, mitä voidaan liittää löydettäessä osaksi kyselgraafia tapauskohtaisesti, mutta ei itsessään ole osana kyselygraafia tai hakijan kysymystä. Kyseessä olisi lisäparametreja, joiden avulla voitaisiin ottaa huomioon ja parantaa korostettavien löydösten laatua ja määrää. Tällä toiminnallisuudella voitaisiin parantaa algoritmin toimivuutta täydentyvässä tietämysverkostossa jossa on muutakin kuin pelkästään absoluuttisesti oleelliseksi nähtävä tieto. Jos kyselygraafi kuvastaa ilmiötä jota etsitään, voisi osailmiö vastaavasti kuvastaa käytöstä tai kyselyn yhteydessä löydettyyn yhteenliittyymään jolla voi olla pisteitä nostava tai laskeva merkitys.

Kyselygraafin osana voisi myös olla negatiivisesti pisteyttäviä osia, tai vastauksessa vältettäviä osia. Mikäli poimittavassa löydöksessä törmätään kielletyyn osaan, se jätettäisiin tulosjoukosta. Negatiivisilla pisteillä voisi jonkin löydöksen arvo jäädä minimipisteiden alle, mutta eivät välttämättä estäisi tulosjoukkoon päätymistä.

5.3.7 Operaatioiden käyttöliittymä jatkokehityksenä

Kun tai jos mikä tahansa algoritmi päätyy tutkijoiden tai kehittäjien tietokoneilta liikeelämän käyttöön, tulisi käytettävyys huomioida. Etenkin jatkokehitetyn algoritmin parametrisointi tulisi olla helppoa ja ymmärrettävää, sekä vastausten kuvaaminen. Täydennetty tietämysverkosto, kyselygraafit, osailmiöt tulisi voida automaattisesti esittää käyttäjälle ja kuvata siten, että käsittelijä voi eritellä täydennetyn ja tiedossa olevan tiedon. Hyödyllistä olisi myös auttaa käyttäjää ymmärtämään täydennetyn tiedon yhteisvaikutus löydöksiin sekä löydösten pisteytykseen. Mahdollisen kaoottisen algoritmin tekemät muutokset ja lisäykset voisi automaattisesti kuvata, jolloin käyttäjä voisi tietää kysymyksen vaikutuksen vastaukseen tai mikä entä–jos-skenaario olisi kyseessä.

Tätä työtä tehdessä huomattiin, että alkuperäisessä julkisesti tiedossa olevassa Neo4J-toteutuksessa ei ollut mitään ihmeellistä käyttöliittymää, ja tätä työtä varten algoritmin kokeiluun tehtiin pieni käyttökelpoisempi ratkaisu. Ensimmäinen käytettävyysasia voisi olla jonkinnäköinen projektirakenne ja mekanismit tiedon tuontia ja päivittämistä varten. Käyttöliittymässä tulisi olla keino kyselygraafien muokkaamiseen ja kehittämiseen. Huomioita tulisi myös kiinnittää graafien tasapainoiseen ja selkeään esittämiseen ja asetteluun näytöllä.

5.3.8 Koneoppimismenetelmien soveltaminen

Koska graafeja voidaan kuvata n-ulotteisina matriiseina, voisi yksi jatkotutkimuksen kohde olla koneoppimismenetelmien soveltaminen graafien louhintaan.

5.4 Lopuksi

Tehdyn tutkimustyön perusteella voitiin todeta, että PINGS-algoritmi voi soveltua geneerisen tiedon käsittelyyn tekemällä pieniä muutoksia, ja mahdollisesti soveltuu vielä paremmin ehdotetulla jatkokehitystyöllä. Alkuperäinen toteutus ei sellaisenaan kuitenkaan toimi mille tahansa tiedolle, joten käyttöönotto vaatii sovelluksen uudelleenkirjoituksen Muramudalige et al kuvausten perusteella ja mahdollisesti tässä työssä ehdotetuilla jatkokehitystoimenpiteillä.

LÄHTEET

[1] Muramudalige, S. R., Hung, B. W. K., Jayasumana, A. P., Ray, I. ja Klausen, J. Enhancing investigative pattern detection via inexact matching and graph databases. eng. *IEEE transactions on services computing* (2021), s. 1–1. ISSN: 1939-1374.

[2] Eriksson, P. ja Koistinen, K. *Monenlainen tapaustutkimus*. fin. Julkaisuja / Kuluttajatutkimuskeskus, 4:2005. Helsinki: Kuluttajatutkimuskeskus, 2005. ISBN: 951-698-149-6.

[3] SUN, Y., LI, G., DU, J., NING, B. ja CHEN, H. A subgraph matching algorithm based on subgraph index for knowledge graph. eng. *Frontiers of Computer Science* 16.3 (2022), s. 163606–. ISSN: 2095-2228.

[4] Lissandrini, M., Mottin, D., Palpanas, T. ja Velegrakis, Y. Multi-Example Search in Rich Information Graphs. eng. *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018, s. 809–820. ISBN: 9781538655207.

[5] Kowaluk, M. ja Lingas, A. Are unique subgraphs not easier to find? eng. *Information processing letters* 134 (2018), s. 57–61. ISSN: 0020-0190.

[6] Diestel, R. *Graph Theory*. eng. Fifth Edition. Vol. 173. Graduate Texts in Mathematics. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2017. ISBN: 978-3-662-57560-4.

[7] Saha Ray, S. *Graph Theory with Algorithms and its Applications In Applied Science and Technology*. eng. 1st ed. 2013. New Delhi: Springer India, 2013. ISBN: 97-81-322-1744-2.

[8] Bona, M. *A Walk Through Combinatorics: An Introduction To Enumeration And Graph Theory (Third Edition)*. World Scientific Publishing Company, 2011. ISBN: 978-981-4460-00-2.

[9] Computer Network Research Laboratory (CNRL), Colorado State University. *PINGS 1.0 (Procedures for INvestigative Graph Search)*. Lähdekoodi. 2020. URL: `https://github.com/cnrl-csu/pings` (viitattu 15. 12. 2021).

[10] Muramudalige, S. R., Hung, B. W. K., Jayasumana, A. P. ja Ray, I. Investigative Graph Search using Graph Databases. eng. *2019 First International Conference on Graph Computing (GC)*. IEEE, 2019, s. 60–67. ISBN: 172814129X.

[11] Benoit, A. *A Guide to Algorithm Design : Paradigms, Methods, and Complexity Analysis*. eng. Boca Raton, FL, 2013 - 2014.

[12] Knuth, D. E. Big Omicron and big Omega and big Theta. eng. *SIGACT news* 8.2 (1976), s. 18–24. ISSN: 0163-5700.

[13] McCulloh, I. *Social network analysis with applications*. eng. Hoboken, N.J: Wiley, 2013. ISBN: 1-118-64468-9.

[14] Scott, J. *Social network analysis*. eng. 4th edition. London: SAGE Publications Ltd, 2017. ISBN: 9781529716597.

[15] Lissandrini, M., Brugnara, M. ja Velegrakis, Y. Beyond Macrobenchmarks: Microbenchmark-Based Graph Database Evaluation. *Proc. VLDB Endow.* 12.4 (joulukuu 2018), s. 390–403. ISSN: 2150-8097. DOI: 10.14778/3297753.3297759. URL: https://doi-org.libproxy.tuni.fi/10.14778/3297753.3297759.

[16] Holford, M. *Let's Write a Stored Procedure in Neo4j – Part I*. Blogi. 2021. URL: https://neo4j.com/developer-blog/lets-write-a-stored-procedure-in-neo4j-part-i/ (viitattu 24. 08. 2023).

[17] *Redis*. Sivusto. 2023. URL: https://redis.com (viitattu 24. 08. 2023).

[18] *etcd*. Sivusto. 2023. URL: https://etcd.io/ (viitattu 24. 08. 2023).

[19] Knight, M. *What Is a Key-Value Database?* Blogi. 2021. URL: https://www.dataversity.net/key-value-database/ (viitattu 24. 08. 2023).

[20] Nguyen, H. *The pros and cons of different data formats: key-values vs tuples*. Blogi. 2018. URL: https://www.freecodecamp.org/news/the-pros-and-cons-of-different-data-formats-key-values-vs-tuples-f526ad3fa964/ (viitattu 24. 08. 2023).

[21] *Document Stores*. Artikkeli. 2023. URL: https://db-engines.com/en/article/Document+Stores (viitattu 24. 08. 2023).

[22] *What Is a Document Database?* Artikkeli. 2023. URL: https://aws.amazon.com/nosql/document/ (viitattu 24. 08. 2023).

[23] Barik, M. S., Sengupta, A. ja Mazumdar, C. Attack graph generation and analysis techniques. eng. *Defense science journal* 66.6 (2016), s. 559–567. ISSN: 0011-748X.

[24] Shao, C. ja Li, Y.-F. Multistage Attack–Defense Graph Game Analysis for Protection Resources Allocation Optimization Against Cyber Attacks Considering Rationality Evolution. eng. *Risk analysis* (2021). ISSN: 0272-4332.

[25] Nandi, A. K., Medal, H. R. ja Vadlamani, S. Interdicting attack graphs to protect organizations from cyber attacks: A bi-level defender–attacker model. eng. *Computers & operations research* 75 (2016), s. 118–131. ISSN: 0305-0548.

[26] Kaynar, K. ja Sivrikaya, F. Distributed Attack Graph Generation. eng. *IEEE transactions on dependable and secure computing* 13.5 (2016), s. 519–532. ISSN: 1545-5971.

[27] Nia, M. A., Bahrak, B., Kargahi, M. ja Fabian, B. Detecting new generations of threats using attribute-based attack graphs. eng. *IET information security* 13.4 (2019), s. 293–303. ISSN: 1751-8709.

[28] Computer Network Research Laboratory (CNRL), Colorado State University. *PINGS (Procedures for INestigative Graph Search) Library*. Esimerkkidata. 2020. URL:

http://www.cnrl.colostate.edu/Projects/RAD/pings.html (viitattu 15. 12. 2021).

[29] neo4j.com. *Importing CSV Data into Neo4j*. Dokumentaatio. 2021. URL: https://neo4j.com/developer/guide-import-csv/ (viitattu 15. 12. 2021).

[30] Keränen, E. *Re-implemented PINGS for a Master's thesis case*. Lähdekoodi. 2024. URL: https://github.com/erkkikeranen/masters-thesis-algorithm-study (viitattu 31. 01. 2024).

LIITE A: CYPHER NEO4J OHJELMALISTAUS TIETOAINEISTON TUONTIA VARTEN

```
1  LOAD CSV WITH HEADERS FROM 'file:///user_rad.csv' AS row
2  MERGE (p:Person {personId: row.ID, name: row.Name}) RETURN count(p);
3  LOAD CSV WITH HEADERS FROM 'file:///smaccount_rad.csv' AS row
4  MERGE (s:SmAccount {smAccountId: row.ID, type: row.Type}) RETURN count(s);
5  LOAD CSV WITH HEADERS FROM 'file:///postnodes_rad.csv' AS row
6  MERGE (p:Post {postId: row.ID, type: row.Type}) RETURN count(p);
7  LOAD CSV WITH HEADERS FROM 'file:///activity_rad.csv' AS row
8  MERGE (a:Activity {activityId: row.ID, name: row.Name, type: row.Type})
9  RETURN count(a);
10 LOAD CSV WITH HEADERS FROM 'file:///exhibits_rad.csv' AS row
11 MATCH (p:Person {personId: row.UserID})
12 MATCH (a:Activity {activityId: row.ActivityID})
13 MERGE (p)-[:EXHIBITS {timestamp: toInteger(row.Timestamp)}]->(a) RETURN *;
14 LOAD CSV WITH HEADERS FROM 'file:///has_rad.csv' AS row
15 MATCH (p:Person {personId: row.UserID})
16 MATCH (s:SmAccount {smAccountId: row.SMID})
17 MERGE (p)-[:HAS]->(s) RETURN *;
18 LOAD CSV WITH HEADERS FROM 'file:///knows_rad.csv' AS row
19 MATCH (p1:Person {personId: row.UserID1})
20 MATCH (p2:Person {personId: row.UserID2})
21 MERGE (p1)-[:KNOWS {timestamp: toInteger(row.Timestamp)}]->(p2) RETURN *;
22 LOAD CSV WITH HEADERS FROM 'file:///posts_rad.csv' AS row
23 MATCH (s:SmAccount {smAccountId: row.SMID})
24 MATCH (p:Post {postId: row.PostID})
25 MERGE (s)-[:POSTS {timestamp: toInteger(row.TimeStamp)}]->(p) RETURN *;
```

Ohjelma A.1. *Pings-esimerkkiaineiston pisteiden ja viivojen tuonti Neo4J-tietokantaan*

LIITE B: KEHITETYN KÄYTTÖLIITTYMÄN PACKAGE.JSON PROJEKTITIEDOSTO

```
1  {
2    "name": "pvis",
3    "private": true,
4    "version": "0.0.0",
5    "proxy": "http://localhost:8080",
6    "type": "module",
7    "scripts": {
8      "dev": "vite",
9      "build": "vite build",
10     "preview": "vite preview"
11   },
12   "dependencies": {
13     "@apollo/client": "^3.7.7",
14     "graphql": "^16.6.0",
15     "react": "^18.2.0",
16     "react-dom": "^18.2.0",
17     "reagraph": "^4.14.1"
18   },
19   "devDependencies": {
20     "@types/react": "^18.0.27",
21     "@types/react-dom": "^18.0.10",
22     "@vitejs/plugin-react": "^3.1.0",
23     "vite": "^4.1.0"
24   }
25 }
```

Ohjelma B.1. *Kehitetyn React-käyttöliittymän projekitiedosto package.json*

LIITE C: KEHITETYN APIN POM.XML PROJEKTITIEDOSTO

```
1  <project
2          xmlns="http://maven.apache.org/POM/4.0.0"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0␣
               http://maven.apache.org/xsd/maven-4.0.0.xsd">
5      <modelVersion>4.0.0</modelVersion>
6
7      <parent>
8          <groupId>org.springframework.boot</groupId>
9          <artifactId>spring-boot-starter-parent</artifactId>
10         <version>3.0.2</version>
11     </parent>
12
13     <groupId>org.erkkikeranen</groupId>
14     <artifactId>pings</artifactId>
15
16     <version>0.0.1-SNAPSHOT</version>
17
18     <properties>
19         <java.version>17</java.version>
20     </properties>
21
22     <dependencies>
23         <dependency>
24             <groupId>org.springframework.boot</groupId>
25             <artifactId>spring-boot-starter</artifactId>
26         </dependency>
27         <dependency>
28             <groupId>org.springframework.boot</groupId>
29             <artifactId>spring-boot-starter-web</artifactId>
```

```
30          </dependency>
31          <dependency>
32              <groupId>org.springframework.boot</groupId>
33              <artifactId>spring-boot-starter-data-jpa</artifactId>
34          </dependency>
35          <dependency>
36              <groupId>org.springframework.boot</groupId>
37              <artifactId>spring-boot-starter-graphql</artifactId>
38          </dependency>
39          <dependency>
40              <groupId>org.projectlombok</groupId>
41              <artifactId>lombok</artifactId>
42          </dependency>
43          <dependency>
44              <groupId>org.postgresql</groupId>
45              <artifactId>postgresql</artifactId>
46              <version>42.5.3</version>
47          </dependency>
48          <dependency>
49              <groupId>org.flywaydb</groupId>
50              <artifactId>flyway-core</artifactId>
51              <version>9.8.1</version>
52          </dependency>
53      </dependencies>
54
55      <build>
56          <plugins>
57              <plugin>
58                  <groupId>org.springframework.boot</groupId>
59                  <artifactId>spring-boot-maven-plugin</artifactId>
60                  <configuration>
61                      <mainClass>org.erkkikeranen.App</mainClass>
62                      <layout>JAR</layout>
63                  </configuration>
64              </plugin>
65          </plugins>
66      </build>
67
68  </project>
```

Ohjelma C.1. *Kehitetyn APIn projektitiedosto pom.xml*

LIITE D: SOVELLETUT PINGS-ALGORITMIT

SimilarityMeasure (Sovellettu PINGS individualSimilarity algoritmista)

```
1:  function SIMILARITYMEASURE(KG,treshold,Q,qflabel)
                                                    ▷ KG : Tietämysgraafi
                                    ▷ treshold : suodatettava pisteraja
                                                    ▷ Q : Kyselygraafi
                                    ▷ qflabel : kohdistussolmutyyppi
2:      maxWeight ← referenceWeight(Q)
3:      matchedGraphs ← searchSimilarGraphs(Q, KG, qflabel)
4:      for all mg ∈ matchedGraphs do
5:          score(mg) ← matchGraphWeight(N(mg), E(mg))/maxWeight
6:          if score(mg) < treshold then
7:              matchedGraphs ← matchedGraphs\{mg}
8:          end if
9:      end for
10:     return matchedGraphs
11: end function
```

Ohjelma D.1. *Similarity Measure-algoritmi*

SearchSimilarGraphs (Sovellettu PINGS searchSimilarGraphs algoritmista)

```
1: function SEARCHSIMILARGRAPHS(KG,Q,qflabel)
                                                          ▷ KG : Tietämysgraafi
                                                          ▷ Q : Kyselygraafi
                                                          ▷ qflabel : kohdistussolmutyyppi
2:     QFN ← queryFocusNodes(KG, qflabel)        ▷ Poimitaan tietämysgraafista
   kohdistusta vastaavat solmut
3:     matchedGraphs ← {}
4:     for all qfn ∈ QFN do                  ▷ Kuljetaan graafi jokaisesta kohdistussolmusta
5:         MPN ← {}                              ▷ MPN sis. löydetyt solmut rekursoidessa
6:         MQN ← matchQueryGraphNodes(Q, KG, {qfn}, MPN)
                                                          ▷ Löydetty solmujoukko MQN
7:         if |MQN| ≥ 1 then
8:             matchedGraphs ← matchedGraphs ∪ {mg}
9:         end if
10:    end for
11:    return matchedGraphs
12: end function
```

Ohjelma D.2. *Search Similar Graphs-algoritmi*

Alialgoritmi: Kyselygraafin referenssipainoarvo

```
1: function REFERENCEWEIGHT(Q)                                  ▷ Kyselygraafi Q
2:     score ← 0
3:     for all n ∈ N(Q) do                                      ▷ Solmu n ja solmut N
4:         score ← score + 1.0
5:         for all np ∈ NP(n) do                  ▷ Solmun attribuutti np ja attribuutit NP
                                  ▷ Konfiguroitava vrt. PINGS-algoritmin red flag multiple
6:             score ← score + weight(np)
7:         end for
8:     end for
9:     score ← score + |E(Q)|                                   ▷ Viivat E
10:    return score
11: end function
```

Ohjelma D.3. *Reference Weight pisteytysalgoritmi*

Alialgoritmi: Vertailtavan graafin pisteytys

```
1: function MATCHGRAPHWEIGHT(MN,ME)  ▷ Löydetyn graafin solmut MN ja viivat ME
2:     score ← 0
3:     score ← +|MN|                          ▷ Piste jokaisesta löydetystä solmusta
4:     for all mn ∈ MN do                                          ▷ Solmu mn
5:         score ← score + 1.0
6:         for all mnp ∈ MNP(mn) do   ▷ Solmun attribuutti mnp ja attribuutit MNP
                                  ▷ Konfiguroitava vrt. PINGS-algoritmin red flag multiple
7:             score ← score + weight(mnp)
8:         end for
9:     end for
10:    score ← score + |ME|
11:    return score
12: end function
```

Ohjelma D.4. *Match Graph Weight pisteytysalgoritmi*

Neighborhood Similarity (Sovellettu PINGS algoritmista)

```
1: function NEIGHBORHOODSIMILARITY(KG,treshold,Q,qflabel)
                                                          ▷ KG : Tietämysgraafi
                                                ▷ treshold : suodatettava pisteraja
                                                          ▷ Q : Kyselygraafi
                                                ▷ qflabel : kohdistussolmutyyppi
2:     maxWeight ← referenceWeight(Q)
3:     SGS ← searchSimilarGraphs(Q, KG, qflabel)
4:     NNS ← searchNeighborNodes(similarGraphs, KG)
                                        ▷ NNS on kokoelma joukkoja joissa naapurustoja
5:     matchedGraphs ← searchNeighborMatchGraphs(KG, Q, qflabel, NNS, SGS)
6:     for all mg ∈ matchedGraphs do
7:         score(mg) ← matchGraphWeight(N(mg), E(mg))/maxWeight
8:         if score(mg) < treshold then
9:             matchedGraphs ← matchedGraphs\{mg}
10:        end if
11:    end for
12:    return matchedGraphs
13: end function
```

Ohjelma D.5. *Neighborhood Similarity-algoritmi*

searchNeighborMatchGraphs (Sovellettu PINGS algoritmista)

```
1: function SEARCHNEIGHBORMATCHGRAPHS(KG,Q,qflabel,NeighborNodeSets,SG)
                                                          ▷ KG : Tietämysgraafi
                                                          ▷ Q : Kyselygraafi
                                                          ▷ qflabel : kohdistussolmutyyppi
                                                          ▷ NeighborNodeSets : vieruspisteet
                                                          ▷ SG : similarGraphs
2:     matchedGraphs ← {}
3:     for all neighborhood ∈ NeighborhoodNodeSets do
4:         result ← neighborhood
5:         for all neighbor ∈ neighborhood do
6:             intermediate ← relatedSG(neighbor, SG)
                                          ▷ relatedSG valitsee naapuriin liittyvän aligraafin
7:             sgs ← searchSimilarGraphs(KG, Q, qflabel, {neighbor})
8:             intermediate ← intermediate ∪ sgs
9:             result ← result ∪ intermediate
10:        end for
11:        matchedGraphs ← result
12:    end for
13:    return matchedGraphs
14: end function
```

Ohjelma D.6. *Search Neighbor Match Graph-algoritmi*

SearchNeighborNodes (Sovellettu PINGS algoritmista)

```
1: function SEARCHNEIGHBORNODES(similarGraphs,KG)
                                                    ▷ KG : Tietämysgraafi
                          ▷ similarGraphs : Aiemmin löydetyt samankaltaiset aligraafit
2:     neighbors ← {}
3:     for all nodeset ∈ similarGraphs do
4:         setneighbors ← {}
5:         for all node ∈ nodeset do
6:             for all edges ∈ E(node) do
7:                 for all edge ∈ edges do
8:                     if targetNode(edge) ∉ nodeset then
9:                         setneighbors ← ∪{targetNode(edge)}
10:                    end if
11:                    if sourceNode(edge) ∉ nodeset then
12:                        setneighbors ← ∪{sourceNode(edge)}
13:                    end if
14:                end for
15:            end for
16:        end for
17:        neighbors ← neighbors + setneighbors
18:    end for
19:    return neighbors
20: end function
```

Ohjelma D.7. *Search Neighbor Nodes-algoritmi*

MatchQueryGraphNodes

```
1: function MATCHQUERYGRAPHNODES(KG,Q,NI,RG)                    ▷ alias funktio
2:     return mqgn(KG, Q, NI, RG)
3: end function

4: function MQGN(KG,Q,NI,RG)
       ▷ Tietämysgraafi KG, kyselygraafi Q, seuraavat käsiteltävät solmut NI, tulos RG
5:     if |NI| = 0 then
6:         return RG                                  ▷ Valmis; ei tutkittavia solmuja jäljellä
7:     else
8:         rg ← {}                                                  ▷ rg iteraation tulos
9:         for all qn ∈ Q do                                        ▷ Kuljetaan kyselygraafi
10:            for all ni ∈ NI do          ▷ Verrataan kyselygraafin solmuja käsiteltäviin
11:                if matchNode(qn, ni) then
12:                    QGE ← E(qn)                ▷ Kyselygraafin solmun qn viivat QGE
13:                    for all qge ∈ QGE do
14:                        NIE ← E(ni)            ▷ Tietämysgraafin solmun ni viivat NIE
15:                        for all nie ∈ NIE do
16:                            ntn ← targetNode(nie)
17:                            nsn ← sourceNode(nie)
18:                            qtn ← targetNode(qge)
19:                            qsn ← sourceNode(qge)
20:                            if matchEdge(qge, nie, ntn, nsn, qtn, qsn) then
21:                                rg ← addEdgeNode(ntn, qtn, rg, NI, RES)
22:                                rg ← addEdgeNode(nsn, qsn, rg, NI, RES)
23:                            end if
24:                        end for
25:                    end for
26:                end if
27:            end for
28:        end for
29:        return mqgn(KG, Q, rg, RG ∪ rg)                          ▷ Rekursio
30:    end if
31: end function
```

Ohjelma D.8. *Match Query Graph Nodes-algoritmi*

Alialgoritmi: matchNode

```
1: function MATCHNODE(source,target)
                                  ▷ source ja target, keskenään vertailtavat solmut
2:     if label(source) ≠ label(target) then
3:         return false
4:     else if |properties(source)| > |properties(target)| then
5:         return false
6:     else
7:         sourceProps ← properties(source)
8:         targetProps ← properties(target)
9:         for all sourceProperty ∈ sourceProps do
10:            if value(sourceProperty) ≠ value(targetProps(key(sourceProperty)))
   then
11:                return false
12:            end if
13:        end for
14:    end if
15:    return true
16: end function
```

Ohjelma D.9. *Match Query Graph Nodes matchNode-algoritmi*

Alialgoritmi: matchEdge

```
1: function MATCHEDGE(qge,nie, ntn, nsn, qtn, qsn)
                                                    ▷ qge kyselygraafin viiva
                                                    ▷ nie tietämysgraafin viiva
                                    ▷ nsn ja ntn tietämysgraafin viivan alku- ja loppupiste
                                    ▷ qsn ja qtn kyselygraafin viivan alku- ja loppupiste
2:     if label(qge) ≠ label(nie) then
3:         return false
4:     end if
5:     if notmatchNode(qsn, nsn) then
6:         return false
7:     end if
8:     if notmatchNode(qtn, ntn) then
9:         return false
10:    end if
11:    return true
12: end function
```

Ohjelma D.10. *Match Query Graph Nodes matchEdge-algoritmi*

Alialgoritmi: addEdgeNode

```
1: function ADDEDGENODE(nodeK,nodeQ,iterationResult,NI,RES)
       ▷ nodeK tietämysgraafin solmu, nodeQ kyselygraafin solmu, iteraatiomuuttujia
2:     if nodeK ∉ iterationResult then
3:         if nodeK ∉ NI then
4:             if nodeK ∉ RES then
5:                 resultNode ← nodeK
6:                 properties(resultNode) ← mergeProperties(nodeK, nodeQ)
7:                 iterationResult ← iterationResult ∪ {nodeK}
8:                 return iterationResult
9:             end if
10:        end if
11:    end if
12:    return iterationResult
13: end function
```

Ohjelma D.11. *Match Query Graph Nodes addEdgeNode-algoritmi*