The semiring-based provenance framework for graph databases

Yann Ramusat

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'École normale supérieure

The Semiring-Based Provenance Framework for Graph Databases

Soutenue par
Yann RAMUSAT
Le 28 avril 2022

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité
Informatique

Composition du jury :

Angela BONIFATI Université Claude Bernard Lyon 1	*Présidente du jury*
Antoine AMARILLI Institut polytechnique de Paris	*Examinateur*
Daniel DEUTCH Tel Aviv University	*Rapporteur*
Silviu MANIU Université Paris-Saclay	*Directeur de thèse*
Pierre SENELLART École normale supérieure	*Directeur de thèse*
Val TANNEN University of Pennsylvania	*Rapporteur*
Michaël THOMAZO Inria	*Examinateur*

ENS | PSL

À Mamie-Jo.

Résumé

L'augmentation du volume de données collectées par des capteurs et générées par des interactions humaines a mené à l'utilisation des bases de données orientées graphes en tant que modèle de représentation efficace pour les données complexes. Les techniques permettant de tracer les calculs qui ont été appliqués aux données au sein d'une base de données relationnelle classique sont sur le devant de la scène, notamment grâce à leur utilité pour faire respecter les régulations sur les données privées telles que le RGPD en Union Européenne. Notre travail de recherche croise ces deux problématiques en s'intéressant à un modèle de provenance à base de semi-anneaux pour les requêtes navigationnelles. Nous commençons par présenter une étude approfondie de la théorie des semi-anneaux et de leurs applications au sein des sciences informatiques en se concentrant sur les résultats ayant un intérêt direct pour notre travail de recherche. La richesse de la littérature sur le domaine nous a notamment permis d'obtenir une borne inférieure sur la complexité de notre modèle. Dans une seconde partie, nous étudions le modèle en lui-même et introduisons un ensemble cohérent d'algorithmes permettant d'effectuer des calculs de provenance et adaptés aux propriétés des semi-anneaux utilisés. Nous introduisons notablement une nouvelle méthode basée sur la théorie des treillis permettant de calculer la provenance pour des requêtes complexes. Nous proposons une implémentation open-source de ces algorithmes et faisons une étude expérimentale sur de larges réseaux de transport issus de la vie réelle pour attester de l'efficacité pratique de notre approche. On s'intéresse finalement au positionnement de ce cadre de travail par rapport à d'autres modèles de provenance à base de semi-anneaux. Nous nous intéressons à Datalog en particulier. Nous démontrons que les méthodes que nous avons développées pour les bases de données orientées graphes peuvent se généraliser sur des requêtes Datalog. Nous montrons de plus qu'elles peuvent être vues comme des généralisations de la méthode semi-naïve. En se basant sur ce fait-là, nous étendons les capacités de Soufflé, un évaluateur Datalog appartenant à l'état de l'art, afin d'effectuer des calculs de provenance pour des requêtes Datalog. Les études expérimentales basées sur cette implémentation open-source confirment que cette approche reste compétitive avec les solutions spécifiques pour les graphes, mais tout en étant plus générale. Nous terminons par une discussion sur les améliorations possibles du modèle et énonçons les questions ouvertes qui ont été soulevées au cours de ce travail.

Abstract

The growing amount of data collected by sensors or generated by human interaction has led to an increasing use of graph databases, an efficient model for representing intricate data. Techniques to keep track of the history of computations applied to the data inside classical relational database systems are also topical because of their application to enforce Data Protection Regulations (e.g., GDPR). Our research work mixes the two by considering a semiring-based provenance model for navigational queries over graph databases. We first present a comprehensive survey on semiring theory and their applications in different fields of computer sciences, geared towards their relevance for our context. From the richness of the literature, we notably obtain a lower bound for the complexity of the full provenance computation in our setting. In a second part, we focus on the model itself by introducing a toolkit of provenance-aware algorithms, each targeting specific properties of the semiring of use. We notably introduce a new method based on lattice theory permitting an efficient provenance computation for complex graph queries. We propose an open-source implementation of the above-mentioned algorithms, and we conduct an experimental study over real transportation networks of large size, witnessing the practical efficiency of our approach in practical scenarios. We finally consider how this framework is positioned compared to other provenance models such as the semiring-based Datalog provenance model. We make explicit how the methods we applied for graph databases can be extended to Datalog queries, and we show how they can be seen as an extension of the semi-naïve evaluation strategy. To leverage this fact, we extend the capabilities of Soufflé, a state-of-the-art Datalog solver, to design an efficient provenance-aware Datalog evaluator. Experimental results based on our open-source implementation entail the fact this approach stays competitive with dedicated graph solutions, despite being more general. In a final round, we discuss on some research ideas for improving the model, and state open questions raised by our work.

Remerciements

Je tiens à remercier en premier lieu Pierre et Silviu, mes deux directeurs de thèse. Du début du stage de master et jusqu'à la fin de cette thèse, j'ai eu la chance inouïe de vous avoir eu comme guides. Avec le recul, je mesure tout le chemin que j'ai parcouru grâce à la pertinence de vos conseils. Je vous remercie pour votre implication, votre attention ainsi que pour votre exceptionnelle disponibilité. Vos compétences scientifiques et techniques sont impressionnantes et inspirantes, tout comme le sont votre volonté et votre aisance à les transmettre.

I would like to sincerely thank my reviewers Daniel Deutch and Val Tannen to have accepted to review this work. Je remercie aussi Angela, Antoine et Michaël de m'avoir fait l'honneur d'accepter de faire partie du jury et d'avoir porté un intérêt à ce travail.

Je remercie aussi toutes les personnes que j'ai croisées à l'ENS, et avec qui j'ai eu l'occasion de partager de bons moments. Je pense tout d'abord aux personnes avec qui j'ai eu l'occasion de partager mon bureau durant ces années: Debabrota, Louis J., Nathan, Julien et Shrey. Vous m'avez rendu les heures de travail plus agréables et vivantes grâce à des discussions enrichissantes. Je pense aussi aux autres personnes croisées dans les Hauts du DI (équipes Valda, Talgo et Data) et avec qui il fut un plaisir de se retrouver lors des pauses repas: Mathieu, Louis T., Chien-Chung, Pierre A., Garance, Guillaume, Luc, Camille et Michaël. Je remercie aussi les personnes avec qui j'ai eu la chance d'enseigner à l'ENS ou à Paris Diderot: Cristina, Michaël et Leonid. Merci aussi à l'équipe du secrétariat pour les bons moments passés pendant la période des oraux: Noureddine, Fabien, Louise, Arnaud, Lionel, Jules et Sébastien. Et bien sûr sans oublier de remercier l'équipe administrative et technique du DI pour leur aide et leur gentillesse: Isabelle, Sophie, Lise-Marie, Linda, Ludovic et Jacques.

J'ai aussi une pensée pour mes collègues, notamment Samy, que j'ai rencontrés au cours de mes études en école d'ingénieur, à l'ENS, ou au MPRI, et avec qui j'ai passé d'extraordinaires moments de travail.

Je tiens à remercier tout particulièrement mes parents et ma défunte grand-mère pour leur dévouement, leur aide et leur croyance indéfectible en moi. Pour tout ce que vous m'avez apporté et qui m'a ultimement permis de faire cette thèse.

Mes derniers mots vont finalement pour toi, Marie-Pierre. Je tiens à te remercier d'avoir été durant ces années la personne avec qui j'ai vécu et partagé l'aventure commune du doctorat qui fut la nôtre.

Contents

Introduction générale

Commençons par étudier les mots qui constituent le titre de la thèse. Le premier mot qui saute aux yeux est le mot *databases* "bases de données". En effet, notre objectif principal est de traiter de l'information stockée sous forme de documents structurés. La théorie des bases de données a initialement connu son essor grâce aux bases de données relationnelles introduites dans les années 70 par Codd [1970]. La représentation logique des données se fait à l'aide de relations n-aires et l'information peut en être extraite à l'aide de langages de données universels tels que l'algèbre relationnelle (vision impérative) ou le calcul relationnel (vision déclarative). Ces deux langages ont été démontrés comme étant équivalents par ce qui est couramment appelé le théorème de Codd. Un SGBDR (Système de Gestion de Base de Données Relationnelle), réalisation logicielle concrète de ce type de bases de données, est communément prévu pour supporter des requêtes SQL, un langage de requête standardisé inspiré des deux langages susmentionnés.

Ces dernières années ont vu apparaître une augmentation exponentielle de données disponibles, capturées par des capteurs comme pour les applications IOT (Internet Of Things), ou générées par des humains comme sur les réseaux sociaux ou le Web 2.0. Ceci a favorisé l'émergence de nouveaux paradigmes pour pouvoir gérer ces quantités très conséquentes de données, sous la forme de nouvelles bases de données appelées NoSQL (Not Only SQL). Elles permettent de pallier les limitations intrinsèques des bases de données relationnelles (peu enclines à être gérées au sein d'un système distribué). Ces bases de données non relationnelles sont catégorisées en quatre grands types, suivant l'application que l'on souhaite en faire : clé-valeur, graphes, colonnes et documents.

Parmi ces quatres grands types, on s'est concentré dans cette thèse sur les bases de données *graphes*. Elles sont communément utilisées pour leur capacité à exprimer élégamment les relations entre les données elles-mêmes : les arêtes du graphe modélisent les relations entre les nœuds de données du graphe et sont donc des informations stockées dans la base de données elle-même. A l'opposé, les bases de données relationnelles nécessitent d'être interrogées en utilisant des jointures coûteuses pour récupérer ce type d'information. Ainsi, des informations comme les relations entre les utilisateurs dans l'analyse des réseaux sociaux ou l'historique des achats des clients (relation entre un utilisateur et un produit) dans le cadre d'applications commerciales conviennent parfaitement à ce type de modèle. Ces bases de données supportent de façon native les requêtes récursives.

Issu des bases de données relationnelles, le concept de *provenance d'une requête* qui a été introduit au début du 21ème siècle [Cheney et al. 2009] ajoute de l'information au résultat d'une requête. Comme son nom l'indique, la provenance capture de l'information se référant à l'origine des données calculées et donne des indices sur les opérations conduites pendant l'évaluation de la requête. Ce concept a connu un certain essor étant

donné sa capacité à exprimer une grande variété de tâches de gestion de données (on peut se référer à [Senellart 2019] pour plus de détails): bases de données probabilistes et incomplètes, gestion des vues ainsi que des explications plus ou moins précises des résultats de requêtes (connues sous les termes anglophones de How, Why et Where provenance).

Green et al. ont introduit la notion de *semi-anneaux de provenance* [2007] en remarquant que les calculs impliqués dans les tâches de gestion de données mentionnées ci-dessus sont fortement similaires. Le modèle fortement générique ainsi proposé utilise la structure de *semi-anneau* comme modèle algébrique pour exprimer les calculs de provenance sur l'algèbre relationnelle et Datalog.

Nous venons de présenter les concepts centraux de la thèse au lecteur, et nous prenons maintenant le temps de discuter des *principes* qui nous ont *guidés* durant ce doctorat. Les bases de données graphes faisant partie des bases de données NoSQL – usuellement utilisées pour des applications de big-data – cela nous force à considérer des solutions pouvant passer à l'échelle, afin de pouvoir les mettre en œuvre dans des cas pratiques. Chaque partie logique de la thèse est accompagnée d'une implémentation librement distribuée, venant avec son lot d'expérimentations conduites sur des ensembles de données minutieusement choisis afin d'attester de l'efficacité pratique des solutions que nous proposons.

Tandis que de profondes connexions ont été établies entre la provenance à base de semi-anneaux et d'autres domaines de recherche, comme par exemple la programmation par contraintes (se référer à la conclusion de [Green et al. 2007]), le modèle que nous avons considéré durant ce doctorat pour capturer les informations de provenance dans les bases de données graphes est lui-même fortement relié à encore plus d'autres domaines. Nous avons effectué en conséquence un travail bibliographique et théorique pour bénéficier de la richesse des sciences informatiques.

Revenons-en maintenant à la genèse du doctorat. Avant qu'il ne commence, j'ai effectué mon stage de master dans l'équipe Valda et à cette occasion, Pierre, Silviu et moi avons travaillé sur les *bases* de la provenance à base de semi-anneaux pour les bases de données graphes. Ce modèle, inspiré de la provenance à base de semi-anneaux pour les bases de données relationnelles et les requêtes Datalog [Green et al. 2007], permet d'exprimer une grande variété de requêtes comme les k-meilleurs chemins, des restrictions d'accès ainsi que le dénombrement de chemins entre deux lieux. En fonction des propriétés du semi-anneau utilisé pour représenter la provenance – ceci permettant plus ou moins d'expressivité – nous avons généralisé trois algorithmes pour graphes déjà existants de généralité et complexité croissantes. Des résultats expérimentaux préliminaires ont attesté que cette approche restait praticable pour de très larges réseaux de transports, pour lesquels, malgré une littérature riche autour du routage efficace, aucune solution donnant des méta-informations sur le chemin optimal entre deux lieux n'a été développée. Les travaux effectués au cours de ce stage, et dont nous venons de faire un bref récapitulatif, ont été publiés dans [Ramusat et al. 2018].

Motivations et organisation du manuscrit

Ce manuscrit est une opportunité de présenter des contributions ne trouvant pas facilement leur place dans un article de conférence. Nous avons poursuit l'objectif de rédiger et présenter ce document comme le serait une monographie centrée sur le modèle de provenance à base de semi-anneaux pour les bases de données graphes, étant donné que l'on s'intéresse aussi au positionnement de ce modèle vis-à-vis d'une vision algébrique des sciences du numérique.

Partie I : Préliminaires

Tandis que la partie I décrit le contexte scientifique autour duquel le document s'articule, nous saisissons l'occasion de faire une étude détaillée sur la théorie des semi-anneaux et leurs applications dans les sciences de l'informatique. Nous en appelions déjà dans [Ramusat et al. 2021a] à la nécessité de faire un point sur la diversité des applications des semi-anneaux, sous peine de perdre l'opportunité de bénéficier de ce qui a déjà été fait dans des domaines connexes à l'aide de structures algébriques similaires, mais dénommées différemment.

Il y a aussi des raisons pédagogiques derrière ce travail de synthèse. Nous avons été confronté pendant ces années à une quantité conséquente de notions et d'applications en rapport avec les semi-anneaux. Certaines étaient trouvables dans des manuels (par exemple dans [Cormen et al. 1989] au chapitre 26.4), mais n'ayant jamais été mises en avant au cours des études supérieures. Notre objectif est de proposer aux futurs chercheurs du domaine, un ensemble cohérent et complet de concepts, propriétés (sous les nombreux noms qui leur sont associés dans la littérature) et de références se rapportant aux semi-anneaux. C'est-à-dire en leur donnant un contexte théorique solide afin de leur permettre de s'attaquer aux futurs problèmes de recherche. C'est un moyen de rendre compte des résultats obtenus après les nombreuses heures que nous avons passées à étudier la littérature afin de trouver des traces d'utilisation des semi-anneaux dans des contextes où ils ne sont même pas nommés (par exemple dans [Knuth 1977]).

Revenons-en à l'organisation de la partie, le chapitre 1 fera un rappel des notations de base sur les graphes et sur le langage de requêtes Datalog. Nous dédions le chapitre 2 à l'étude des semi-anneaux et plus particulièrement la section 2.2 à l'étude des calculs de points fixes, des sommes infinies et des opérateurs continus au sein d'un semi-anneau. Finalement, nous abordons dans le chapitre 3 les travaux précurseurs de [Green et al. 2007] introduisant la provenance à base de semi-anneaux pour le modèle relationnel et Datalog.

Nous soulignons le fait que cette partie, et spécifiquement le chapitre 2, sont des contributions à part entière de cette thèse.

Partie II : Modèle de provenance pour les bases de données orientées graphes

L'essentiel du contenu décrit dans cette partie a été publié à la conférence EDBT'21 [Ramusat et al. 2021b]. L'implémentation open-source des algorithmes décrits est une autre contribution de cette thèse [Maniu et al. 2020].

La seconde partie de ce manuscrit est dédiée aux contributions directement reliées au modèle de provenance pour les bases de données orientées graphes. Nous commençons par définir le modèle dans le chapitre 4 et nous faisons au chapitre 5 un récapitulatif des algorithmes que nous avons introduits dans [Ramusat et al. 2018]. Nous discutons aussi de résultats supplémentaires qui n'ont pas fait l'objet de publication. Nous présentons ensuite les contributions de la thèse à partir du chapitre 6, qui introduit la nouvelle approche que nous avons développée dans [Ramusat et al. 2021b] pour combler les limitations du précédent ensemble d'algorithmes que nous avons décrit. Nous avons proposé une implémentation open-source de tous ces algorithmes [Maniu et al. 2020], et nous avons conduit une étude expérimentale afin d'évaluer leur efficacité sur des réseaux de transports issus du monde réel. Nous avons notablement étudié l'impact de propriétés de graphes telles que la *highway dimension* [Abraham et al. 2016] et la largeur d'arbre sur l'efficacité de notre approche. L'ensemble des résultats de cette étude peut être trouvé dans le chapitre 7. Une taxonomie complète des semi-anneaux et des algorithmes qui leur sont adaptés peut être trouvée dans le chapitre 8, nous y discutons aussi des limitations du modèle, principalement énoncées sous la forme d'une borne inférieure sur le calcul de provenance pour *toutes paires*.

Partie III : Provenance pour Datalog appliquée aux requêtes sur des graphes

L'ensemble du travail présenté dans cette partie n'est pas encore publié mais peut être consulté sur ArXiv [Ramusat et al. 2021a]. Nous avons aussi proposé une implémentation open-source des méthodes décrites qui est disponible sur GitHub [Ramusat 2021].

Nous nous intéressons maintenant aux possibilités de généraliser nos précédentes solutions initialement destinées aux bases de données orientées graphes. Nous voulons aussi dépasser les limitations théoriques, les bornes inférieures, étudiées à la partie précédente. Une façon naturelle d'étendre nos résultats est de passer par la provenance pour les programmes Datalog. En effet, les requêtes navigationnelles peuvent être encodées simplement dans ce modèle. Nous commençons par le chapitre 9 qui établit une traduction entre un modèle de programmation dynamique sur les hypergraphes [Huang 2008] et la provenance pour Datalog. Ceci nous permet d'obtenir une généralisation d'un algorithme introduit par Knuth [1977], directement applicable dans notre modèle de provenance. Le chapitre 10 décrit cet algorithme et montre comment les idées à la base de la méthode semi-naïve peuvent être réutilisées pour accélérer cette méthode. Nous discuterons aussi

du fait que l'approche basée sur les treillis distributifs peut aussi s'appliquer à ce cas plus général. Pour poursuivre avec une étude expérimentale, nous adaptons Soufflé [Jordan et al. 2016], un évaluateur Datalog à l'état de l'art, pour qu'il puisse retourner la provenance du résultat. Ceci a conduit à une implémentation open-source [Ramusat 2021]. Nos résultats expérimentaux sont présentés dans le chapitre 11. Le chapitre 12 aborde la traduction inverse de celle présentée en début de partie et propose des idées d'ouverture.

Autres contributions

Nous citons maintenant diverses autres contributions effectuées au cours de cette thèse. J'ai eu l'opportunité de participer à un atelier des doctorants dans une des conférences les plus renommées du domaine, VLDB [Ramusat 2019], pour y présenter les résultats préliminaires du doctorat. J'ai donné une présentation [Ramusat et al. 2020] à la conférence française autour des bases de données, BDA "Gestion de Données – Principes, Technologies et Applications" qui favorise les discussions informelles entre les chercheurs du domaine. Et c'est finalement avec grand plaisir que j'ai donné durant deux ans des travaux dirigés aux étudiants de première année à l'ENS. Les cours portaient sur la théorie des bases de données ainsi que sur les langages formels, la calculabilité et la complexité.

General Introduction

Let us take a look at the words that constitute the title of this PhD. You may notice at first the word *databases*, as our main concern is to process information stored in some kind of structured documents. Database theory has been primarily focused on the relational data model introduced in the seventies by Codd [1970]. The data is logically represented as tuples inside n-ary relations and can be queried using a universal data sublanguage, such as the relational algebra (imperative) or the relational calculus (declarative), two logically equivalent languages due to Codd's theorem. RDBMSs (Relational DataBase Management Systems) are queried using SQL (Structured Query Language) wich is based upon these two languages.

Undoubtedly, while being the most prominent model for storing data and retrieving information, the 21st century has come with a dramatically increasing amount of available data, captured by sensors (e.g., IoT applications), or human-generated (e.g., social networks, Web 2.0), which required to introduce new database systems to scope with a very-large amount of data. Such new models are called NoSQL (Not Only SQL) databases. These non-relational databases are broadly classified into four categories, depending on their target applications: key-value, graph, column, and document.

Among them, we focus in this PhD on *graph*-based models. Commonly used when the relationships between the data is of primary interest: edges with the semantics of relations between the data nodes are pieces of information contained in the database system itself, opposite to relational databases where those links have to be retrieved (using costly JOIN operations) by a query. Thus, strongly-linked information such as relationships between users for social network analysis or the consumption of customers (relationships between users and products) in retail industry applications perfectly fit into this model. These databases provide native support for recursive languages, and offer the possibility to perform efficient traversal queries.

Stemming from the field of relational databases, the *provenance of a query* (also called the lineage), a concept introduced at the beginning of the 21st century [Cheney et al. 2009], consists in additional information provided alongside a query result. As the name suggests, the provenance collects information about the origin of the data and provides clues on the operations processing it at evaluation time. It has become a trending field as its different formalisms encompass many data management tasks (refer to [Senellart 2019] for more details): probabilistic and incomplete databases, view management, and coarse to fine-grained explanations of query results (How, Why, and Where provenance).

Green et al. introduced the notion of *provenance semirings* [2007] by taking into consideration the data management tasks above share strong similarities in the way computations are performed over the provenance annotations. This work has led to a comprehensive framework using *semirings* as an algebraic basis to model the computa-

tions for relational query languages and Datalog programs.

We have so far introduced the reader to the many concepts this thesis revolves around, and we now take a break to discuss the *guiding principles* of this PhD. Graph databases being part of the many types of NoSQL databases – being commonly used in traditional big-data applications – it forces us to consider scaling-up solutions to practical scenarios. Each part of the research work comes along with its own freely-available implementation we made, together with many experimentations conducted over carefully choosen datasets, to witness the practical efficiency of our methods.

Diametrically opposite, whereas deep connections with other research areas have been exhibited between the provenance semiring framework and semiring-based constraint-solving programming for instance (acknowledged in the conclusion of [Green et al. 2007]), the model we have considered during this PhD to capture provenance information in graph databases turns out to also be strongly connected to many other representation frameworks in the computer science literature. We have pursued a deep bibliographical and theoretical work to benefit from the richness of computer sciences.

To contextualize this research, prior to this PhD, I did my graduation internship within the Valda team where, Pierre, Silviu, and I, have set the *foundations* for the semiring-based provenance over graph databases. The model, akin to the semiring provenance model for relational databases and Datalog queries [Green et al. 2007], expresses a wide range of queries such as k-optimal routes, access restrictions, and number of paths between two given locations. Depending on the restrictions over the underlying semiring of use – permitting more or less expressivity – we generalized three existing graph algorithms of increased generality and complexity. Preliminary experimental results are evidence of the practical efficiency of this approach scaling-up to large-scale transportation networks, for which, despite rich literature for efficient routing, no solution providing meta-information about the optimal path between two locations were developed. The work done during this internship, summarised above, has been published in [Ramusat et al. 2018].

Rationale and manuscript organization

This manuscript is an opportunity to present developments that are not easily integrated into publications. We pursue the objective of writing this document as if it were a monograph around the semiring-based provenance framework for graph databases, as we look more generally at how the framework we consider is positioned in an algebraic vision of computer sciences.

Part I: Preliminaries

This part sets the context of this document, but we also take the opportunity to survey in great detail the widespread field of the theory and applications of semirings in computer sciences. We have been already calling in [Ramusat et al. 2021a] to focus on the presentation of the wealth of the scientific knowledge around this ubiquitous mathe-

matical structure. This would avoid loosing the benefit of investigations pursued under similar algebraic frameworks by other research communities.

There are also pedagogical reasons to provide such an extensive review, as we have faced throughout these years a tremendous amount of semiring notions and many applications, some being present in standard textbooks (e.g., [Cormen et al. 1989, Chapter 26.4]), but never emphasized during graduate studies. We hope to provide future researchers interested in this topic with a comprehensive set of concepts, properties (under their various names and definitions occuring in research works and textbooks) and references related to semirings. In other words, the aim is to give them a sound theoretical background so they can address future challenges over the field. We want to leave for posterity a valuable account of the many hours we spent finding evidence of the usage of semirings in diverse contexts, sometimes not being properly acknowledged by their authors (e.g., [Knuth 1977]).

To come back to the topic, this chapter will also be a reminder in Chapter 1 of the basic notation for graphs and the Datalog query language. Any researcher curious about semirings could also find valuable insights into their uses accross the literature in Chapter 2. We dedicate Section 2.2 to the study of the small but crucial differences when it comes to capture the semantics of fixed-point computations, infinite sums or continuous operators inside a semiring. Finally, we introduce in Chapter 3 the seminal work of [Green et al. 2007] introducing the provenance framework for the relational model and for Datalog programs.

We stress the fact this whole part, and especially Chapter 2, are contributions per se of this thesis.

Part II: The graph database provenance model

The bulk of the work content outlined in this part has been published at EDBT'21 [Ramusat et al. 2021b]. Another contribution is the open-source implementation of all the described algorithms which can be found at [Maniu et al. 2020].

The second part of the manuscript focuses on our contributions purely related to the graph database provenance model. We start by defining the model in Chapter 4 and we recapitulate the algorithms we introduced in [Ramusat et al. 2018] in Chapter 5. We also provide some additional results and discussions that have not been included in any paper. We then start describing the contributions of this thesis in Chapter 6, which introduces the novel algorithm we proposed in [Ramusat et al. 2021b] to address the limitations of the set of algorithms described later. We have made an open-source implementation of all the above-mentioned algorithms [Maniu et al. 2020], and have conducted a comprehensive set of experiments comparing their efficiency over real-world transportation networks. We have notably studied whether the network properties such as *highway dimension* [Abraham et al. 2016] and *treewidth* have an impact on their effectiveness. All the results of this study can be found in Chapter 7. A comprehensive taxonomy of semirings and corresponding algorithms, establishing which practical approaches are

needed in different cases can be found in Chapter 8, and we discuss the limitations of the model, notably providing a lower bound for *full provenance* computations.

Part III: Datalog Provenance for Graph Queries

All of the work outlined in this part is still undergoing the peer review process but can be consulted at [Ramusat et al. 2021a]. We also made a contribution as an open-source implementation of the method we describe which is available at [Ramusat 2021].

We now focus on how our solutions dedicated to graph databases can be further generalized. We also want to overcome the limitations of the lower bound we previously established. A natural way of extending our results is to consider provenance for Datalog queries, because navigational queries can be captured in this model. We start this part of the manuscript with Chapter 9 establishing a translation between a dynamic programming framework over hypergraphs [Huang 2008] and Datalog provenance. We obtain as result a generalization of an algorithm introduced by Knuth [1977], direcly applicable to Datalog provenance computations. Chapter 10 describes this algorithm and shows how the same ideas behind the semi-naïve evaluation strategy can be applied to speed up this method. We also discuss how the ideas underlying the approach based on distributive lattices we designed for graph databases also works in this context. To pursue with an experimental study, we adapted Soufflé [Jordan et al. 2016], a state-of-the-art Datalog solver, to make it able to process provenance-aware queries. This led to an open-source implementation [Ramusat 2021]. Our experimental results are showcased in Chapter 11. Chapter 12 provides the reverse translation of the previously presented one, and discusses on some open research ideas.

Other Contributions

We now outline the other contributions of this PhD. I had the opportunity to participate in a PhD Workshop of one of the best conference of the field, namely VLDB, to present the preliminary research results [Ramusat 2019]. I also gave a talk at a French conference on databases called BDA "Gestion de Données – Principes, Technologies et Applications" which favors informal discussions between researchers in the field [Ramusat et al. 2020]. Finally, I had the great honor to teach for two years the tutorials of the database theory course, and of the formal languages, calculability, and complexity course to the first year ENS students.

Personal Bibliography

Maniu, Silviu, Yann Ramusat, and Pierre Senellart (2020). *Graph Provenance*. URL: `https://bitbucket.org/smaniu/graph-provenance`.

Ramusat, Yann (2019). "Provenance-Based Routing in Probabilistic Graph Databases". In: *PhD@VLDB*. Vol. 2399.

Ramusat, Yann (2021). *Semiring-Based Provenance within Soufflé*. Version 0.0.1. URL: `https://github.com/yannramusat/souffle-prov`.

Ramusat, Yann, Silviu Maniu, and Pierre Senellart (2018). "Semiring Provenance over Graph Databases". In: *TaPP*.

Ramusat, Yann, Silviu Maniu, and Pierre Senellart (2020). "Algorithmes à base de provenance pour des requêtes enrichies sur les bases de données graphes". In: *BDA*.

Ramusat, Yann, Silviu Maniu, and Pierre Senellart (2021a). *A Practical Dynamic Programming Approach to Datalog Provenance Computation*. arXiv: `2112.01132`.

Ramusat, Yann, Silviu Maniu, and Pierre Senellart (2021b). "Provenance-Based Algorithms for Rich Queries over Graph Databases". In: *EDBT*.

Part I.

Preliminaries

1. Notation and Basic Structures

1.1. Graphs

Let $\mathcal{V}$ be a countably infinite set, whose elements are called *node identifiers* or *node ids*. A *graph* G over V is a pair (V, E), where V is a finite set of node ids (i.e., $V \subseteq \mathcal{V}$) and $E \subseteq V \times V$.

Given an edge $e \in E$, we denote by $n[e]$ its second component and we call it its *destination* (or next) vertex, by $p[e]$ its first component and we call it its *origin* (or previous vertex). Given a vertex $v \in V$, we denote by $E[v]$ the set of edges leaving v. A path $\pi = e_1 e_2 \cdots e_k$ in G is an element of E^* with consecutive edges: $n[e_i] = p[e_{i+1}]$ for $i = 1, \ldots, k-1$. The path corresponding to the empty list is called the *empty path*, written ϵ.

We extend n and p to paths by setting $p[\pi] \coloneqq p[e_1]$, and $n[\pi] \coloneqq n[e_k]$. A *cycle* is a path starting and ending at the same vertex: $n[c] = p[c]$. If $\pi = e_1 e_2 \cdots e_k$ and $\pi' = e'_1 e'_2 \cdots e'_{k'}$ are two paths and $n[\pi] = p[\pi']$, the *concatenation* $\pi\pi' \coloneqq e_1 e_2 \cdots e_k e'_1 e'_2 \cdots e'_{k'}$ is a path with $p[\pi\pi'] = p[e_1]$ and $n[\pi\pi'] = n[e'_{k'}]$.

Let $s \in V$ be a fixed vertex of V called the *source*. We denote by $P_{sv}(G)$ the set of paths from s to $v \in V$. By extension, $P_{ij}(G)$ is the set of paths from i to j.

1.1.1. Treewidth

Given a graph $G = (V, E)$, a *tree decomposition* $D_G = (S_G, T_G)$ of G is a tree T_G whose vertices are elements of S_G, where S_G is a collection of subsets of V, such that:

1. $\bigcup_{S_i \in S} S_i = V$;
2. for all edge e of E, $\exists S_i \in S_G$ such that $n[e], p[e] \in S_i$;
3. for all element v of V, the elements of S_G that contain v form a subtree of T.

The *treewidth* is then defined as $tw(G) \coloneqq \min_{D_G} \max_{S_i \in S_G} |S_i| - 1$. For example, the treewidth of a tree is 1, the treewidth of a cycle is 2, and the treewidth of a $k+1$-clique is k.

The only property of treewidth relevant to this manuscript is that treewidth can be characterized in terms of elimination orderings. The *elimination of a vertex* in a graph is the operation of removing the vertex from the graph and adding a new edges between all of its neighbors. An *elimination ordering* is then an order on the vertices defining a sequential elimination of all vertices. Over all possible choices of an elimination ordering,

we are interested in the one minimizing the maximum neighborhood size encountered during the process – the *width*. We call it a *simplicial elimination order*. It turns out the width of a simplicial elimination ordering for G is precisely the treewidth of G.

1.2. Datalog

We recall some basics about the Datalog query language and refer the reader to [Abiteboul et al. 1995] for more details.

A *Datalog rule* is of the form $R(\vec{x})$:- $R_1(\vec{x_1})$, ..., $R_n(\vec{x_n})$ with R's representing relations of a given arity and the $\vec{x}$'s tuples of variables of corresponding arities. Variables occurring on the left-hand side, the *head* of the rule, are required to occur in at least one of the atoms on the right-hand side, the *body* of the rule. A *Datalog program* is a finite set of Datalog rules. We call *fact* a rule with an empty body and variables replaced by constants. We divide relations into *extensional* ones (which can only occur as head of a fact, or in rule bodies) and *intensional* ones (which may occur as heads of a non-fact rule). The set of extensional relations is called the *extensional database* (EDB). The set of intensional facts is called the *intensional database* (IDB). These two sets are disjoints and their union is called the *schema* of the datalog program. We sometimes distinguish one particular relation occurring in the head of a rule, this relation being the *output predicate* of the Datalog program.

Semantics: A Datalog program q describes a mapping from an *instance* $\mathbf{I}$ (a set of facts in the EDB) to a set of facts $q(\mathbf{I})$ in the IDB. It is often more convenient to call the input data the extensional database and the program the intensional database.

We outline as Algorithm 1 an example Datalog query leveraging recursivity for computing the transitive closure of a directed graph: the relation edge constitutes the EDB, and path the IDB.

Algorithm 1 Datalog program computing the transitive closure

```
1: path(x, y) :- edge(x, y).
2: path(x, y) :- path(x, z), edge(z, y).
```

1.2.1. Derivation Trees

A derivation (aka. a proof) for a fact in $q(\mathbf{I})$ can be represented by a labeled tree whose leaves are facts in $\mathbf{I}$ and internal nodes correspond to instantiations of rules of q.

Example 1.1. *We show in Figure 1.1 an example Datalog program (right) as well as the (only) two derivation trees of the fact* path(Paris, London)*.*

1.2.2. Fixpoint-Theoretic Approach

We now introduce the *immediate consequence operator* T_q of a Datalog program q. This operator, given an instance $\mathbf{I}$, computes the new facts that are *immediate consequence*

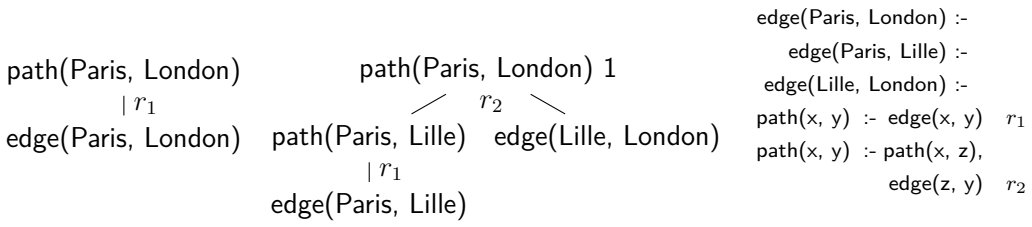


Figure 1.1.: Derivation trees for the fact path(Paris, London) using the transitive closure Datalog program with an EDB containing 3 facts.

(i.e., derivable from $\mathbf{I}$ using a single application of a rule of q). This operator is *monotone* and its least fixpoint is precisely $q(\mathbf{I})$ [Abiteboul et al. 1995]. This yields a constructive definition for $q(\mathbf{I})$, that is to compute $\mathbf{I} \subsetneq T_q(\mathbf{I}) \subsetneq T_q^2(\mathbf{I}) \subsetneq \cdots \subsetneq T_q^i(\mathbf{I}) = T_q^{i+1}(\mathbf{I}) = q(\mathbf{I})$ until convergence.

Example 1.2. *Given the Datalog program and the instance $\boldsymbol{I}$ from Figure 1.1, we have:* $T_q(\boldsymbol{I}) = \boldsymbol{I} \cup \{\mathit{path(Paris, London)}, \mathit{path(Paris, Lille)}, \mathit{path(Lille, London)}\}$ *and* $T_q^2(\boldsymbol{I}) = T_q(\boldsymbol{I})$.

2. Elements of Semiring Theory and Their Applications

While not being properly acknowledged in graduate studies, *semirings* (colloquially called **rigs**, because they are **rings** without **n**egatives) are mathematical structures, perfectly suited for the representation of computations, playing a central role in the algebraic foundations of computer sciences. One of the prominent uses of such structures is made by the notion of *provenance* introduced by Green et al. [2007], stemming from the field of relational databases and addressing the lack of information about the atomic database operations leading to the results of a query.

We start with the basis of the theory of semirings. A good starting point for an introduction on semirings is the chapter *Semirings and Formal Power Series: Their Relevance to Formal Languages and Automata* from the textbook *Handbook of Formal Languages* [Kuich 1997].

Definition 2.1 (Monoid)**.** *A* monoid *is an algebraic structure* $(M, \oplus, \bar{0})$ *where* S *is some set,* $\oplus$ *is a binary operator over* M*, and* $\bar{0}$ *is an element of* M *such that* $\oplus$ *is associative and* $\bar{0}$ *is the identity element for* $\oplus$*.*

Definition 2.2 (Semiring)**.** *A* semiring *is an algebraic structure* $(S, \oplus, \otimes, \bar{0}, \bar{1})$ *where* S *is some set,* $\oplus$ *and* $\otimes$ *are binary operators over* S*, and* $\bar{0}$ *and* $\bar{1}$ *are elements of* S*, satisfying the following axioms:*

- $(S, \oplus, \bar{0})$ *is a* commutative monoid*:* $(a \oplus b) \oplus c = a \oplus (b \oplus c)$*,* $a \oplus b = b \oplus a$*,* $a \oplus \bar{0} = \bar{0} \oplus a = a$*;*
- $(S, \otimes, \bar{1})$ *is a* monoid*:* $(a \otimes b) \otimes c = a \otimes (b \otimes c)$*,* $\bar{1} \otimes a = a \otimes \bar{1} = a$*;*
- $\otimes$ *distributes over* $\oplus$*:* $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$*,* $(b \oplus c) \otimes a = (b \otimes a) \oplus (c \otimes a)$*;*
- $\bar{0}$ *is an annihilator for* $\otimes$*:* $\bar{0} \otimes a = a \otimes \bar{0} = \bar{0}$*.*

The $\oplus$ operation is always meant to be commutative, thus, we call a semiring *commutative* if, for all $a, b \in S$, $a \otimes b = b \otimes a$. A semiring is *idempotent* if for all $a \in S$, $a \oplus a = a$. One can define a *pre-order* $\leq$ over a semiring by defining $a \leq b \coloneqq \exists h$, $a \oplus h = b$. We always have for all a, b, $c \in S$, that $\bar{0}$ is the *smallest element* ($\bar{0} \leq a$) and that $\oplus$ and $\otimes$ are *monotone*: $a \leq b$ implies $a \oplus c \leq b \oplus c$ and $a \otimes c \leq b \otimes c$. When $\leq$ is a partial order it is called the *natural order*.

Property 2.3. *If the semiring is idempotent, then* $\leq$ *is a partial order, equivalently defined by* $a \leq b \coloneqq a \oplus b = b$*.*

Proof. For the first part of the statement, we are only to verify $\leq$ is an antisymmetric relation: if $x \leq y$ and $y \leq x$ then it exist c and d such that $a \oplus c = y$ and $y \oplus d = x$, thus $x = x \oplus c \oplus d$ and then $y = x \oplus c = x \oplus c \oplus d \oplus c = x$. For the equivalence of the two orders, we observe that if exists h such that $a \oplus h = b$ then $a \oplus b = a \oplus a \oplus h = b$. □

Definition 2.4 (Morphism of semirings)**.** *For two semirings* $(R, \oplus_R, \otimes_R, \bar{0}_R, \bar{1}_R)$ *and* $(S, \oplus_S, \otimes_S, \bar{0}_S, \bar{1}_S)$*, a* semiring morphism *is a function* $h\colon R \to S$ *such that:*

- $h(\bar{0}_R) = \bar{0}_S$ *and* $h(\bar{1}_R) = \bar{1}_S$*;*
- $h(r \oplus_R r') = h(r) \oplus_S h(r')$ *and* $h(r \otimes_R r') = h(r) \otimes_S h(r')$ *for all* r, $r' \in R$*.*

A morphism of semirings which is both injective and surjective is called an *isomorphism*. If there exists an isomorphism between semirings R and S we say that R and S are *equivalent*.

We give below examples of three important semirings.

Semiring 1 (Boolean semiring)**.** *The* Boolean semiring $(\{\bot, \top\}, \vee, \wedge, \bot, \top)$ *where* $\vee$ *denotes the disjunction,* $\wedge$ *the conjuction,* $\bot$ *is the false value, and* $\top$ *the true value.*

Semiring 2 (Tropical semiring on the integers)**.** *The* $(\min, +)$-semiring on the integers $(\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$ *with, for all* $c \in \mathbb{N}_+$*,* $\infty + c = c + \infty = \infty$ *and* $\min(c, \infty) = \min(\infty, c) = c$*.*

Semiring 3 (Tropical semiring on the reals)**.** *The* $(\min, +)$-semiring on the reals $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$ *with, for all* $c \in \mathbb{R}_+$*,* $\infty + c = c + \infty = \infty$ *and* $\min(c, \infty) = \min(\infty, c) = c$*.*

Semirings 1, 2, and 3 are all commutative and idempotent semirings. They are thus naturally ordered: for Semiring 1 the natural order gives $\bot \leq \top$ and for Semirings 2, and 3 the natural order $\leq$ is the reversed order on the numbers (0 is the greatest element and ∞ the least).

2.1. Closed Semirings and the Floyd-Warshall Scheme

This section refers to what is called the *Algebraic Path Problem* (APP) [Rote 1990], the *(General) Path Problem* [Aho et al. 1974; Mehlhorn 1984], or the *Path-finding Problem* [Aho et al. 1974; Maggs et al. 1988].

Instead of introducing a plethora of definitions and properties, we go straight into showcasing semirings in action through a conceptually simple use case, commonly found in some standard algorithmic textbooks. The reader might already been familiar with the algebraic formulation of the dynamic programming approach to the computation of all-pairs shortest paths in directed graphs (APSP), subsuming the transitive-closure algorithm. These two problems are solved by instantiating the dynamic programming scheme on a dedicated semiring for each one: the Boolean semiring (Semiring 1) for the

transitive closure of the graph, and the Tropical semiring (Semiring 2 or 3) for the APSP problem.

We recall here the main ideas, introduced in the textbook of Cormen et al. [1989, Chapter 26.4]. The aim is to understand which are the required properties of an algebraic structure to express *path problems* such as the transitive closure or APSP. This algebraic framework needs to encompass all of the semirings used to express these problems, but also needs to be as general as possible. That is to say, the objective is restrict them only to a set of properties under which the problem is well-formed; this will lead to the notion of *closed semirings*.

We consider a directed graph $G = (V, E)$. Some labels over the edges are given by a labeling function $\lambda : E \to S$, with range a semiring S. The $\otimes$ operator of the semiring will be used to construct labels for paths: $\lambda(e_1 e_2 \cdots e_k) \coloneqq \lambda(e_1) \otimes \lambda(e_2) \otimes \cdots \otimes \lambda(e_k)$. The empty path has label $\bar{1}$. Associativity of $\otimes$ allows to forget the computation order and ensures the concatenation for paths to be well-defined: $\lambda(\pi\pi') \coloneqq \lambda(\pi) \otimes \lambda(\pi')$.

For the Boolean semiring, labeling an edge with value $\top$ indicates its availability (we can go through), hence, implicitly all the elements of E are positively labeled and the remaining elements in $(V \times V) \setminus E$ have label $\bot$. It will be understood for the rest of the section that under each semiring, missing edges have label $\bar{0}$. The Tropical semiring enriches the positive labels by offering the opportunity to provide scalar values to available connections in the graph (such as the traveling time between two locations), unavailable edges are now labeled by ∞ (the $\bar{0}$ element of any Tropical semiring). Somehow, any version of a Tropical semiring seems to be an *extension* of the Boolean semiring: this is formally stated as the fact it exists a surjective morphism (mapping any non ∞ value to $\top$) from any Tropical semiring to the Boolean one.

The $\otimes$ operator permits to give the expected semantic for paths: $(\wedge)$ ensures each of the constituent edges to be available for the path to be praticable, while the traveling time is the sum $(+)$ of the travel time for each connections.

The $\oplus$ operator aggregates labels over a given (possibly infinite) set of paths. Because there are countably many paths in a graph, we thus assume for the rest of this example that infinite stands for countably infinite. This allows us to define the aggregate of the labels between two vertices.

$$l_{ij} \coloneqq \bigoplus_{\pi \in P_{ij}(G)} \lambda(\pi) \tag{2.1}$$

Because no summation order is provided in Equation 2.1, $\oplus$ has to be associative and commutative. Following Cormen et al. [1989], it is mandatory for $\oplus$ to be idempotent to remain consistent with set semantics. This limitation has been criticized in the literature (e.g., Lehmann [1977]) and sometimes [see Mehlhorn 1984, Chapter 5] a closed semiring is not required to be idempotent. For the sake of simplicity, we will follow the design choice of Cormen et al. and use an idempotent structure until the end of the section. Nevertheless, we will get rid of this limitation in the forthcoming chapters.

While the sum in Equation 2.1, being possibly infinite, is not an issue using the Boolean semiring, special attention ought to be paid when using real values. In the case of the Tropical Semiring, this "formal sum" turns out to be the notion of *infimum* for countable

sequences. Therefore, additional properties are needed for infinite sums to behave as expected. We assume the summation operator to be extended for countably infinite sums (i.e. $\bigoplus_{i\in I} a_i$ is well-defined). Further more, computations are still independent on the computation order (associativity extends to infinite sums), the terms can be reordered and duplicates can be removed (commutativity and idempotence also extend to infinite sequences). The distributivity must hold to be able to factorize diverging paths (paths starting by a common subpath), it also needs to hold in case of infinite sums to handle cycles.

The notion of *closed semiring* encompasses the above properties:

Definition 2.5 (Closed semiring [Cormen et al. 1989])**.** *A* closed semiring *is an idempotent semiring S having additional properties:*

- *If $(a_i)_{i\in I}$ is a countable sequence of elements of S, $\bigoplus_{i\in I} a_i$ is well-defined and belongs to S;*
- *Associativity, commutativity and idempotence extend to infinite sums;*
- *$\otimes$ distributes over infinite sums: $a \otimes (\bigoplus_{i\in I} b_i) = \bigoplus_{i\in I} (a \otimes b_i)$ and $(\bigoplus_{i\in I} b_i) \otimes a = \bigoplus_{i\in I} (b_i \otimes a)$.*

In closed semirings an additional operation can be defined, the *closure operator*: for each $a \in S$, $a^* \coloneqq \bar{1} \oplus a \oplus a^2 \oplus \cdots = \bigoplus_{0\leq i} a^i$. The reader may recall the notion of *Kleene star* for regular languages. Indeed they are strongly related; Section 2.2 will cover the links between those notions. To distinguish between the infinite summation operator and $\oplus$ we will also denote the former by $\sum$.

Property 2.6. *The boolean semiring $(\{\bot, \top\}, \vee, \wedge, \bot, \top)$ is a closed semiring.*

Property 2.7. *The $(\min, +)$-semiring on the integers $(\mathbb{N}\cup\{\infty\}, \min, +, \infty, 0)$ is a closed semiring.*

Property 2.8. *The $(\min, +)$-semiring on the reals $(\mathbb{R}_+ \cup \{\infty\}, \min, +, \infty, 0)$ is a closed semiring.*

Algorithm 2 introduces what is also called in the Kleene method for solving the APSP problem [Mehlhorn 1984]. According to Cormen et al., the Floyd-Warshall algorithm has been proposed by Floyd [1962] inspired by Warshall's theorem [1962] for computing the closure of a graph.

We have so far introduced some notions from the textbook of Cormen et al. [1989, Chapter 26.4]. Note this chapter is only present in the first edition of the book and removed from the subsequent one. The authors acknowledged the fact the underlying structure of *closed semirings* has been previously introduced by Aho et al. [1974]. A more in depth analysis of such framework can be found in the Mehlhorn's book [1984, Chapter 5]; Section 2.3 will discuss their results related to theoretical lower-bounds. The first part of the thesis focuses on the semiring-based provenance framework for graph databases, which generalizes what we have just presented there. This section permitted us to give valuable insights, on a simpler model, on how semirings permit to capture the expected semantics of real life concerns.

Algorithm 2 Floyd–Warshall [Cormen et al. 1989] – All-pairs shortest-distance

Input: $G = (V, E, \lambda)$ with $n = |V|$ a directed graph with labels over a closed semiring S.
Output: The matrix $L^n = [l_{ij}^n]$ of aggregated labels between every pair of nodes.

for $i \in V$ **do**
  for $j \in V$ **do**
    $l_{ij}^0 = \begin{cases} \lambda(i,j) & \text{if i} \neq \text{j}, \\ \bar{1} \oplus \lambda(i,j) & \text{if i} = \text{j} \end{cases}$
  end for
end for
for $k \leftarrow 1$ **to** n **do**
  for $i \in V$ **do**
    for $j \in V$ **do**
      $l_{ij}^k = l_{ij}^{k-1} \oplus \left(l_{ik}^{k-1} \otimes (l_{kk}^{k-1})^* \otimes l_{kj}^{k-1} \right)$
    end for
  end for
end for

2.1.1. Going Deeper and Deeper: Equational Systems and Matrices

We have so far discussed some problems related to weighted sets of paths. Nevertheless, this representation – much more appealing to computer scientists – is in fact a particular instance of the resolution of equational systems and of matrix operations, fundamental tools for mathematicians. We shall now discuss the same problem, but rephrased as a solution of a system of equations.

Semiring 4 (Semiring of formal languages over Σ)**.** *The* semiring of formal languages $\mathcal{F}_\Sigma = (2^{\Sigma^*}, \cup, \cdot, ^*, \emptyset, \{\epsilon\})$ *with* $L^* = \bigcup_{n \geq 0} L^n$ *for all* $L \subseteq \Sigma^*$.

Semiring 4 contains all the languages over Σ, not only the relational, or even context-free, languages.

Property 2.9 ([Aho et al. 1974])**.** *The semiring of formal languages* $\mathcal{F}_\Sigma$ *is a closed semiring.*

We can turn an automaton $\mathcal{A} = (Q, \Sigma, q_0, \delta, F \subseteq Q)$ into a labeled graph $G = (V, E)$ over $\mathcal{F}_\Sigma$ whose vertices are the states of the automaton (i.e., $V = Q$) and the labels are given by the transition function of the automaton:

$$\forall e \in E,\ \lambda(e) \coloneqq \{a \in \Sigma \mid n[e] \in \delta(p[e], a)\}.$$

Notice the label of an edge e representing an absent transition in the automata (i.e., $\lambda(e) = \emptyset$) is labeled with the zero element of the semiring of formal languages and can be considered as a missing edge. Let us now come back to a bottom-up definition of *regular languages*, as languages built from finite set of words using $\cup$, $\cdot$, and * operators. To avoid dealing with infinite sets of words, we can alternatively consider *regular expressions* to describe the operators instead of evaluating them.

The well-known Bellman-Ford algorithm [Bellman 1958] led to a dynamic programming formulation of the shortest-path problem using a system of equations (functional equation approach). In the following, we will exhibit the system of equations expressing the shortest path constraints for a weighted graph and the system of equations defining the regular language of a given automaton. These two problems have been acknowledged in [Rote 1990] to be both expressible as equational systems over closed semirings.

$$\begin{aligned} x_{ij} &= \bigoplus_{k=1}^{n} (l_{ik} \otimes x_{kj}) \qquad \text{for all } i \neq j, \\ x_{ii} &= \bigoplus_{k=1}^{n} (l_{ik} \otimes x_{kj}) \oplus \bar{1} \end{aligned} \tag{2.2}$$

In Equation 2.2 the x_{ij} variables denote, depending on the problem:

- the shortest-path between vertices i and j, using a Tropical semiring;
- the language recognized by an automaton from state i to state j, using $\mathcal{F}_\Sigma$.

Thus, in both cases, computing the aggregated labels defined by Equation 2.1 is the same as solving the system of equations displayed as Equation 2.2. The latter equation becomes $X = AX \oplus I$ in matrix notation and its solution, the *star of the matrix* A, is denoted A^*.

Algorithm 2 then provides a constructive proof that the accepted language of an automaton is regular (by taking the union of all returned values $l^n_{q_o q_F}$ for each $q_F \in F\}$) and this is in fact one half of the famous Kleene's theorem [Sakarovitch 1987] stating the equivalence between finite automata and regular expressions. Understood in terms of semirings, this means that the entries of the star of a matrix over closed semirings are regular combinations of the entries of the initial matrix (Theorem 5 in [Conway 1971, Chapter 3]).

From an equational perspective, the first part of Arden's lemma [1961] turns out to be implied by the properties of closed semirings (see Property 2.10). Thus, the iterative application of Arden's lemma for solving an equational system (known as the Brzozowski and Mc Cluskey method [1963]) under the formal language semiring generalizes to arbitrary closed semirings.

Property 2.10 (Theorem 3 in [Conway 1971, Chapter 3])**.** *In a closed semiring a^*b is the least solution to $x = ax \oplus b$.*

2.1.2. Other Related Problems

So far, we exemplified the *algebraic path problem* using concrete applications arising from graph theory (transitive closure, all-pairs shortest paths) or formal language theory (equivalent language of an automaton). The intended result was simply obtained from solving Equation 2.1 without further operations over the aggregated labels. In fact, the range of applicability of such framework is much broader if we allow to "query" the aggregated labels. We will not cover in detail all of the possible applications, instead, we carefully select and briefly cover two use cases.

Testing if a graph is bipartite [Rote 1990] Let us work on the set $S = \{\emptyset, E, O, EO\}$ which records for sets of paths if they contain only even paths, odd paths, or both. We label edges with O or $\emptyset$ depending on whether they are present or not in the graph. We can munish S with suitable definitions for operators $\cup$, $\cdot$, and * encoding the desired semantic and satisfying the semiring axioms. The semiring of use is thus $(\{\emptyset, E, O, EO\}, \cup, \cdot, \emptyset, EO)$.

By inspecting all l_{ii}^n returned by Algorithm 2, we can check if a directed graph has an even cycle. Moreover, this definition permits to count path length modulo 2 and can clearly be extended to modulus counting over an arbitrary integer $k \geq 2$. It is then possible to compute shortest-path of length m mod k if whoever wants to.

Note. The semiring-based provenance framework for graph databases we have studied during this PhD subsumes completely those results pointed out by Rote [1990] to arbitrary regular constraints (RPQs) over paths. We refer the reader to the extended discussion around this at the end of Section 4.3.

Finding the bridges of a graph [Rote 1990] A *bridge* is an edge in a graph $G = (V, E)$ whose removal breaks a connected component into two connected components. We introduce the semiring $(2^E \cup \{\bar{0}\}, \cap, \cup, \bar{0}, \emptyset)$ with underlying set the subsets of E with one fresh new zero element $\bar{0}$. Semiring axioms completely determine how $\bar{0}$ interacts with the other elements. We label an edge e of the graph with the set $\{e\}$, and with $\bar{0}$ if it was missing. Equation 2.1 then gives:

$$l_{ij} = \bigcap_{\pi \in P_{ij}(G)} \bigcup_{e \in \pi} \lambda(e).$$

Edges in l_{ij} are shared amongst all paths from i to j and are then bridges. Conversely, any bridge appears in at least one set l_{ij}.

The new element was added to distinguish between the cases where all edges are bridges and where the two vertices are not connected. Such an idea to create a new $\bar{0}$ element is quite commonly encountered in semiring theory and is know as the *one-point closure*, as we will see some other examples later on (e.g., Section 2.2.1).

These annotations have been called *aggregated labels* up to this point. We will provide more context to these annotations in Part II and introduce a unifying framework providing a concise semantic for querying graph databases. All the intuitions described there will continue to apply to this next framework.

We hope to have properly introduced to the reader this bird's-eye view using algebraic patterns providing unifying frameworks for what are, at first glance, unrelated problems. This constitutes, to my own point of view, the essence of the usage of semirings in computer science research.

2.2. Infinite Sums and Continuous Operators in Semirings

The provenance semiring framework of Green et al. [2007], introducing semirings to represent provenance annotations in relational databases (and beyond), relies on the notion of *ω-continuity*, a property of some semirings we informally justify now and later formally define. The provenance of a Datalog query in their line of work is then defined as the least fixpoint of a system of fixpoint equations over ω-continuous semirings. Continuous structures are of choice in this context because permitting to define an iterative algorithm to solve the system and to obtain the least fixpoint.

Weaker semiring structures than the continuous ones are also relevant to the provenance context, and we have dealt with many of them during the PhD. In the following, we introduce the reader to the most crucial semiring structures, in increasing order of complexity. We eventually reach the *ω-continuous* structure at the basis of the provenance framework and study their properties pertaining to the resolution of fixpoint equations.

When adapting this definition to graph databases, more akin to semiring-valued matrices, we eventually noticed strong similarities with other research areas in computer sciences, such as the algebraic path problem, formal languages, weighted automata, and even linear algebra and the fundamental algorithmic framework of dynamic programming (understood as discrete optimization problems). Each of these research areas have their own theoretical line of work in which semirings play a crucial role. By the end of this section, the reader will also have an overview of semirings and their applications, and will understand how the underlying theoretical framework for this doctoral work is positioned across the computer science literature.

The amount of textbooks, research papers, definition, properties, etc, covered in this "survey" is large, but it is by no mean supposed to be exhaustive. *A lot* of the knowledge acquired by the author during this thesis is not presented here, as we have chosen to only introduce notions that have eventually found some concrete *practical* application during this PhD. As a summary for this section, Table 2.1 provides a translation between the semiring structure names encountered in the literature and our nomenclature.

We start with the notion of *embedding* that will be key for our work. We are generally interested to find an embedding of S into T, with T enriching the structure of S. T will have the property of interest, for instance, to justify the use of an algorithm requiring this property to hold. Morphism properties permit to avoid doing computations over T, and still be performing computations over the simpler structure of S, thus leading to a possibly more efficient implementation.

Definition 2.11 (Embedding)**.** *A semiring S is said to be* embedded *into a semiring T whenever S is isomorphic to a subsemiring of T, or equivalently, if there exists an injective semiring morphism from S to T.*

Rationale. Throughout this thesis, we were constantly searching for a consistent way to define and name the semirings of interest for our research work. Following the proposition made by Kozen [1990], we could use an alternative name for the closed semirings from

Definition 2.5. Kozen has proposed *ω-complete idempotent semirings* but we find *ω-continuous idempotent star semirings* is a better name. The bulk of this section is to convince the reader of the necessity of this nomenclature. In fact, the sole purpose of $\sum$ in Definition 2.5 was to define *, so we use the term "star" in *ω-continuous idempotent star semirings* to stress the fact we use the infinite summation operator to define the star of an element to be the sum of its powers. We now introduce in increasing order of specificity the notions of *star semirings* (Section 2.2.1), *complete semirings* (Section 2.2.2), and *continuous semirings* (Section 2.2.4).

2.2.1. Towards the Simplest Notion of a Closure Operator

This section refers to what has previously been called *closed semirings* in [Lehmann 1977] (but here we assume the presence of a $\bar{0}$ element) and **-semirings* in [Abdali 1994; Abdali et al. 1985]. A more in-depth analysis of such structures and algorithms can be found in [Minoux et al. 2008].

Definition 2.12 (Star semiring [Abdali et al. 1985], and under the name of *closed semiring* [Lehmann 1977])**.** *A* star semiring *is a structure* $(S, \oplus, \otimes, ^*, \bar{0}, \bar{1})$ *where* $(S, \oplus, \otimes, \bar{0}, \bar{1})$ *is a semiring and* * *satisfies additional axioms:*

$$\bar{1} \oplus aa^* = \bar{1} \oplus a^*a = a^*.$$

This is the most basic structure where we can define an *asterate* a^* (also known as a *closure*) for its elements. We also note $a^+ = aa^* = a^*a$. In [Abdali 1994; Abdali et al. 1985] the structure may not be closed for the asteration (i.e. * might be a partial function). Nevertheless, the *one-point-closure* (adding a new element ∞ to be the asterate of each element for which the operator was not previously defined) can easily solve the problem.

The *classical axioms* introduced thereafter permit to simplify the reasoning on the properties of star semirings and the other more sophisticated structures occuring in this manuscript.

Definition 2.13 (Classical axioms [Conway 1971, pp. 25, 35])**.**

$C1.$ $a \oplus \bar{0} = a$;

$C2.$ $a \oplus b = b \oplus a$;

$C3.$ $(a \oplus b) \oplus c = a \oplus (b \oplus c)$;

$C4.$ $a \otimes \bar{0} = \bar{0}$;

$C5.$ $\bar{0} \otimes a = \bar{0}$;

$C6.$ $a \otimes \bar{1} = a$;

$C7.$ $\bar{1} \otimes a = a$;

$C8.$ $a(b \oplus c) = ab \oplus ac$;

$C9.$ $(b \oplus c)a = ba \oplus ca$;

$C10.$ $a(bc) = (ab)c$;

$C11.$ $(a \oplus b)^* = (a^*b)^*a^*$;

$C12.$ $(ab)^* = \bar{1} \oplus a(ba)^*b$;

$C13.$ $(a^*)^* = a^*$;

$C13^\circ.$ $\bar{1}^* = \bar{1}$;

$C14.\ a^* = (a^n)^* \oplus (\bar{1} \oplus a \oplus \cdots \oplus a^{n-1})$; $C19.\ \bar{0}^* = \bar{1}$.

$C15.\ \bar{1} \oplus aa^* = a^*$;

It is not required for $\otimes$ to be commutative, neither for $\oplus$ to be idempotent. The *sum-star equation* ($C11$) and the *product-star equation* ($C12$) are also not required (neither are $C13$ and $C14$). Star semirings verify the first ten "classical" axioms (basic semiring axioms) plus, by definition, axiom $C15$. $C19$ also holds in star semirings and is deduced from $C15$.

The primary interest of those structures is to consider computations over their associated matrices. There are two equivalent definitions of such matrices, one by induction [Lehmann 1977] and the other one involving eliminants [Abdali 1994; Abdali et al. 1985]. We will only discuss the inductive definition as the other one mainly focuses on providing a suitable definition for parallel computations.

Definition 2.14 (Matrices over star semirings [Lehmann 1977])**.** *Given a star semiring S and a positive integer $n \geq 1$ we define the set S_n of $n \times n$ matrices over S with:*
$\bar{0}_{S_n} = \begin{bmatrix} \bar{0} & \cdots & \bar{0} \\ \vdots & \ddots & \vdots \\ \bar{0} & \cdots & \bar{0} \end{bmatrix}$, $\bar{1}_{S_n} = \begin{bmatrix} \bar{1} & \cdots & \bar{0} \\ \vdots & \ddots & \vdots \\ \bar{0} & \cdots & \bar{1} \end{bmatrix}$, *for $A = [a_{ij}]_{i,j\in[1,n]}$ and $B = [b_{ij}]_{i,j\in[1,n]}$:*
$A \oplus B = [a_{ij} \oplus b_{ij}]_{i,j\in[1,n]}$ *and* $A \otimes B = \left[\bigoplus_{k=1}^{n} (a_{ik} \otimes b_{kj})\right]_{i,j\in[1,n]}$,
for $n = 1$: $A^* = [a^*]$ and for $n \geq 2$ and any subdivision of $A = \begin{bmatrix} B & C \\ D & E \end{bmatrix}$ where B and E are square matrices:
$A^* = \begin{bmatrix} B^* \oplus B^*C\Delta^*DB^* & B^*C\Delta^* \\ \Delta^*BD^* & \Delta \end{bmatrix}$ *with $\Delta = (E \oplus DB^*C)^*$.*

Property 2.15 ([Lehmann 1977])**.** *For all $n \geq 1$, $(S_n, \oplus, \otimes, ^*, \bar{0}_{S_n}, \bar{1}_{S_n})$ are star semirings.*

A wise reader will notice the top-left cell of A^* is not simplified to $(B \oplus CE^*D)^*$. In fact this simplification can be done if and only if the Conway equalities (see Definition 2.16) hold [Lehmann 1977].

Definition 2.16 (Conway semiring [Droste et al. 2009])**.** *A* conway semiring *is a semiring satisfying the sum-star equation ($C11$) and the product-star equation ($C12$) or, equivalently, a star semiring satisfying $(a \oplus b)^* = (a^*b)^*a^*$ and $(ab)^*a = a(ba)^*$.*

Property 2.17 ([Conway 1971, p. 35])**.** *In star semirings $C13$ implies $C13^\circ$. In Conway semirings $C13$ is equivalent to $C13^\circ$. Moreover, all idempotency laws $\bar{1} \oplus \bar{1} = \bar{1}$ ($C21$), $a \oplus a = a$ ($C22$), $(\bar{1} \oplus a)^* = a^*$ ($C23$), and $a^*a^* = a^*$ ($C24$) follow from $C13^\circ$ in Conway semirings.*

Let us now turn our attention to the analogous form of Algorithm 2, stated in terms of matrix operations. Here, star semirings play the role of the most general algebraic

structure on which an asteration of matrices can be defined. A very curious reader could also find a valuable insight in [Minoux et al. 2008, Chapter 4.5] on how this method can be understood as a generalization of the Gauss-Jordan method in linear algebra.

Algorithm 3 Warshall–Floyd–Kleene [Lehmann 1977] – Asteration of a matrix

Input: $A = A[i,j]_{i,j\in[1,n]}$ a matrix over a star semiring S.
Output: The closure A^* of A.

```
1: A_0 = A
2: for k ← 1 to n do
3:    for i ∈ [1...n] do
4:       for j ∈ [1...n] do
5:          A_k[i,j] = A_{k-1}[i,j] ⊕ (A_{k-1}[i,k] ⊗ (A_{k-1}[k,k])* ⊗ A_{k-1}[k,j])
6:       end for
7:    end for
8: end for
9: A* = A_n ⊕ 1̄_{S_n}
```

Finally, the last interesting usage of star semirings we touch on is the possibility to solve systems of equations [Abdali 1994]. In this case, a^* is not necessarily the unique solution to $x = \bar{1} \oplus ax$, and neither is a^*b to $x = ax \oplus b$. We provide at the end of this section an example (related to Semiring 7) were an alternative definition of the * operator leads to another solution to these equations. Historically, *complete* and *continuous semirings* we discuss later have been introduced to provide additional contraints to star semirings in order to restrict the asterates to be the least solutions to such systems of equations.

As a final remark before providing examples of star semirings, we stress the fact they are not necessarily *commutative*, thus leading to different possible solutions to left and right-handed systems of equations.

Examples.

Semiring 5 (Security semiring)**.** *The semiring* $(\{0,\dots,n\}, \min, \max, n, 0)$ *with* $a^* = 0$ *for all* a*.*

Also exists in the $(\max, \min)$ variant, and over $\mathbb{N}$ or $\mathbb{R}$. It turns out $(\max, \min)$ and $(\min, \max)$ are both distributive lattices with reversed meet and join operations.

Semiring 6 (Reliability semiring [Rote 1990])**.** *The semiring* $([0,1], \max, \cdot, 0, 1)$ *with* $a^* = 1$ *for all* a*.*

Semiring 7 (The idempotent commutative semiring – Example (ix) in [Droste et al. 2009, p. 9])**.** *The semiring* $(\{0,1,a,\infty\}, +, \cdot, 0, 1)$ *with* $0 \le 1 \le a \le \infty$*,* $a \cdot a = a$*,* $0^* = 1$*,* $1^* = 1$*,* $a^* = \infty$*, and* $\infty^* = \infty$*.*

Semiring 8 (Real numbers semiring)**.** *The semiring* $(\mathbb{R} \cup \{\infty\}, +, \cdot, 0, 1)$ *with* $a^* = \frac{1}{1-a}$ *when* $|a| < 1$ *and* $1^* = \infty$ *either.*

All previous introduced closed semirings (Semirings 1, 2, 3, and 4) are also star semirings using the $\sum$-based definition of the star operator. Semirings 5, 6, 7, and 8 are also star semirings. It also exists for Semirings 5 and 6 a proper infinite summation operator compatible with the star. An alternative definition for $a^* := a$ in Semiring 7 also provides a solution to $\bar{1} \oplus ax = x$, with $a \leq \infty$. Semiring 8 is in fact a ring and we will see right after it is not possible to embed any ring into a complete semiring and thus, to any closed semiring.

2.2.2. Completeness: Working with an Infinite Summation Operator

This section refers to *complete semirings* of Krob [1987], Goldstern [2002], and in [Droste et al. 2009; Eilenberg 1974; Kuich 1997]. *ω-Complete* structures appear under the name of *closed semirings* in [Mehlhorn 1984, Chapter 5.1].

Definition 2.18 (Complete monoid [Krob 1987])**.** *Let $(M, \oplus, \bar{0})$ be a commutative monoid, M is a* complete monoid *(resp. ω-*complete monoid*) if for every (resp. at most countable) family set I of elements of M we can define an element $\bigoplus_{i \in I} x_i$ satisfying:*

1. *if I is empty: $\bigoplus_{i \in \emptyset} x_i = \bar{0}$;*
2. *if I is non-empty finite: $\bigoplus_{i \in \{1, \ldots, n\}} x_i = x_1 \oplus \cdots \oplus x_n$;*
3. *for every (resp. at most countable) index set I and every partition J of I:*

$$\bigoplus_{i \in I} x_i = \bigoplus_{j \in J} \left(\bigoplus_{i \in I_j} x_i \right).$$

Definition 2.19 (Complete semiring [Krob 1987])**.** *Let $(S, \oplus, \otimes, \bar{0}, \bar{1})$ be a semiring, S is a* complete semiring *(resp. ω-*complete semiring*) if for every (resp. at most countable) family set I of elements of S we can define an element $\bigoplus_{i \in I} x_i$ satisfying:*

1. *$(S, \oplus, \bar{0})$ is a complete (resp ω-complete) monoid;*
2. *for every (resp. at most countable) index set I and every $s \in S$: $s \otimes \left(\bigoplus_{i \in I} x_i \right) = \bigoplus_{i \in I} (s \otimes x_i)$ and $\left(\bigoplus_{i \in I} x_i \right) \otimes s = \bigoplus_{i \in I} (x_i \otimes s)$.*

We provide basic properties of complete semirings, Property 2.20 establishes the link to the definition of complete semirings from Goldstern [2002] and Property 2.21 the link to the definition from the Melhorn's book [1984, Chapter 5.1].

Property 2.20 ([Krob 1987]). *Let $(S, \oplus, \otimes, \bar{0}, \bar{1})$ be a complete semiring (resp. ω-complete semiring) for every (resp. at most countable) family set I of elements of S we have:*

$$\bigoplus_{i \in I} \bar{0} = \bar{0}.$$

Property 2.21 ([Krob 1987]). *Let $(S, \oplus, \otimes, \bar{0}, \bar{1})$ be a complete semiring (resp. ω-complete semiring), we have for every (resp. at most countable) family set I and J:*

$$\left(\bigoplus_{i \in I} x_i\right) \otimes \left(\bigoplus_{j \in J} x_j\right) = \bigoplus_{(i,j) \in I \times J} (x_i \otimes x_j).$$

The next properties entail the fact that complete semirings are instances of star semirings and permit us to get a formal viewpoint via the equalities satisfied by such structures.

Property 2.22 (Theorem 2.2 in [Kuich 1997]). *Each complete star semiring is a star semiring and satisfies $a^* = (a^n)^* \oplus (\bar{1} \oplus a \oplus \cdots \oplus a^{n-1})$, for $n > 0$ ($C14$).*

Property 2.23 (Theorem 3.4 in [Droste et al. 2009]). *Each complete star semiring is a Conway semiring.*

To sum up the results of Properties 2.22 and 2.23 in terms of classical axioms, we can say that complete semirings verify all the classical axioms presented in Definition 2.13, excepting $C13$ and $C13^\circ$.

It is claimed in [Droste et al. 2009, Chapter 1] that matrices over complete semirings are themselves complete and similarly for matrices over Conway semirings.

In the following, an *embedding of complete semirings* is a semiring embedding ϕ preserving the additional structure of $\sum$: $\phi\left(\sum A\right) = \sum \phi(A)$. All complete semirings are ω-complete but the opposite does not hold (Krob provides an example in [1987]). Nevertheless, we will now turn our attention on an elementary property of complete semirings which will ultimately lead to an embedding of ω-complete into complete ones.

Property 2.24 (Zero-sum free [Krob 1987]). *Let $(S, \oplus, \otimes, \bar{0}, \bar{1})$ be an ω-complete semiring, for all $x, y \in S$, $x \oplus y = \bar{0} \implies (x = \bar{0} \wedge y = \bar{0})$.*

Property 2.24 entails the fact non-trivial rings can neither be ω-complete nor can be embedded inside larger ω-complete semirings. This fact also rules out rings from the lower bounds on monotone operations over matrices (Section 2.3).

Definition 2.25 (Positive semiring [Krob 1987]). *A semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ is said* positive *when:*

1. *for all $x, y \in S$, $x \oplus y = \bar{0} \implies (x = \bar{0} \wedge y = \bar{0})$;*
2. *for all $x, y \in S$, $x \otimes y = \bar{0} \implies (x = \bar{0} \vee y = \bar{0})$.*

Semiring 9. *The* semiring $(2^{\mathbb{N}}, \cup, \cap, \emptyset, \mathbb{N})$ *where* $\cup$ *denote arbitrary union and* $\cap$ *finite intersection.*

Semiring 9 provides an example of a complete semiring not being positive: $\mathbb{N}^* \cap \{0\} = \emptyset$. In [Krob 1987] one can find a construction to embed any positive semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ into a complete semiring by adjoining an infinite element ∞, absorptive for $\oplus$ and for $\otimes$ (except for $\bar{0}$: $\bar{0} \otimes \infty = \infty \otimes \bar{0} = \bar{0}$), and define for any I:

$$\bigoplus_{i \in I} x_i \coloneqq \begin{cases} \infty, & \text{if } J = \{i \in I \mid x_i \neq \bar{0}\} \text{ is infinite,} \\ \bigoplus_{j \in J} x_j, & \text{otherwise.} \end{cases}$$

Semiring 10 (Complete version of Semiring 7)**.** *The semiring* $(\{0, 1, a, \infty\}, +, \cdot, 0, 1)$ *with* $0 \leq 1 \leq a \leq \infty$*,* $a \cdot a = a$*,* $0^* = 1$*,* $1^* = \infty$*,* $a^* = \infty$*, and* $\infty^* = \infty$*.*

Important note. Sometimes this infinite summation operator does not convey much interest. As exemplified in Semiring 10, fixing all sums involving an infinite number of non-zero elements to a new ∞ element provides a suitable definition for a summation operator. Such examples of semiring structures satisfying all formal axioms on a^* for an element a, on the other hand not representing the intuitive idea of the supremum of the powers of the element, is very common in the literature: an example of such is the four-element **R**-algebra[1] R_4 of Conway [1971, p. 102] which is not an **S**-algebra. This ultimately leads us to a discussion on *continuous semirings*.

Examples.

Semiring 11 (Integers semiring [Droste et al. 2009])**.** *The semiring* $(\mathbb{N} \cup \{\infty\}, +, \cdot, 0, 1)$ *with* $0^* = 1$ *and* $a^* = \infty$ *either.*

Semiring 12 (Positive real numbers semiring [Droste et al. 2009])**.** *The semiring* $(\mathbb{R}_+ \cup \{\infty\}, +, \cdot, 0, 1)$ *with* $a^* = \frac{1}{1-a}$ *when* $a < 1$ *and* $1^* = \infty$ *either.*

Semiring 4 is also a complete semiring which is not commutative. Semirings 11 and 12 are complete and non-idempotent. We acknowledge the fact that R_4 and Semiring 7 are actually isomorphic. A work of note is the possibility to extend a *ring* into a complete semiring (refer to Example 2.2 from [Karner 1992]) which shows there exist some complete semirings which are not orderable).

2.2.3. Case Study of Closed Semirings and S-Algebras

This section mainly refers to the **S**-algebras of Conway [1971]. It is also partially covered how **S**-algebras and closed semirings relate to each other using the results from Kozen [1990].

[1] Formal systems with operators $\oplus$, $\otimes$ and * satisfying formal laws universally valid in all **S**-algebras [Conway 1971, Chapter 4].

We now tackle an ubiquitous inconsistency (or *hiatus*) in the literature. Many authors such as Krob [1987], Droste et al. [2009] and Kozen [1990] strongly suggest the **S**-algebras of Conway are an alternative name for *complete semirings* (in fact Kozen uses the name *closed* but ended up naming them *complete idempotent semirings*). Whereas for Kozen an **S**-algebra is necessarily idempotent, it is not the case for Krob and Droste et al. That is already suspicious. So, one can ask a question: what really are **S**-algebras? Let us take a closer look at their definition in [Conway 1971].

Definition 2.26 (**S**-algebra [Conway 1971])**.** *An* **S**-algebra *is a set S with three operations defined on it $\sum$, $\otimes$, and *, and two particular elements $\bar{0}$ and $\bar{1}$, such that:*

$S1.$ $\sum_{i\in I} \bar{0} = \bar{0}$;

$S2.$ $\sum_{i\in I} s_i = \sum_{j\in J} \left(\sum_{i\in I_j} s_i \right)$, $\bigcup_{j\in J} I_j = I$;

$S3.$ $\bar{1} \otimes s = s \otimes \bar{1} = s$;

$S4.$ $(r \otimes s) \otimes t = r \otimes (s \otimes t)$;

$S5.$ $\left(\sum_{i\in I} s_i \right) \otimes \left(\sum_{j\in J} s_j \right) = \sum_{(i,j)\in I\times J} (s_i \otimes s_j)$;

$S6.$ $s^* = \sum s^i$.

One can thus define the $\oplus$ operator to retrieve a semiring structure: $s_0 \oplus s_1 = \sum_{i\in\{0,1\}} s_i$. Taking a look at $S2$, not requesting the union to be disjoint seems to be an oversight. By fixing the definition adding this requirement, one can obtain the classical definition for *complete semirings*. Following the many previously mentioned authors, this has been my first belief. But this strong misunderstanding leads to many incoherences. Because the classical axioms must hold in **S**-algebras (acknowledged by Conway [1971, p. 28]), we deduce the validity of $a \oplus a = a$ by using C13. We recall Semirings 11 and 12 are complete but not idempotent. Now, if the structure satisfies $a \oplus a = a$, is it always the case for $\sum a = a$? Surely not because of Semiring 7 (aka. R_4). In fact, by analyzing Theorem 2 in [Conway 1971, Chapter 3] we observed it was admitted in the proof that idempotence extends to infinite sums. And this is precisely what we can deduce from a strict interpretation of the definition (not requiring the I_j to be all disjoint).

Property 2.27 (Finite and infinite idempotence)**.** *For any finite or infinite index set I the following holds for* ***S****-algebras:*

$$\sum_I a = a.$$

Proof. Let $I = \{1\}$ and $s_1 = a$. For any finite or infinite index set J with $I_j = \{1\}$ for all $j \in J$, we deduce using $S2$, $\sum_J a = a$. □

It is now clear that closed semirings of Definition 2.5 and the **S**-algebras are almost identical structures. The only difference lies in the fact infinite sums are only defined for countable sequences for closed semirings. We now study the shared properties of $\sum$ in both cases. Since $\sum$ is associative, commutative, and idempotent, it might as well be defined on finite or infinite subsets of S.

Property 2.28 ([Kozen 1990], Theorem 2 in [Conway 1971, Chapter 3]). *In closed semirings (resp. **S**-algebras),* $\sum A = \sup_{\leq} A$.

Proof. Upper bound: if $x \in A$, then $x \oplus \sum A = \sum A \cup \{x\} = \sum A$ thus $x \leq \sum A$. Supremum: if $x \leq y$ for all $x \in A$, then $x \oplus y = y$ for all $x \in A$, so

$$\begin{aligned}\left(\sum A\right) \oplus y &= \left(\sum_{x\in A} x\right) \oplus \left(\sum_{x\in A} y\right)\\ &= \sum_{x\in A} (x \oplus y)\\ &= \sum_{x\in A} y\\ &= y, \text{ then } \sum A \leq y.\end{aligned}$$

□

Property 2.29 (Theorem 3 in [Conway 1971, Chapter 3]). *In closed semirings (resp. **S**-algebras),* a^*b *is the least solution to* $x = ax \oplus b$.

Property 2.30 (Theorem 4 in [Conway 1971, Chapter 3]). *If* $A = \begin{bmatrix} B & C \\ D & E \end{bmatrix}$ *is a matrix over a closed semiring (resp. **S**-algebra) then* $A^* = \begin{bmatrix} (B \oplus CE^*D)^* & B^*C\Delta^* \\ \Delta^*BD^* & \Delta \end{bmatrix}$ *with* $\Delta = (E \oplus DB^*C)^*$.

Let us now make a short discussion about the slight differences between the first two algorithms. Algorithm 2 working over closed semirings computes $(\bar{1}_{S_n} \oplus A)^+$ the transitive (but not reflexive) closure of the matrix $\bar{1}_{S_n} \oplus A$, whereas Algorithm 3 working over star semirings computes $\bar{1}_{S_n} \oplus A^+ = A^*$. This is due to the fact that because $C13^\circ$ holds for closed semirings and closed semirings are Conway semirings, then $(\bar{1}_{S_n} \oplus A)^+ = (\bar{1}_{S_n} \oplus A)^* = A^*$ follows from Property 2.17.

Semiring 13 (Continuous version of Semiring 7). *The semiring* $(\{0, 1, a, \infty\}, +, \cdot, 0, 1)$ *with* $0 \leq 1 \leq a \leq \infty$, $a \cdot a = a$, $0^* = 1$, $1^* = 1$, $a^* = a$, *and* $\infty^* = \infty$.

2.2.4. Continuity: Constraining to Least Solutions

In this section we review the notions of *continuity* and *finitariness* from [Goldstern 2002; Karner 1992; Krob 1988]. We also link to other resources in the literature where equivalent definitions for *continuity* have been provided such as in [Kuich 1997, Theorem 2.3].

As we said before, we want to be able to use non-idempotent semirings in our settings. Thus, we need to come up with a new approach preserving Properties 2.28, 2.29, and 2.30. Restricting to least solutions of equational systems impose to have a notion of ordering inside the semiring. Structures proposed in the literature intersect both notions of *complete* and *ordered semirings*.

Definition 2.31 (Ordered semiring [Goldstern 2002])**.** *An* ordered semiring *is a semiring* $(S, \oplus, \otimes, \bar{0}, \bar{1}, \preceq)$ *where* $\preceq$ *is a partial order,* $\oplus$ *and* $\otimes$ *are monotone in each argument, and* $\bar{0}$ *is the least element.*

An *embedding of ordered semirings* is a semiring embedding ϕ preserving the additional structure of the order $\preceq$: $a \preceq b \implies \phi(a) \preceq \phi(b)$. In Definition 2.31 the order is *positive*: $\bar{0} \preceq a$ for all $a \in S$. There also exist orders on semirings which are not positive as it is sometimes more convenient to work with a reversed order [Mohri 2002]. It is clear from the definition that every positively ordered semiring is zero-sum free, but the reverse is not true (see Example 2.2 from [Karner 1992]). In order to simplify things, Lemma 2.32 permits us to only consider the natural order $\leq$ when dealing with ordered semirings.

Lemma 2.32 ([Goldstern 2002])**.** *Let* $(S, \oplus, \otimes, \bar{0}, \bar{1})$ *be a semiring, the following are equivalent:*

1. *S can be embedded into an ordered semiring;*
2. *there exists a partial order on S making S an ordered semiring;*
3. *the natural pre-order is antisymmetric (i.e. a partial order);*
4. *for all $s, x, y \in S$: $s \oplus x \oplus y = s$ implies $s \oplus x = s$.*

Definition 2.33 (Finitary semiring [Goldstern 2002])**.** *A complete ordered semiring* $(S, \oplus, \otimes, \bar{0}, \bar{1}, \preceq)$ *is called* finitary *(resp.* ω-finitary*) when for every (resp. at most countable) index set I:*

$$\bigoplus_{i \in I} a_i = \sup_{\preceq} \left\{ \bigoplus_{i \subseteq F} a_i \mid F \subseteq I \text{ finite} \right\}. \tag{2.3}$$

Definition 2.34 (Continuous semiring [Karner 1992])**.** *A semiring is called* continuous *(resp.* ω-continuous*) when it is finitary (resp. ω-finitary) with respect to the natural order* $\leq$*.*

There are various ways of introducing continuous semirings in the literature. In [Droste et al. 2009; Krob 1988] the approach is to consider first *continuous monoids* (*complete* and *naturally ordered monoids* satisfying Equation 2.3) and to request the semiring to be as well complete for the infinite summation operator of the monoid (i.e., asking for $\otimes$ to be bicontinuous w.r.t $\sum$). There are also many equivalent alternative definitions for (ω-)continuous semirings, refer to [Kuich 1997, Theorem 2.3] for a compilation and [Karner 1992, Proposition 5.6] for a characterization of (ω-)continuous not mentionning any order.

The provenance semiring framework [Green et al. 2007] seems to follow a nomenclatura for *complete partial order* (CPO) where *complete* (resp. *ω-complete*) means every chain (resp. countable chain) has an upper bound. It is similar to the approaches from [Droste et al. 2009; Krob 1988] for defining continuous semirings.

As we wanted, the analogous of Properties 2.28, 2.29, and 2.30 also hold for continuous structures: Property 2.28 turns out to be already encompassed by the definition of

continuity, Property 2.35 extends the result of Property 2.29 to finitary semirings, and from [Droste et al. 2009, Theorem 3.4], we learn that each *complete star semiring* satisfies the Conway's equations, thus, Property 2.30 can be generalized to (ω-)continuous semirings (Property 2.36).

Property 2.35 (Proposition 5.3 in [Karner 1992])**.** *In a finitary semiring, a^*b is the least solution (w.r.t the order $\preceq$) to $x = ax \oplus b$.*

Property 2.36 (Matrices over continuous semirings)**.** *If $A = \begin{bmatrix} B & C \\ D & E \end{bmatrix}$ is a matrix over a continuous semiring then $A^* = \begin{bmatrix} (B \oplus CE^*D)^* & B^*C\Delta^* \\ \Delta^*BD^* & \Delta \end{bmatrix}$ with $\Delta = (E \oplus DB^*C)^*$.*

In fact, idempotent semirings always have exactly one partial order: the natural order [Karner 1992]. Thus, for idempotent semirings, finitariness coincides with continuity and so, **S**-algebras and *closed semirings* are continuous (resp. ω-continuous).

We finally end this survey with some fundamental remarks.

Theorem 2.37 ([Goldstern 2002])**.** *Let $(S, \oplus, \otimes, \bar{0}, \bar{1}, \preceq)$ be an ordered semiring. Then there is a finitary (complete ordered) semiring $\bar{S}$ such that S is embedded into $\bar{S}$ as an ordered semiring.*

We have voluntarily eluded the many different notions of limits in semirings such as *d-complete*, *l-complete*, and *t-complete* as long as only *continuous semirings* have been studied for provenance annotations. Those notions can notably be found in [Goldstern 2002; Krob 1987; Krob 1988]. An extensive study with a schematic overview of how they compare with each other and with *complete* and *continuous semirings* can be found in [Karner 1992]. We now discuss on the assumption we can *up-to-embedding* always assume working on a continuous structure.

Important note. The original version of Theorem 2.37 is in fact much stronger, but we have not introduced the vocabulary from *category theory* on which the theorem statement relies on to fully reproduce it there. It additionally mentions, every embedding of S into a finitary semiring T eventually *extends* the embedding of S into $\bar{S}$. It is worth recalling a finitary semiring need not be necessarily continuous (e.g., [Karner 1992, Fact 5.1]). Moreover, the proof of Theorem 2.37 constructs $\bar{S}$ as a quotient of a continuous structure. We believe it could be possible to adapt the proof to turn the finitary structure of $\bar{S}$ into a continuous one. We will come back to this open question in Chapter 12.2.

2.2.5. Summary

Semirings 7, 10, and 13 examplified how given the same semiring we can modify the definition of the * operator to turn the semiring into a complete star structure with all sums over an infinite number of non-zero elements being ∞ (resp. to a continuous structure fixing the infinite sum to be the supremum of the sequence).

Table 2.1 provides a visual summary of the structures names occuring in the literature linked to their unified nomenclure name we choosed in this manuscript. Finally,

Table 2.1.: Summary of the structure names occuring in the literature.

Book or paper	Name of their structure	Our name in this manuscript
Mehlhorn [1984]	closed semiring	ω-complete star semiring
Aho et al. [1974], Kozen [1990], and Cormen et al. [1989]	closed semiring	ω-continuous idemp. star semiring *or* closed semiring
Conway [1971]	**S**-algebra	continuous idemp. star semiring
Eilenberg [1974]	complete semiring	complete semiring
Krob [1987]	(ω-)complete semiring	(ω-)complete semiring
Krob [1988]	(ω-)continuous semiring	(ω-)continuous semiring
Droste et al. [2009]	complete (star) semiring continuous semiring	complete (star) semiring continuous semiring
Kuich [1997]	(ω-)continuous semiring	(ω-)continuous semiring
Abdali et al. [1985] and Abdali [1994]	*-semiring	star semiring
Lehmann [1977]	closed semiring	star semiring
Goldstern [2002]	finitary semiring	finitary semiring
Karner [1992]	finitary semiring	finitary semiring

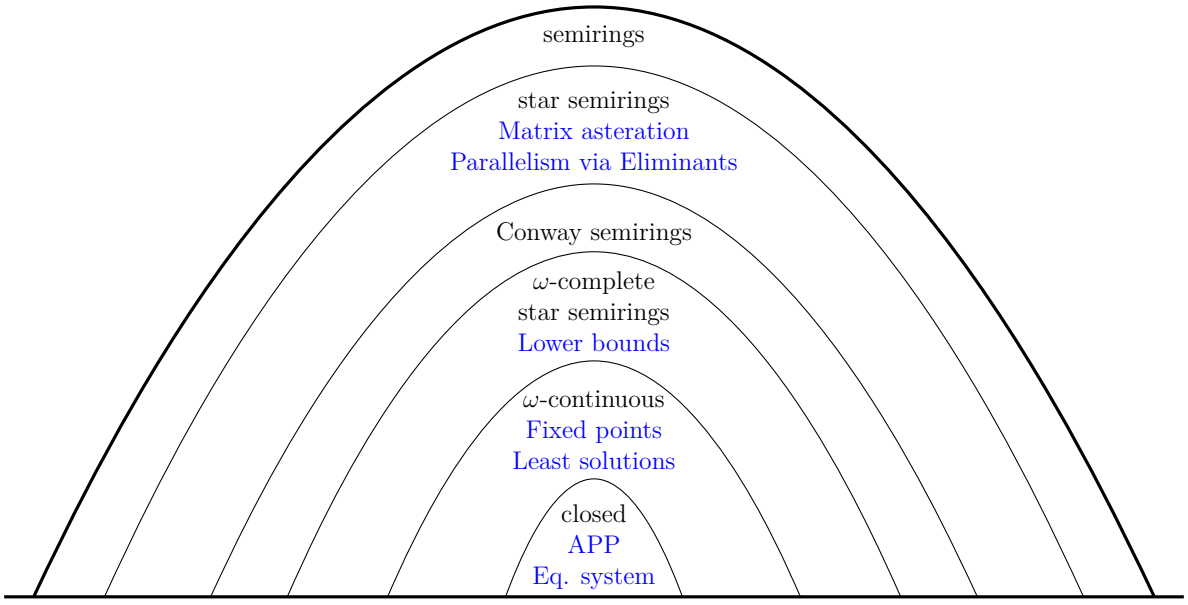


Figure 2.1.: Summary of the theoretical structures along with their key properties or usages.

Figure 2.1 indicates what are the key properties of the structures we studied in this chapter.

2.3. Theoretical Bounds

We have seen so far computing the asteration of a semiring-valued matrix is key for solving a vast amount of different problems. In particular, solving a system of linear equations over a star semiring consists in computing the asterate of the coefficient matrix [Abdali 1994; Minoux et al. 2008]. The *Kleene method* (Algorithms 2 and 3) provides a cubic time method for doing so. We now introduce in Section 2.3.1 a fundamental result relating the complexity of matrix multiplication to the complexity of the asteration of a matrix, and in Section 2.3.2 we provide a lower-bound on the *monotone complexity* of matrix multiplication. Thanks to Section 2.2, we deduce by the end of the discussion a nice *up-to-embedding* characterization for the semirings on which this limitation holds.

2.3.1. On the Complexities of Matrix Multiplication and Matrix Asteration

Let A and B two $n \times n$ matrices over an ω-complete star semiring S. For the rest of the section the semiring S will be assumed to be ω-complete. The following lemma shows how to encode the matrix multiplication of A and B into the asteration of a third, carefully designed, matrix C.

Lemma 2.38 ([Mehlhorn 1984])**.** *Let us consider the following* $(3n \times 3n)$ *matrix* $C = \begin{bmatrix} \bar{0}_{S_n} & A & \bar{0}_{S_n} \\ \bar{0}_{S_n} & \bar{0}_{S_n} & B \\ \bar{0}_{S_n} & \bar{0}_{S_n} & \bar{0}_{S_n} \end{bmatrix}$*, then* $C^* = \begin{bmatrix} \bar{1}_{S_n} & A & AB \\ \bar{0}_{S_n} & \bar{1}_{S_n} & B \\ \bar{0}_{S_n} & \bar{0}_{S_n} & \bar{1}_{S_n} \end{bmatrix}$.

Proof. $C^0 = \begin{bmatrix} \bar{1}_{S_n} & \bar{0}_{S_n} & \bar{0}_{S_n} \\ \bar{0}_{S_n} & \bar{1}_{S_n} & \bar{0}_{S_n} \\ \bar{0}_{S_n} & \bar{0}_{S_n} & \bar{1}_{S_n} \end{bmatrix}$, $C^2 = \begin{bmatrix} \bar{0}_{S_n} & \bar{0}_{S_n} & AB \\ \bar{0}_{S_n} & \bar{0}_{S_n} & \bar{0}_{S_n} \\ \bar{0}_{S_n} & \bar{0}_{S_n} & \bar{0}_{S_n} \end{bmatrix}$, and for $k > 2$: $C^k = \bar{0}_{S_{3n}}$. □

Theorem 2.39 (Theorem 5.2 in [Mehlhorn 1984])**.** *If an algorithm computes the closure of a matrix using* $A(n)$ *semiring operations, then the multiplication of two* $n \times n$ *matrices can be computed with* $M(n) = \mathcal{O}(A(n))$ *semiring operations.*

When referring to *semiring operations*, we either mean $\oplus$, $\otimes$, or *. Quite surprisingly, the reverse also holds: we can reduce matrix asteration to many matrix multiplications under the same global amount of semiring operations performed.

Lemma 2.40 ([Mehlhorn 1984])**.** *If* $A = \begin{bmatrix} B & C \\ D & E \end{bmatrix}$ *is an* $n \times n$ *matrix over an* ω*-complete star semiring then* $A^* = \begin{bmatrix} F & G \\ H & K \end{bmatrix}$ *with* $F = (B \oplus (CE^*D))^*$, $G = FCE^*$, $H = E^*DF$, *and* $K = E^* \oplus (E^*DFCE^*)$*. All of which are* $\frac{n}{2} \times \frac{n}{2}$ *square matrices. Moreover,* A^* *can be computed using* 2 *additions,* 2 *closures, and* 6 *multiplications over* $\frac{n}{2} \times \frac{n}{2}$ *matrices.*

Proof. The rough ideas for obtaining such identities are explained in the book. Formally, they can be deduced from the classical axioms universally valid in ω-complete star semirings. The following *straight-line* program computes F, G, H, and K by reusing common subexpressions:

$$\begin{aligned} T_1 &\leftarrow E^* \\ T_2 &\leftarrow CT_1 \\ F &\leftarrow (B \oplus (T_2 D))^* \\ G &\leftarrow FT_2 \\ T_3 &\leftarrow T_1 D \\ H &\leftarrow T_3 F \\ K &\leftarrow T_1 \oplus (T_3 G) \end{aligned}$$

□

Theorem 2.41 (Theorem 5.3 in [Mehlhorn 1984])**.** *If the product of two $n \times n$ matrices can be computed with $M(n)$ additions and multiplications of semiring elements, then the closure of an $n \times n$ matrix can be computed with $A(n) = \mathcal{O}(M(n))$ semiring operations.*

Theorem 2.39 together with Theorem 2.41 establish that matrix multiplication and matrix asteration over an ω-complete star semiring have the same order of complexity. To establish lower bounds related to matrix asteration under our semiring framework, we can now focus on the complexity for matrix multiplication.

2.3.2. Lower Bounds

The proof of Lemma 2.40 has already made use of *straight-line* programs [Mehlhorn 1984]. Those programs are restricted programs with no *loop* and no *conditional* statement. All the previous algorithms can be seen as such kind of programs restricting to fixed input size and duplicating code inside iteration loops. The structure of a straight-line program is in one-to-one correspondance with a *circuit representation* in which gates indicate performed operations and leaves the initial data.

A *monotone* straight-line program restricts to *monotone operations*: in our case $\oplus$ and $\otimes$ for any semiring. This rules out for example the *negation* operator when dealing with boolean values, *negative* values in the reals, or more generally the *inverse* operator in any ring.

Theorem 2.42 (Theorem 5.4 in [Mehlhorn 1984])**.** *Let $(S, \oplus, \otimes, \bar{0}, \bar{1})$ be a zero-sum free semiring. Any straight-line program which computes the (r, p, q) matrix product of two matrices over S using only $\oplus$ and $\otimes$ has at least $r \cdot p \cdot q$ multiplications and $r \cdot (p-1) \cdot q$ additions. Moreover the naïve definition of such product is optimal.*

Original version of Theorem 2.42 requires *semirings of characteristic zero*: $\bar{1} \oplus \cdots \oplus \bar{1} \neq \bar{0}$ for any number of one to be added, instead of zero-sum free semirings. It is clear that any zero-sum free semiring is of characteristic zero.

3. Using Semirings to Express Provenance

The considerable generalization power associated with semiring structures found a notable application in database systems. Many different notions of annotated relations concerning incomplete and probabilistic databases [Imieliński et al. 1984; Fuhr et al. 1997], lineage [Cui et al. 2000], and bag semantics have all been shown to be expressible using commutative semirings [Green et al. 2007].

In the following, we provide an overview of the seminal paper of [Green et al. 2007], and discuss on the key ideas introduced there. It starts with the definition of provenance for *positive relational algebra queries*, and extends this to *Datalog queries*. Here, we emphasize Datalog provenance, as it was one of the main focuses of this thesis, but we will also shortly touch on the non-recursive case. We then focus on the notion of *provenance semirings* in [Green et al. 2007], which allows for a symbolic representation of the provenance.

3.1. System of Fixpoint Equations

A *system of fixpoint equations* over an ω-continuous semiring S is a finite set of equations: $X_1 = f_1(X_1, X_2, \ldots, X_n), \ldots, X_n = f_n(X_1, X_2, \ldots, X_n)$, where $X_1, \ldots, X_n$ are variables and $f_1, \ldots, f_n$ are polynomials with coefficients in S. We extend the notion of natural order from semiring elements to tuples of semiring elements by simply considering the product order. We then have the following on solutions of a system of equations over an ω-continuous semiring:

Theorem 3.1 (Theorem 3.1 of [Kuich 1997])**.** *Every system of fixpoint equations* $\mathbf{X} = f(\mathbf{X})$ *over a commutative* ω*-continuous semiring has a least solution* $\mathsf{lfp}(f)$ *w.r.t.* $\leq$*, and* $\mathsf{lfp}(f)$ *is equal to the supremum of the Kleene sequence:* $\mathsf{lfp}(f) = \sup_{m \in \mathbb{N}} f^m(\bar{\mathbf{0}})$.

3.2. Datalog Provenance

Let us now recall our running example – Example 1.1 – from Section 1.2.1, but this time with labels (semiring values) over the EDB facts. The provenance for an IDB tuple is then the sum for each of its derivation trees, of the product of the provenance annotations associated to each leaf of the tree. This is exemplified in Figure 3.1 (an incremented version of Figure 1.1) featuring provenance annotations over the tropical semiring.

Definition 3.2 (Proof-theoretic definition for Datalog provenance [Green et al. 2007])**.** *Let $(S, \oplus, \otimes, \bar{0}, \bar{1})$ be a commutative ω-continuous semiring and q a Datalog program with output predicate T and such that all extensional facts $R(t')$ are annotated with an element of S, denoted as $\mathsf{prov}_R^q(t')$. Then the* provenance *of $T(t)$ for q, where $T(t)$ is in the output of q, is defined as:*

$$\mathsf{prov}_T^q(t) = \bigoplus_{\tau \text{ yields } t} \left(\bigotimes_{t' \in \text{ leaves}(\tau)} \mathsf{prov}_R^q(t') \right).$$

This definition is well-founded using any commutative semiring for union of conjunctive queries (i.e., non-recursive queries), but in case of recursivity, a tuple can have infinitely many derivation trees. In this case, we need to consider specific semirings to properly deal with those infinite sums. The choice made in [Green et al. 2007] for Definition 3.2 is to consider *commutative ω-continuous semirings* following formal language theory [Kuich 1997].

Example 3.3. *The* tropical semiring *is* $(\mathbb{R}^+ \cup \{\infty\}, \min, +, \infty, 0)$*. We show in Figure 3.1 an example Datalog program (right) with tropical semiring annotations on extensional facts, as well as the (only) two derivation trees of the fact* path(Paris, London) *along their weight. This witnesses that the provenance of* path(Paris, London) *is* $\min(1, 3) = 1$*.*

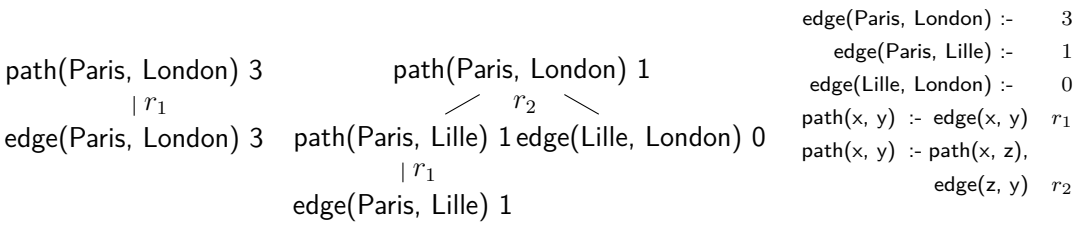


Figure 3.1.: Derivation trees along their weights for the fact path(Paris, London) using the transitive closure Datalog program over the tropical semiring with an EDB containing 3 facts

Since some tuples can have infinitely many derivations, hence the Datalog semantics given above cannot be used as an algorithm. As pointed out in [Green et al. 2007] it is possible instead to use a fixpoint-theoretic definition of the provenance of a Datalog query q: introduce a fresh variable for every possible *intensional* tuple (i.e., every possible *ground atom*), and produce for this variable an equation that reflects the *immediate consequence operator T_q – extensional* facts appearing as their semiring annotations in these equations. This yields a system of fixpoint equation f_q. The provenance of $T(t)$ for q is now simply the value of the variable corresponding to $T(t)$ in $\mathsf{lfp}(f_q)$. The fixpoint-theoretic definition directly yields an algorithm, albeit a very inefficient one because of the need of generating a rule for every intensional tuple. We investigate in Part III more efficient algorithms, for specific types of semirings.

Because we have spent a lot of time in Chapter 2 discussing on infinite sums and continuous operators in semirings, we will give additional insights on their choice to use commutative ω-continuous semirings. We extend the short discussion in [Green et al. 2007] about the decision process leading to use such semirings for expressing provenance for Datalog programs. It is clear that closed semirings (see Section 2.2.3) are too limited, due to the necessity of idempotence, and it is also quite natural to focus only on countable sums as we will never deal with non countable patterns. The interesting question is whether ω-complete semirings can suffice for having a proper definition for provenance.

Let us consider the Datalog program r outlined as Algorithm 4 over Semiring 10 denoted S in the following. This semiring is ω-complete but not ω-continuous[1]. We consider an instance $\mathbf{I}$ containing only $E(0)$ with $\mathsf{prov}^r_E(0) = a$ over S. The system of fixpoint equations associated to r and $\mathbf{I}$ is $x = a \oplus ax$ with x the target objective for $\mathsf{prov}^r_I(0)$. It is easily seen, that for both definitions of the provenance, $\mathsf{prov}^r_I(0) = \infty$. But the fixpoint computation converges towards a because $a = a \oplus aa$ holds true in S. The reason for this to fail comes from the fact that idempotence does not extend to infinite idempotence ($a \oplus a = a$ but $\sum a = \infty$). We made a case in Chapter 12 for not using a least fixpoint semantics for Datalog provenance.

Algorithm 4 Datalog program r with $\mathsf{EDB} = \{E\}$ and $\mathsf{IDB} = \{I\}$

```
1: I(x) :- E(x).
2: I(x) :- I(x), E(x).
```

3.3. Provenance Semirings

We now turn our focus on the notion of *provenance semirings* [Green et al. 2007]. Given a query language (e.g., positive relational algebra or Datalog) and a class of semirings on which the definition of the provenance for such query language is well-founded – for instance commutative semirings (resp. commutative ω-continuous semirings) – the *semiring of provenance* is an abstract structure (i.e., a *free algebra*) allowing for a symbolic representation of the provenance and permitting to factorize computations.

Definition 3.4 (Positive algebra provenance semiring [Green et al. 2007])**.** *Let X be the set of tuple ids of a (usual) database instance $\boldsymbol{I}$. The* positive algebra provenance semiring *for $\boldsymbol{I}$ is the semiring of polynomials with variables (a.k.a. indeterminates) from X and coefficients from $\mathbb{N}$, with the operations defined as usual:* $(\mathbb{N}[X], +, \cdot, 0, 1)$.

Property 3.5 (Free algebraic structure [Green et al. 2007])**.** *Let S be a commutative semiring and X a set of variables. For any valuation $\nu : X \to K$ there exists a unique homomorphism of semirings $\mathsf{Eval}_\nu : \mathbb{N}[X] \to S$ such that for the one-variable monomials we have $\mathsf{Eval}_\nu(x) = \nu(x)$.*

[1] An alternative definition with $1^* \coloneqq 1$ and $a^* \coloneqq a$ makes it continuous (see Semiring 13).

Theorem 3.6 ([Green et al. 2007]). *The provenance for any $\mathcal{RA}^+$ query q* factors through $\mathbb{N}[X]$*: $q(R) = \mathsf{Eval}_\nu \circ q(\bar{R})$.*

Here, R is a relation with provenance indication over S, and $\bar{R}$ is its *abstractly tagged* version, i.e., $\bar{R}$ is a particular kind of $\mathbb{N}[X]$-relation where each tuple is tagged by its own id. Whereas for $\mathcal{RA}^+$ queries [Green et al. 2007] polynomials suffice, in case of Datalog with the possibility of infinite sums to appear, we need to consider *the semiring of formal power series* $\mathbb{N}^\infty[[X]]$ [Kuich 1997]. The two types of infinite sums – an infinite number of *distincts* monomials and an infinite copy of the *same* monomial – are encompassed within $\mathbb{N}^\infty[[X]]$. We now recall from [Green et al. 2007] the analogous of Definition 3.4, Property 3.5, and Theorem 3.6, but in case of Datalog and formal power series.

Definition 3.7 (Datalog provenance semiring [Green et al. 2007]). *Let X be the set of tuple ids of a database instance $\boldsymbol{I}$. The* datalog provenance semiring *for $\boldsymbol{I}$ is the commutative ω-continuous semiring of formal power series $\mathbb{N}^\infty[[X]]$.*

Property 3.8 (Free algebraic structure [Green et al. 2007]). *Let S be a commutative ω-continuous semiring and X a set of variables. For any valuation $\nu : X \to K$ there exists a unique ω-continuous homomorphism of semirings $\mathsf{Eval}_\nu : \mathbb{N}^\infty[[X]] \to K$ such that for the one-variable monomials we have $\mathsf{Eval}_\nu(x) = \nu(x)$.*

Theorem 3.9 ([Green et al. 2007]). *The semantics of Datalog provenance for any commutative ω-continuous semiring* factors through *the provenance semirings of formal power series.*

3.4. Free Path-Related Semiring

Given a graph $G = (V, E)$ let us consider the *free idempotent semiring generated by the arcs of G* as the semiring $S_E = (2^{P(G)}, \cup, \cdot, \emptyset, \epsilon)$ and the *free semiring generated by the arcs of G* [Rote 1990] as the multiset semiring $(\mathbb{N}^{P(G)}, \cup, \cdot, \emptyset, \epsilon)$. To further simplify the notation we treat only the former case, all definitions extending to multiset semantics in the usual way. ϵ denotes the empty path, $\cup$ the union of paths, and

$$E_1 \cdot E_2 = \{\pi_1\pi_2 \mid \pi_1 \in E_1,\ \pi_2 \in E_2,\ n[\pi_1] = p[\pi_2]\}.$$

The * operator is defined as expected using both $\cup$ and $\cdot$. The definition is sound: $P(G)$ contains only valid path in G and the definition of $\cdot$ prohibits creating invalid paths. We identify each edge e of G with the set $\{e\} \in 2^{P(G)}$ and for any valuation of the edges in a closed semiring (aka. ω-continuous idempotent semiring) S there is a unique homomorphism ϕ (also called Eval in the line of [Green et al. 2007]) from S_E to S having $\phi(\{e\}) = \lambda(e)$. The computation of aggregated labels between vertices s and t factor through ϕ, i.e., $l_{st} = \phi(P_{st}(G))$. The use of multisets permits to relax the condition of idempotence and to work on ω-continuous semirings.

This construction has been pointed out by Rote [1990] to share strong similarities with the notion of *path expressions* (i.e., regular expressions over the alphabet E describing

the set of paths between two vertices of the graph) introduced by Tarjan [1981]. ϕ can be computed by structural induction over a given path expression for $P_{st}(G)$, thus allowing to work on a finite representation of the (possibly) infinite set of paths.

Part II.

A Provenance Model for Graph Databases

Prior to this PhD, we have adapted during a Master's internship the framework of provenance semirings – algebraic structures that can capture different forms of provenance – already in use for relational databases and Datalog [Green et al. 2007] to focus on *annotated graph databases*. We have then proposed three generalizations of existing graph algorithms to compute the *provenance* of regular path queries (RPQs) under this model. Each algorithm yields a different trade-off between time complexity and generality, as each requires different properties over the semiring. Together, these algorithms already covered a large class of semirings commonly used for provenance (top-k, security, etc.). Experimental results conducted at the time strongly suggested these approaches were complementary and practical for various kinds of provenance indications, even on a relatively large transport network. This preliminary work, summarised above, has been published in TaPP 2018 [Ramusat et al. 2018].

Such encouraging results convinced us to dig into the practical and theoretical details of such framework and this thesis has eventually been the opportunity to pursue investigations on this research topic.

The first part of the thesis has been dedicated to investigate the efficient computation of the provenance of rich queries over graph databases. We made clear how semiring-based provenance annotations enrich the expressiveness of routing queries over graphs. We notably addressed the limitations of the three algorithms by introducing a new one, partially bridging a complexity and expressiveness gap and adding to the algorithmic toolkit for solving this problem. Importantly, we provided a comprehensive taxonomy of semirings and corresponding algorithms, establishing which practical approaches are needed in different cases. We implemented and comprehensively evaluated several practical applications of the problem (e.g., shortest distances, top-k shortest distances, Boolean or integer path features), each corresponding to a specific semiring and algorithm, that depends on the properties of the semiring. On several real-world and synthetic graph datasets, we showed the fact that the algorithms we propose exhibit large practical benefits for processing rich graph queries. This research has been published at EDBT 2021 [Ramusat et al. 2021b].

This part is organized as follows. We start with Chapter 4 introducing the provenance framework for graph databases. Essentially based on [Ramusat et al. 2018], but augmented with complementary results which were omitted there due to lack of space. We also give an alternative semiring-based statement for some of the structural results, to provide a more algebraic oriented view of the introduced constructions. Chapter 5 recalls basic provenance-aware algorithms and Chapter 6 introduces the new algorithm we designed during this PhD. Chapter 7 focuses on the experiments based on our implementation made freely available at [Maniu et al. 2020]. Finally, we provide the taxonomy we established in [Ramusat et al. 2021b] and also discuss on the theoretical limitations of our framework. We also explain the rough ideas that led us to turn our attention to Datalog provenance that will be the focus of Part III.

4. Introduction to the Model

This chapter is built upon the notation already introduced in Chapter 1.

4.1. Graph Databases with Provenance Indication

Given a finite alphabet Σ, a *graph database* G over Σ is a pair (V, E), where V is a finite set of node ids (i.e., $V \subseteq \mathcal{V}$) and $E \subseteq V \times \Sigma \times V$. Thus, each edge in G is a triple $(v, a, v') \in V \times \Sigma \times V$, whose interpretation is an a-labeled edge from v to v' in G. When Σ is clear from the context, we shall speak simply of a graph database. A *graph database with provenance indication* (V, E, w) over S is a graph database (V, E) together with a *weight function*, w: $E \to S$ for $(S, \oplus, \otimes, \bar{0}, \bar{1})$ a semiring. The weight function w can also be extended to paths by defining the weight of a path as the result of the $\otimes$-multiplication of the weights of its constituent edges: $w[\pi] \coloneqq \bigotimes_{i=1}^{k} w[e_i]$ and can in fact be extended to any finite set of paths by $w[\bigcup_{i=1}^{n} \pi_i] \coloneqq \bigoplus_{i=1}^{n} w[\pi_i]$. Given an edge $e \in E$, we denote by $\rho(e)$ its label. The label $\rho(\pi) \in \Sigma^*$ of a path π is defined as the concatenation of its labels:

$$\rho(\pi) = \rho(e_1)\rho(e_2)\cdots\rho(e_{k-1})\rho(e_k).$$

4.2. Regular Path Queries (RPQs)

RPQs [Barceló 2013] provide a way to query a graph database using its topology and constitue the basic navigational mechanism for graph databases. An RPQ Q has the form $\text{RPQ}(x, y) \coloneqq (x, L_Q, y)$ where L_Q is a regular language, with the semantics that Q is satisfied iff there exists at least a path having the sequence of its labels' edges in L_Q. From now on, we use the shorthand L_Q for the RPQ Q. We now introduce the notion of *provenance of an RPQ* based on the notion of provenance semiring for the Datalog language [Green et al. 2007].

Definition 4.1 (Provenance of an RPQ [Ramusat et al. 2018])**.** *For a graph database G with provenance indication over S, such a query Q associates to each pair (x, y) of nodes an element $\mathsf{prov}_S^Q(G)(x, y)$ of the semiring called the* provenance of the RPQ Q between x and y *defined as*

$$\mathsf{prov}_S^Q(G)(x, y) \coloneqq \bigoplus_{\substack{\pi \in P_{xy}(G),\\ \rho(\pi) \in L_Q}} w[\pi].$$

Definition 4.1 involves an infinite sum, we will explain later how to deal with it. We will consider and name many variants of this problem. Given two vertices s and t, the *single-pair provenance* problem computes the provenance between s and t. Given a vertex s, the *single-source provenance* problem computes the provenance between s and each vertex of the graph. The *full provenance*, or *all-pairs provenance* problem computes the provenance between each pair of vertices of the graph.

4.3. Graph Transformation

We now show how to simplify the computation of the provenance of an RPQ whose language is non-trivial. That is, given a graph database G over S and an RPQ Q, we want to compute the function $\mathsf{prov}_S^Q(G)$. For this, we suppose we have a complete deterministic automaton $\mathcal{A}_Q$ to represent L_Q. We reduce the problem of computing the provenance of a query Q over labeled graph G to the problem of computing shortest distances over an unlabeled graph G'.

Let $\mathcal{A}_Q = (K, \Delta, k_0, F)$ where $\Delta \subseteq K \times \Sigma \times K$ is the transition relation, $k_0 \in K$ is the initial state and $F \subseteq K$ the set of final states. If $(k, a, k') \in \Delta$ we note $k \xrightarrow{a} k'$. In order to take into account path restrictions we will transform the graph G by taking the *product graph* $P_{G\times\mathcal{A}_Q}$ between itself and the automaton $\mathcal{A}_Q$. The product graph is defined as $P_{G\times\mathcal{A}_Q} = (V \times K, E')$ over $\Sigma' = \{\star\}$, where

$$E' = \{((v,k), \star, (v',k')) \mid \exists a \in \Sigma, (v,a,v') \in E \wedge k \xrightarrow{a} k'\},$$

and the weight function, $w'\colon E' \to S$:

$$e = ((v,k), \star, (v',k')) \mapsto \bigoplus_{\substack{a\in\Sigma,\\ k\xrightarrow{a}k'}} w[(v,a,v')]^1.$$

We now show that we can compute the provenance of the query Q, between x and y in the graph G by computing the provenance of the reachability query R (the query with underlying language $L_{\mathsf{R}} = \Sigma^*$) over $P_{G\times\mathcal{A}_Q}$ between all pairs having source (x, k_0) and a target (y, k_F) for $k_F \in F$ and performing a last step consisting in summing each of them. The next lemma characterizes the kind of paths from the source in $P_{G\times\mathcal{A}_Q}$.

Lemma 4.2 ([Ramusat et al. 2018])**.** *If $\mathcal{A}_Q$ is a complete and deterministic automaton, there is a one-to-one mapping c between paths from x in G onto paths from (x, k_0) in $P_{G\times\mathcal{A}_Q}$. Moreover for π with $p[\pi] = x$ and $n[\pi] = y$ we have $n'[c(\pi)] = (y, k)$ with $k_0 \xrightarrow{\rho(\pi)} k$ and $w[\pi] = w'[c(\pi)]$ where n' is the destination for edges in $P_{G\times\mathcal{A}_Q}$ and w' the weight function of this graph.*

Proof. We construct $c(\pi)$ by induction over the length of the path π:

[1] Or $P_{G\times\mathcal{A}_Q}$ can be equivalently defined as a multigraph to keep a separate edge for each label.

Base case, $\pi_1 = e_1 = (x, a, x_1)$ then $c(\pi_1) = e'_1 \coloneqq ((x, k_0), \star, (x_1, k_1))$ with k_1 such that $k_0 \xrightarrow{\rho(e_1)} k_1$ which exists because the automaton is complete and is uniquely defined by determinism of the automaton.

Induction step, $\pi_n = \pi_{n-1}e_n$, then $c(\pi_n) \coloneqq c(\pi_{n-1})e'_n$, with $e'_n = (n'[c(\pi_{n-1})], \star, (y, k_n))$ and k_n such that $k_{n-1} \xrightarrow{\rho(e_n)} k_n$ which exists because the automaton is complete and is uniquely defined by determinism of the automaton.

Finally, we obtain a one-to-one function c onto paths from (x, k_0) in $P_{G\times\mathcal{A}_Q}$ and by construction of the graph $n'[c(\pi)] = (y, k)$ with $k_0 \xrightarrow{\rho(\pi)} k$ and $w[\pi] = w'[c(\pi)]$ holds for each π. □

Using the above lemma, we can show:

Theorem 4.3 ([Ramusat et al. 2018])**.** *The following equality holds:*

$$\mathsf{prov}^Q_S(G)(x, y) = \bigoplus_{k_F\in F} \mathsf{prov}^{\mathsf{R}}_S(P_{G\times\mathcal{A}_Q})((x, k_0), (y, k_F)).$$

Proof. Using the definition of provenance and Lemma 4.2 we obtain successively:

$$\begin{aligned}
\mathsf{prov}^Q_S(G)(x, y) &= \bigoplus_{\substack{\pi\in P_{xy}(G),\\ \rho(\pi)\in L_Q}} w[\pi] \\
&= \bigoplus_{\substack{\pi\in P_{xy}(G),\\ \rho(\pi)\in L_Q}} w'[c(\pi)] \text{ by properties of } c \\
&= \bigoplus_{k_F\in F} \bigoplus_{\pi\in P_{(x,k_0)(y,k_F)}(P_{G\times\mathcal{A}_Q})} w'[\pi] \\
&\quad \text{because for each } \pi,\ n'[c(\pi)] = (y, k) \text{ with } k_0 \xrightarrow{\rho(\pi)} k \\
&= \bigoplus_{k_F\in F} \mathsf{prov}^{\mathsf{R}}_S(P_{G\times\mathcal{A}_Q})((x, k_0), (y, k_F))
\end{aligned}$$

□

Let us now take a look at the two constraints over the automaton $\mathcal{A}_Q$ (i.e., completeness and determinism). Both are required by Lemma 4.2 in order to ensure a one-to-one correspondence from paths in the initial graph to paths in the product graph. Nevertheless, no one is interested in vertices in the product graph with a second component not an accessible or not a coaccessible state of the automaton (see Remark 4.4). In practice, for efficiency reasons, either the product graph will be computed *on-the-fly*, or those non-relevant vertices will be subsequently removed without any impact on provenance computation. Property 4.5 moreover entails the fact the automaton need not be necessarily complete for the construction from Lemma 4.2 to work.

Remark 4.4. *If $k \in K$ is not a coaccessible state in $\mathcal{A}_Q$, then for any $v, v' \in V^2$, $k_F \in F$ there is no path from (v, k) to (v', k_F) in $P_{G\times\mathcal{A}_Q}$.*

Property 4.5. *Removing vertices (v, k) in $P_{G\times\mathcal{A}_Q}$, with $v \in V$ and $k \in K$ not a coaccessible state in $\mathcal{A}_Q$, does not affect the provenance of the initial query Q.*

Proof. Using the equality (already derived in the proof of Theorem 4.3):

$$\mathsf{prov}_S^Q(G)(x,y) = \bigoplus_{k_F\in F} \bigoplus_{\pi\in P_{(x,k_0)(y,k_F)}(P_{G\times\mathcal{A}_Q})} w'[\pi],$$

we observe no such (v, k) belongs to any path contributing to the right hand side sum. □

We have seen so far that a deterministic automaton is required in the general case, but this restriction can be obviated when the semiring of interest is idempotent. This is the purpose of the following Property 4.7.

Lemma 4.6. *If $\mathcal{A}_Q$ is a complete and non-deterministic automaton, given a path $\pi = e_1e_2\cdots e_n$ from x to y in G, where π_p denote the longest proper prefix of π (i.e. $e_1e_2\cdots e_{n-1}$) and itself in case of empty path, each path π' in the set of paths:*

$$\Pi_k(\pi) = \{\pi' \mid p'[\pi'] = (x,k_0),\ n'[\pi'] = (y,k),\ k_0 \xrightarrow{\rho(\pi_p)} k' \xrightarrow{\rho(e_n)} k,\ \text{and } \pi'_p \in \Pi'_k(\pi_p)\}$$

verifies $w'[\pi'] = w[\pi]$, therefore $w'[\Pi_k(\pi)] = w[\pi]$ if $\Pi_k(\pi)$ is not empty, $\bar{0}$ either.

Proof. The proof is similar to proof of Lemma 4.2 but, due to non-determinism, instead of defining $c(\pi)$ we construct the set $\Pi(\pi)$ corresponding to each possible run in the automaton. □

Property 4.7. *The following equality holds when S is idempotent and $\mathcal{A}_Q$ an NFA:*

$$\mathsf{prov}_S^Q(G)(x,y) = \bigoplus_{k_F\in F} \mathsf{prov}_S^{\mathsf{R}}(P_{G\times\mathcal{A}_Q})((x,k_0),(y,k_F)).$$

Proof. Using the definition of provenance and Lemma 4.6 we obtain successively:

$$\begin{aligned}
\mathsf{prov}_S^Q(G)(x,y) &= \bigoplus_{\substack{\pi\in P_{xy}(G),\\ \rho(\pi)\in L_Q}} w[\pi] \\
&= \bigoplus_{\substack{\pi\in P_{xy}(G),\\ \rho(\pi)\in L_Q}} \bigoplus_{k_F\in F} w'[\Pi_{k_F}(\pi)] \text{ by Lemma 4.6} \\
&= \bigoplus_{k_F\in F} \bigoplus_{\pi\in P_{(x,k_0)(y,k_F)}(P_{G\times\mathcal{A}_Q})} w'[\pi] \\
&= \bigoplus_{k_F\in F} \mathsf{prov}_S^{\mathsf{R}}(P_{G\times\mathcal{A}_Q})((x,k_0),(y,k_F))
\end{aligned}$$

□

This construction generalizes the ideas from [Rote 1990] that were recalled in Section 2.1.2. We were using the semiring $(\{\emptyset, E, O, EO\}, \cup, \cdot, \emptyset, EO)$ with all edges labeled with O. An automaton over $\Sigma = \{a\}$ for counting modulo 2 has two states, say E and O. The carrier set of the above mentioned semiring is the powerset of the set of states of the parity automaton. It is clear that their construction is precisely what we obtain doing the product of the graph with the parity automaton when the semiring S is trivial (we forget about the weights).

4.4. Path Provenance and Its Semantics

We previously showed that computing the single-pair provenance of an RPQ could be reduced in polynomial time to the single-source provenance problem of the reachability query. Performing this construction on-the-fly – avoiding generation of inaccessible vertices – using an (usually quite small) automaton leads to an affordable overhead; even for large graphs as it has been showed experimentally in [Ramusat et al. 2018]. We will implicitly use this reduction from now on, and consequently also ignore edge labels and see a graph database as defined by its vertices, edges, and semiring weights. We now consider the subsequent definition of the *path provenance* problem.

Definition 4.8 (Path provenance [Ramusat et al. 2021b])**.** *Let G be a graph database with provenance indication over some semiring S. The* provenance between x and y*, for x and y two vertices of G is defined as the (possibly infinite) sum:*

$$\mathsf{prov}_S(G)(x,y) \coloneqq \mathsf{prov}^{\mathsf{R}}_S(G)(x,y) = w\left[P_{xy}(G)\right] = \bigoplus_{\pi \in P_{xy}(G)} w[\pi].$$

We have still not discussed about the conditions over the underlying semiring of use in Definitions 4.1 and 4.8 to make the (possibly infinite) sum over the provenances of all paths from the source vertex to the target vertex well-defined and semantically sound. We observed in [Ramusat 2019] that the only possible source of non-finiteness in the sum is due to cycles in the graph. Thus, we only need to be able to sum all the powers of a given semiring value. The first step to give an algebraic basis to the expressions we will manipulate when dealing with provenance values for graph databases is to consider *star semirings* (Definition 2.12 in Section 2.2.1). We additionally need the star operator to verify for all semiring element a: $a^* = \bigoplus_{n=0}^{\infty} a^n$ for some well-behaved infinitary sum operation $\bigoplus$ (namely, associativity, and distributivity of $\otimes$ over this infinitary sum operator). We have hopefully studied in the preliminaries this class of semirings, commonly known as *ω-complete star semirings* (Definition 2.19 in Section 2.2.2). Moreover, for semantic-based considerations, the semiring will usually be assumed to be ω-continuous. Most of the constructions and results in the literature focus on such structures, it is notably the case for both free path-related semirings in Section 3.4. In the next part of the thesis, where a graph database will be converted into a Datalog query, it will also be required to work on continuous structures. Nevertheless, we have seen in preliminaries this assumption does not restrict our model because application semirings generally either verify this property or can be embedded into larger continuous semirings.

Example 4.9. *We now provide example semirings with their associated semantics for graph provenance. Let G be a graph database over some semiring S, and s and t fixed source and target vertices in G. The provenance between s and t corresponds to the following notions, depending on the semiring S:*

Tropical semiring: defined as Semirings 2 and 3.

Semantics: length of shortest path between s and t.

Top-k semiring: for $k \geq 1$ some integer,

$$((\mathbb{R}^+ \cup \{\infty\})^k, \overset{k}{\min}, +^k, (\infty, \ldots, \infty), (0, \infty, \ldots, \infty)),$$

where

$$\overset{k}{\min}((a_1, \ldots, a_k), (b_1, \ldots, b_k)) = \overset{k}{\min}\{a_1, \ldots, a_k, b_1, \ldots, b_k\}$$

returns the k smallest entries (with duplicates) among those in a and b, in increasing order, and

$$(a_1, \ldots, a_k) +^k (b_1, \ldots, b_k) = \overset{k}{\min}\{a_i + b_j \mid 1 \leq i, j \leq k\}.$$

We further impose that only tuples that are in increasing order are valid elements of the semiring. Note that the top-1 semiring is the same as the tropical semiring.

Semantics: lengths of k shortest paths between s and t.

Counting semiring: defined as Semiring 5.

Semantics: total number of paths between s and t, edge weights being interpreted as number of edges between two vertices.

Boolean semiring: defined as Semiring 1.

Semantics: existence of a path between s and t, depending on the existence of edges denoted by their Boolean weights.

k-feature semiring: for $k \geq 1$ some integer,

$$((\mathbb{R}^+)^k, \min, \max, (\infty, \infty, \infty), (0, 0, 0))$$

where min and max are applied pointwise; it also exists in dual form, with min and max exchanged.

Semantics: minimum feature value along each dimension of all paths between s and t; if min and max are exchanged, maximum feature value along some path from s to t.

Integer polynomial semiring. $(\mathbb{N}[X], +, \times, 0, 1)$ where X is a finite set of variables, and $+$, $\times$, 0, 1 have their standard interpretations as polynomial operators and polynomial values.

Semantics: this is the most general provenance semiring in use for tracking non-recursive queries such as relational algebra queries [Green et al. 2007].

Formal power series: how-provenance, see [Green et al. 2007].

Shortest-path semiring: let $((\mathbb{R}^+ \cup \{\infty\}) \times \Sigma^*, \oplus, \otimes, (\infty, \epsilon), (0, \epsilon))$ with the following operators $\oplus$ and $\otimes$:

- $(d, \pi) \oplus (d', \pi') = (\min(d, d'), \pi'')$ where π'' is π if $d < d'$, π' if $d > d'$, and $\min(\pi, \pi')$ (in lexicographic order, assuming some order on Σ) if $d = d'$;
- $(d, \pi) \otimes (d', \pi') = (d + d', \pi \cdot \pi')$ if neither d nor d' is ∞; and $(d, \pi) \otimes (d', \pi') = (\infty, \epsilon)$ if either d or d' is ∞.

Semantics: pair formed of a length l and path label π such that π is the shortest path from s to t, of length l (if there are multiple shortest paths, π is the first in lexicographic order).

All semirings in Example 4.9 are commutative except for the shortest-path semiring (indeed, concatenation is not a commutative operation). All of them are idempotent, except for the top-k, counting, and integer polynomial semirings. The natural order of the tropical semiring is the total order $\geq$ (note that this is the *reverse* of the standard order on $\mathbb{R}^+ \cup \{\infty\}$).

Whereas many of these examples are quite simple, the framework of semiring provenance also allows modeling of intricate issues, e.g., when the problem of interest can be decomposed into several sub-problems and when the resulting provenance does not necessarily correspond to a particular path in the graph.

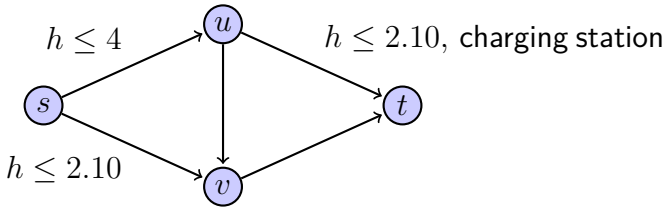


Figure 4.1.: Example road network represented by a graph with provenance annotations along two dimensions: maximum height h (as a positive number) a vehicle must have to use the road segment, and a Boolean indicating the presence of an electrical charging station. When a dimension is not mentioned, the annotations are assumed to be, respectively, $h \leq \infty$ and $\neg$(charging station).

Working example Consider the example of a road transportation network modeled as a directed graph with provenance annotations on edges. We can for example encode the presence of points of interests (such as gas stations, restaurants, or electrical charging stations) as Boolean features on edges, and road properties (e.g., maximal height or weight for a bridge or tunnel) as real-valued features.

We will show that, using semiring provenance, we can deal with graph queries that take into account multiple such features: a pair of vertices is valid for the queries if there exists at least one valid path for each restriction between the two locations. An application of this would be to ensure that different vehicle categories (say, a high-clearance truck and an electric car that requires charging on the way) can properly reach a common destination from the same origin.

Another possible semantics for semiring provenance is to check that all paths between two vertices verify (or exclude) some properties (e.g., absence of tolls, or presence of

gas stations on the route) thus providing road administrators crucial information on the global state of the roads between two points.

This is illustrated in Figure 4.1, a road network where some road segments have restrictions on the height on vehicles; this is a first dimension of provenance. The second dimension records whether there exists an electrical charging station on the road segment – in our example, this is the case for only one edge. We model the charging station Boolean feature as an integer feature by simply setting $\top = 1$ and $\bot = 0$. We take the $(\max, \min)$ definition of the k-feature semiring where we compute the maximum value of each feature among some path from origin to destination, and we order heights in decreasing order (e.g., by taking their inverse) so that a higher feature value means a (more restrictive) lower height.

Consider two types of vehicles of interest that want to reach the vertex t from the vertex s: one has height between 3 and 4 meters, the second is a small ($h \leq 1.5$) electric car that needs at least one charging station on the road to t. In the presence of the edge from u to v, both of them can reach t from s; without that edge, only the electric car is able to. This is reflected in the provenance: $\mathsf{prov}(G)(s,t) = (4, \text{charging station})$ while $\mathsf{prov}(G \backslash \{(u,v)\})(s,t) = (2.10, \text{charging station})$.

5. Provenance-Based Algorithms

We now provide a review of three algorithms to solve path provenance problems, initially described in [Ramusat et al. 2018]. Each of these algorithms yields a different trade-off between time complexity and applicability to various types of semirings, as summarized in Table 5.1.

Table 5.1.: Required semiring properties and asymptotic complexity for each studied algorithm, where $\mathrm{T}_\bullet$ is the complexity of the elementary semiring operation $\bullet$. The last column assumes constant cost for all semiring operations.

Name	Semiring property	Time complexity (with semiring op.)	Time complexity
MatrixAsteration	star	$\mathcal{O}(\lvert V\rvert \mathrm{T}_* + \lvert V\rvert^3(\mathrm{T}_\oplus + \mathrm{T}_\otimes))$	$\mathcal{O}(\lvert V\rvert^3)$
NodeElimination	c-complete star	$\mathcal{O}(\lvert V\rvert \mathrm{T}_* + \lvert V\rvert^3(\mathrm{T}_\oplus + \mathrm{T}_\otimes))$	$\mathcal{O}(\lvert V\rvert^3)$
Mohri	k-closed	Exponential	Exponential
MultiDijkstra	0-closed $\otimes$-idempotent	$\mathcal{O}\left(\ell \times (\mathrm{T}_\oplus \lvert V\rvert \log \lvert V\rvert + \lvert E\rvert\,(\mathrm{T}_\oplus + \mathrm{T}_\otimes))\right)$	$\mathcal{O}(\ell \times (\lvert V\rvert \log \lvert V\rvert + \lvert E\rvert))$
Dijkstra	0-closed total ordered	$\mathcal{O}(\mathrm{T}_\oplus \lvert V\rvert \log \lvert V\rvert + \lvert E\rvert(\mathrm{T}_\oplus + \mathrm{T}_\otimes))$	$\mathcal{O}(\lvert V\rvert \log \lvert V\rvert + \lvert E\rvert)$

Table 5.1 has five entries because we also compare the studied algorithms to the method we will introduce in Chapter 6, and to the MatrixAsteration algorithm described in the preliminaries; it has a wider scope than NodeElimination despite being less likely to allow structure-based heuristic for increased efficiency. We will discuss this in Chapter 7.

5.1. Mohri's Framework for Shortest-Distances

We now introduce *k-closed semirings* whose properties are widely used in the design of the Mohri's algorithm presented further.

Definition 5.1 (k-closed semirings [Mohri 2002])**.** *A semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ is k-closed for a given $k \geq 0$ if:*

$$\forall a \in S,\ \bigoplus_{n=0}^{k+1} a^n = \bigoplus_{n=0}^{k} a^n.$$

Remark 5.2. *Any 0-closed semiring is idempotent, since, for all a,*

$$a = a \otimes \bar{1} = a \otimes (\bar{1} \oplus \bar{1}) = a \oplus a.$$

The following property of k-closed semirings is standard:

Property 5.3. *Let $(S, \oplus, \otimes, \bar{0}, \bar{1})$ be a k-closed semiring, then for any integer $l > k$,*

$$\forall a \in S, \bigoplus_{n=0}^{l} a^n = \bigoplus_{n=0}^{k} a^n.$$

It is clear that k-closed semirings are star semirings (Definition 2.12). A specific caveat occurs there if we do not restrict to continuous structures. By looking at R_4 (aka. Semiring 7), we see it is a 1-closed semiring, but the star operator is not obtained by any finite number of steps. Thus MOHRI would compute the provenance as if $a^* \coloneqq a$. But it is also worth knowing they also are Conway semirings [Droste et al. 2009, Theorem 2.4]. In fact, any *locally closed semiring*[1] verifies the Conway equalities (Definition 2.16).

Back to Example 4.9, the tropical, Boolean, k-feature, and shortest-path semirings are 0-closed. The top-k semiring is $(k-1)$-closed. The counting and integer polynomial semirings are not k-closed for any k.

Mohri [2002] introduced an algorithm for computing single-source path provenance over k-closed semirings (denoted single-source shortest-distance problem in his line of work). Outlined in Algorithm 5, this algorithm performs in a manner similar to the Bellman–Ford algorithm with step-by-step relaxations over the edges of the graph (lines 13–14), maintaining a queue to decide in which order the elements are inspected. The queue can be chosen in different ways: based on the topology of the graph, e.g., if the graph is acyclic; or a queue prioritized by weight when, e.g., one wishes to compute top-k shortest paths using the top-k semiring. It is worth noting this algorithm does not require the semiring to be idempotent.

In the worst case, the theoretical complexity of this approach is exponential in the size of the graph [Mohri 2002], mainly due to the fact that the algorithm may have to visit the same cycle in the graph multiple times. However, the complexity heavily depends on the implementation of the queue. For instance, for top-k shortest paths, implementing a priority queue allows for an efficient algorithm, having polynomial complexity. For road transportation networks and top-k shortest paths, experiments we conducted in [Ramusat et al. 2021b] show an almost linear-time behavior in k and the size of the graph.

In contrast, the algorithm may be much more inefficient in practice for other types of networks (such as social networks). We have conjectured this may be due to the fact that transport networks have relatively low *treewidth* [Maniu et al. 2019]. The treewidth is a parameter measuring how much a graph (or more generally any relational instance) resembles a tree[2]. Many intractable problems over graphs have tractable solutions on instances of fixed treewidth. We have experimentally confirmed [Ramusat et al. 2021b] that many of the algorithms for provenance computation strongly benefit – in terms of running time – from low treewidth.

Another important graph parameter – stemming from the active research community around computing routing for, e.g, driving directions – the *highway dimension* [Abraham

[1] For any element a of the semiring there exist a k such that $\bigoplus_{n=0}^{k+1} a^n = \bigoplus_{n=0}^{k} a^n$.

[2] See Section 1.1.1 for formal definition and basic properties of treewidth.

et al. 2016] has been introduced to provide a theoretical basis for the efficiency observed in practice in state-of-the-art heuristics for computing optimal transport paths. This parameter relies heavily on weights on the edges of the graphs and the distribution of shortest distances in the graph. In the same set of experiments from [Ramusat et al. 2021b] we have evaluated whether this parameter also explains the practical efficiency of our algorithms for computing the provenance of routing queries. But it turns out the algorithms perform equally well using random weights. Chapter 7 presents our experimental results from [Ramusat et al. 2021b] related to these questions.

Algorithm 5 MOHRI – single-source path provenance [Mohri 2002]

Input: $(G = (V, E, w), s)$ a graph database with provenance indication over S and the source s.

Output: Array $\mathbf{w}$ representing the single-source path provenance from s.

for $i \in \{1, \ldots, |V|\}$ **do**
  $\mathbf{w}[i] \leftarrow r[i] \leftarrow \bar{0}$
end for
$\mathbf{w}[s] \leftarrow r[s] \leftarrow \bar{1}$
$Q \leftarrow \{s\}$
while $Q \neq \emptyset$ **do**
  $q \leftarrow \text{head}(Q)$
  dequeue(Q)
  $r' \leftarrow r[q]$
  $r[q] \leftarrow \bar{0}$
  for each $e \in E[q]$ **do**
    if $\mathbf{w}[n[e]] \neq \mathbf{w}[n[e]] \oplus (r' \otimes w[e])$ **then**
      $\mathbf{w}[n[e]] \leftarrow \mathbf{w}[n[e]] \oplus (r' \otimes w[e])$
      $r[n[e]] \leftarrow r[n[e]] \oplus (r' \otimes w[e])$
      if $n[e] \notin Q$ **then**
        enqueue$(Q, n[e])$
      end if
    end if
  end for
end while
$\mathbf{w}[s] \leftarrow \bar{1}$
return $\mathbf{w}$

5.2. Generalized Dijkstra's Algorithm

In preliminaries, we said we will sometimes consider the reversed of the natural order when it is more convenient to work with. When dealing with the generalized Dijkstra's algorithm we will consider the *reversed natural order*: $a \leq_S b \coloneqq a \oplus b = a$. As long as the semiring of interest is 0-closed, $\leq_S$ is indeed an order and for all $a \in S$, $\bar{1} \leq_S a \leq_S \bar{0}$.

Dijskstra's algorithm is generally used to solve shortest-distance problems in directed graphs. However, as shown also in [Ramusat et al. 2018], the algorithm readily generalizes to our semiring context, by placing some restrictions on the semirings used. For instance, the tropical semiring is exactly the semiring that allows to compute the shortest distance, as in the original algorithm. The general flow of the algorithm – using general semiring operations – is outlined in Algorithm 6, and Table 5.1 indicates its running time (in terms of the graph size and the costs of the semiring operations $\oplus$ and $\otimes$). Dijkstra's algorithm is known to be a very efficient algorithm. However, this efficiency comes from the fact that it uses a priority queue: once a value is extracted from it, we know that it is the correct one – this allows us to only visit each vertex in the graph once. This only works if we apply DIJKSTRA to semirings which are 0-*closed* (or absorptive) and in which an additional condition is satisfied: the natural order is a *total order* [Ramusat et al. 2018].

As we shall discuss later, there is a large complexity gap between DIJKSTRA on the one hand and the other two algorithms we discuss in this section – NODEELIMINATION and MOHRI – on the other. This is the main motivation to introduce the new algorithm we present in Chapter 6.

Algorithm 6 DIJKSTRA – single-source path provenance [Ramusat et al. 2018]

Input: $(G = (V, E, w), s)$ a graph database with provenance indication over S and the source s.

Output: Array $\mathbf{w}$ representing the single-source path provenance from s.

```
1: P ← ∅
2: w[a] ← 0̄, ∀a ∈ V
3: w[s] ← 1̄
4: while P ≠ V do
5:    Select a ∉ P with minimal w[a]
6:    P ← P ∪ {a}
7:    for each neighbor b of a not in P  do
8:       w[b] = w[b] ⊕ (w[a] ⊗ w[ab])
9:    end for
10: end while
11: return  w
```

Theorem 5.4 ([Ramusat et al. 2018])**.** *If the semiring of provenance is 0-closed and $\leq_S$ is a total order then Algorithm 6 computes the single-source path provenance from the source s.*

Proof. We show that for each vertex $a \in P$, $\mathbf{w}[a]$ is the shortest-distance between s and a. Note that when a vertex is added to P its value in $\mathbf{w}$ will no longer change.

Base case Because $\forall a \in S$, $\bar{1} \leq_S a$, s is the first vertex added in P and $\bar{1}$ is its provenance (corresponding to the empty path).

Induction case At each moment of the algorithm we consider for each $v \in V$, $L_{sv} \subseteq P_{sv}(G)$ the set of paths seen by the algorithm, i.e. $\bigotimes_{\pi \in L_{sv}} w[\pi] = \mathbf{w}[v]$. We have for each $p \in P$, $w[L_{sp}] = \bigoplus_{\pi \in P_{sp}(G)} w[\pi]$ the provenance of the reachability query. This set is updated only on line 8: $L_{sb} \leftarrow L_{sb} \cup (L_{sa} \cdot \{ab\})$.

We now show, that, when t is added in P, L_{st} is the provenance between s and t; at this moment, it holds that for each $c \notin P$, $w[L_{sc}] = \bigoplus_{p \in P} \left(\bigoplus_{\pi \in L_{sp}} w[\pi] \otimes w[pc] \right)$ and $\forall c \in \bar{P} \setminus \{t\}$, $w[L_{st}] \leq_S w[L_{sc}]$:

$$
\begin{aligned}
\bigoplus_{\pi \in P_{st}} w[\pi] &= \bigoplus_{p \in P} \left[\bigoplus_{\pi \in P_{sp}} w[\pi] \otimes \bigoplus_{\pi \in P_{pt}} w[\pi] \right] \\
&= \bigoplus_{p \in P} \left[\bigoplus_{\pi \in L_{sp}} w[\pi] \otimes \left[w[pt] \otimes (\bar{1} \oplus \bigoplus_{\pi \in P_{tt}} w[\pi]) \right] \right] \text{ (by IHP)} \\
&\oplus \bigoplus_{p \in P} \left[\bigoplus_{\pi \in L_{sp}} w[\pi] \otimes \bigoplus_{c \in \bar{P} \setminus \{t\}} \left(w[pc] \otimes \bigoplus_{\pi \in P_{ct}} w[\pi] \right) \right] \\
&= \bigoplus_{p \in P} \left[\bigoplus_{\pi \in L_{sp}} w[\pi] \otimes w[pt] \right] \text{ (0-closedness)} \\
&\oplus \bigoplus_{p \in P} \left[\bigoplus_{\pi \in L_{sp}} w[\pi] \otimes \bigoplus_{c \in \bar{P} \setminus \{t\}} \left(w[pc] \otimes \bigoplus_{\pi \in P_{ct}} w[\pi] \right) \right] \\
&\oplus \bigoplus_{p \in P} \left[\bigoplus_{\pi \in L_{sp}} w[\pi] \otimes \bigoplus_{c \in \bar{P} \setminus \{t\}} w[pc] \right] \text{ (because of total order)} \\
&= \bigoplus_{p \in P} \left[\bigoplus_{\pi \in L_{sp}} w[\pi] \otimes w[pt] \right] \\
&\oplus \bigoplus_{p \in P} \left[\bigoplus_{\pi \in L_{sp}} w[\pi] \otimes \bigoplus_{c \in \bar{P} \setminus \{t\}} w[pc] \right] \text{ (0-closedness)} \\
&= \bigoplus_{p \in P} \left[\bigoplus_{\pi \in L_{sp}} w[\pi] \otimes w[pt] \right] \text{ (because of total order)} \\
&= w[L_{st}]
\end{aligned}
$$

□

Note. It was not known to us by the time we published it in [Ramusat et al. 2018] that a similar generalization of the Dijkstra's algorithm has already been presented in [Minoux et al. 2008, Section 4.4.3]. The terminology was different: 0-closed totally ordered semirings were called *selective dioids*. One can also find in [Rote 1990, Section 5.4] a short paragraph about the possibility to generalize the Dijkstra's algorithm under similar

assumptions (i.e *semiring* which comes from *linearly ordered semigroups* with $\bar{1}$ *the largest element*); nevertheless, neither an algorithm nor a proof was provided alongside this claim. This is an example of the issues we faced when dealing with semiring theory: the large discrepancy of notation and names made it exceedingly difficult to assemble the relevant results from the literature.

5.3. Node-Elimination Technique

The most general algorithm available is based on the idea of Brzozowski and McCluskey for obtaining a formal language expression (i.e., a regular expression) equivalent to the language of an automaton [Brzozowski et al. 1963]. The algorithm is outlined in Algorithm 7. The algorithm works by eliminating vertices one by one and computing the "shortcut" values for each vertex pair, until only the source and target vertices remain. This algorithm works for any *ω-complete star semiring* instead of the weaker structure of star semirings because *infinite associativity*, i.e.,

$$ab^*c = a\left(\bigoplus_{i\geq 0} b^i\right)c = \bigoplus_{i\geq 0} ab^ic$$

must hold for the shortcuts computed in the algorithm to be correct.

In general, the complexity of the algorithm is at least cubic in the number of vertices in the graph, which makes it practically unusable on large graphs. Importantly, however, it also can be shown that its complexity is closely related to the treewidth parameter of the graph. Following a *simplicial elimination order*[3] (unfortunately not tractable to compute) one can rephrase the complexity shown in Table 5.1 in terms of the treewidth parameter w by $\mathcal{O}(|V|\mathrm{T}_* + w^2|V|(\mathrm{T}_\oplus + \mathrm{T}_\otimes))$. Thus, if the treewidth is small over, e.g., transportation networks, one can benefit from heuristics for finding a suitable elimination order to optimize this algorithm. We have dedicated a part of our experiments in [Ramusat et al. 2021b] to demonstrate the impact of some heuristics (for instance, focusing on vertices of higher degrees) on the running time of this algorithm. The results we obtained will be outlined in Chapter 7.

Another advantage of using NodeElimination is the possibility to compute the provenance for multiple pairs in a single run of the algorithm – we create t' for each target vertex t and return all such $\mathbf{w}_{s't'}$. Thus, the modified NodeElimination returns the provenances between (s, k_0) and each (t, k_F), for $k_F \in F$ and solves with same complexity the provenance for an RPQ using the graph product technique.

[3]Refer to Section 1.1.1 for a discussion relating simplicial elimination orders and treewidth.

Algorithm 7 NodeElimination – single-pair path provenance

Input: $(G = (V, E, w), s, t)$ a graph database with provenance indication over S, the source s, and the target t.

Output: Single value $\mathbf{w}_{s't'}$ representing the single-pair path provenance between s and t.

1: $V' \leftarrow V \cup \{s', t'\}$
2: $E' \leftarrow E \cup \{(s', s), (t, t')\}$
3: **for** $i \in V'$ **do**
4: **for** $j \in V'$ **do**
5: $\mathbf{w}_{ij}^{(0)} \leftarrow \begin{cases} w[ij] & \text{if } i \neq j, \\ \bar{1} \oplus w[ij] & \text{if } i = j \end{cases}$
6: **end for**
7: **end for**
8: **for** k in V **do**
9: **for** each (p, q) s.t. $(p, k), (k, q) \in E'$ **do**
10: $\mathbf{w}_{pq} \leftarrow \mathbf{w}_{pq} \oplus (\mathbf{w}_{pk} \otimes \mathbf{w}_{kk}^{*} \otimes \mathbf{w}_{kq})$
11: **end for**
12: **end for**
13: **return** $\mathbf{w}_{s't'}$

6. Lattice Decomposition

As explained in Section 5.2, DIJKSTRA requires a total natural order on the elements of a 0-closed semiring. This is quite a restrictive setting (among the examples listed in Example 4.9, only the *tropical semiring* fits), while using a more generally available algorithm such as MOHRI can lead to practical inefficiency. The question we addressed in [Ramusat et al. 2021b] is whether we can bridge this complexity gap and still obtain practical algorithms for 0-closed semiring without total orders.

First, we present an example semiring setting, with non-total natural order, where DIJKSTRA cannot be readily applied.

Example 6.1. *Let us consider the* 3*-feature semiring*

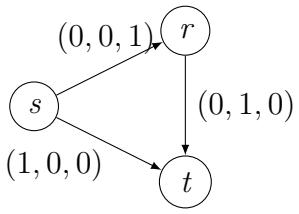
$$(\{0,1\}^3, \min, \max, (1,1,1), (0,0,0)).$$

In the example graph below, the provenance between s *and* t *is:*

$$\min\left(\max\left((0,0,1),(0,1,0)\right),(1,0,0)\right) = (0,0,0)$$

and that between s *and* r *is:*

$$\min\left(\max\left((1,0,0),(0,1,0)\right),(0,0,1)\right) = (0,0,0)$$



Assuming the existence of an order for which DIJKSTRA *computes this provenance. Then, starting from* s*,* DIJKSTRA *would select either* r *and assign it provenance* $(0,0,1)$*, which is wrong, or* t *and assign it provenance* $(1,0,0)$*, which is also wrong.*

We addressed this problem in [Ramusat et al. 2021b] by designing a new algorithm, MULTIDIJKSTRA (for *Multidimensional Dijkstra*) that applies to the more general case of 0-closed semirings for which multiplication is idempotent (such as the k-feature semiring, but also the Boolean function semiring used in probabilistic databases, see [Senellart 2017]). As it turns out, such semirings satisfy the axioms of *bounded distributive lattices* [Bistarelli et al. 1997, Theorem 10]; this allowed us to design an efficient algorithm for answering queries using these types of semirings.

6.1. Mathematical Background

In the following we recall the basic notions about finite distributive lattices already given in [Ramusat et al. 2021b]. We assume the lattices we use are finite because we are only ever using the subsemiring generated by edge annotations. As we shall see, this subsemiring is finite when both operations of the semiring are idempotent.

We refer the reader to [Siggers 2020] for more details regarding the theory behind distributive lattices. The basic introduction to lattices we propose thereafter follows the same line as the background section from [Siggers 2020].

6.1.1. Definitions and Notation

A *lattice* $(L, <)$ is a partially ordered set (*poset*) where every pair of elements have a unique infimum (their *meet*, $\wedge$) and supremum (their *join*, $\vee$). A *lattice embedding* of a lattice L into a lattice K is a one-to-one join and meet homomorphism from L to K. In a poset, an element y covers x (denoted $x \lessdot y$) if $x < y$ and there are no such z such that $x < z < y$. A lattice embedding ℓ is *tight* if $x \lessdot y$ implies $\ell(x) \lessdot \ell(y)$.[1]

An element x of a lattice L is *join-irreducible* if $x = a \vee b$ implies that $x = a$ or $x = b$. The set of non-zero join-irreducible elements of L is denoted $J(L)$. It induces a subposet of L which is also denoted by $J(L)$.

For a subset S of a lattice L, we let $\bigvee S = \bigvee_{x \in S} x$ be the join of the elements of S. We often write $\bigvee_L S$ to specify that the join takes place in L. A subset S of a poset is a *downset* or *ideal* if $x \in S$ and $y \leq x$ implies $y \in S$. The minimum downset containing an element x is denoted $\mathtt{id}\ x$. We note $\mathcal{D}(P)$, for a poset P, the family of downsets of P ordered by inclusion.

A *chain* C of length n in a poset P is a subposet isomorphic to the linear order $\mathbb{Z}_n$ on the n elements $\{0, 1, \ldots, n-1\}$. A *chain decomposition* of a poset P is a partition of its elements into a family $\mathcal{C}$ of chains $C_1, \ldots, C_d$. For a family $\mathcal{C} = \{C_1, \ldots, C_d\}$ of disjoint chains, the product $\prod \mathcal{C} := \prod_{i=1}^{d} C_i$ consists of all d-tuples $x = (x_1, \ldots, x_d)$ where $x_i \in C_i$ for each $i \in \{1, \ldots, d\}$. It is ordered by $x \leq y$ if $x_i \leq y_i$ for each i.

6.1.2. Results

A classical result from Birkhoff [1937] establishes a lattice isomorphism between L and $\mathcal{D}(J(L))$:

Theorem 6.2 ([Birkhoff 1937])**.** *The map* $\mathbf{S} : x \mapsto \mathtt{id}\ x \cap J(L)$ *is an isomorphism of* L *to* $\mathcal{D}(J(L))$*. Its inverse is* $S \mapsto \bigvee_L S$ *.*

For a chain decomposition $\mathcal{C}$ of a poset, let $\mathcal{C}_0$ be the family of chains we get from the chains in $\mathcal{C}$ by adding a new minimum element to each. Dilworth [1950] proved the following embedding theorem:

[1] Implicitly from lattice notation to poset notation: $x \vee y = y$ means $x \leq y$.

Theorem 6.3 ([Dilworth 1950]). *For any chain decomposition $\mathcal{C}$ of a poset P the map $S \mapsto \bigvee_P S$ is an embedding of $\mathcal{D}(P)$ into $P = \prod \mathcal{C}_0$.*

Then, we obtain the following corollary we will use later:

Corollary 6.4 ([Siggers 2020, from Corollaries 2.4 and 2.6]). *Given a chain decomposition $\mathcal{C}$ of a distributive lattice L, there is a tight embedding of L into $\prod \mathcal{C}_0$.*

6.2. Application to Provenance Computation

Corollary 6.4 provided us with a way to compute provenance over distributive lattices using a multidimensional version of Dijkstra. Because an embedding is a homomorphism, we can compute each component of $\prod \mathcal{C}_0$ independently. And because the homomorphism is one-to-one, we can easily recover the provenance at the end of the computation.

Example 6.5. *If we take a look at distributive lattice of the divisors of* 60 *with greatest common divisor (gcd) and least common multiple (lcm) as join and meet operators, we notice that the divisors of* 60 *are either powers of* 2, 3, 5 *or an lcm of these integers. Thus, they can be represented using three dimensions representing the factorization of* 60 *along these prime numbers: decompose*$(4) = (2, 0, 0)$*, recompose*$(0, 1, 0) = 3$*, and recompose*$(2, 1, 0) = 12$*. We can then compute* independently *each dimension of the result using Dijkstra since each component is totally ordered; then, partial results are combined.*

In other words, we can run separately, ℓ times, Dijkstra for each dimension of this product, where ℓ is the number of chains in the chain decomposition. This gives us a parameterized algorithm, where ℓ depends on the semiring. For example, for the semiring used in Example 6.1, $\ell = 3$. We outline the algorithm in pseudo-code in Algorithm 8. We need the following routines that are highly specific to the semiring: *decompose*(e) takes as parameter an element e of L and returns its image $v(e) \in \mathcal{P}$. For the opposite direction *recompose*$(d_1, \ldots, d_n) = \bigvee_{0 \leq i \leq n} d_i$ returns as expected an element of L.

We use as a subroutine a slightly modified version of Dijkstra, parameterized by the semiring dimension and working with semirings having elements in vector form, corresponding to the decomposition. *Dijkstra(s,t,i)* $\in J(L)$ computes the provenance between s and t corresponding to the i^{th} dimension of the decomposition.

Example 6.6. *We describe the working of Algorithm 8 in the example presented in Example 6.1: first, each edge value is decomposed; this step is easy to follow as the* 3*-feature values are already presented in decomposed form. A second step consists in calculating values along each dimensions. Algorithm 6 is launched a first time over the graph with edge values corresponding to the first dimension:* 0 *for* (s, r) *and* (r, t)*,* 1 *for* (s, t)*. The result is* 0*. Algorithm 6 is launched a second time over the graph with edge values corresponding to the second dimension:* 0 *for* (s, r) *and* (s, t)*,* 1 *for* (r, t)*. The result is, again,* 0*. Finally, Algorithm 6 is launched a third time over the graph with*

edge values corresponding to the third dimension: 0 *for* (s,t), 1 *for* (s,r) *and* (r,t). *The result is* 0. *This ends the second step. The third step consists in recomposing partial values obtained by successive applications of the Dijkstra subroutine. This ends up to the final provenance value of* $(0,0,0)$.

Algorithm 8 MultiDijkstra – single-pair path provenance [Ramusat et al. 2021b]

Input: $(G=(V,E,w),s,t)$ a graph database with provenance indication over S, the source s, and the target t.
Output: Single-pair path provenance from s to t.

```
for each edge e ∈ E do
    decompose(w(e))
end for
for each dimension i do
    d_i ← Dijkstra(s, t, i)
end for
return recompose(d_1, ..., d_n)
```

Theorem 6.7 ([Ramusat et al. 2021b])**.** *If the semiring of provenance is 0-closed and multiplicatively idempotent, then Algorithm 8 computes the single-pair path provenance from* s *to* t *in time* $\mathcal{O}(\ell \cdot (m + n \log n))$.

For the sake of simplicity, we presented the single-pair version of our algorithm. To extend it to the single-source version one only needs to perform the *recompose* subroutine for each vertex in the graph.

To minimize accesses to the *decompose* subroutine – which can be very costly – we optimize MultiDijkstra by adopting a lazy approach, where the *Dijkstra* subroutine calls *decompose* only when needed, storing the decomposition across calls. This avoids scanning the whole graph when s and t are close.

Two other optimizations implemented are a stopping condition that ends the *Dijkstra* subroutine when a visited vertex has value $\bar{0}$, and lazy initialization of the priority queue. These two optimizations led to vastly improved computation times over the naive implementation.

6.3. Practical Use Case

As exemplified in Figure 4.1, k-feature semirings can be used to ensure that all paths from s to t verify a combination of features (they all go through a specific set of points of interests, or verify some road properties) or either ensure the existence of valid paths up to some collection of restrictions. We showed in [Ramusat et al. 2021b, Section 6] that this is tractable for practical use cases (continental-sized areas, around 10^7 vertices). Those experiments will be reproduced in the manuscript in Chapter 7. To the best of our knowledge, no solution for this that scales even to graphs of thousands of vertices has been previously proposed.

7. Practical Efficiency

In [Ramusat et al. 2021b] we performed experiments on real-world graph data, using an Inria computing cluster running the OAR task manager. The individual vertices of the cluster have a minimum of 48 GB of RAM, and run Intel Xeon X5650 or E5-26xx CPUs.

We used datasets[1] from a variety of domains, mostly representing infrastructure networks: the OpenStreetMaps network of Paris (Paris), the Paris public transport network (Stif), and the power grid of the continental US (USPowerGrid). For comparison, we have also evaluated on other types of datasets: a small subset of the Facebook social network (Facebook) and the yeast protein-to-protein interaction network (Yeast). All these datasets came without provenance annotations, that we added in different ways depending on experiments. We also used a real weighted road transportation network dataset Rome99, with tropical semiring annotations, from the 9th DIMACS Implementation Challenge[2]. This dataset consists of a large portion of the directed road network of the city of Rome, Italy, from 1999. Basic information about the resulting graphs are summarized in Table 7.1.

For datasets without provenance annotations, unless specified differently, we randomly generated weights in the tropical semiring for benchmarks, uniformly between 1 and 3 000. To be able to compare the impact of the weights on the performance of the algorithms, we also use a constant-weight setting, where all weights equal to 1. Each experiment generally represents the average over 10 runs (random choices of origin and destination vertices).

Table 7.1.: Graph datasets: size and treewidth lower and upper estimates from [Maniu et al. 2019]

type	name	# of vertices	# of edges	tw
infrastructure	Paris	4 325 486	5 395 531	55–521
	Stif	17 720	31 799	28–86
	USPowerGrid	4 941	6 594	10–18
	Rome99	3 353	4 831	5–50
social	Facebook	4 039	88 234	142–237
biology	Yeast	2 284	6 646	54–255

[1] These datasets were used in [Maniu et al. 2019] for treewidth computation experiments, and are downloadable from `https://github.com/smaniu/treewidth/`; some of them originate from `http://snap.stanford.edu/data/index.html`.

[2] `http://users.diag.uniroma1.it/challenge9/download.shtml`

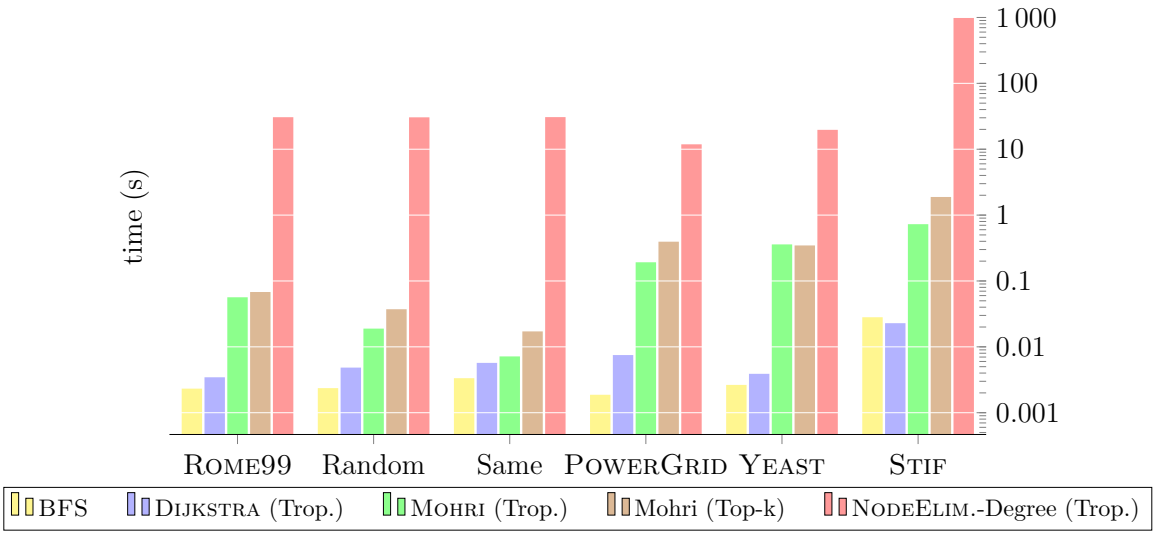


Figure 7.1.: Comparison between algorithms for shortest distances

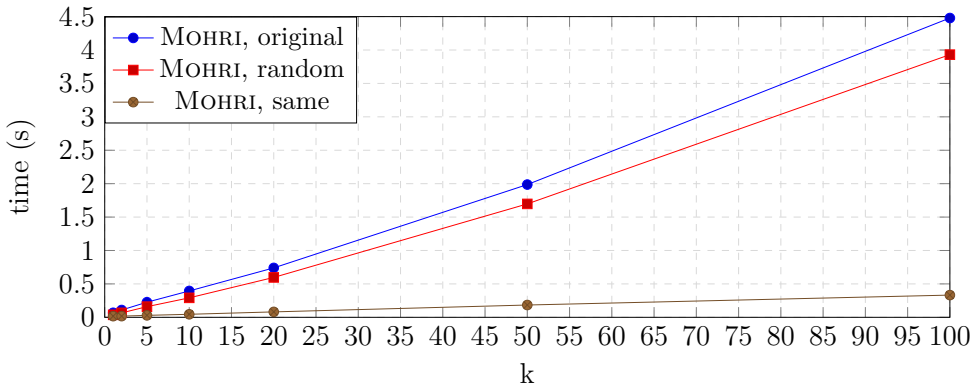


Figure 7.2.: Computation time for Mohri over the top-k distances semiring, for varying values of k and varying weight assignments (Rome99)

The experimental study was focused on comparing the four algorithms we have previously presented, over several semirings. We provide a comparison of all of our algorithms for the computation over the tropical semiring (shortest distance), since all algorithms can be used in this setting. We investigate the running time and the number of relaxation steps performed by Mohri and MultiDijkstra algorithm, using initial weights provided by the dataset Rome99, as well as custom weights (all identical and all random); we then study over all datasets the impact of the elimination order heuristic on the overall performance for NodeElimination. We then finish with the comparison between the new algorithm and the previous solutions to demonstrate its efficiency.

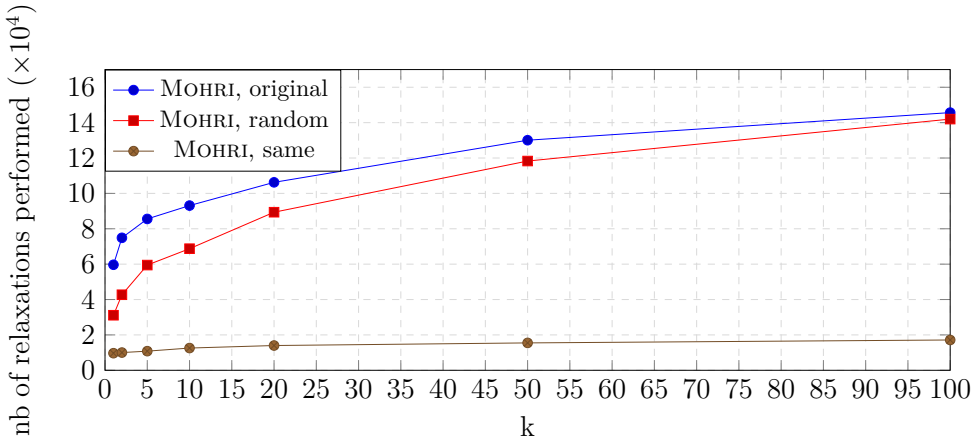


Figure 7.3.: Number of relaxations performed by Mohri over the top-k distances semiring, for varying values of k and varying weight assignments (Rome99)

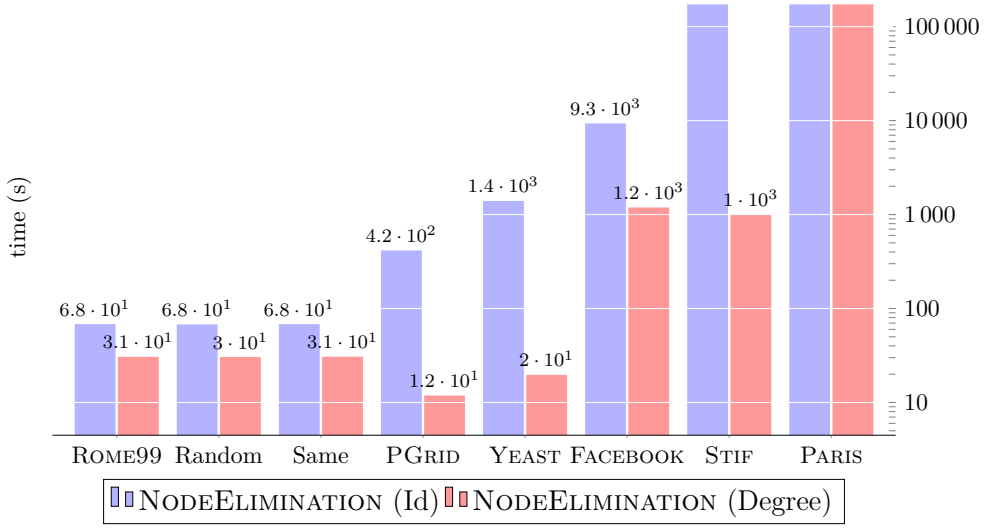


Figure 7.4.: Comparison between elimination orders for NodeElimination algorithm (tropical semiring). Values greater than 100 000 s are timeouts.

7.1. Evaluating Shortest Distances

We start by evaluating how the algorithms deal with the shortest distance semiring, i.e., the tropical and top-k semiring (by setting $k = 1$). The properties of this semiring allow their implementation for the first three algorithms: Dijkstra, Mohri, and NodeElimination, whereas MultiDijkstra reduces to Dijkstra in that case. We also implemented a breadth-first-search traversal for computing accessibility with no provenance information (BFS). This also allows us to compare the performance of algorithms against non-annotated graph databases.

Figure 7.1 shows, on a logarithmic scale, the result for our graphs, and for some settings of weights (original, random, or same weights). It is immediately clear from the

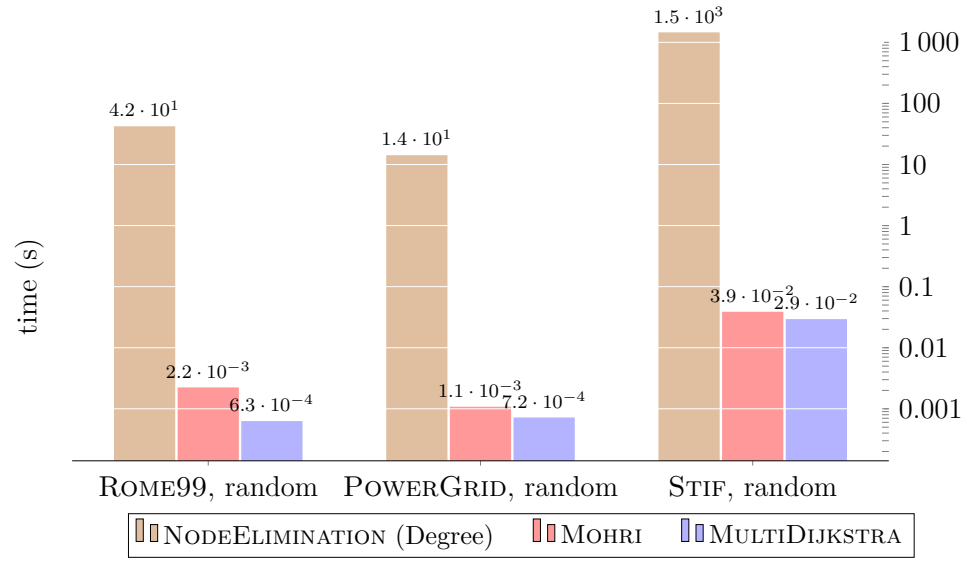


Figure 7.5.: Comparison between NODEELIMINATION, MOHRI, and MULTIDIJKSTRA (3-feature semiring)

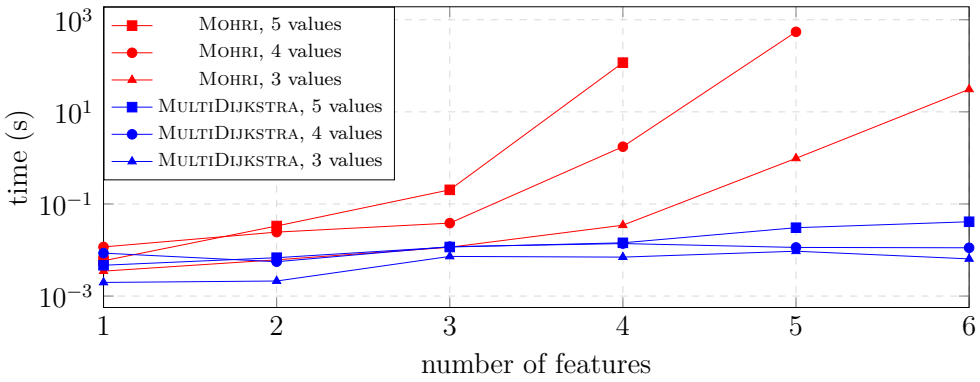


Figure 7.6.: Computation time for MOHRI and MULTIDIJKSTRA depending on the number of dimensions (ROME99)

figure that the choice of algorithm is crucial: we need the most specialized algorithm for the semiring we use: DIJKSTRA is more efficient than MOHRI which is more efficient than NODEELIMINATION. Even for MOHRI, we notice that using it configured for the top-k semiring with $k = 1$ does introduce an overhead in execution; the use of the tropical semiring mitigates the overhead. We also show the overhead introduced when using provenance annotations is quite limited, as the difference between DIJKSTRA and BFS is less than an order of magnitude for each dataset, and DIJKSTRA sometimes even outperforms BFS. Finally, NODEELIMINATION is always several orders of magnitude slower than Dijkstra. Another encouraging result is that MOHRI – which allows more classes of semirings than DIJKSTRA – has a reasonable running time in practice, despite the stated exponential complexity bound in the original paper. We turn to evaluating

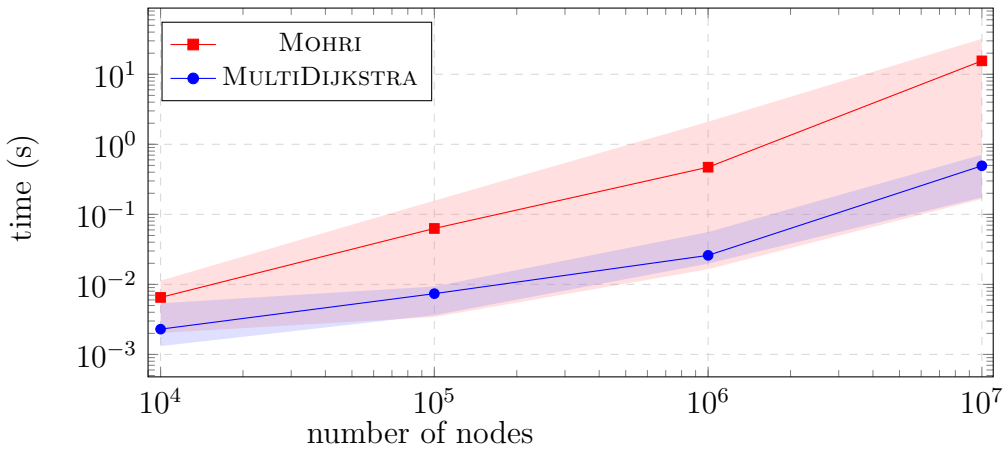


Figure 7.7.: Average computation time for Mohri and MultiDijkstra over random graphs depending on the number of nodes; shaded areas indicate minimum and maximum computation times observed (3-feature semiring)

its performance next.

7.2. Mohri's Algorithm in Practice

In Figure 7.2 and in Figure 7.3 we respectively study the impact of the factor k on the running time and on the number of computations performed by the algorithm. Our results show that the computational time is linear in k, though this is not the case for the number of relaxations, which increases sublinearly in k. This means that for large values of k the algorithm spends most of its time maintaining the queue.

We also compare the performance of the algorithm depending on weight assignment (original, random, same). It seems that considering random values instead of "real" values has almost no significant impact over the efficiency of the algorithm. This is a somewhat disappointing result because it rules out the possibility to parameterize the complexity of the algorithm through network parameters, for instance, in terms of the *highway dimension* [Abraham et al. 2016] – a graph parameter that has been successfully applied for understanding the efficiency of state-of-the-art shortest-distance algorithms in road networks. However, the performance significantly increases when all weights are uniform, which may be expected since computation of shortest distances become far simpler, and far more paths have equal distance.

As pointed out in Section 5.1, this algorithm performs extremely well over transportation networks. We wanted to provide a comparison of its working time for different kinds of graphs (especially graphs whose treewidth is relatively large for their size). For this purpose we used a social network dataset: who-trusts-whom network of people who trade using Bitcoin on a platform called Bitcoin Alpha [Kumar et al. 2016; Kumar et al. 2018] (3 783 vertices and 24 186 edges). The algorithm times out after 48 hours.

What we can learn from this is that the key property making Mohri so efficient over transportation networks is not due to distance properties (e.g., highway dimension) –

impacted by the weights of the connections – but rather by topological properties of the underlying graph (e.g., treewidth).

7.3. Orderings for Node-Elimination Technique

NodeElimination's perfomance, due to its main loop of creating "shortcuts" in the graph, is heavily dependent on the order in which the vertices are eliminated. This elimination ordering is strongly linked to the *treewidth* parameter of the graph. For instance, following a degree based elimination order gives an upper bound on this parameter.

Hence, we have compared different elimination orders for NodeElimination and found out that the minimum degree based elimination order (*Degree*) greatly improves the efficiency of this algorithm compared to having no such heuristic (*Id*). This improvement can be dramatic, as for the Yeast dataset where the algorithm is two orders of magnitude faster. As expected, weights over the edges do not impact the running time, as shown in Figure 7.4.

This is important in practice: running NodeElimination on low-treewidth graphs (e.g., infrastructure and transport networks) can be the difference between the algorithm being unusable and allowing reasonable running times. Taking into account that NodeElimination allows for a large class of semirings, this can have a significant real-world application impact.

7.4. Evaluation of MultiDijkstra

We now evaluate MultiDijkstra, our solution for bridging the gap between absorptive semirings and more general ones. We compare it to Mohri and NodeElimination in the case of the k-feature semiring, which is kind of the canonical semiring that is 0-closed and multiplicatively idempotent. Figure 7.5 showcases this on 3 datasets. In all cases, our new algorithm is between 3 and 4 orders of magnitude faster than NodeElimination, depending on the network we use, and significantly faster than Mohri.

We then performed an additional experiment (Figure 7.6), examining the impact of the number of features and values actually used in each feature on the running time of both algorithms. We found out that when either one of the two criteria reaches 4, Mohri times out while MultiDijkstra keeps scaling.

Finally, Figure 7.7 presents a comparison between Mohri and MultiDijkstra on large Erdős–Rényi random generated graphs (generated using Python networkx's *fast_gnp_generation* method, using an average of 1.7 edges per vertex) show that our new algorithm is still tractable for continental-sized graphs of millions of vertices. Interestingly, MultiDijkstra also exhibits a much smaller variance than that of Mohri, whose performance varies by more than one order of magnitude between runs.

8. A Taxonomy of Semirings for Provenance over Graph Databases

This last chapter will be divided in two parts. In Section 8.1, we first make the formal link between the results of Section 2.3.1 and our framework, while obtaining a lower bound for full path provenance computations. The second section mainly deals with Figure 8.1, representing a summary of the algorithms we can use, depending on the properties of the semiring of interest. Based on that, we finish by discussing some pending research questions.

8.1. Lower Bounds

The main focus of this work has been to provide general methods for computing semiring provenance for very large graph databases. Being able to provide an improved solution – in terms of asymptotical complexity – to NodeElimination for full provenance computation would have had a great impact. Very unfortunately, it became more and more apparent, whilst establishing links to other problems arising in computer sciences, that such an algorithm has never been discovered for related issues.

We finally found out results in the literature linking the complexity of performing the asterate of a semiring-valuated matrix to the complexity of carrying out matrix multiplication [Mehlhorn 1984]. This result allows a characterization of semirings on which the monotone complexity – only using monotone operators $\oplus$ and $\otimes$ – for matrix multiplication is cubic time. This work has been presented in Section 2.3, and we can deduce the following corollary.

Corollary 8.1. *The monotone complexity for square matrix asteration over an ω-complete star semiring requires $\Omega(n^3)$ semiring operations.*

Proof. We can assume the size of the matrix to be fixed, given a specific entry. We can thus reformulate any algorithm satisfying the above condition as a straight-line program. We conclude using Property 2.24, Theorem 2.39, and Theorem 2.42. □

This entails the fact that computing full provenance without considering any additional property of the semiring of use is not tractable in practice for large graphs such as continental-sized transportation networks. This leaves open many research opportunities, of which some have been investigated in this manuscript: finding new semiring properties permitting better optimizations as in Chapter 6, discovering graph parameters having the most impact on the efficiency of the algorithm as in Chapter 7, and later

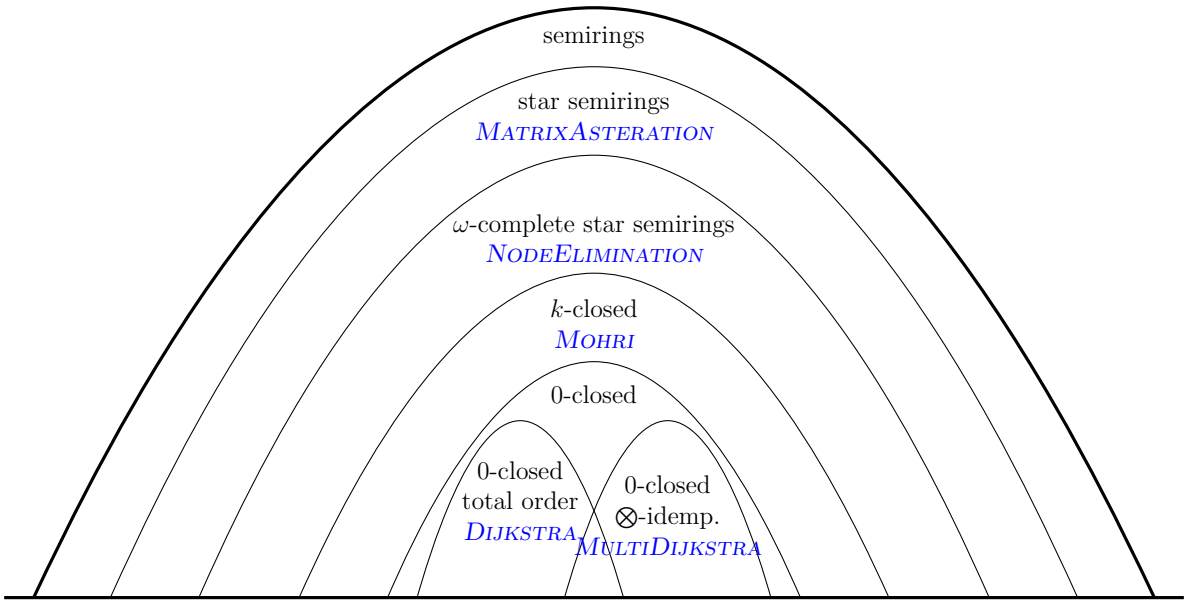


Figure 8.1.: Taxonomy of the semirings used for graph provenance along with algorithms that work on them [Ramusat et al. 2021b].

on in Part III linking to Datalog provenance. Others opportunities will be discussed as open research questions in Chapter 12.2.

8.2. A Hierarchy of Semirings

Figure 8.1 provides a high-level view linking the properties and classes of semiring we presented in Section 2.2 and Chapter 5 together and their associated algorithms, presented in Chapter 5 and Chapter 6. The figure shows a clear hierarchy of classes of semirings, both in terms of the complexity of the algorithm and the expressive power of the semirings.

An important practical application that is similar to our setting is the provenance for Datalog queries introduced in [Green et al. 2007] and further optimized using circuits [Deutch et al. 2014]. Datalog – refer to Section 1.2 for a quick introduction – is a language derived from Prolog, useful for infering new knowledge given existing facts and a set of inference rules. In the papers above, the semiring classes for which optimization of queries is possible are strikingly similar: PosBool(X) and Sorp(X) discussed in [Green et al. 2007; Deutch et al. 2014] correspond respectively to the positive fragment of the Boolean function semiring, and to the free (i.e., most general) 0-closed semiring.

We have seen in Section 8.1 that the computation of the asteration of a matrix over an arbitrary ω-complete semiring (i.e, using only semiring operations and not some specific properties of the semiring of interest) is at least involving a cubic number of semiring operations. Thus, the NodeElimination algorithm is optimal for computing

the full path provenance. While the formal proof of this result was not known to us by the midpoint of the thesis, we were strongly convinced it would not be possible to significantly improve full provenance computation in the general case. We have decided to turn our focus on Datalog provenance to benefit from the richness of the literature and to propose new ways for solving path provenance by converting a graph database into a Datalog query.

The reader will find in Perspectives some discussion about two other research opportunities we unfortunately did not had time to address during our doctoral studies:

- while Corollary 8.1 concerns the full matrix asteration, it could be interesting to consider lower bounds on the computation of a restricted subset of the asteration matrix (e.g., one single row for single-source, only one cell for single-pair);
- devising parallel algorithms built upon the notion of eliminants [Abdali 1994] to provide an efficient computation of the full path provenance (understood as the asteration of a matrix over a star semiring).

Part III.

Datalog Provenance for Graph Queries

There are two major notions of provenance for information systems currently in use in the literature, each focusing on different usages. The first notion can be broadly categorized as "informational": the provenance encapsulates information about the deductive process leading to a specific result. Declarative debugging is one application of this concept. Analysis of provenance information in an interactive manner simplifies the user's burden of identifying which part of the rule specification is responsible for a *faulty* derivation.

In this manuscript we consider a "computational" notion of provenance, where operations (and queries) over provenance values are permitted. The initial definition of the *provenance* of a Datalog program has been defined in Section 3.2 as a system of equations over an ω-continuous semiring following the seminal paper of [Green et al. 2007].

Beyond that, some research has proposed other representation frameworks leveraging the underlying properties of the semiring leading to optimized algorithms. It has been shown in [Deutch et al. 2014] that, for a Datalog program having n candidate IDB tuples, a circuit for representing Datalog provenance in the semiring $\mathbf{Sorp}(X)$ (the most general absorptive semiring) only needs $n + 1$ layers. For binary relations, e.g., representing the edge relation of a graph, this construction is at least quadratic in the number of vertices, thus not practically applicable for the large real-world networks we target in our research work. Similarly, in [Esparza et al. 2011], absorptive semirings (i.e., 0-closed semirings) have the property that derivation trees of size $\geq n$ are "pumpable" (they do not contribute to the final result). A concrete implementation [Esparza et al. 2014] computes the provenance for commutative and idempotent semirings using n Newton iterations.

Fairly recently, a paper of Khamis et al. [2021] introduced POPS (*Partially Ordered, Pre Semiring*), a structure decoupling the order on which the fixed-point is computed from the semiring structure. Complex and recursive computations over vectors, matrices, tensors are now expressible using this framework. This approach and those of [Deutch et al. 2014] and [Esparza et al. 2011] rarely go all the way to the implementation level and are internally considering all possible IDB tuples, thus raising the question of the applicability to real-world data.

A work of note that does come with an implementation is [Deutch et al. 2015; Deutch et al. 2018] in cases where keeping the *full* provenance of a program (*how*-provenance) is still prohibitively large. A relevant subset of the derivation trees is selected using *selection criteria* based on tree patterns and ranking over rules and facts occurring in the derivation. First, given a Datalog program P and a pattern q, an *offline* instrumentation is performed, leading to an *instrumented program* P_q. Then, given any database D, an efficient algorithm can be used to retrieve only the top-k best derivation trees for $P_q(D)$.

We will present our contributions from [Ramusat et al. 2021a] in the four next chapters. In Chapter 9 we focus on a correspondence between dynamic programming over hypergraphs (as introduced in [Huang 2008] under the name of *weighted hypergraphs*) and the *proof-theoretic* definition of the provenance for Datalog programs. We provide both-way translations and characterize for which class of semirings the *best-weight derivation* in the hypergraph corresponds to the provenance of the initial Datalog program.

The above mentioned translation led us to an adaption of Knuth's generalization of Dijkstra's algorithm [Knuth 1977] to the *grammar problem*, adapted to the case of Datalog provenance computation. This will be described in Chapter 10. In the special setting where all hyperedges are of arity 1, we obtain the notion of semiring-based provenance for graph databases described in Chapter 4. In the general setting, the algorithm steadily generalizes to Datalog the adapted Dijkstra's algorithm we presented in Section 5.2, under the same assumptions on the properties of the underlying semiring.

Such algorithm is unlikely to be efficient as-is in practical contexts. The main issue is closely related to the inefficiency of basic Datalog evaluation: many computations of facts (provenance values) have already been deduced, leading to redundant computations. In the same chapter we show that the *semi-naïve* evaluation strategy for Datalog is also applicable in our setting.

In comparison to the line of work of [Khamis et al. 2021], our method is restricted to semirings that are totally ordered (a subclass of distributive dioids[1]), leveraging the invariant that once a fact is first labeled with a provenance value, we are certain it is the correct one. The top-1 algorithm of [Deutch et al. 2015; Deutch et al. 2018] is also closely related to our solution, but does not mention the use of a priority queue nor does it take into account the optimization provided by the semi-naïve evaluation strategy we describe in Chapter 10. The best-first method we outline in this manuscript can thus be seen as a hybrid of the ideas introduced in [Khamis et al. 2021] and [Deutch et al. 2018].

An added advantage is that it greatly facilitates the extension of existing Datalog solvers to compute provenance annotations. We discuss in Chapter 11 of our implementation based on Soufflé [Jordan et al. 2016], a state-of-the-art Datalog solver, for the best-first method. We applied our solution to process rich graph queries (translated into Datalog programs) on several real-world and synthetic graph datasets, as well as to Datalog programs used in previous works [Deutch et al. 2018]. Still in the same chapter, we provide experimental results witnessing that the performance of the implementation competes with the efficiency of dedicated solutions for graph databases that have been evaluated in Chapter 7.

The link we established with the framework of weighted hypergraphs, and hence, with the grammar problem, opens up new opportunities and challenges for semiring-based provenance for Datalog programs. We will discuss in Chapter 12 selected topics and problems that can now be understood as closely linked to the provenance framework for Datalog programs of Green et al. [2007].

[1]Distributive dioids are POPS structures over a distributive lattice being the natural order of the dioid.

9. Datalog Provenance and Dynamic Programming over Hypergraphs

In [Ramusat et al. 2021a] we showed how to convert a Datalog program into a weighted hypergraph (as introduced in [Huang 2008]) and characterized the semirings where the best-weight derivation in the hypergraph corresponds to the provenance for the initial Datalog program, mimicking the proof-theoretic definition.

9.1. Best-Weight Derivation

We first recall basic definitions and notation related to hypergraphs.

Definition 9.1 (Weighted hypergraph [Huang 2008])**.** *Given a semiring S, a* weighted hypergraph *on S is a pair $H = \langle V, E\rangle$, where V is a finite set of vertices and E is a finite set of hyperedges, where each element $e \in E$ is a triple $e = \langle h(e), \mathrm{T}(e), f_e\rangle$ with $h(e) \in V$ its* head *vertex, $\mathrm{T}(e) \in V$ an ordered list of* tail *vertices and f_e a* weight function *from $S^{|\mathrm{T}(e)|}$ to S.*

We note $|e| = |\mathrm{T}(e)|$ the *arity* of a hyperedge. If $|e| = 0$, we say e is nullary and then $f_e()$ is a constant, an element of the semiring; we assume there exists at most one nullary edge for a given vertex. In that case, $v = h(e)$ is called a *source vertex* and we note $f_e()$ as f_v. The *arity of a hypergraph* is the maximum arity of any hyperedge.

The *backward-star* $\mathrm{BS}(v)$ of a vertex v is the set of incoming hyperedges $\{e \in E \mid h(e) = v\}$. The *graph projection* of a hypergraph $H = \langle V, E\rangle$ is a directed graph $G = (V, E')$ where $E' = \{(u, v) \mid \exists e \in \mathrm{BS}(v), u \in \mathrm{T}(e)\}$. A hypergraph H is acyclic if its graph projection G is acyclic; then a topological ordering of H is an ordering of V that is a topological ordering of G.

With these definitions in place, we can encode a Datalog program with semiring annotations as a weighted hypergraph in a straightforward manner:

Definition 9.2 (Hypergraph representation of a Datalog query [Ramusat et al. 2021a])**.** *Given a Datalog program q as a set of rules $\{q_1, \cdots, q_n\}$ and the semiring S used for annotations, we define the* weighted hypergraph representation of q *as $H_q = \langle V_q, E_q\rangle$ with V_q being all ground atoms and, for each instantiation of a rule $t(\vec{x}) \leftarrow r_1(\vec{x_1}), \ldots, r_n(\vec{x_n})$, a corresponding edge $\langle t(\vec{x}), (r_1(\vec{x_1}), \ldots, r_n(\vec{x_n})), \otimes\rangle$. For a fact $R(\vec{x}) \in \mathsf{EDB}(q)$ we add a nullary edge e with $h(e) = R(\vec{x})$ and $f_e = \mathsf{prov}^q_R(\vec{x})$.*

The notion of *derivations* is the hypergraph counterpart to paths in graph. We recall the definition of derivations and we define it in a way that is reminiscent of Datalog-related notions.

Definition 9.3 (Derivation in hypergraph [Huang 2008])**.** *We recursively define a* derivation *D of a vertex v in a hypergraph H (as a pair formed of a hyperedge and a list of derivations), its size $|D|$ (a natural integer) and its weight $w(D)$ (a semiring element) as follows:*

- *If $e \in \mathrm{BS}(v)$ with $|e| = 0$, then $D = \langle e, \langle\rangle\rangle$ is a derivation of v, $|D| = 1$, and $w(D) = f_e()$.*
- *If $e \in \mathrm{BS}(v)$ with $|e| \geq 0$, D_i is a derivation of $T_i(e)$ for $i = 1 \dots |e|$, then $D = \langle e, \langle D_1 \cdots D_{|e|}\rangle\rangle$ is a derivation of v, $|D| = 1 + \sum_{i=1}^{|e|} |D_i|$, and its associated weight is $w(D) = f_e(w(D_1), \dots, w(D_{|e|}))$.*

We note $\mathcal{D}_H(v)$ the set of derivations of v in H.

When modeling Datalog provenance in a semiring S as weighted hypergraphs on S, all non-source weight functions are bound to the $\otimes$ operation of the semiring. Note that, if S is idempotent, the natural order on S induces an ordering on derivations: $D \leq D'$ if $w(D) \leq w(D')$.

We now show that in this formalism, the Datalog provenance of an output predicate can be understood as the best-weight for the corresponding vertex in the hypergraph.

Definition 9.4 (Best-weight [Huang 2008])**.** *The* best-weight *$\delta_H(v)$ of a vertex v of a hypergraph H on a semiring $(S, \oplus, \otimes, \bar{0}, \bar{1})$ is the weight of the best derivation of v:*

$$\delta_H(v) = \begin{cases} f_v & \text{if } v \text{ is a source vertex;} \\ \bigoplus_{D \in \mathcal{D}_H(v)} w(D) & \text{otherwise.} \end{cases}$$

The best-weight generally requires additional properties of either the hypergraph or the semiring to be well-defined. Acyclicity for the hypergraph is a sufficient condition. Existence of an infinitary summation operator in the semiring extending $\oplus$, guaranteed in ω-continuous semirings, is also a sufficient condition. To guarantee semantics compatible with the intuitive meaning of provenance, we need to consider, at least, *ω-complete star-semirings*.

9.2. Equivalence with Datalog Provenance

We can now show that Datalog provenance can be computed through the formalism of weighted hypergraphs. Let us start with a lemma exhibiting a one-to-one mapping between derivations in the hypergraph and proof trees in Datalog.

Lemma 9.5 ([Ramusat et al. 2021a])**.** *For any Datalog query q and grounding of an atom v of q, there is a bijection between $\mathcal{D}_{Hq}(v)$ and $\{\tau \mid \tau \text{ yields } v\}$.*

Proof. By definition of H_q each instantiation of a rule corresponds to a unique hyperedge. Then, we can inductively construct for a given derivation D its associated (unique) Datalog proof tree τ_D:

- If $|D| = 1$, then v is a source vertex and thus an extensional tuple, we get the empty proof.
- If $|D| \geq 1$, then there exists $e \in \mathrm{BS}(v)$ where $|e| \geq 0$ and D_i a derivation of $T_i(e)$ for $1 \leq i \leq |e|$, where $D = \langle e, D_1 \cdots D_{|e|} \rangle$. By definition, this hyperedge corresponds to the grounding of a rule $t(\vec{x}) \leftarrow r_1(\vec{x_1}), \ldots, r_n(\vec{x_n})$. By induction, for $1 \leq i \leq |e|$, τ_{D_i} is the corresponding proof of the derivation D_i. Then by composition we obtain τ_D the proof for D.

□

We then show that the weight of each derivation of a tuple is equal to the corresponding proof tree weight in Datalog.

Lemma 9.6 ([Ramusat et al. 2021a])**.** *For any Datalog query q and grounding of an atom v of q, for any derivation D of v in H_q $w(D) = \bigotimes_{t' \in \mathrm{leaves}(\tau_D)} \mathsf{prov}_R^q(t')$ where τ_D is the proof tree corresponding to D in the bijection given by Lemma 9.5.*

Proof. By induction on the size of the derivation D:

- If $|D| = 1$ then, there exists a nullary edge $e \in E_q$ with $h(e) = v$ and $w(D) = f_v = \mathsf{prov}_R^q(r(\vec{x})) = \bigotimes_{t' \in \mathrm{leaves}(\tau_D)} \mathsf{prov}_R^q(t')$.
- If $|D| \geq 1$ then there exists $e \in E_q$ and D is of the form $\langle e, D_1 \cdots D_{|e|} \rangle$ with D_i a derivation of $T_i(e)$ for $1 \leq i \leq |e|$. We have $w(D) = f_e(w(D_1), \ldots, w(D_{|e|}))$ and by definition of $f_e = \otimes$ and by IHP $w(D) = \bigotimes_{t' \in \mathrm{leaves}(\tau_D)} \mathsf{prov}_R^q(t')$.

□

Finally, we obtain:

Theorem 9.7 ([Ramusat et al. 2021a])**.** *Let t be a tuple of a Datalog program q with output predicate G and H_q its hypergraph representation, then $\mathsf{prov}_G^q(t) = \delta_{H_q}(G(t))$.*

Proof.

$$
\begin{aligned}
\delta_{H_q}(t) &= \bigoplus_{D \in \mathcal{D}_{H_q}(G(t))} w(D) && \text{and by Lemma 9.6,} \\
&= \bigoplus_{D \in \mathcal{D}_{H_q}(G(t))} \left(\bigotimes_{t' \in \mathrm{leaves}(\tau_D)} \mathsf{prov}_R^q(t') \right) && \text{and by Lemma 9.5,} \\
&= \bigoplus_{\tau \text{ yields } t} \left(\bigotimes_{t' \in \mathrm{leaves}(\tau)} \mathsf{prov}_R^q(t') \right) && = \mathsf{prov}_G^q(t)
\end{aligned}
$$

□

10. Best-First Method

We start this section with a little reminder of semiring theory. Huang [2008] introduced the notion of *superiority* of a semiring S with respect to a partial order $\leq$, defined by: $\forall a, b \in S\ a \leq a \otimes b,\ b \leq a \otimes b$. We prove the natural order satisfies this notion for 0-closed semirings:

Lemma 10.1 ([Ramusat et al. 2021a])**.** *Let S be an idempotent semiring and $\leq$ the natural order over S. Then S is* superior *with respect to $\leq$ if and only if S is 0-closed.*

Knuth [1977] generalized the Dijkstra algorithm to what he calls the *grammar problem* (i.e., finding the *best-weight derivation* from a given non-terminal, where each terminal has a specific weight and each rule comes with an associated weight function). This has been identified as corresponding to the search problem in a monotonic superior hypergraph – for each $e \in H$, f_e is monotone and superior in each argument (see Table 3 in [Huang 2008]). Lemma 10.1 shows that superiority corresponds to 0-closedness in semirings with natural orders. The definition of the grammar problem assumes a total order on weights as the weights are real numbers. In the special case where all hyperedges are of arity 1 (and all weight functions bound to $\otimes$), we obtain the classical notion of semiring-based provenance for graph databases [Ramusat et al. 2021b]. Thus, Knuth's algorithm can be seen as a generalization to hypergraphs (and therefore, by the results of the previous section, to Datalog provenance computation) of the modified Dijkstra algorithm we presented in [Ramusat et al. 2021b], working on 0-*closed totally-ordered semirings*, which are generalizations of the tropical semiring.

10.1. Naïve Version and Optimizations

We present as Algorithm 9 the Best-first method, the reformulation of the Knuth's algorithm in terms of semiring-based Datalog provenance, the basis of the optimizations that will be introduced later.

Optimized version of Best-first method In the original paper of Knuth [1977], the question of efficient construction of the set of candidate facts for the extraction of the minimal-valued fact is not dealt with. A lot of redundant work may be carried out if the implementation is not carefully designed.

In the following, we show how to obtain a ready-to-be-implemented version incorporating ideas from the *semi-naïve* evaluation of Datalog programs. Semi-naïve evaluation of Datalog, as described in [Abiteboul et al. 1995, Chapter 13] introduces a number of

Algorithm 9 Naïve version of Best-first method for Datalog provenance [Ramusat et al. 2021a]

Input: Datalog query q, EDB D with provenance indications over a 0-closed totally-ordered semiring S.
Output: full Datalog provenance for the IDB of q over D.

1: $I \leftarrow \emptyset$
2: Let ν be the function that maps all facts of D to their annotation in S and all potential facts of the intensional schema of q to $\bar{0}$
3: **repeat**
4: **for** each intensional fact $r(\vec{x}) \notin I$ **do**
5: $\nu(r(\vec{x})) \quad \oplus= \bigotimes_{1 \leq i \leq n} \nu(r_i(\vec{x_i}))$ for each instantiation of a rule $r(\vec{x}) \leftarrow r_1(\vec{x_1}), \ldots, r_n(\vec{x_n})$ with $r_i(\vec{x_i}) \in D \cup I$
6: **end for**
7: Add to I the tuple $r(\vec{x_{\min}})$ such that $\nu(r(\vec{x_{\min}}))$ is $\leq$-minimal among all potential intensional facts $r(\vec{x})$ not in I
8: **until** $\nu(r(\vec{x_{\min}})) = \bar{0}$ or I contains all potential intensional facts
9: **return** $\nu_{|I}$

ideas aiming at improving the efficiency of the *naïve* Datalog evaluation method; we show how to leverage these tricks in our setting.

The *naïve* evaluation of a Datalog program q processes iteratively, applying at each step the *consequence operator* T_q. Many redundant derivations are computed, leading to practical inefficiency. The *semi-naïve* evaluation addresses this problem by considering only facts derived using at least one new fact found at the previous step. Note, however, whereas many new facts can be found at one step of the semi-naïve evaluation, only one is to be added by the Best-first method to respect the $\leq$-minimality ordering of added facts.

This algorithm starts by initializing the priority queue with IDB facts that are derivable from EDB facts. Then, at each step, the minimum valued-fact is added, and only derivations using this new fact are computed to update the value of the facts in I. This algorithm stops whenever the maximal value is reached for a candidate fact, or the list is empty (because the minimal value of the list is by default the maximal value of the semiring).

Theorem 10.2 ([Ramusat et al. 2021a])**.** *Algorithm 10 computes the full Datalog provenance for* 0*-closed totally-ordered semirings.*

Proof. We show the algorithm verifies the following invariant: whenever a tuple is added to I in Line 12, it has optimal value. This implies that I is populated in increasing order: each new derivation computed in the Relax procedure only updates the priority queues with values greater than the value of the tuple relaxed (by superiority of $\otimes$).

Suppose by contradiction that some output tuples are not correctly labeled and take such a minimal tuple $\nu = r(\vec{x})$. At the moment where ν is extracted with value n let

Algorithm 10 Basic semi-naïve version of Best-first method for Datalog provenance [Ramusat et al. 2021a]

Input: Datalog query q, EDB D with provenance indications over a 0-closed totally-ordered semiring S.

Output: full Datalog provenance for the IDB of q.

1: **def** RELAX($r_0(\vec{x_0})$, E):
2: **for** each instantiation of a rule $r(\vec{x}) \leftarrow r_1(\vec{x_1}), \ldots, r_m(\vec{x_m}), \cdots, r_n(\vec{x_n})$ where $r_i(\vec{x_i}) \in D \cup E \cup \{r_0(\vec{x_0})\}$, $1 \leq i < m$, $r_m(\vec{x_m}) = r_0(\vec{x_0})$ and $r_i(\vec{x_i}) \in D \cup E$, $m < i \leq n$ **do**
3: $\nu(r(\vec{x})) \quad \oplus= \bigotimes_{1 \leq i \leq n} r_i(\vec{x_i})$
4: **end for**
5:
6: $I \leftarrow \emptyset$
7: Let ν be the function that maps all facts of D to their annotation in S and all potential facts of the intensional schema of q to $\bar{0}$
8: **for** each intensional atom $r(\vec{x}) \notin I$ **do**
9: $\nu(r(\vec{x})) \quad \oplus= \bigotimes_{1 \leq i \leq n} r_i(\vec{x_i})$ for each instantiation of a rule $r(\vec{x}) \leftarrow r_1(\vec{x_1}), \ldots, r_n(\vec{x_n})$ with $r_i(\vec{x_i}) \in D$
10: **end for**
11: **while** $\min_{\nu \setminus I} r(\vec{x}) \neq \bar{0}$ **do**
12: Add such minimal $r(\vec{x})$ to I
13: RELAX($r(\vec{x})$, $I \setminus r(\vec{x})$)
14: **end while**
15: **return** ν

us consider an optimal derivation path of ν that leads to the optimum value $opt < n$. By superiority each tuple occurring in the tail of the rule has value less than opt. Thus a tuple occurring in the tail is either wrong-valued or not present in I at the moment where ν is found. In both cases and because tuples are added to I in increasing order we obtain a new minimal tuple incorrectly labeled by the algorithm, contradicting the hypothesis. □

The structure of the Datalog program can be analysed to provide clues about the predicates to focus on. Following [Abiteboul et al. 1995], we introduce the notion of *precedence graph* G_P of a Datalog program P. The nodes are the IDB predicates and the edges are pairs of IDB predicates (R, R') where R' occurs at the head of a rule of P with R belonging to the tail. P is a *recursive* program if G_P has a directed cycle. Two predicates R and R' are mutually recursive if $R = R'$ or R and R' participate in the same cycle of G_P. This defines equivalence classes. Putting it together, we obtain as Algorithm 11 our final algorithm. This approach reduces the load of the priority queue thus mitigating the cost of extracting the minimal element (Line 12 of Algorithm 10).

Algorithm 11 Semi-naïve version of Best-first method for Datalog provenance [Ramusat et al. 2021a]

Input: q a Datalog query with provenance indication over a 0-closed totally-ordered semiring S.
Output: full Datalog provenance for the IDB of q.

```
1: Compute the equivalence classes of q
2: for each equivalence class in a topological order do
3:     Apply Algorithm 10 over IDB predicates in the equivalence class
4:     considering previous equivalence classes as EDB predicates
5: end for
6: return ν
```

10.2. Generalization to Distributive Lattices

We outlined a new algorithm in Chapter 6 based on Dijkstra's algorithm and solving the single-source provenance in graph databases with provenance indications over 0-closed multiplicatively idempotent semirings (equivalents to distributive lattices). This new method stems from a tentative to bridge the strong complexity gap for computing the provenance in the case of a semiring not 0-closed and totally ordered. A similar gap also appears when we consider provenance over Datalog queries. Thus, we show how to apply this method for computing provenance for Datalog queries over distributive lattices.

We provide a brief review of the key ideas presented in Chapter 6. Any element of a distributive lattice is decomposable into a product of *join-irreducible* elements of the lattice, and there exists an embedding of the distributive lattice into a chain decomposition of its join-irreducible elements. This ensures we can 1) work on a totally ordered environment and apply algorithms that require total ordering over the elements, 2) independently compute partial provenance annotations for each dimension to form the final provenance annotation. Given m the number of dimensions in the decomposition, our solution (described in Algorithm 12) performs m launches of the Best-first method and thus, roughly has a cost increased by a factor m.

Algorithm 12 Generalized Best-first method for Datalog provenance [Ramusat et al. 2021a]

Input: q a Datalog query with provenance indication over a 0-closed multiplicatively idempotent semiring S.
Output: full Datalog provenance for the IDB of q.

```
1: for each EDB fact R(x⃗) do
2:     DECOMPOSE(R(x⃗))
3: end for
4: for each dimension i do
5:     ν_i ← BEST-FIRST(q, i)
6: end for
7: return RECOMPOSE(ν_1, ..., ν_n)
```

11. Implementation and Experiments

In numerous application domains, Datalog is used as a *domain specific language* (DSL) to express logical specifications, e.g. for static program analysis. A formal specification, written as a *declarative* Datalog program is usually translated into an efficient *imperative* implementation by a *synthesizer*. This process simplifies the development of program analysis compared to hand-crafted solutions (highly optimized C++ applications specialized in enforcing a fixed set of specifications). Soufflé [Jordan et al. 2016; Scholz et al. 2016] has been introduced to provide efficient synthesis of Datalog specifications to executable C++ programs, competing with state-of-the-art handcrafted code for program analysis. The inner workings of Soufflé were of interest to our work from [Ramusat et al. 2021a]; the algorithm implementations are similar to the evaluation strategy followed by the Best-first method we introduced here. We present a brief overview of the architecture of Soufflé and discuss how we extended it.

11.1. Architecture and Implementation

Following what is described in [Jordan et al. 2016], an input datalog program q goes through a staged specialization hierarchy. After parsing, the first stage of Soufflé specializes the semi-naïve evaluation strategy applied to q, yielding a relational algebra machine program (RAM). Such a program consists in basic relational algebra operations enriched with I/O operators and fix-point computations. As a final step, the RAM program is finally either interpreted or compiled into an executable. For this work, we have only used the interpreter. Our code was inserted in two different stages of Soufflé: a new *translation strategy* from the parsed program to the RAM program, a *priority queue*, replacing the code in charge of adding at run-time the tuples to the relations.

We showcase the result of our translation strategy in Algorithms 13, 14, and 15 for a Datalog query computing the transitive closure of a graph; this program is given in Algorithm 13 in its Soufflé syntax. Algorithm 14 presents the corresponding Soufflé RAM program resulting from applying the semi-naïve evaluation strategy and Algorithm 15 our modification to the RAM program to provide provenance annotation via the Best-first strategy and use the priority queue pq for provenance computation. The $\perp$ notation corresponds to a wildcard. Importantly, modifying directly at the RAM level of Soufflé allowed us to benefit of all implemented optimizations.

Algorithm 13 Input Datalog program computing the transitive closure (SOUFFLÉ syntax) [Ramusat et al. 2021a]

```
1: .decl edge(s:number, t:number[, @prov:semiring value])
2: .input edge
3: .decl path(s:number, t:number[, @prov:semiring value])
4: .output path
5: path(x, y) :- edge(x, y).
6: path(x, y) :- path(x, z), edge(z, y).
```

Algorithm 14 Corresponding SOUFFLÉ RAM program for Algorithm 13 [Ramusat et al. 2021a]

```
1: if ¬(edge = ∅) then
2:    for t0 in edge do
3:       add (t0.0, t0.1) in path
4:       add (t0.0, t0.1) in δpath
5:    end for
6: end if
7: loop
8:    if ¬(δpath = ∅) ∧ ¬(edge = ∅) then
9:       for t0 in δpath do
10:         for t1 in edge on index t1.0 = t0.1 do
11:            if ¬(t0.0, t0.1) ∈ path then
12:               add (t0.0, t0.1) in path'
13:            end if
14:         end for
15:      end for
16:   end if
17:   if path' = ∅ then
18:      exit
19:   end if
20:   for t0 in path' do
21:      add (t0.0, t0.1) in path
22:   end for
23:   swap δpath with path'
24:   clear path
25: end loop
```

Algorithm 15 Modification of RAM program of Algorithm 14 to implement Best-first strategy [Ramusat et al. 2021a]

```
1: if ¬(edge = ∅) then
2:    for t0 in edge do
3:       update (t0.0, t0.1, t0.prov) in path
4:    end for
5:    for t0 in path do
6:       add (t0.0, t0.1, t0.prov) in δpath
7:    end for
8: end if
9: loop
10:   if ¬(δpath = ∅) ∧ ¬(edge = ∅) then
11:      for t0 in δpath do
12:         for t1 in edge on index t1.0 = t0.1 do
13:            if ¬(t0.0, t1.1, ⊥) ∈ path then
14:               update (t0.0, t0.1, t0.prov ⊗ t1.prov) in pq
15:            end if
16:         end for
17:      end for
18:   end if
19:   clear δpath
20:   if pq is empty then
21:      exit
22:   end if
23:   add pq.top() in pq.top().relation and in pq.top().δrelation
24: end loop
```

11.2. Experiments

Our implementation was tested on an Intel Xeon E5-2650 computer with 176 GB of RAM. The source code is freely available on GITHUB[1].

The initial motivation for this work stemmed from a key observation we outlined at the end of Chapter 8. We pointed out the similarity between Datalog and the classes of semirings and their optimized provenance algorithms discussed in that work, focused on graph provenance algorithms. To translate this graph setting into Datalog, the graph structure has been encoded into an EDB with one binary predicate *edge* encoding the edges, and with edge annotations depending on the provenance semiring we chose. We run the *transitive closure* Datalog program outlined in Algorithm 13. Full information on the graph datasets used can be found in Chapter 7. We provide, in Figure 11.1, a comparison between the best-first method introduced here (SOUFFLÉ-PROV), the plain

[1] `https://github.com/yannramusat/souffle-prov`

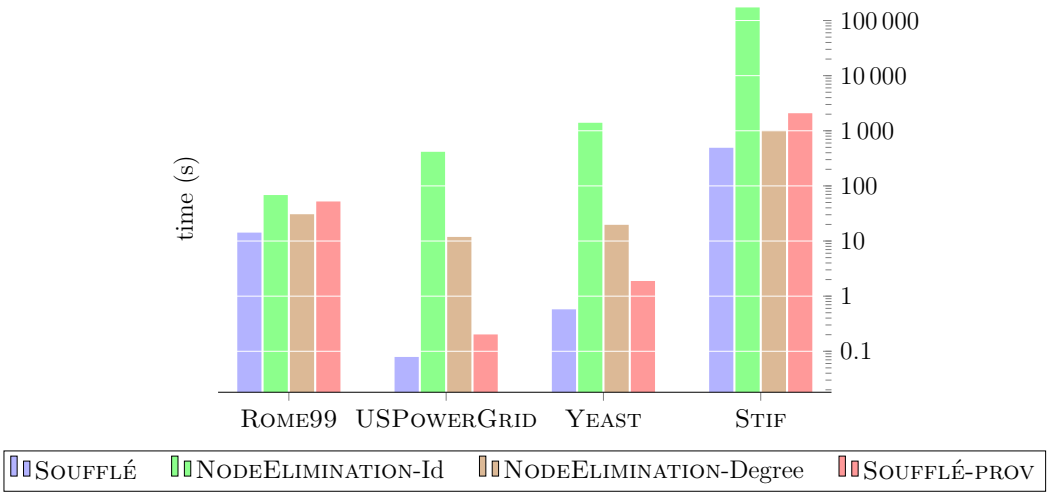


Figure 11.1.: Comparison between algorithms for all-pairs shortest-distances (Tropical). Values greater than 100 000 s are timeouts.

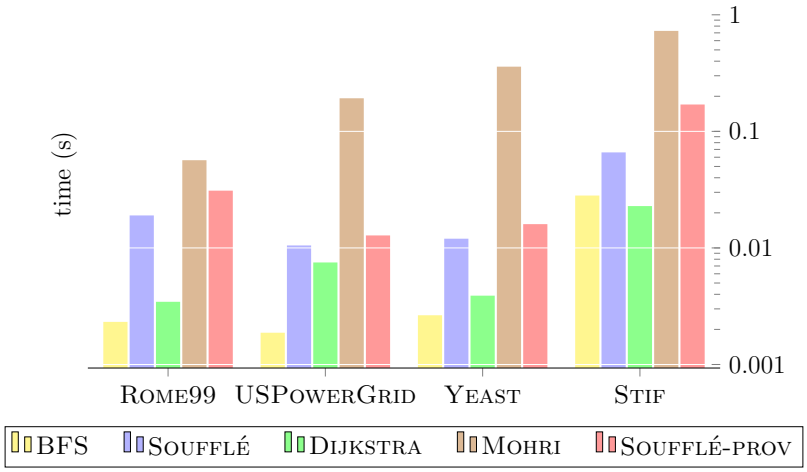
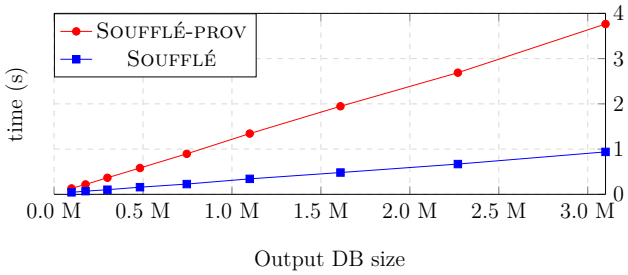
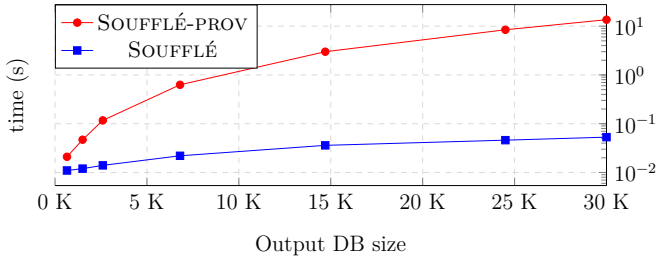


Figure 11.2.: Comparison between algorithms for single-source shortest-distances (Tropical).




```
ra(v, w, x, y, z)  :- p(v), p(w), p(x), p(y), p(z).
rb(v, w, x, y, z)  :- p(v), p(w), p(x), p(y), p(z).
q(v)               :- r(v, w, x, y, z).
r(v, w, x, y, z)   :- ra(v, w, x, y, z),
                      rb(v, w, x, y, z).
q(z)               :- r(v, w, x, y, z).
```

Figure 11.3.: Comparison between Soufflé and Soufflé-prov (IRIS)




```
hc(x,y)  :- hasChild(x, y).
imt(x,y) :- isMarriedTo(x, y).

hc(x,y)  :- imt(z,x), imt(z, y).
hc(x,y)  :- imt(x,z), imt(z, y).
imt(x, y) :- imt(y, x).
imt(x, y) :- hc(x, z), hc(y, z).
```

Figure 11.4.: Comparison between Soufflé and Soufflé-prov (AMIE)

Soufflé without provenance computation, and a previous provenance-based algorithm for graphs introduced in Chapter 5 computing all-pairs shortest-distances over graph databases (the NodeElimination algorithm, with a choice of node to eliminate based on its id or its degree), in the tropical semiring. Similarly, in Figure 11.2, we compare with previous solutions for single-source shortest-distances, in the same semiring, in particular the adaptation of Dijkstra algorithm described in Chapter 5, and, for comparison purposes, a Bfs algorithm that does not compute provenance. The main focus of this work was to provide an effective Datalog based solution for all-pairs provenance in graph databases. For the all-pairs problem, depending on the dataset, (see, e.g., Yeast), Soufflé-Prov is significantly faster than the previous best known algorithm, NodeElimination. Unsurprisingly, BFS and Dijkstra perform respectively better than Soufflé and Soufflé-prov in the single-source context. What favors both graph algorithms strongly is the fact that they reduce redundant computation: the algorithms abort whenever the target vertex has been reached. Soufflé-prov performs between 1 and 2 orders of magnitude faster than Mohri, yet another provenance-based algorithm from Chapter 5. This fact highlights the potential of adapting the best-first method to also handle k-closed semirings.

We now turn to evaluating the overhead induced by adding provenance computations to Datalog via Soufflé. This appears fairly modest in the experiments of Figures 11.1, 11.2. We further consider two datasets that go beyond the setting of graph databases: a non-recursive query over synthetic data (IRIS [Deutch et al. 2015], obtained from the authors of that paper) in Figure 11.3 and a program over a knowledge base (AMIE [Galárraga et al. 2013], available online[2]) populated with real-world data and automatic rules in Figure 11.4. The evaluation was made by varying the output database size. In IRIS, the overhead induced is a modest constant factor of approximately 4 times the original cost. This is not the case for AMIE, where the overhead tends to increase with the size of the answer. We conjecture this is due to the lack of native support for tuple updates in Soufflé (in Datalog, the fixed-point operator is inflationary); this leads to inefficiency in updating provenance values associated to each tuple.

[2] `https://bitbucket.org/amirgilad/selp/src/Journal/`

12. Reverse Translations and Opportunities

Our focus so far in the study from [Ramusat et al. 2021a] was to express Datalog programs into hypergraphs. We also considered the opposite direction: given a weighted hypergraph H with all its weight functions derived from a semiring S via its $\otimes$ operator, we aim to design an equivalent Datalog program which computes the provenance on the given semiring. We proposed two translations: one quite straightforward, and another one with a fixed program, to benefit of the low data complexity of Datalog.

12.1. From Weighted Hypergraphs to Datalog Programs

Definition 12.1 (Weighted hypergraph to Datalog program [Ramusat et al. 2021a])**.** *Given a hypergraph $H = \langle V, E\rangle$ of maximal hyperedge arity n, we define a Datalog query q_H with extensional predicates E_n of arity $n+1$ for $0 \leq i \leq n$ and a unary intensional predicate R. For each $1 \leq i \leq n$, we add a single rule of the form: $R(x) \leftarrow E_{i+1}(x, x_1, \ldots, x_n), R(x_1), \ldots, R(x_n)$, degenerating into $R(x) \leftarrow E_1(x)$ for $i = 0$. For each $e \in E$ of arity $n \geq 1$, we populate E_{n+1} with $E_{n+1}(h(e), \mathrm{T}(e))$ having provenance value $\bar{1}$. Since source vertices (i.e., heads of nullary edges) may have an initial constant value $s \in S$, for each $e \in E$ of arity 0 we tag $E_1(h(e))$ with the provenance value f_e.*

Let us start with a lemma showing the one-to-one correspondence between derivations in the hypergraph and proof trees in Datalog.

Lemma 12.2 ([Ramusat et al. 2021a])**.** *Given a vertex $v \in V$ of the hypergraph, $\mathcal{D}_H(v)$ is in one-to-one correspondance with $\{\tau \mid \tau \text{ yields } v\}$.*

Proof. By definition of q_H each rule corresponds to a unique hyperedge. Then, we can inductively construct for a given derivation D its associated (unique) Datalog proof τ_D:

- If $|D| = 1$, then v is a source vertex (head of a nullary hyperedge). Thus, the corresponding proof is the single instantiation of the rule $R(v) \leftarrow E_1(v)$.
- If $|D| \geq 1$, then it exists $e \in BS(v)$ where $|e| \geq 0$ and D_i is a derivation of $T_i(e)$ for $1 \leq i \leq |e|$, then $D = \langle e, D_1 \cdots D_{|e|}\rangle$ is a derivation of v. By definition, this edge corresponds to a single rule:

  $$R(v) \leftarrow E_{i+1}(v, T_1(e), \ldots, T_{|e|}(e)), R(T_1(e)), \ldots, R(T_{|e|}(e)).$$

  By IHP, for $1 \leq i \leq |e|$, τ_{D_i} is the corresponding proof of the derivation D_i. Then by composition we obtain τ_D the proof for D.

□

We then show the weight of each derivation of a tuple is equal to the corresponding proof weight in Datalog.

Lemma 12.3 ([Ramusat et al. 2021a]). $w(D) = \bigotimes_{t' \in \text{leaves}(\tau_D)} \mathsf{prov}^{q_H}_{E_1,\ldots,E_{n+1}}(t')$

Proof. By induction on the size of the derivation D:

- If $|D| = 1$ then, there exists a nullary edge $e \in E$ with $h(e) = v$ and $w(D) = f_e = \mathsf{prov}^{q_H}_{E_1}(e) = \bigotimes_{t' \in \text{leaves}(\tau_D)} \mathsf{prov}^{q_H}_{E_1,\ldots,E_{n+1}}(t')$.
- If $|D| \geq 1$ then it exists $e \in E$ and D is of the form $\langle e, D_1 \cdots D_{|e|} \rangle$ with D_i a derivation of $T_i(e)$ for $1 \leq i \leq |e|$. We have $w(D) = f_e(w(D_1), \ldots, w(D_{|e|}))$ and by definition of $f_e = \otimes$ and by IHP $w(D) = \bigotimes_{t' \in \text{leaves}(\tau_D)} \mathsf{prov}^{q_H}_{E_1,\ldots,E_{n+1}}(t')$.

□

And then we have:

Theorem 12.4 ([Ramusat et al. 2021a]). *Let v be a vertex of a weighted hypergraph H with all weight functions derived from the $\otimes$ operation of a semiring S and q_H its translation into a Datalog query, then $\delta_H(v) = \mathsf{prov}^{q_H}_R(v)$.*

Proof of Theorem 12.4.

$$\begin{aligned}
\delta_H(t) &= \bigoplus_{D \in \mathcal{D}_H(v)} w(D) && \text{and by Lemma 12.3,} \\
&= \bigoplus_{D \in \mathcal{D}_H(v)} \left(\bigotimes_{t' \in \text{leaves}(\tau_D)} \mathsf{prov}^{q_H}_{E_1,\ldots,E_{n+1}}(t') \right) && \text{and by Lemma 12.2,} \\
&= \bigoplus_{\tau \text{ yields } t} \left(\bigotimes_{t' \in \text{leaves}(\tau)} \mathsf{prov}^{q_H}_{E_1,\ldots,E_{n+1}}(t') \right) \quad = \quad \mathsf{prov}^{q_H}_R(t)
\end{aligned}$$

□

Let us know consider the case of programs where we have a fixed schema and arity:

Definition 12.5 (Weighted hypergraph to Datalog program with fixed schema [Ramusat et al. 2021a]). *Given an hypergraph $H = \langle V, E \rangle$ let us consider the following Datalog program q_{H_f} over unary predicates H, R and Nullary, binary predicates E, N, First and End, and ternary predicate Next:*

$$\begin{aligned}
&R(x) \leftarrow E(x, e), H(e) \\
&H(e) \leftarrow \mathit{First}(e, x), R(x), N(e, x) \qquad N(e, x) \leftarrow \mathit{Next}(e, x, y), R(y), N(e, y) \\
&\qquad \leftarrow \mathit{Nullary}(e) \qquad\qquad\qquad\qquad \leftarrow \mathit{End}(e, x)
\end{aligned}$$

For each $e \in E$ of arity $n \geq 1$, we populate E with $E(h(e), e)$ with provenance value $\bar{1}$. Let $x_1, \ldots, x_n$ be the elements of $\mathrm{T}(e)$), we populate First with $First(e, x_1)$, Next with $Next(e, x_i, x_{i+1})$ for $1 \leq i \leq n-1$ and End with $End(e, x_n)$; all have provenance value $\bar{1}$.

Source vertices (heads of nullary edges) may have an initial constant value $s \in S$, then for each $e \in E$ of arity 0 we tag $E(h(e), e)$ with the provenance value f_e and $Nullary(e)$ with provenance value $\bar{1}$.

The reasoning follows a similar flow:

Lemma 12.6 ([Ramusat et al. 2021a])**.** *Given a vertex $v \in V$ of the hypergraph, $\mathcal{D}_{H_f}(v) \simeq \{\tau \mid \tau \text{ yields } v\}$.*

Proof. We inductively construct for a given derivation D its associated (unique) Datalog proof τ_D:

- If $|D| = 1$, then v is a source vertex (head of a nullary hyperedge e). Thus, the corresponding proof is composed of $R(v) \leftarrow E(v, e), H(e)$ and $H(e) \leftarrow Nullary(e)$.
- If $|D| \geq 1$, then there exists $e \in \mathrm{BS}(v)$ where $|e| \geq 0$ and D_i is a derivation of $T_i(e)$ for $1 \leq i \leq |e|$, then $D = \langle e, D_1 \cdots D_{|e|} \rangle$ is a derivation of v. Let $v_1, \ldots, v_n$ be the elements of $\mathrm{T}(e)$), this hyperedge corresponds to the following proof tree:

$$\begin{array}{lcl} R(v) & \leftarrow & E(v,e), H(e) \\ H(e) & \leftarrow & First(e, v_1), R(v_1), N(e, v_1) \\ For\ 1 \leq i \leq n-1: & & \\ N(e, v_i) & \leftarrow & Next(e, v_i, v_{i+1}), R(v_{i+1}), N(e, v_{i+1}) \\ N(e, v_n) & \leftarrow & End(e, v_n) \end{array}$$

  By IHP, for $1 \leq i \leq |e|$, τ_{D_i} is the corresponding proof of the derivation D_i (derivation of the fact $R(v_i)$). Then, by composition, we obtain τ_D the proof for D.

□

Lemma 12.7 ([Ramusat et al. 2021a])**.** $w(D) = \bigotimes_{t' \in \text{ leaves}(\tau_D)} \mathsf{prov}^{q_{H_f}}_{E,Nullary,First,Next,End}(t')$

Proof. By induction on the size of the derivation D:

- If $|D| = 1$ then there exists a nullary edge $e \in E$ with $h(e) = v$ and its associated (unique) derivation is $R(v) \leftarrow E(v, e), H(e)$ and $H(e) \leftarrow Nullary(e)$. Thus $w(D) = f_e = \mathsf{prov}^{q_{H_f}}_{E}(v, e) \otimes \mathsf{prov}^{q_{H_f}}_{Nullary}(e)$.
- If $|D| \geq 1$ then there exists $e \in E$ and D is of the form $\langle e, D_1 \cdots D_{|e|} \rangle$ with D_i a derivation of $T_i(e)$ for $1 \leq i \leq |e|$. The derivation tree given in Lemma 12.6 corresponding to the hyperedge has provenance:

$$\begin{aligned} &\mathsf{prov}^{q_{H_f}}_{E}(v, e) \otimes \mathsf{prov}^{q_{H_f}}_{First}(e) \otimes \mathsf{prov}^{q_{H_f}}_{R}(v_1) \otimes \\ &\bigotimes_{1 \leq i \leq n-1} \left(\mathsf{prov}^{q_{H_f}}_{Next}(e, v_i, v_{i+1}) \otimes \mathsf{prov}^{q_{H_f}}_{R}(v_{i+1}) \right) \otimes \mathsf{prov}^{q_{H_f}}_{End}(e, v_n) \end{aligned}$$

By IHP, for $1 \leq i \leq |e|$, $w(D_i) = \mathsf{prov}_R^{q_{H_f}}(v_i)$, thus, we obtain

$$w(D) = \bigotimes_{t' \in \text{ leaves}(\tau_D)} \mathsf{prov}_{E,Nullary,First,Next,End}^{q_{H_f}}(t').$$

□

Theorem 12.8 ([Ramusat et al. 2021a])**.** *Let v be a vertex of a weighted hypergraph H with all weight functions derived from the $\otimes$ operation of a semiring S and q_{H_f} its translation into a Datalog query with fixed schema, then $\delta_{H_f}(v) = \mathsf{prov}_R^{q_{H_f}}(v)$.*

Proof of Theorem 12.8. We note EDB = $\{E, Nullary, First, Next, End\}$.

$$\begin{aligned}
\delta_{H_f}(t) &= \bigoplus_{D \in \mathcal{D}_{H_f}(v)} w(D) && \text{and by Lemma 12.7,} \\
&= \bigoplus_{D \in \mathcal{D}_{H_f}(v)} \left(\bigotimes_{t' \in \text{ leaves}(\tau_D)} \mathsf{prov}_{\text{EDB}}^{q_{H_f}}(t') \right) && \text{and by Lemma 12.6,} \\
&= \bigoplus_{\tau \text{ yields } t} \left(\bigotimes_{t' \in \text{ leaves}(\tau)} \mathsf{prov}_{\text{EDB}}^{q_{H_f}}(t') \right) = \mathsf{prov}_R^{q_{H_f}}(t)
\end{aligned}$$

□

12.2. Case Study: AND/OR Graphs

Section 4.3 of [Huang 2008] showcases some formalisms in which equivalent hypergraphs can be constructed in the dynamic programming framework proposed in their work. One advantage is that we can directly benefit from these translations in our Datalog provenance framework. However, there are some issues with these translations, as we now discuss.

One example formalism that can be easily translated to our setting are the AND/OR graphs, a special case of graphs – e.g., used in scheduling tasks – in which nodes can express two types of restrictions: AND restrictions (in which tasks have to executed sequentially) and OR restrictions (in which tasks can be overlapping). These two operations, not unlike the $\otimes$ and $\oplus$ operations on semirings, suggest our translation strategy is readily usable. Indeed, by our translation, vertices of the hypergraph would represent OR vertices and the hyperedges correspond to the AND vertices. This translation does not add any other restriction to the shape of the AND/OR graph. AND and OR vertices can have multiples outgoing edges and the AND/OR graph is not necessarily bipartite. However, not all applications in which AND/OR graphs are useful can be used with semiring operations. For instance, scheduling using AND/OR graphs [Adelson-Velsky et al. 2002; Dinitz et al. 2011] requires three operations (`min`, `max` and $+$) instead of the two allowed using semirings.

Another crucial issue, not obvious at first glance, is that when expressing these graphs into our Datalog framework the semantics can change. As explained in [Dinitz et al.

2011], zero-edge cycles express mutual events (occurring in groups, with no causal relations): either they all occur together or none of them do. In a deductive system, where semantics is given by a least fixed-point, they are believed to not occur at all. The semantics of Datalog provenance or derivation trees cannot express this situation: in Datalog there would be no finite proof of a set of given facts. However, adding all of these facts to the solution would not break the deductive rules; simply they do not belong to the least fixed-point. Thus we wonder if using a carefully choosen ω-complete semiring under the proof-theoretic definition for Datalog provenance could provide sound semantics for this case.

Perspectives

In conclusion to the research we pursued during this PhD, we emphasize some of the issues that are left open and the resulting research opportunities.

We surveyed in the preliminaries a considerable amount of technical results having strong impact on our work. We have seen that ω-continuous structures connect an infinite summation operator given by an ω-complete structure to the natural order. This has notably been pointed out in [Karner 1992]; their work also provides a characterization of continuous structures, without mentioning any order (see Proposition 5.6 therein). This structure permits to give a sound theoretical basis to the notion of least fixpoint equations involving semirings. One question is if all ordered semirings can be extended to a continuous structure. Theorem 2.37 has shown this is almost true by proving that each ordered semiring can be embedded into a finitary semiring. We recall that a finitary semiring need not be necessarily continuous (e.g., [Karner 1992, Fact 5.1]). If all naturally ordered semirings can be embedded into a continuous structure, it would surely simplify our task: we would have to only exhibit the natural order to make sure to conform to the specifications of our model. This is certainly easier than finding an embedding into a structure and then proving the new structure to be continuous.

While working on the provenance model for graph databases, we underlined two points of future focus. We established Corollary 8.1, which concerns only the full matrix asteration. It could be interesting to consider lower bounds on the computation of a restricted subset of the asteration matrix (e.g., one single row for single-source, only one cell for single-pair). We observed it was rarely possible to obtain an algorithm such as Dijkstra's, being able to solve only one row of the asteration matrix. This requires a specific set of properties over the semiring. In the general case (for ω-complete semirings) it seems that computing only a subset of the cells of the asteration matrix is at least as difficult as computing the full asteration matrix. To the best of our knowledge, no result formally stating this fact has ever appeared in the literature. The interest in our setting would be to obtain formal lower bounds for the single-source or single-pair provenance. On the other hand, devising parallel algorithms built upon the notion of eliminants [Abdali 1994] to provide an efficient computation of the full path provenance (understood as the asteration of a matrix over a star semiring) seems quite promising, and could lead to efficient solutions to compute the provenance over large transportation networks. Existing solutions in the spirit of GraphBlas [Kepner et al. 2011], a framework for expressing efficient parallel algorithms over large sparse graphs in the language of linear algebra (notably involving semirings to describe the operations to be performed) are natural candidates for designing an efficient parallel implementation of the full path provenance in general semirings. Nevertheless, priority queue management is not trivially expressible into linear algebra operators, and also not easily amenable to parallelism. Thus,

the provenance-aware algorithms we have focused on in our experiments such as Dijkstra, MultiDijkstra, or even Mohri are not subject to an implementation based on GraphBlas (or any equivalent).

While working with Datalog provenance, we have shown in Section 12.2 a possible application of the provenance model where the fixed-point semantics seems to not be relevant. Describing a carefully chosen ω-complete semiring under the proof-theoretic definition for Datalog provenance to provide sound semantics for AND/OR graphs could make a case for generalizing the provenance model of [Green et al. 2007] to other semirings structures, but at the cost of losing the equivalence between the proof-theoretic and fixed-point semantics. Another issue has been raised concerning the optimization of our implementation. The internals of Soufflé, targeting the inflationary computation of the fixed-point operator lack support for updating tuples. We conjecture we could mitigate the overhead induced by provenance computations within Soufflé-prov by adding primitives in their data structures. What remains to be established is to what extent these data structures [Jordan et al. 2016] could be extended to handle updates, without reducing the efficiency of Soufflé's current set of operators.

Acknowledgments This work has been funded by the French government under management of Agence Nationale de la Recherche as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

Bibliography

Abdali, S. Kamal (1994). "Parallel Computations in *-Semirings". In: *Computational Algebra*. Chap. 1, pp. 1–16.

Abdali, S. Kamal and B. David Saunders (1985). "Transitive closure and related semiring properties via eliminants". In: *Theoretical Computer Science* 40, pp. 257–274.

Abiteboul, Serge, Richard Hull, and Victor Vianu (1995). *Foundations of Databases*. Addison-Wesley.

Abraham, Ittai et al. (2016). "Highway Dimension and Provably Efficient Shortest Path Algorithms". In: *J. ACM* 63.5.

Adelson-Velsky, George M., Alexander Gelbukh, and Eugene Levner (2002). "On fast path-finding algorithms in AND-OR graphs". In: *Math. Problems in Engineering*.

Aho, Alfred V., John E. Hopcroft, and Jeffrey D. Ullman (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley.

Arden, Dean N. (1961). "Delayed-logic and finite-state machines". In: *SWCT*, pp. 133–151.

Barceló, Pablo (2013). "Querying Graph Databases". In: *PODS*, pp. 175–188.

Bellman, Richard E. (1958). "On a routing problem". In: *Quarterly of Applied Mathematics* 16.1, pp. 87–90.

Birkhoff, Garrett (1937). "Rings of sets". In: *Duke Math. J.* 3.3, pp. 443–454.

Bistarelli, Stefano, Ugo Montanari, and Francesca Rossi (1997). "Semiring-based constraint satisfaction and optimization". In: *J. ACM* 44.2, pp. 201–236.

Brzozowski, Janusz A. and Edward J. McCluskey (1963). "Signal Flow Graph Techniques for Sequential Circuit State Diagrams". In: *IEEE Trans. Electr. Comp.* EC-12.2, pp. 67–76.

Cheney, James, Laura Chiticariu, and Wang-Chiew Tan (2009). "Provenance in Databases: Why, How, and Where". In: *Found. Trends Databases* 1.4, pp. 379–474.

Codd, Edgar F. (1970). "A Relational Model of Data for Large Shared Data Banks". In: *Commun. ACM* 13.6, pp. 377–387.

Conway, John H. (1971). *Regular algebra and finite machines*. Chapman and Hall.

Cormen, Thomas H., Charles E. Leiserson, and Ronald L. Rivest (1989). *Introduction to Algorithms*. 1st Ed. MIT Press.

Cui, Yingwei, Jennifer Widom, and Janet L. Wiener (2000). "Tracing the Lineage of View Data in a Warehousing Environment". In: *ACM Trans. Database Syst.* 25.2, 179–227.

Deutch, Daniel, Amir Gilad, and Yuval Moskovitch (2015). "selP: Selective tracking and presentation of data provenance". In: *ICDE*, pp. 1484–1487.

Deutch, Daniel, Amir Gilad, and Yuval Moskovitch (2018). "Efficient provenance tracking for datalog using top-k queries". In: *The VLDB Journal* 27, pp. 245–269.

Deutch, Daniel et al. (2014). "Circuits for Datalog Provenance". In: *ICDT*, pp. 201–212.

Dilworth, Robert P. (1950). "A Decomposition Theorem for Partially Ordered Sets". In: *Annals of Mathematics* 51.1, pp. 161–166.

Dinitz, Yefim et al. (2011). "An $O(|V||E|)$ Algorithm for Scheduling with AND/OR Precedence Constraints". In: *Haifa Workshops*.

Droste, Manfred, Werner Kuich, and Heiko Vogler (2009). *Handbook of Weighted Automata*. Springer.

Eilenberg, Samuel (1974). *Automata, Languages, and Machines*. Vol. A. Academic Press.

Esparza, Javier and Michael Luttenberger (2011). "Solving fixed-point equations by derivation tree analysis". In: *CALCO*, pp. 19–35.

Esparza, Javier, Michael Luttenberger, and Maximilian Schlund (2014). "FPsolve: A Generic Solver for Fixpoint Equations over Semirings". In: *CIAA*, pp. 1–15.

Floyd, Robert W. (1962). "Algorithm 97: Shortest Path". In: *Commun. ACM* 5.6, p. 345.

Fuhr, Norbert and Thomas Rölleke (1997). "A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems". In: *ACM Trans. Inf. Syst.* 15.1, 32–66.

Galárraga, Luis et al. (2013). "AMIE: Association rule mining under incomplete evidence in ontological knowledge bases". In: *WWW*, pp. 413–422.

Goldstern, Martin (2002). *Completion of semirings*. arXiv: `math/0208134 [math.RA]`.

Green, Todd J., Grigoris Karvounarakis, and Val Tannen (2007). "Provenance Semirings". In: *PODS*, pp. 31–40.

Huang, Liang (2008). "Advanced Dynamic Programming in Semiring and Hypergraph Frameworks". In: *COLING*, pp. 1–18.

Imieliński, Tomasz and Witold Lipski (1984). "Incomplete Information in Relational Databases". In: *J. ACM* 31.4, 761–791.

Jordan, Herbert, Bernhard Scholz, and Pavle Subotić (2016). "Soufflé: On Synthesis of Program Analyzers". In: *Computer Aided Verification*, pp. 422–430.

Karner, Georg (1992). "On limits in complete semirings." In: *Semigroup forum* 45.2, pp. 148–165.

Kepner, Jeremy and John Gilbert (2011). *Graph Algorithms in the Language of Linear Algebra*. SIAM.

Khamis, Mahmoud A. et al. (2021). "Convergence of Datalog over (Pre-) Semirings". In: *CoRR* abs/2105.14435. arXiv: `2105.14435`.

Knuth, Donald E. (1977). "A generalization of Dijkstra's algorithm". In: *Information Processing Letters* 6.1.

Kozen, Dexter (1990). "On kleene algebras and closed semirings". In: *Math. Found. of Comp. Sc.* Pp. 26–47.

Krob, Daniel (1987). "Monoides et semi-anneaux complets." In: *Semigroup forum* 36, pp. 323–340.

Krob, Daniel (1988). "Monoides et semi-anneaux continus." In: *Semigroup forum* 37.1, pp. 59–78.

Kuich, Werner (1997). "Semirings and Formal Power Series". In: *Handbook of Formal Languages*. Springer, pp. 609–677.

Kumar, Srijan et al. (2016). "Edge weight prediction in weighted signed networks". In: *ICDM*, pp. 221–230.

Kumar, Srijan et al. (2018). "Rev2: Fraudulent user prediction in rating platforms". In: *WSDM*, pp. 333–341.

Lehmann, Daniel J. (1977). "Algebraic structures for transitive closure". In: *Theoretical Computer Science* 4.1, pp. 59–76.

Maggs, Bruce M. and Serge A. Plotkin (1988). "Minimum-cost spanning tree as a path-finding problem". In: *Information Processing Letters* 26.6, pp. 291–293.

Maniu, Silviu, Yann Ramusat, and Pierre Senellart (2020). *Graph Provenance*. URL: `https://bitbucket.org/smaniu/graph-provenance`.

Maniu, Silviu, Pierre Senellart, and Suraj Jog (2019). "An Experimental Study of the Treewidth of Real-World Graph Data". In: *ICDT*. Vol. 127, 12:1–12:18.

Mehlhorn, Kurt (1984). *Data Structures and Algorithms: Graph Algorithms and NP-Completeness*. Vol. 2. Springer.

Minoux, Michel and Michel Gondran (2008). *Graphs, Dioids and Semirings. New Models and Algorithms*. Springer.

Mohri, Mehryar (2002). "Semiring Frameworks and Algorithms for Shortest-Distance Problems". In: *J. Autom. Lang. Comb.* 7.3, pp. 321–350.

Ramusat, Yann (2019). "Provenance-Based Routing in Probabilistic Graph Databases". In: *PhD@VLDB*. Vol. 2399.

Ramusat, Yann (2021). *Semiring-Based Provenance within Soufflé*. Version 0.0.1. URL: `https://github.com/yannramusat/souffle-prov`.

Ramusat, Yann, Silviu Maniu, and Pierre Senellart (2018). "Semiring Provenance over Graph Databases". In: *TaPP*.

Ramusat, Yann, Silviu Maniu, and Pierre Senellart (2020). "Algorithmes à base de provenance pour des requêtes enrichies sur les bases de données graphes". In: *BDA*.

Ramusat, Yann, Silviu Maniu, and Pierre Senellart (2021a). *A Practical Dynamic Programming Approach to Datalog Provenance Computation*. arXiv: `2112.01132`.

Ramusat, Yann, Silviu Maniu, and Pierre Senellart (2021b). "Provenance-Based Algorithms for Rich Queries over Graph Databases". In: *EDBT*.

Rote, Günter (1990). "Path Problems in Graphs". In: *Computational Graph Theory*, pp. 155–189.

Sakarovitch, Jacques (1987). "Kleene's theorem revisited". In: *Trends, Techniques, and Problems in Theoretical Computer Science*, pp. 39–50.

Scholz, Bernhard et al. (2016). "On Fast Large-Scale Program Analysis in Datalog". In: *International Conference on Compiler Construction*, 196–206.

Senellart, Pierre (2017). "Provenance and Probabilities in Relational Databases: From Theory to Practice". In: *SIGMOD* 46.4.

Senellart, Pierre (2019). "Provenance in Databases: Principles and Applications". In: *Reasoning Web*.

Siggers, Mark (2020). "On the representation of finite distributive lattices". In: *Kyungpook Math. J.* 60.1, pp. 1–20.

Tarjan, Robert E. (1981). "A Unified Approach to Path Problems". In: *J. ACM* 28.3, 577–593.

Warshall, Stephen (1962). "A Theorem on Boolean Matrices". In: *J. ACM* 9.1, pp. 11–12.

RÉSUMÉ

L'augmentation du volume de données collectées par des capteurs et générées par des interactions humaines a mené à l'utilisation des bases de données orientées graphes en tant que modèle de représentation efficace pour les données complexes. Les techniques permettant de tracer les calculs qui ont été appliqués aux données au sein d'une base de données relationnelle classique sont sur le devant de la scène, notamment grâce à leur utilité pour faire respecter les régulations sur les données privées telles que le RGPD en Union Européenne. Notre travail de recherche croise ces deux problématiques en s'intéressant à un modèle de provenance à base de semi-anneaux pour les requêtes navigationnelles. Nous commençons par présenter une étude approfondie de la théorie des semi-anneaux et de leurs applications au sein des sciences informatiques en se concentrant sur les résultats ayant un intérêt direct pour notre travail de recherche. La richesse de la littérature sur le domaine nous a notamment permis d'obtenir une borne inférieure sur la complexité de notre modèle. Dans une seconde partie, nous étudions le modèle en lui-même et introduisons un ensemble cohérent d'algorithmes permettant d'effectuer des calculs de provenance et adaptés aux propriétés des semi-anneaux utilisés. Nous introduisons notablement une nouvelle méthode basée sur la théorie des treillis permettant de calculer la provenance pour des requêtes complexes. Nous proposons une implémentation open-source de ces algorithmes et faisons une étude expérimentale sur de larges réseaux de transport issus de la vie réelle pour attester de l'efficacité pratique de notre approche. On s'intéresse finalement au positionnement de ce cadre de travail par rapport à d'autres modèles de provenance à base de semi-anneaux. Nous nous intéressons à Datalog en particulier. Nous démontrons que les méthodes que nous avons développées pour les bases de données orientées graphes peuvent se généraliser sur des requêtes Datalog. Nous montrons de plus qu'elles peuvent être vues comme des généralisations de la méthode semi-naïve. En se basant sur ce fait-là, nous étendons les capacités de SOUFFLÉ, un évaluateur Datalog appartenant à l'état de l'art, afin d'effectuer des calculs de provenance pour des requêtes Datalog. Les études expérimentales basées sur cette implémentation open-source confirment que cette approche reste compétitive avec les solutions spécifiques pour les graphes, mais tout en étant plus générale. Nous terminons par une discussion sur les améliorations possibles du modèle et énonçons les questions ouvertes qui ont été soulevées au cours de ce travail.

MOTS CLÉS

Provenance, Graphes, Bases de données, Semi-anneaux, Datalog

ABSTRACT

The growing amount of data collected by sensors or generated by human interaction has led to an increasing use of graph databases, an efficient model for representing intricate data. Techniques to keep track of the history of computations applied to the data inside classical relational database systems are also topical because of their application to enforce Data Protection Regulations (e.g., GDPR). Our research work mixes the two by considering a semiring-based provenance model for navigational queries over graph databases. We first present a comprehensive survey on semiring theory and their applications in different fields of computer sciences, geared towards their relevance for our context. From the richness of the literature, we notably obtain a lower bound for the complexity of the full provenance computation in our setting. In a second part, we focus on the model itself by introducing a toolkit of provenance-aware algorithms, each targeting specific properties of the semiring of use. We notably introduce a new method based on lattice theory permitting an efficient provenance computation for complex graph queries. We propose an open-source implementation of the above-mentioned algorithms, and we conduct an experimental study over real transportation networks of large size, witnessing the practical efficiency of our approach in practical scenarios. We finally consider how this framework is positioned compared to other provenance models such as the semiring-based Datalog provenance model. We make explicit how the methods we applied for graph databases can be extended to Datalog queries, and we show how they can be seen as an extension of the semi-naïve evaluation strategy. To leverage this fact, we extend the capabilities of SOUFFLÉ, a state-of-the-art Datalog solver, to design an efficient provenance-aware Datalog evaluator. Experimental results based on our open-source implementation entail the fact this approach stays competitive with dedicated graph solutions, despite being more general. In a final round, we discuss on some research ideas for improving the model, and state open questions raised by our work.

KEYWORDS

Provenance, Graphs, Databases, Semirings, Datalog