



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΕΛΟΠΟΝΝΗΣΟΥ
UNIVERSITY *of the* PELOPONNESE

Ph.D. Thesis

Social Data Mining and Spatial Data Management for Marketing Applications on Web

Author:

Konstantinos Theocharidis

Supervisor:

Professor Spiros Skiadopoulos

April 9, 2022

Κοινωνική Εξόρυξη Δεδομένων και Χωρική Διαχείριση Δεδομένων για Εφαρμογές Μάρκετινγκ στο Διαδίκτυο

Διδακτορική Διατριβή
του
Κωνσταντίνου Θεοχαρίδη

Διπλωματούχου Ηλεκτρονικών Μηχανικών και Μηχανικών Υπολογιστών Πολυτεχνείου
Θεσσαλίας (2012) και MSc Πολυτεχνείου Θεσσαλίας (2013)

Συμβουλευτική Επιτροπή: Σπύρος Σκιαδόπουλος Επιβλέπων καθηγητής
Χρήστος Τρυφωνόπουλος
Θεόδωρος Δαλαμάγκας

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την .../.../2022

Όνομα	Βαθμίδα	Τόρυμα
Σπύρος Σκιαδόπουλος	Καθηγητής	Παν. Πελοποννήσου
Κωνσταντίνος Βασιλάκης	Καθηγητής	Παν. Πελοποννήσου
Θεόδωρος Δαλαμάγκας	Ερευνητής Α	ΙΠΣΥ / Ε.Κ. ΑΘΗΝΑ
Χρήστος Τρυφωνόπουλος	Αν. Καθηγητής	Παν. Πελοποννήσου
Μανώλης Τερροβίτης	Ερευνητής Β	ΙΠΣΥ / Ε.Κ. ΑΘΗΝΑ
Παναγιώτης Καρράς	Αν. Καθηγητής	Παν. Ωρχους
Γιάννης Λιαγούρης	Επ. Καθηγητής	Παν. Βοστώνης

Copyright © Κωνσταντίνος Θεοχαρίδης, 2022.

Διδάκτωρ τμήματος Πληροφορικής και Τηλεπικοινωνιών, Παν. Πελοποννήσου.

Με επιφύλαξη παντός δικαιώματος - All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας διατριβής, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη διατριβή για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Η έγκριση της διδακτορικής διατριβής από το Πανεπιστήμιο Πελοποννήσου δε δηλώνει αποδοχή των απόψεων του συγγραφέα.

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Δρ Σπύρο Σκιαδόπουλο, για την πολύτιμη βοήθεια του καθ'όλη τη διάρκεια της διδακτορικής διατριβής μου. Ιδιαίτερα, επίσης, ευχαριστώ τον Δρ Μανώλη Τερροβίτη από το Ινστιτούτο Πληροφοριακών Συστημάτων του Ερευνητικού Κέντρου Αθηνά, τον Δρ Παναγιώτη Καρρά από το Πανεπιστήμιο του Όρχους, και τον Δρ. Γιάννη Λιαγούρη από το Πανεπιστήμιο της Βοστώνης, για την συνεργασία μου μαζί τους στο περιεχόμενο της διατριβής μου. Τέλος, πάνω απ'όλα ευχαριστώ θερμά την οικογένεια μου για την αδιάκοπη στήριξη.

Abstract

Nowadays, there is an increasing need for brands (stakeholders) to effectively and efficiently connect with their customers in both spatial and social domains so as to grow their revenues. In spatial field, there is a variety of location-based services (e.g., Google Maps, Uber, Foursquare) for utilization by brands, whereas in social area, there are several social networks (e.g., Facebook, Instagram, VK) in which brands can maintain their own social network pages for advertising. In this thesis, we propose and study fundamental social and spatial data operations that can significantly contribute to successful connection among brands and customers.

In spatial domain, our contribution is the development of a spatial RDF system, named SRX and built upon the popular RDF-3X store, that provides efficient spatial RDF data operations. In particular, SRX supports three types of *spatial queries*: range selections (e.g. find entities within a given polygon), spatial joins (e.g. find pairs of entities whose locations are close to each other), and spatial k nearest neighbors (e.g. find the three closest entities from a given location). Further, SRX supports *spatial updates* (e.g., insertions or deletions of RDF triples). SRX relies its good performance on a grid-scheme that approximates the geometries of the RDF spatial entities inside their integer IDs. We extensively evaluate the performance of SRX for both queries and updates by comparing it with RDF-3X, Virtuoso, GraphDB, and Strabon on LGD and YAGO datasets. Results show that SRX outperforms other systems for queries and incurs a little overhead to RDF-3X for updates.

In social domain, we contribute by studying three novel content-aware recommendation problems relative to the *Influence Maximization* (IM) problem that asks for the k users who can maximize the influence of a *fixed* post in a social network.

The first problem we introduce, named *Content-Aware Influence Maximization* (CAIM), is the inverse variant of IM and asks for the k features that can form the content of a *tunable* post so as the latter to become viral in a social network, starting its diffusion from a fixed set of initial adopters (*subscribers* of brand's social network page). We prove that CAIM does not have influence guarantees, and our experimental results on Gnutella and VK datasets showing that our deployed heuristic algorithm is more practical than baseline and conventional greedy approaches for CAIM.

The second problem we propose is an adaptive (online) version of CAIM, named *Adaptive Content-Aware Influence Maximization* (ACAIM), that takes place in many rounds. In each round, the content of a post is sought, comprising k features, and the influence feedback of posts in previous rounds is utilized for the content decision of posts in next rounds. The target is to maximize the cumulative influence over all rounds. To solve ACAIM, we integrate *Online Learning To Rank* (OLR) techniques to our machine learning IM framework, and we further deploy one propagation model, one simulator running that model to generate realistic feedback, three ACAIM learners, and a thorough experimental study on various VK datasets for several brands.

Finally, the third problem we study relates with how brands can maximize their *subscription* (instead of *influence* as happens in CAIM and ACAIM) in social networks. Specifically, we propose a *content recommendation policy* to brands for *Gaining Subscribers by Messaging* (GSM). The GSM problem applies in many rounds. In each round, GSM recommends to brands what content (consisting of k features) to publish in their social network pages and which m users to notify of that content. The aim is to maximize the cumulative subscription gain over all rounds. We develop three GSM solvers, and by conducting a rich experimental evaluation on different VK datasets, we ascertain the importance and practical value of GSM.

Περίληψη

Στις μέρες μας, υπάρχει ολοένα και μεγαλύτερη ανάγκη για τις εταιρείες να συνδέονται αποτελεσματικά και αποδοτικά με τους πελάτες τους, και σε χωρικό και σε κοινωνικό επίπεδο, έστι αώστε να αυξήσουν τα έσοδα τους. Στο χωρικό πεδίο, υπάρχουν διάφορες εφαρμογές που κάνουν χρήση τοποθεσίας (π.χ., Google Maps, Uber, Foursquare) από τις οποίες μπορούν να επωφεληθούν οι εταιρείες, ενώ στο κοινωνικό πεδίο, υπάρχουν αρκετά κοινωνικά δίκτυα (π.χ., Facebook, Instagram, VK) στα οποία οι εταιρείες μπορούν να διατηρούν τις κοινωνικές σελίδες τους για διαφημιστικούς λόγους. Στη διατριβή αυτή, παρουσιάζουμε θεμελιώδεις κοινωνικές και χωρικές λειτουργίες δεδομένων που μπορούν σημαντικά να συνεισφέρουν στην επιτυχή σύνδεση των εταιρειών με τους πελάτες τους.

Στο χωρικό πεδίο, η συνεισφορά μας έγκειται στην ανάπτυξη ενός χωρικού RDF συστήματος, με όνομα SRX και βασισμένο στο δημοφιλές RDF-3X σύστημα, το οποίο παρέχει αποδοτικές λειτουργίες χωρικών RDF δεδομένων. Συγκεκριμένα, το SRX υποστηρίζει τρία είδη χωρικών ερωτημάτων: επιλογές εμβέλειας (π.χ., βρείτε τις οντότητες εντός ενός πολυγώνου), χωρικές ενώσεις (π.χ., βρείτε ζευγάρια οντοτήτων που έχουν κοντινές τοποθεσίες), και *k* κοντινότερων χωρικών γειτόνων (π.χ., βρείτε τις τρεις πλησιέστερες οντότητες σε μια τοποθεσία). Επιπλέον, το SRX υποστηρίζει χωρικές ενημερώσεις (π.χ., εισαγωγές ή διαγραφές RDF τριπλέτων). Η καλή απόδοση του SRX βασίζεται σε ένα σχήμα πλέγματος που προσεγγίζει τις γεωμετρίες των χωρικών RDF οντοτήτων μέσα στα ακέραια αναγνωριστικά τους. Αξιολογούμε εκτενώς την απόδοση του SRX και στα ερωτήματα και στις ενημερώσεις, συγκρίνοντας το με τα συστήματα RDF-3X, Virtuoso, GraphDB, και Strabon στα σύνολα δεδομένων LGD και YAGO. Τα αποτελέσματα δείχνουν ότι το SRX υπερτερεί των άλλων συστημάτων στα ερωτήματα και επιφέρει μια μικρή επιβάρυνση στο RDF-3X στις ενημερώσεις.

Στο κοινωνικό πεδίο, η συνεισφορά μας έγκειται στη μελέτη τριών καινοτόμων προβλημάτων σύστασης βάσει περιεχομένου που σχετίζονται με το πρόβλημα *Μεγιστοποίησης Επιρροής* (ME) το οποίο φάχνει τους *k* χρήστες που μπορούν να μεγιστοποιήσουν την επιρροή μιας δεδομένης ανάρτησης σε ένα κοινωνικό δίκτυο.

Το πρώτο πρόβλημα που εισάγουμε, ονομάζεται *Μεγιστοποίηση Επιρροής Βάσει Περιεχομένου* (MEBPI), το οποίο αποτελεί την αντίστροφη παραλλαγή του ME και φάχνει τα *k* χαρακτηριστικά που μπορούν να σχηματίσουν το περιεχόμενο μιας ρυθμιζόμε-

νης ανάρτησης έτσι ώστε αυτή να μπορεί να γίνει πολύ μεταδοτική (δημοφιλής) σε ένα κοινωνικό δίκτυο, ξεκινώντας τη διάδοση της από ένα δεδομένο σύνολο συνδρομητών της κοινωνικής σελίδας της εκάστοτε εταιρείας. Αποδεικνύουμε ότι το MEBII δεν έχει εγγυήσεις επιρροής, ενώ τα πειραματικά μας αποτελέσματα στα σύνολα δεδομένων Gnutella και VK δείχνουν ότι ο ευρετικός αλγόριθμος που υλοποιήσαμε είναι περισσότερο πρακτικός από απλές και άπληστες προσεγγίσεις για το MEBII.

Το δεύτερο πρόβλημα που προτείνουμε είναι μια προσαρμοστική (πραγματικού χρόνου) εκδοχή του MEBII, ονομάζεται *Προσαρμοστική Μεγιστοποίηση Επιρροής Βάσει Περιεχομένου* (ΠΜΕΒII), το οποίο λαμβάνει χώρα σε πολλούς γύρους. Σε κάθε γύρο, αναζητείται το περιεχόμενο μιας ανάρτησης που αποτελείται από *k* χαρακτηριστικά, ενώ η ανάδραση επιρροής των αναρτήσεων σε προηγούμενους γύρους αξιοποιείται στη δημιουργία περιεχομένου για τις αναρτήσεις των επόμενων γύρων. Ο στόχος είναι η μεγιστοποίηση της συνολικής επιρροής σε όλους τους γύρους. Για να λύσουμε το ΠΜΕΒII, ενσωματώνουμε τεχνικές *Πραγματικού Χρόνου Μάθησης Βάσει Κατάταξης* (ΠΧΜΒΚ) στο μηχανικής μάθησης πλαίσιο εργασίας μας που σχεδιάσαμε για μεγιστοποίηση επιρροής, και περαιτέρω υλοποιήσαμε ένα μοντέλο διάδοσης, έναν προσομοιωτή που τρέχει το εν λόγω μοντέλο για την παραγωγή ρεαλιστικής ανάδρασης, τρεις αλγορίθμους μάθησης, και μια εξονυχιστική πειραματική μελέτη σε ποικίλα σύνολα δεδομένων VK για αρκετές εταιρείες.

Τέλος, το τρίτο πρόβλημα που μελετάμε σχετίζεται με το πως οι εταιρείες μπορούν να μεγιστοποιήσουν την συνδρομή τους (αντί για την επιρροή τους όπως συμβαίνει στα MEBII και ΠΜΕΒII) στα κοινωνικά δίκτυα. Συγκεκριμένα, προτείνουμε μια πολιτική σύστασης περιεχομένου στις εταιρείες για την *Απόκτηση Συνδρομητών μέσω Μηνυμάτων* (ΑΣΜ). Το ΑΣΜ πρόβλημα εφαρμόζεται σε πολλούς γύρους. Σε κάθε γύρο, το ΑΣΜ συστήνει στις εταιρείες τι περιεχόμενο (αποτελούμενο από *k* χαρακτηριστικά) να δημοσιεύσουν στις κοινωνικές σελίδες τους και ποιους *m* χρήστες να ειδοποιήσουν για το εν λόγω περιεχόμενο. Ο στόχος είναι η μεγιστοποίηση της συνολικής απόκτησης συνδρομητών σε όλους τους γύρους. Για να λύσουμε το ΑΣΜ, υλοποιήσαμε τρεις αλγορίθμους, ενώ πραγματοποιώντας μια εκτενή πειραματική αξιολόγηση σε διαφορετικά σύνολα δεδομένων VK, διαπιστώνουμε την σημαντική και πρακτική αξία του ΑΣΜ.

Contents

Abstract	vii
Περιλήψη	ix
1 Introduction	1
1.1 Concept and Motivation	1
1.1.1 SRX System	1
1.1.2 CAIM Problem	2
1.1.3 ACAIM Problem	2
1.1.4 GSM Problem	3
1.2 Contributions	3
1.2.1 SRX System	3
1.2.2 CAIM Problem	5
1.2.3 ACAIM Problem	5
1.2.4 GSM Problem	6
1.3 Publications	7
1.4 Organization	7
2 Related Work	9
2.1 RDF Indexing, Querying, and Spatial Support	9
2.2 Influence Maximization and Variants	12
2.3 Online Learning To Rank	15
3 SRX: Efficient Management of Spatial RDF Data	17
3.1 Introduction	17
3.2 Preliminaries	20
3.3 A Basic Spatial Extension	22
3.4 Encoding the Spatial Dimension	25
3.5 Query Evaluation	27
3.5.1 Spatial Range Filtering	28
3.5.2 Spatial Join Filtering	29

3.5.3	Spatial Merge Join on Encoded Entities	30
3.5.4	Spatial Hash Join on Encoded Entities	33
3.5.5	Spatial kNN on Encoded Entities	33
3.6	Query Optimization	38
3.6.1	Augmenting the Query Graph	39
3.6.2	Spatial Join Operators	41
3.6.3	Spatial Query Optimization	42
3.6.4	Selectivity Estimation	42
3.6.5	Runtime Optimizations	42
3.7	Updates	43
3.8	Experimental Evaluation	46
3.8.1	Queries Setup	46
3.8.2	Queries Comparison	48
3.8.3	Updates Setup	56
3.8.4	Updates Comparison	57
3.9	Conclusion	61
3.10	Appendix	62
3.10.1	Spatial distribution of geometries	62
3.10.2	Queries	62
3.10.3	Deltas between different dataset versions	67
4	Content Recommendation for Viral Social Influence	71
4.1	Introduction	71
4.2	Motivation	73
4.2.1	Idea Habitats	73
4.2.2	Digital Influence	74
4.2.3	Distinctiveness	75
4.3	Problem Statement	75
4.3.1	Content-Aware Cascade Model	75
4.3.2	Content-Aware Influence Maximization	77
4.4	Hardness and Inapproximability	77
4.5	The Explore-Update Algorithm	82
4.6	Experimental Study	86
4.6.1	Influence spread	88
4.6.2	Runtime	89
4.6.3	Effect of Seed Size	89
4.6.4	Effect of θ	89
4.6.5	Comparison to the Optimal Solution	90
4.6.6	Real-World Examples	91

Contents

4.7 Conclusion	92
5 Adaptive Content-Aware Influence Maximization through Online Learning To Rank	93
5.1 Introduction	94
5.2 Problem Statement	95
5.2.1 TRM Click Model in Social Networks	95
5.2.2 CATRID Propagation Model	96
5.2.3 ACAIM Problem	97
5.3 CATRID Simulator	98
5.4 ACAIM Learners	100
5.4.1 The Learner RANDOM	100
5.4.2 The Learner TRIM_C	101
5.4.3 The Learner TRIM_E	103
5.5 Experimental Evaluation	108
5.5.1 Setup	109
5.5.2 Reliability	110
5.5.3 Scalability	116
5.5.4 Business Applicability	117
5.6 Conclusion	121
6 A Content Recommendation Policy for Gaining Subscribers	123
6.1 Introduction	123
6.2 GSM Solvers	125
6.2.1 The Solver RANDOM	125
6.2.2 The Solver SCAN	125
6.2.3 The Solver SUBSTITUTE	126
6.3 Experimental Evaluation	129
6.3.1 Setup	129
6.3.2 Results	129
6.4 Conclusion	133
7 Conclusion	135
7.1 Summary	135
7.2 Future Work	136
Bibliography	137

Contents

List of Figures

2.1	Use of Dictionary.	10
3.1	Example of RDF data and three spatial queries.	21
3.2	Possible query plans in the basic extension.	23
3.3	Spatial encoding of entity IDs.	26
3.4	Plan for the query of Fig. 3.1b.	29
3.5	Plan for the query of Fig. 3.1c.	30
3.6	Example of SMJ.	32
3.7	An example grid (a) with 64 cells at the bottom level (11-bit encoding) ordered according to the Hilbert curve and organized in CPM zones (L_i, R_i, U_i, D_i) around a query point p in cell 28. The entity IDs are shown on the right (b) in binary and decimal format.	38
3.8	Augmenting a query graph.	39
3.9	Latency (ms) of processing batches of size b on LGD.	58
3.10	Latency (ms) of processing batches of size b on YAGO.	59
3.11	Latency (ms) of processing batches of size 1K triples on two different subsets (a) and (b) of LGD; the former dataset is a subset of the latter.	60
3.12	Spatial distribution of geometries in LGD. LGD contains geometries for entities in the United Kingdom with highest density in the area around London.	63
3.13	Spatial distribution of geometries in YAGO. YAGO’s geometries spread all over the globe with highest density in North America and Europe.	63
4.1	A graph instance demonstrating that the CAIM problem is NP-hard.	78
4.2	Increasing and decreasing marginal returns.	79
4.3	A graph instance demonstrating that it is NP-hard to approximate the optimal solution to the CAIM problem.	81
4.4	Participating and non-participating edges.	84
4.5	Influence spread and runtime results.	88

4.6	Influence spread and runtime vs. seed set size and θ on Gnutella in (a), (b), and (c). Influence spread on reduced network in (d).	90
5.1	Flow of TRIM_E . Each step digests new evidence (feature path, red); in step 7, (K, D) eliminates features in grey; in step 9, the eliminated D remains as intermediate; in step 11, the cycle caused by (W, B) cancels paths containing subpath <B, (D), E, W>.	104
5.2	Reliability results of RANDOM , TRIM_C , and TRIM_E in datasets A1, A2, and A3 for $k = 3, 4$, and 5.	111
5.3	Reliability results of RANDOM , TRIM_C , and TRIM_E in datasets B1, B2, and B3 for $k = 3, 4$, and 5.	112
5.4	Scalability results of TRIM_E for the nine most popular VK brands (depicted in increasing $ S $) in datasets A1, A2, and A3 for $k = 3, 4$, and 5.	115
5.5	Scalability results of TRIM_E for the nine most popular VK brands (depicted in increasing $ S $) in datasets B1, B2, and B3 for $k = 3, 4$, and 5.	116
6.1	SG per round and Time per round results of RANDOM , SCAN , and SUBSTITUTE for $ L = 27$, $ V = 80$, and $k = 1, 2$, and 3.	130
6.2	SG per round and Time per round results of RANDOM , SCAN , and SUBSTITUTE for $ L = 27$, $ V = 100$, and $k = 1, 2$, and 3.	130
6.3	SG per round and Time per round results of RANDOM , SCAN , and SUBSTITUTE for $ L = 54$, $ V = 80$, and $k = 1, 2$, and 3.	131
6.4	SG per round and Time per round results of RANDOM , SCAN , and SUBSTITUTE for $ L = 54$, $ V = 100$, and $k = 1, 2$, and 3.	131

Chapter 1

Introduction

1.1 Concept and Motivation

Social and spatial marketing problems applied on Web are very common nowadays. In spatial domain, there is a variety of services (e.g., Google Maps, Uber, Foursquare) that efficiently process spatial queries so as *brands* (stakeholders) to connect with their customers. E.g., *Uber Eats* connects food & beverage brands with customers that order goods from these brands by fast delivering requested products to customer locations. So, there is a need for efficient spatial data management operations that our deployed SRX system covers in this thesis. In social domain, brands connect with their customers (social network users) via their social network pages. The main need here is brands to publish interesting content in their pages so as to attract the attention of their subscribers (earning their *like* responses to published posts), and so the latter propagate the brands' content further in the network. In this thesis, we address this need by studying three novel content-aware recommendation problems that maximize either *influence* or *subscription* of brands in social networks.

1.1.1 SRX System

Despite the large volume of work on indexing and querying large RDF knowledge bases [AMMH07, ACZH10, BDK⁺13, BKvH01, CDES05, NW08, WKB08, WSKR03, YWZ⁺09, YLW⁺13, ZYW⁺13, ZMC⁺11], only a few works focus on the effective handling of spatial semantics in RDF data. In particular, the current spatial extensions of RDF stores (e.g., Virtuoso [vir], GraphDB [owl], Parliament [par], Strabon [KKK12], and others [BNM10, WKC12, WZF⁺13]) focus mainly on supporting GeoSPARQL features, and less on performance optimization. The features and weaknesses of these systems are reviewed in Chapter 2. Yet, there is a large number of spatial entities (i.e., resources) in RDF knowledge bases (e.g., YAGO [YAG]). Thus, the power of the

state-of-the-art RDF stores is limited by the inadequate handling of spatial semantics, given that it is not uncommon for user queries to include spatial predicates. Also, spatial data management systems [EM16] can only be used to index and search the spatial semantics of the entities, but do not support graph pattern search. We fill this gap by presenting SRX (Spatial RDF-3X), a system built on top of the open-source RDF-3X store [NW08] to efficiently support spatial queries and updates.

1.1.2 CAIM Problem

The *Influence Maximization* (IM) problem [KKT03] asks for the k users that can maximize the influence of a *fixed* post in a social network. Here, we propose and study the inverse variant of IM where the *initial adopters* are fixed and the *content* of post, comprising k features¹ (or attributes), is sought; we name this problem as *Content-Aware Influence Maximization* (CAIM). The initial adopters in CAIM are the *subscribers* of a social network page maintained by a selected brand, and constitute the set of social network users that *follow* that page to get awarded of latest products, services, events, or general material and news around brand.

Assume the brand *GoPro* wants to launch its next big campaign in the social network where it maintains a page. Let say that for $k = 4$, the optimal feature set is {“Rafael Nadal”, “Paris”, “Artistic photography”, “Street city life”}. Based on this knowledge, *GoPro* can create a post with found feaures. Such a campaign could be a video post that depicts Rafael Nadal (wearing a *GoPro* camera) exercising for the next tennis tournament by running in the central streets of Paris located in artistic background areas. So, *GoPro* can make viral its campaign and maximize its reach.

1.1.3 ACAIM Problem

The *Adaptive Content-Aware Influence Maximization* (ACAIM) problem is an adaptive version of CAIM that applies online under a multi-round setting. ACAIM searches in each round, k features to form an influential post, aiming to maximize the cumulative influence of created posts over all rounds. The propagation of each post always starts with the subscribers’ activation of respective brand, and the real-time feedback of previous posts is utilized for the k -size content decisions of next posts.

Assume the brand *GoPro* has now a more vital need that takes place on a daily basis. Namely, *GoPro* now wants to constantly publish *everyday* posts (low k) in its page to keep in touch with its subscribers and gradually increase the number of *likes* it receives per post so as to grow its social network presence in the long-term.

¹We consider that each *feature* corresponds to a specific social network page.

To find the content of those posts for maximum cumulative influence purposes, the agent/advertiser of *GoPro* solves the ACAIM problem.

We stress that brands can benefit from CAIM problem when they want to create their next big campaign (high k), whereas ACAIM solves a different and more crucial need for their social network presence that daily requests influential content (low k).

1.1.4 GSM Problem

Besides studying how brands can increase their *influence* in social networks, we also propose a problem that dictates how brands can grow their *subscription* in such networks. The relative problem named *Gaining Subscribers by Messaging* (GSM) and pertains to a *content recommendation policy* for brands to earn new subscribers to their pages. This policy applies to many rounds, and in each round, it guides the brand advertiser to invite a number of users to visit the last published content in brand's page so as these users to really find attractive the content of page and subscribe to it. So, GSM in each round, recommends to brands what content to publish, comprising k features, and which users to invite by messaging to see that content; a user is notified only once. The aim of GSM is to maximize the subscription gain over all rounds.

In social networks, there is a *limited* number of allowed messages per day; VK allows 20 messages per 12 hours to each user having a VK account (here, advertiser) to send to any other VK user that is not included in her connections. So, advertiser by solving GSM can effectively prioritize her messages, while on the contrary, by following a random policy of messaging she would attract subscribers in a much lower pace and also facing the danger of temporal blocking by VK company due to spamming reports sent by invited users who find not interesting the brand's content.

1.2 Contributions

The overall goal of this thesis is to empower brands with social and spatial operations to efficiently and effectively connect with their customers; in spatial domain, the customers are mobile users, while in social sphere, the customers are social network users. In what follows, we summarize the technical contributions of this thesis.

1.2.1 SRX System

SRX inherits the basic design principles of RDF-3X [NW08], which encodes all values that appear in SPO triples by identifiers with a help of a dictionary, and models the RDF knowledge base as a single long table of ID triples. Yet, SRX includes the following extensions over RDF-3X:

Index Support for Spatial Queries. SRX includes a spatial index (i.e., an R-tree [Gut84]) for the geometries associated to the spatial entities. This facilitates the efficient evaluation of queries with very selective spatial components.

Spatial Encoding of Entities. SRX encodes spatial approximations inside the IDs of entities associated to geometries, yielding several benefits. First, the IDs of resources are used as cheap filters and data are pruned without accessing the entities' geometries. Second, our encoding does not affect the ordering (i.e., sorting) of triples used by the RDF-3X evaluation engine. Finally, our encoding can handle spatially skewed datasets and updates without the need to re-assign IDs for all entities.

Spatial Join Algorithms. We design spatial join algorithms tailored to our encoding scheme. Our *Spatial Merge Join* (SMJ) algorithm extends the traditional merge join algorithm to process the filter step of a spatial join at the approximation level of our encoding, while (i) preserving *interesting orders* of the qualifying triples that can be used by succeeding operators, and (ii) not breaking the pipeline within the operator tree. Our *Spatial Hash Join* (SHJ-ID) operates with unordered inputs, using their encodings to identify fast candidate join pairs.

Spatial kNN Algorithms. We design two k nearest neighbors (kNN) algorithms that make use of our encoding scheme. Both are based on previous work on grid-based kNN query evaluation. The first one operates on unordered input whereas the second exploits interesting orders and can be combined with other order-preserving operators to improve performance and further reduce the memory footprint.

Spatial Query Optimization. We extend the query optimizer of RDF-3X to consider spatial filtering operations that can be applied on the spatially encoded entities. To do that, we augment the original join query graph of a SPARQL expression to include binding of spatial variables via spatial join conditions.

Dynamic Spatial Re-encoding. When new triples are added and old ones are removed or updated, the need for re-encoding spatial entities arises naturally. To tackle this problem with a low overhead in performance, we carefully integrate a dynamic re-encoding technique with the original update mechanism of RDF-3X.

Extensive Experimental Evaluation. To evaluate queries, we compare SRX with RDF-3X [NW08], Virtuoso [vir], GraphDB [owl], and Strabon [KKK12] on two real datasets: LinkedGeoData (LGD) [LGD] and YAGO [YAG]. To evaluate updates, we generated a realistic update benchmark based on the deltas we collected between different versions of LGD and YAGO. Results show that SRX outperforms other systems for queries and incurs a little overhead to RDF-3X for updates.

1.2.2 CAIM Problem

The CAIM problem constitutes the first work that studies the *Influence Maximization* (IM) problem [KKT03] in which the initial adopters are given and the post content is sought. Our related contributions are as follows:

Problem Setting. We motivate the *Influence Maximization* problem in settings where the set of initial adopters is fixed, or even a single point of origin, and the content of a propagated post can be tuned. We formulate the concept of *digital influence* as a special case of social influence.

Propagation Model. We devise a *content-aware* propagation model, whereby the probability of influence across edges depends on content. We show that, with this model: (i) the problem of choosing content features that maximize influence is NP-hard; (ii) the spread function is not submodular, hence no submodularity-based approximation algorithm applies; and (iii) it is NP-hard to approximate the optimal solution within a factor of $n^{1-\varepsilon}$ for $\varepsilon > 0$.

Algorithm. We design a fast algorithm, *Explore-Update*, which achieves higher influence spread than baselines; its effectiveness is based on the iterative estimation of the marginal spread achieved by each feature, while its efficiency is gained by limiting such computations only to nodes within a probability-based distance threshold θ and features potentially affecting such nodes.

Experiments. We compare *Explore-Update* to two baselines on Gnutella² and VK³ datasets, and show that it always achieves better propagation results, while it is significantly faster than a basic Greedy approach by achieving competitive efficacy.

1.2.3 ACAIM Problem

The ACAIM problem is an adaptive version of CAIM tailored for online and real-time applicability to social networks. In addition, ACAIM includes the first integration of OLR techniques to maximize influence in a network, and ascertains its utilization value via a thorough industry-oriented experimental evaluation on real-world brands. We summarize the contributions of ACAIM as follows:

Problem. We propose and study the ACAIM problem that is the first work that searches for influential features under adaptive (multi-round) settings. To do that, we introduce the CATRID propagation model that enables for first time the utilization of an *Online Learning To Rank* (OLR) framework for IM purposes.

²<https://snap.stanford.edu/data/p2p-Gnutella04.html>

³<https://vk.com/>

Simulator. We deploy a simulator (named SimCATRID) that runs the CATRID model so as to represent a realistic feedback environment for ACAIM guided by a proposed training-testing scheme that is based on real VK posts.

Learners. We develop the learners `RANDOM`, *TopRank IM Classic* (TRIM_C), and *TopRank IM Eliminator* (TRIM_E) to solve the ACAIM problem. `RANDOM` is a baseline learner with non-trivial application semantics, TRIM_C is the integrated version of `TOPRANK` [LKLS18] for ACAIM, and TRIM_E also uses the TRM click model as TRIM_C , but its logic on leveraging clicks is different; TRIM_E eliminates features over rounds by using a *transitive* structure during *exploration*, and further ranks them during *exploitation* by using an auxiliary *participation* structure.

Experiments. We present a plethora of experiments that examines the ACAIM performance of multiple brands on several VK datasets under the metrics of *reliability* and *scalability*, and we also provide a *business applicability* study that demonstrates the ACAIM results of worldwide known brands. The performance of TRIM_E is impressive in all cases, making it suitable for utilization in social network industry.

1.2.4 GSM Problem

Differently from CAIM and ACAIM that focus on the influence maximization for brands, the GSM problem studies how brands can maximize their subscribers in social networks. The contributions of this work mentioned as follows:

Problem. We propose that brands can use a *content recommendation policy* to gain subscribers to their social network pages via messaging. This problem, named *Gaining Subscribers by Messaging* (GSM), applies in many rounds, where in each round, there is a recommendation to brands for what content to publish and which users to notify of that content so as to maximize their subscription gain over all rounds.

Solvers. We deploy three GSM solvers, named `RANDOM`, `SCAN`, and `SUBSTITUTE`. `RANDOM` opts to notify users uniformly at random. `SCAN` finds the best pair of users and features by searching all the possible combinations of both of them. `SUBSTITUTE` importantly reduces the exhaustive search burden of `SCAN` by using fast retrieval techniques based on sorting of users in regards to their achieved subscription gains.

Experiments. We evaluate the performance of deployed GSM solvers on VK data by considering different user sets and feature sets. The results show that `SUBSTITUTE` provides the best solution, as it is significantly more efficient than `SCAN` with a minor loss of efficacy and clearly more efficacious than `RANDOM` with competitive efficiency.

1.3 Publications

The results presented in this thesis have been published at the following venues:

- Sergei Ivanov, Konstantinos Theocharidis, Manolis Terrovitis, Panagiotis Karras. Content Recommendation for Viral Social Influence. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR*: 565-574, 2017.
- Konstantinos Theocharidis, John Liagouris, Nikos Mamoulis, Panagiotis Bouros, Manolis Terrovitis. SRX: Efficient Management of Spatial RDF Data. In *The International Journal on Very Large Data Bases - The VLDB Journal*: 703-733, 2019.
- Konstantinos Theocharidis, Manolis Terrovitis, Spiros Skiadopoulos, Panagiotis Karras. A Content Recommendation Policy for Gaining Subscribers. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR*, 2022.
- Konstantinos Theocharidis, Manolis Terrovitis, Spiros Skiadopoulos, Panagiotis Karras. Adaptive Content-Aware Influence Maximization through Online Learning To Rank. Submitted to *31st ACM International Conference on Information and Knowledge Management - CIKM*, 2022. The extended journal version of this work will be submitted to *IEEE Transactions on Knowledge and Data Engineering - TKDE*, 2022.

1.4 Organization

The rest of this thesis is structured as follows:

Chapter 2. This chapter discusses related work about RDF stores and their spatial extensions, as also *Influence Maximization* (IM) and its variants. We also include the *Online Learning To Rank* (OLR) literature as it is part of our ACAIM problem.

Chapter 3. In this chapter, we present the SRX system that extends the RDF-3X store for spatial data management. SRX supports WITHIN, DISTANCE, kNN spatial queries, as also spatial updates. For both queries and updates, we compare SRX with RDF-3X, Virtuoso, GraphDB, and Strabon on LGD and YAGO datasets.

Chapter 4. This chapter presents the *Content-Aware Influence Maximization* (CAIM) problem that constitutes the inverse variant of the classic IM problem. We prove that CAIM does not have influence guarantees, and so we deploy heuristic solutions to solve it. Experimental results are provided on Gnutella and VK datasets.

Chapter 5. This chapter presents the *Adaptive Content-Aware Influence Maximization* (ACAIM) problem that addresses CAIM in an adaptive/online setting with many rounds. To solve ACAIM, we follow a machine learning approach based on the years 2010-2019 of VK, and we show how an OLR framework can be utilized for IM purposes. For that, we deploy one propagation model, one simulator that runs that model, three ACAIM learners, and a thorough experimental study on VK brands.

Chapter 6. In this chapter, we present the *Gaining Subscribers by Messaging* (GSM) problem that pertains to a content recommendation policy for brands to effectively increase their subscribers. We present three GSM solvers and evaluate their performance by issuing a rich experimental process on different VK datasets.

Chapter 7. In this chapter, we summarize the contributions of this thesis, as also we provide interesting directions for future social and spatio-social research problems.

Chapter 2

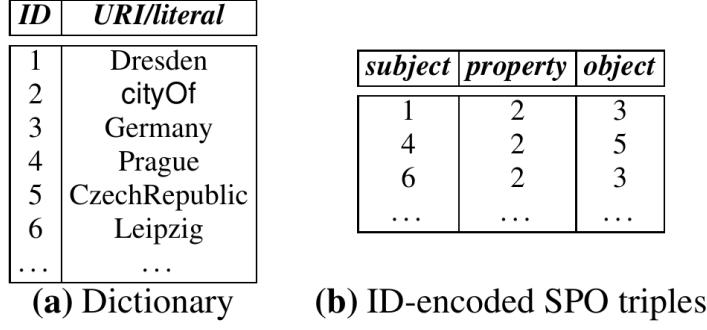
Related Work

In this chapter, we present the related work of this thesis. We start the discussion from the spatial domain and afterwards proceed to the social field. Our work relative to spatial data management (presented in Chapter 3) compares with similar works about storing and indexing of RDF data and executing RDF queries on them, as also with spatial extensions of those RDF systems. On the other hand, our works relative to social data mining (presented in Chapters 4, 5, and 6) mainly depend on works that study the *Influence Maximization* (IM) problem that searches for the k users who can maximize the influence of a given post in a social network. We also discuss related work of *Online Learning to Rank* (OLR) problem that seeks for the k most clickable items to satisfy users, since we utilize OLR in Chapter 5 for IM purposes.

2.1 RDF Indexing, Querying, and Spatial Support

RDF Storage and Query Engines. There have been many efforts toward the efficient storage and indexing of RDF data. The most intuitive method is to store all $\langle \text{subject}, \text{property}, \text{object} \rangle$ (SPO) statements in a single, very large *triples* table. The RDF-3X system [NW08] is based on this simple architecture. RDF-3X (following an idea from previous work) uses a *dictionary* to encode URIs and literals as IDs. Indexing is then applied on the ID-encoded SPO triples. Figure 2.1 illustrates a dictionary and the ID-encoded triples for the RDF base of Figure 3.1a. RDF-3X creates a clustered B⁺-tree index for each of the six SPO permutations (i.e., SPO, SOP, PSO, POS, OSP, OPS). A SPARQL query is transformed to a multi-way self-join query on the triples table; the query engine binds the query variables to SPO values and joins them (if the query contains literals or filter conditions, these are included as selection conditions). A query is first translated by replacing URIs or literals by the respective IDs and then evaluated using the six indices; finally, the

2.1. RDF Indexing, Querying, and Spatial Support



ID	URI/literal
1	Dresden
2	cityOf
3	Germany
4	Prague
5	CzechRepublic
6	Leipzig
...	...

subject	property	object
1	2	3
4	2	5
6	2	3
...

(a) Dictionary **(b)** ID-encoded SPO triples

Figure 2.1: Use of Dictionary.

query results (in the form of ID-triples) are translated back to their original form. The six indices offer different ways for accessing and joining the triples; RDF-3X includes a query optimizer to identify a good query evaluation plan. The system favors plans that produce *interesting orders*, where merge joins are pipelined without intermediate sorts. In addition, a run-time *sideways information passing* (SIP) mechanism [NW09] reduces the cost of long join chains. RDF-3X maintains nine additional aggregate indices, corresponding to the nine projections of the SPO table (i.e., SP, SO, PS, PO, OS, OP, S, P, O), which provide statistics to the query optimizer and are also useful for evaluating specialized queries. The query optimizer was extended in [NM11] to use more accurate statistics for star-pattern queries. RDF-3X employs a compression scheme to reduce the size of the indices by differential storage of consecutive triples in them. Hexastore [WKB08] is a contemporary to RDF-3X proposal, which also indexes SPO permutations on top of a triples table. An earlier implementation of a triples table by Oracle [CDES05] uses materialized join views to improve performance.

An alternative storage scheme is to decompose the RDF data into *property* tables: one binary table is defined per distinct property, storing the SO pairs that are linked via this property. To avoid the case of having a huge number of property tables, this extreme approach was refined to a *clustered-property* tables approach (used by early RDF stores, like Jena [WSKR03] and Sesame [BKvH01]), where correlated tables are clustered into the same table and triples with infrequent properties are placed into a *left-over* table. Abadi et al. [AMMH07] use a column-store database engine to manage one SO table for each property, sorted by subject and optionally indexed on object.

A common drawback of the column-store approach and RDF-3X is the potentially large number of joins that have to be evaluated, together with the potentially large intermediate results they generate. Atre et al. [ACZH10] alleviate this problem by introducing a 3D compressed bitmap index, which reduces the intermediate results before joining them. A similar idea was recently proposed in [YLW⁺13]; the participation of subjects and objects in property tables is represented as a sparse 3D matrix, which is compressed. Yet, another storage architecture was proposed in [BDK⁺13].

The idea is to first cluster the triples by subject and then combine multiple triples about the same subject into a single row. Thus, the system saves join cost for star-pattern queries, but it may suffer from redundancy due to repetitions and null values.

Trinity [ZYW⁺13] is a distributed memory-based RDF data store, which focuses on graph query operations such as random walk distance, reachability, etc. RDF data are represented as a huge (distributed) graph and query evaluation is done in an exploration-based manner; starting from the most selective predicates, query variables are bound progressively, while the RDF graph is browsed. Trinity’s power lies on the fact that memory storage eliminates the otherwise very high random access cost for graph exploration. More information on distributed RDF systems can be found in [Ö16]. gStore [ZMC⁺11] is an earlier, graph-based approach, which models SPARQL queries as graph pattern matching queries on the RDF graph. gStore owes its efficiency to a signature generation scheme for the RDF (non-literal) graph nodes. The signature of a vertex is determined by its neighboring edges (i.e., properties) and nodes (i.e., entities or literals). The same encoding scheme is used for query patterns and can be used to effectively filter nodes that do not match with a query. The signatures of all nodes are hierarchically indexed by a search tree, which can help to find query pattern results. The index is also appropriate for *wildcard* queries where variables are bound to literals with wildcards via filter conditions. A similar, but simpler graph-partitioning approach for RDF data was proposed in [YWZ⁺09]. More recently, EmptyHeaded [ATOR, ATOR16] employed novel worst-case optimal join algorithms to accelerate pattern matching queries on RDF graphs.

Spatial Extensions of RDF Stores. Although the efficient management of RDF data has been a hot research topic for almost two decades, there have been only a few recent efforts toward extending RDF stores to support spatial data. The Parliament [BK12], built on top of Jena [WSKR03], implements most of the features of GeoSPARQL. Strabon [KKK12], developed contemporarily with Parliament, extends Sesame [BKvH01] to manage spatial RDF data stored in PostGIS. Strabon adopts a column-store approach, implementing two SO and OS indices for each property table. Spatial literals (e.g., points, polygons) are given an identifier and are stored in a separate table, which is indexed by an R-tree [Gut84]. Strabon extends the query optimizer of Sesame to consider spatial predicates and indices. The optimizer applies simple heuristics to push down (spatial) filters or literal binding expressions in order to minimize intermediate results. Strabon is shown to outperform Parliament, however, both systems suffer from the poor performance of the RDF stores they are based on (i.e., Jena and Sesame) compared to faster engines (e.g., RDF-3X [NW08]). For that reason, Strabon and Parliament lack sophisticated query evaluation and optimization techniques.

Brodt et al. [BNM10] extend RDF-3X [NW08] to support spatial data. The extension is limited, since range selection is the only supported spatial operation. Furthermore, query evaluation is restricted to either processing the non-spatial query components first and then verifying the spatial ones or the other way around. Finally, the opportunity of producing an interesting order from a spatial index (in order to facilitate subsequent joins) is not explored.

Geo-Store [WKC12] is another spatial extension of RDF-3X. Geo-Store divides the space by a grid and orders the cells using a *Hilbert* space-filling curve. Each geometry literal g (e.g. “POINT (...)”) is approximated by the Hilbert order $g.ID$ of the cell that includes it. Then, for all triples of the form $s \text{ hasGeometry } g$, a triple $s \text{ hasPos } g.ID$ is added to the data. During query evaluation, an extra join with the `hasPos` triples is applied to perform the filter step of spatial queries. Geo-Store supports only spatial range and k nearest neighbor queries, but not spatial joins. In addition, it does not extend the query optimizer of RDF-3X to consider spatial query components. Finally, besides increasing the size of the original database with the introduction of `hasPos` triples, it is not clear how its encoding can handle complex spatial literals, such as “POLYGON (...)”, which may span multiple cells of the grid.

S-Store [WZF⁺13] is a spatial extension of gStore [ZMC⁺11], which appends to the signatures of spatial entities their minimum bounding rectangles (MBRs). The hierarchical index of gStore is then adapted to consider both non-spatial and spatial signatures. Although S-Store was shown to outperform gStore for spatial queries, it handles spatial information only at a high level (i.e., the data are primarily indexed based on their structure) and there is no study on how spatial statistics can be used to improve query evaluation. Spatial RDF queries are also supported by many commercial systems, such as Oracle, Virtuoso [[vir](#)], and GraphDB [[owl](#)], however, details about their internal design are not public.

Finally, [NVDV18] recently introduced DiStRDF that adapts the encoding scheme of [LMBT14] to support RDF queries with spatio-temporal filters on top of Spark.

2.2 Influence Maximization and Variants

Classic Influence Maximization. The first solutions to the *Influence Maximization* (IM) problem were proposed by Domingos and Richardson [DR01, RD02], yet had no guarantees on influence spread. Then, Kempe et al. [KKT03] formulated the problem based on the *Independent Cascade* (IC) and *Linear Threshold* (LT) propagation models, proved its NP-hardness, and proposed a greedy algorithm with a $(1 - 1/e - \epsilon)$ approximation guarantee. Subsequent works investigated efficiency and scalability questions, either with heuristics [CWW10, CYZ10, CSH⁺14] or preserving

an approximation guarantee [LKG⁺07, GBL11, CPL12, BBCL14, TXS14]. IM is a very popular problem that has been extensively studied the last two decades due to its lucrative viral marketing applicability in social networks, its efficiency and efficacy challenges [AGR17, LFWT18], as also its plethora of variants [LFWT18].

Topic-Aware Influence Maximization. Barbieri et al. [BBM12] were the first to look at social influence taking content characteristics into consideration. They proposed methods that learn propagation model parameters such as topic-aware influence strength from a query log of past propagation traces, and verified experimentally that a larger influence spread can be engendered when taking item characteristics into consideration via their *Topic-Aware Influence Maximization* (TIM) models.

Aslay et al. [ABBY14] studied online TIM queries; the incentive for this online scenario is that many independent advertisers wish to instantly detect the k most influential users for advertising purposes; each advertisement contains a different set of keywords and hence induces a new probabilistic graph creating a separate TIM instance; an offline-online solution, INFLEX, based on an index used to identify similarities among a new and log TIM queries; pre-computed solutions for log queries are aggregated online so as to provide an approximate solution for a TIM query.

The online TIM problem is also studied in [CLY14, CFL⁺15]. Chen et al. [CLY14] studied topic-aware influence results on two real networks and utilized the derived properties to form three preprocessing-based algorithms, of which MIS is the best; its main difference from INFLEX is that, in MIS, pre-computed seed sets are based on each separate topic rather than on a mixture of topics from different log queries. Chen et al. [CFL⁺15] utilized the *maximum influence arborescence* (MIA) model [CWW10] to achieve high influence spread with a theoretical guarantee. The core idea is to utilize upper- and lower-bounding techniques, so that an exact marginal influence is computed only for the most promising nodes. This work provides the state-of-the-art solution for the online TIM problem [ABBY14].

Recently, Li et al. [LZT15] proposed a variation on the online TIM problem, namely the alternative problem of *Keyword-Based Targeted Influence Maximization* (KB-TIM). By KB-TIM, each user is associated with a weighted vector of preferences for distinct keywords, which stand for topics. This vector can be generated by applying topic modeling techniques [HD10] on aggregated user social activities, such as posts, likes, etc. An advertisement then achieves an impact determined by its own topic-oriented keywords. The KB-TIM problem aims to maximize an advertisement's impact, expressed in terms of its spread to target users relevant to its keywords. The solution in [LZT15] draws from previous work in [TXS14], with the main difference being that, while in [TXS14] θ users in a sampled *Reverse Reachable* (RR) set [BBCL14] are counted without prejudice, in [LZT15] these sampled users

are accounted in terms of exerted advertisement impact; [LZT15] also employs two indexing methods to precompute RR sets for different keywords, so as to obtain RR sets associated with the query keywords on the fly. Nevertheless, results in [LZT15] are not compared to those in [CFL⁺15].

Finally, authors in [KLK20] consider each topic (they call it *feature*) as a specific social network page and propose an alternative version of LT model based on features, named *Content-Aware Linear Threshold* (CALT) model. Particularly, they first learn the influence parameters of CALT associated with activation probabilities on network edges, and then they use that knowledge to find the k most influential features in order to form a viral post that starts its diffusion from a fixed set of initial adopters.

Influence Maximization with Viral Product Design. Aral and Walker [AW11] investigated the problem of *viral product design* (VPD) under randomized trials focusing on product features like *personalized referrals* and *broadcast notifications*. Thereafter, Barbieri and Bonchi [BB14] studied the problem of IM in conjunction with that of VPD, aiming to detect a combination of seed nodes and product attributes that maximize influence in a network. The proposed solutions are generic methods named *Local Update* and *Genetic Update*; the former is a greedy algorithm allowing for both addition and removal of attributes at each greedy iteration; the latter is a brute-force method that randomly selects a subset of all attributes.

Adaptive Influence Maximization. The *Adaptive Influence Maximization* (AIM) problem attracted significant interest lately due to its high applicability to many IM use-cases in real world. We mention two main reasons for that.

First, it is easily verified in [GBL11] that, given a network with known activation probabilities on its edges, one-round propagations of classic IM are often less influential than multi-round ones of AIM. This is logical to happen, since in AIM, the influential candidates are gradually selected over rounds based on a network feedback [GK11, CK13, YT17, SHYC18, HTH⁺20]. Golovin and Krause [GK11] introduced an adaptive optimization framework with approximation guarantees under a *dependent* (incremental) over rounds seed selection setting; only one seed node can be selected in each round and the propagation capabilities of earlier seeds are not reconsidered for later seeds. To do that, they utilize a *full-adoption feedback* model that notifies the learner of which nodes activated in each round. All the subsequent works are motivated by the adaptive formulation in [GK11]. In particular, under the full-adoption feedback model, Chen and Krause [CK13] study adaptive seed selection by assuming that more than one seed nodes can be selected in each round. Yuan and Tang [YT17] solve AIM under a *partial-feedback* model that allows selecting seeds without time restrictions in the sense that there is no need to wait for total influence results of previous seeds so as to select the next seeds. The state-of-the-art work for

AIM, under the dependent seed selection setting, is the recent work in [HTH⁺20] that use approximation algorithms which depend on selecting nodes in equal batches. Differently from previous works, Sun et al. [SHYC18] proposed an alternative setting for AIM under which influence propagates in multiple rounds *independently* from possibly different seed sets and the aim is to maximize the expected number of unique activated nodes over all rounds.

The second reason for the popularity of AIM is the uncertain factor. In many realistic scenarios, the parameters of activation probabilities in the edges of a network or even the network topology (nodes and their edges), are partially or completely agnostic to the learner. Several approaches have been proposed [LMM⁺15, CWYW16, VLS16, VKW⁺17, WKVV17, WLW⁺19], which given a network, they learn the underlying diffusion parameters while simultaneously running *independent* (i.e., non-incremental) propagation campaigns. To balance between *exploration* steps (of yet uncertain model aspects) and *exploitation* ones (focusing on the most promising seeds), these approaches rely on *multi-armed bandits* techniques similar to [CWY13]. Among these works, only in [LMM⁺15] authors target the activation of unique nodes across rounds as in [SHYC18], while all the others share the same non-unique node influence objective. Yet, all these multi-armed bandit approaches still focus on the learning aspect of edge probabilities to find influential seeds. Further, regarding the case where the network topology is unknown, there is an interesting line of adaptive works [SS13, HS15, LCCM19] looking for promising seeds to solve AIM under an agnostic network perspective.

2.3 Online Learning To Rank

The *Online Learning To Rank* (OLR) problem mentions to a learner that adaptively selects, in each round, a ranked list of k out of L items so as to maximize the *user satisfaction* over all rounds; each round relates with a single user and users are often different among rounds. Most OLR works [KSWA15, CMPL15, KWAS15, ZNS⁺16, LWZC16, KSW16, LVC16] measure that user satisfaction in clicks, by utilizing a *click model* (common to all users in all rounds) that is a stochastic model of how a user examines and clicks on a ranked list of items. There is a variety of click models [CMdR15]. Among them, the most popular ones for the OLR problem are the *Position-Based Model* (PBM) [RDR07] and the *Cascade Model* (CM) [CZTR08]. In PBM, the probability that users click on an item of a ranked list depends on the position of item in the list and the inherent attractiveness of item, while CM assumes that users scan the ranked list from top to bottom, clicking on the first item they find attractive. Yet, it has been experimentally observed that no single existing click

model captures the behavior of an entire population of users [GCM⁺15]. To address this problem, Zoghi et al. [ZTG⁺17] introduced an algorithm that solves OLR in different click models (including PBM and CM) under reasonable click probability assumptions. After that, Lattimore et al. [LKLS18] enhanced the independent click model setting of previous work by relaxing further its click assumptions and proposed a more practical algorithm, named `TopRank`, to solve OLR in multiple click models under stronger regret guarantees. Recently, Li et al. [LLS19] proposed an interesting learning framework to handle OLR cases where the collection of items (L) is importantly higher than earlier works.

Chapter 3

SRX: Efficient Management of Spatial RDF Data

In this chapter, we present a general encoding scheme for the efficient management of spatial RDF data. The scheme approximates the geometries of the RDF entities inside their integer IDs and can be used, along with several operators and optimizations we introduce, to accelerate queries with spatial predicates and to re-encode entities dynamically in case of updates. We implement our ideas in SRX, a system built on top of the popular RDF-3X system [NW08]. SRX extends RDF-3X with support for three types of spatial queries: range selections (e.g. find entities within a given polygon), spatial joins (e.g. find pairs of entities whose locations are close to each other), and spatial k nearest neighbors (e.g. find the three closest entities from a given location). We evaluate SRX on spatial queries and updates with real RDF data, and we also compare its performance with the latest versions of three popular RDF stores. The results show SRX’s superior performance over the competitors; compared to RDF-3X, SRX improves its performance for queries with spatial predicates while incurring little overhead during updates.

3.1 Introduction

The Resource Description Framework (RDF) has become a standard for expressing information that does not conform to a crisp schema. Semantic-Web applications manage large knowledge bases and data ontologies in the form of RDF. RDF is a simple model, where all data are in the form of $\langle \text{subject}, \text{property}, \text{object} \rangle$ (SPO) triples, also known as *statements*. The subject of a statement models a *resource* (e.g., a Web resource) and the property (a.k.a. *predicate*) denotes the subject’s relationship to the object, which can be another resource or a simple value (called *literal*). A resource is specified by a uniform resource identifier (URI) or by a *blank node* (denoting an

unknown resource). An RDF knowledge base can be modeled as a graph, where nodes are resources or literals and edges are properties.

SPARQL is the standard query language for RDF data, used to express *query graph patterns* that have to be matched in the RDF data graph. The GeoSPARQL standard [BK12], defined by the Open Geospatial Consortium (OGC), extends RDF and SPARQL to represent geographic information and support spatial queries. Geospatial filter functions are used to express spatial predicates between entities in SPARQL queries. stSPARQL [KK10] has similar features.

Despite the large volume of work on indexing and querying large RDF knowledge bases [AMMH07, ACZH10, BDK⁺13, BKvH01, CDES05, NW08, WKB08, WSKR03, YWZ⁺09, YLW⁺13, ZYW⁺13, ZMC⁺11], only a few works focus on the effective handling of spatial semantics in RDF data. In particular, the current spatial extensions of RDF stores (e.g., Virtuoso [vir], GraphDB [owl], Parliament [par], Strabon [KKK12], and others [BNM10, WKC12, WZF⁺13]) focus mainly on supporting GeoSPARQL features, and less on performance optimization. The features and weaknesses of these systems are reviewed in Chapter 2. On the other hand, there is a large number of spatial entities (i.e., resources) in RDF knowledge bases (e.g., YAGO [YAG]). Thus, the power of the state-of-the-art RDF stores is limited by the inadequate handling of spatial semantics, given that it is not uncommon for user queries to include spatial predicates. At the same time, spatial data management systems [EM16] can only be used to index and search the spatial semantics of the entities, but do not support graph pattern search.

We fill this gap by presenting SRX (Spatial RDF-3X), a system built on top of the open-source RDF-3X store [NW08] to efficiently support spatial queries and updates. SRX inherits the basic design principles of RDF-3X, which encodes all values that appear in SPO triples by identifiers with a help of a dictionary, and models the RDF knowledge base as a single long table of ID triples. A SPARQL query can then be modeled as a multi-way join on the triples table. The system creates a clustered B⁺-tree for each of the six SPO permutations; the query optimizer identifies an appropriate join order, considering all the available permutations and advanced statistics [NM11]. RDF-3X is known to have robust performance in comparison studies on various RDF datasets and query benchmarks [BDK⁺13, NW08, YLW⁺13]. Although we have chosen RDF-3X as a basis for SRX, our techniques are also applicable to other RDF stores, e.g. [YLW⁺13]. In a nutshell, SRX includes the following extensions over RDF-3X.

Index Support for Spatial Queries. Similar to previous spatial extensions of RDF stores (e.g., [BNM10]), SRX includes a spatial index (i.e., an R-tree [Gut84]) for the geometries associated to the spatial entities. This facilitates the efficient evaluation

of queries with very selective spatial components.

Spatial Encoding of Entities. The identifiers given to RDF resources in the dictionary of RDF-3X (and other RDF stores) do not carry any semantics. Taking advantage of this fact, we encode spatial approximations inside the IDs of entities (i.e., resources) associated to spatial locations and geometries. This mechanism has several benefits. First, for queries that include spatial components, the IDs of resources can be used as cheap filters and data can be pruned without having to access the exact geometries of the involved entities. Second, our encoding scheme does not affect the standard ordering (i.e., sorting) of triples used by the RDF-3X evaluation engine, therefore it does not conflict with the RDF-3X query optimizer; in other words, the original system’s performance on non-spatial queries is not compromised. Finally, our encoding scheme adopts a flexible hierarchical space decomposition so that it can easily handle spatially skewed datasets and updates without the need to re-assign IDs for all entities.

Spatial Join Algorithms. We design spatial join algorithms tailored to our encoding scheme. Our *Spatial Merge Join* (SMJ) algorithm extends the traditional merge join algorithm to process the filter step of a spatial join at the approximation level of our encoding, while (i) preserving *interesting orders* of the qualifying triples that can be used by succeeding operators, and (ii) not breaking the pipeline within the operator tree. In typical SPARQL queries which usually involve a large number of joins, the last two aspects are crucial for the overall performance of the system. Our *Spatial Hash Join* (SHJ-ID) operates with unordered inputs, using their encodings to identify fast candidate join pairs.

Spatial kNN Algorithms. We design two k nearest neighbors (kNN) algorithms that make use of our encoding scheme. Both are based on previous work on grid-based kNN query evaluation. The first one operates on unordered input whereas the second exploits interesting orders and can be combined with other order-preserving operators to improve performance and further reduce the memory footprint.

Spatial Query Optimization. In addition to including standard selectivity estimation models and techniques for spatial queries, we extend the query optimizer of RDF-3X to consider spatial filtering operations that can be applied on the spatially encoded entities. To do that, we augment the original join query graph of a SPARQL expression to include binding of spatial variables via spatial join conditions.

Dynamic Spatial Re-encoding. Changes in real RDF datasets are the rule rather than the exception. Such changes occur as new triples are added and old ones are removed or updated, and the need for re-encoding spatial entities arises naturally. To tackle this problem with a low overhead in performance, we carefully integrate a dynamic re-encoding technique with the original update mechanism of RDF-3X.

We evaluate SRX by comparing it with the latest versions of two commercial spatial RDF management systems: Virtuoso [vir] and GraphDB [owl], and a popular free RDF management system: Strabon [KKK12]. For query evaluation, we use two real datasets: LinkedGeoData (LGD) [LGD] and YAGO [YAG]. To evaluate dynamic re-encoding, we generated a realistic update benchmark — the first one using real data — based on the deltas we collected between different versions of LGD and YAGO. The results demonstrate the superior performance and robustness of SRX over the competitors; SRX improves the performance of the original RDF-3X for queries with spatial predicates, while incurring insignificant overhead when performing updates.

3.2 Preliminaries

The SPARQL queries we consider follow the format:

Select [projection clause]
Where [graph pattern]
Filter [condition]

The **Select** clause includes a set of variables that should be instantiated from the RDF knowledge base (variables in SPARQL are denoted by a `? prefix`). A graph pattern in the **Where** clause consists of triple patterns in the form of $s \ p \ o$ where any of the s , p and o can be either a constant or a variable. Finally, the **Filter** clause includes one or more *spatial predicates*. For the ease of presentation, in our discussion and examples, we consider only **WITHIN** range predicates (for spatial selections), **DISTANCE** predicates (for spatial joins), and **kNN** predicates (for k nearest neighbors). However, we emphasize that the results of our work are directly applicable to all spatial predicates defined in the GeoSPARQL standard [BK12]. In addition, we use a simplified syntax for expressing queries and not the one of the GeoSPARQL standard because the latter is verbose.

As an example, consider the RDF knowledge base partially listed in Figure 3.1a. Literals and *spatial literals* (i.e., geometries) are in quotes. An exemplary query with a range predicate is:

```
Select ?s ?o  
Where ?s cityOf Germany . ?s hosted ?o .  
      ?s hasGeometry ?g .  
Filter WITHIN(?g, “POLYGON(...)”);
```

This query finds the cities of Germany within a specified polygonal range together with the persons they hosted. Note that there are three variables involved ($?s$, $?o$,

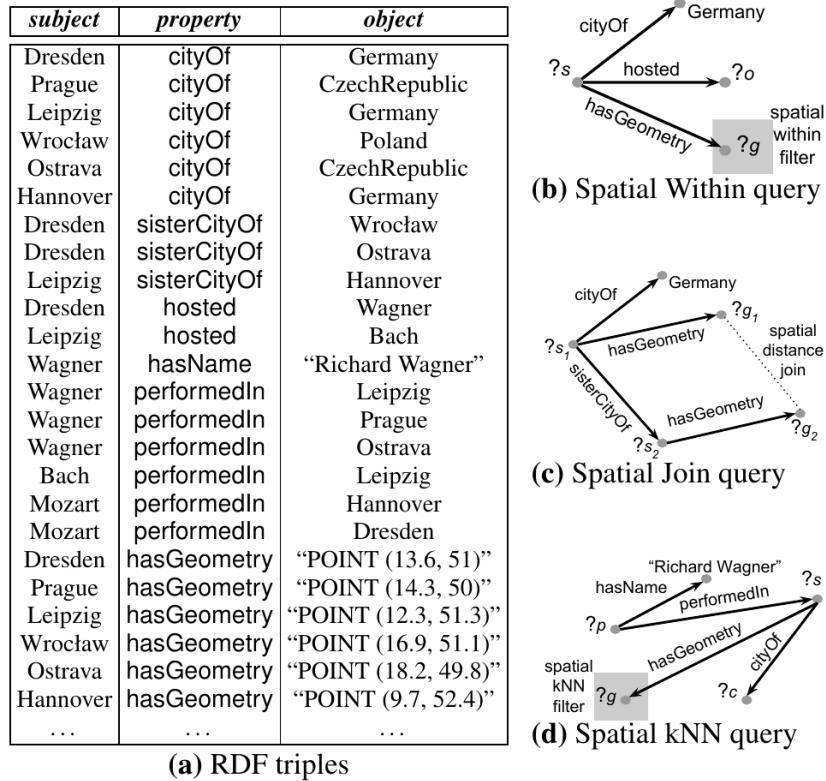


Figure 3.1: Example of RDF data and three spatial queries.

and $?g$) connected via a set of triple patterns which also include constants, i.e., Germany. For example, if POLYGON(...) covers the area of East Germany, (Dresden, Wagner) and (Leipzig, Bach) are results of this query. The query is represented by the pattern graph of Figure 3.1b. In general, queries can be represented as graphs with *chain* (e.g., $?s_1$ hosted $?s_2$. $?s_2$ performedIn $?s_3$.) and *star* (e.g., $?s$ cityOf $?o$. $?s$ hosted Wagner.) components. Another exemplary query, which includes a spatial join predicate, represented by the pattern graph of Figure 3.1c, is:

```
Select ?s1 ?s2
Where ?s1 cityOf Germany . ?s1 sisterCityOf ?s2 .
      ?s1 hasGeometry ?g1 . ?s2 hasGeometry ?g2 .
Filter DISTANCE(?g1, ?g2) < “300km”;
```

This query asks for pairs of sister cities (i.e., $?s_1$ and $?s_2$) such that the first city (i.e., $?s_1$) is in Germany and the distance between them does not exceed 300km. In the exemplary RDF base of Figure 3.1a, (Dresden, Wrocław) and (Leipzig, Hannover) are results of this query while (Dresden, Ostrava) is not returned as the distance between Dresden and Ostrava is around 500km. Finally, an exemplary query with a kNN predicate, represented by the pattern graph of Figure 3.1d, is the next:

```

Select ?s ?c
Where ?p hasName "Richard Wagner". ?p performedIn ?s .
    ?s cityOf ?c . ?s hasGeometry ?g .
Filter kNN(?g, "POINT(...)", 2);

```

This query asks for the two closest to the specified point cities where Richard Wagner has performed, together with their respective countries. E.g., if `POINT(...)` refers to the city of Chemnitz (12.8, 50.8), then the result of the query in the RDF base of Figure 3.1a consists of the tuples (Leipzig, Germany) and (Prague, CzechRepublic).

Besides queries, we also consider delete, insert, and update operations on RDF data. Updates in SPARQL (and GeoSPARQL) are expressed via `DELETE` and `INSERT` statements following the format:

`Delete|Insert [triples]`

For example, to update the name of the entity `Wagner` in the RDF base of Figure 3.1a, one can simply apply the following two statements:

```

Delete Wagner hasName "Richard Wagner";
Insert Wagner hasName "Wilhelm Richard Wagner";

```

3.3 A Basic Spatial Extension

In the remainder of the chapter, we present the steps of extending a standard query evaluation framework for triple stores (i.e., the framework of RDF-3X) to efficiently handle the spatial components of RDF queries. In RDF-3X, a query evaluation plan is a tree of operators applied on the base data (i.e., the set of RDF-triples). The leaves of the tree are any of the 6 SPO clustered indices. The operators apply either selections or joins. Each operator addresses a triple of the query pattern and instantiates the corresponding variables; the instantiated triples (or query subgraphs) are passed to the next operator, until they reach the root operator, which computes instances for the entire query graph.

This section outlines the basic (but essential) spatial extension to RDF-3X, which improves the spatial RDF-3X extension of Brodt et al. [BNM10] to support spatial join and kNN query evaluation. We also discuss drawbacks of the basic extension that motivated us to design the spatial encoding scheme described in Section 3.4 and the query evaluation algorithms that use it in Section 3.5.

Spatial Indexing. Spatial entities i.e., resources associated to spatial literals like `POINT` and `POLYGON`, are indexed by an R-tree [Gut84]. For each entity associated to a polygon, there is an entry at a leaf of the R-tree of the form (*mbr*, ID), where *mbr*

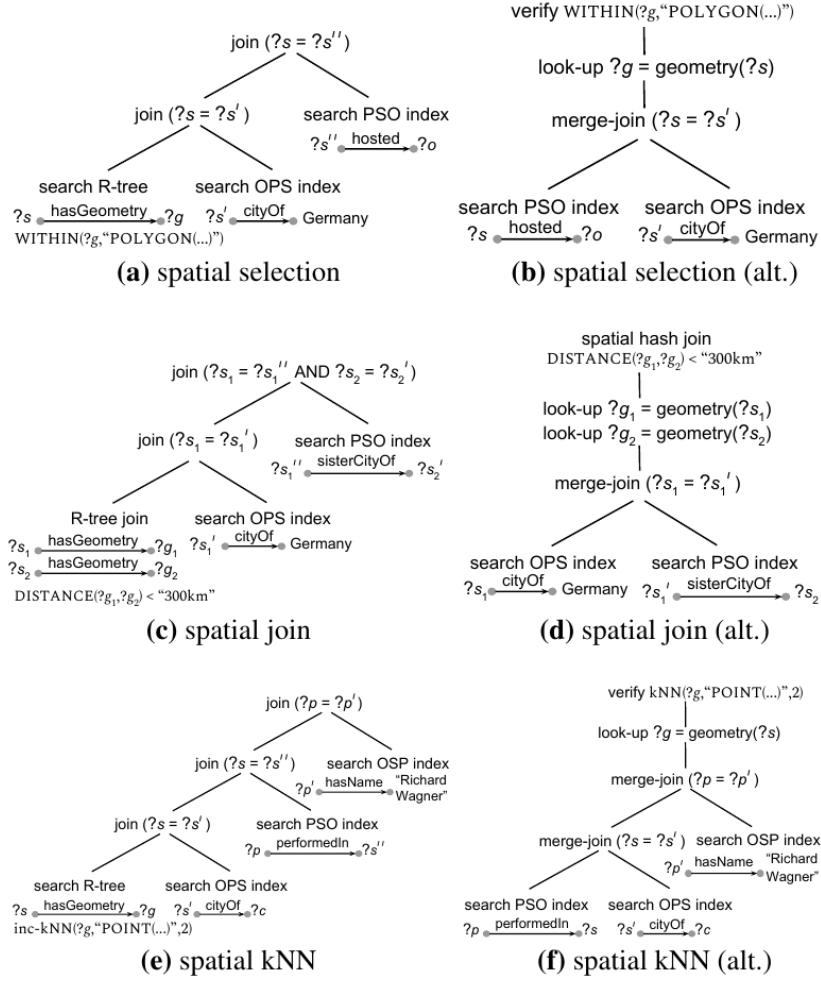


Figure 3.2: Possible query plans in the basic extension.

is the minimum bounding rectangle (MBR) of the polygon. For each entry associated to a point pt , there is a (pt, ID) entry.

Spatial Selections. Given a query with a spatial selection Filter condition, the optimizer may opt to use the R-tree to evaluate this condition first and retrieve the IDs of all entities that satisfy it.¹ However, the output fed to the operators that follow (i.e., those that process non-spatial query components) is in a random order. Thus, query evaluation algorithms that rely on the input being in an *interesting order* (such as merge-join) are inapplicable. On the other hand, if the spatial selection is evaluated after another (i.e., non-spatial) operator, the R-tree cannot be used because the input is no longer indexed. So, in this case, the system must look up the geometries of the entities that qualify the preceding operator at the dictionary, incurring

¹For entities that have point geometries, the spatial selection can be evaluated using only the R-tree. If the entities have non-point geometries, the R-tree search may result in false positives, thus, the final results of the spatial filter are confirmed by retrieving the exact geometries from the dictionary.

significant cost. Figure 3.2a and Figure 3.2b illustrate two alternative plans for the spatial selection query of Figure 3.1b. The plan of Figure 3.2a uses the R-tree to perform the spatial selection and joins the result with the instances of triple $?s$ cityOf Germany. Finally, the join results are joined with the results of $?s$ hosted $?o$. The plan of Figure 3.2b first evaluates the non-spatial part of the query and then looks up and verifies the geometries of all $?s$ instances in it (i.e., the R-tree is not used here).

Spatial Joins. The R-tree can also be used to evaluate spatial join Filter conditions, by applying join algorithms based on R-trees. We implemented three algorithms for this purpose. First, the R-tree join algorithm [BKS93] can be used in the case where both spatially joined variables involved in the Filter condition are instantiated directly from the base data and do not come as outputs of other query operators. Second, we use the SISJ algorithm [MP03] for the case where the R-tree can be used only for one variable. Finally, we implemented a spatial hash join (SHJ) algorithm [LR96] for the case where both inputs of the spatial join filter condition are output by other operators.² As in the case of spatial selections, spatial join algorithms do not produce interesting orders and for spatial join inputs that are instantiated by preceding query operators, the system has to perform dictionary look-ups in order to retrieve the geometries of the entities before the join. Figure 3.2c and Figure 3.2d illustrate two alternative plans for the spatial join query of Figure 3.1c. The plan of Figure 3.2c applies an R-tree self-join [BKS93] to retrieve nearby $(?s_1, ?s_2)$ pairs and then binds $?s_1$ with the result of $?s_1$ cityOf Germany. The output is then joined with the result of $?s_1$ sisterCityOf $?s_2$. The plan of Figure 3.2d first evaluates the non-spatial part of the query and then looks up the geometries of all $(?s_1, ?s_2)$ pairs, and joins them using SHJ. In the following, we briefly describe SISJ and SHJ for completeness.

SISJ joins a spatial input A which is not indexed, with an R-tree B . Assuming that we want to use H hash buckets, SISJ first divides the entries at the uppermost level of B that contains at least H entries into H groups based on their spatial proximity. The i -th group has as spatial extent the MBR of all entries in group i . Bucket B_i contains all objects in the subtrees of B pointed by the entries in the i -th group. The objects from A are hashed to buckets such that bucket A_i contains all objects that intersect the spatial extent of the i -th group. Finally, each A_i is spatially joined in memory with B_i (e.g., using plane sweep). Our SHJ implementation pulls the smallest of the two join inputs (based on the query optimizer’s estimation) and constructs from it a spatial hash table in memory. Each hash bucket corresponds to a cell in a 2D grid with side equal to the distance join threshold ϵ . Each entity from the hashed join input is assigned to all buckets (cells) that it spatially overlaps. Then, SHJ pulls the

²If the spatial join inputs are very small, we simply fetch the geometries of the input entity sets and do a nested-loops spatial join.

records from the other input one by one and, for each spatial entity e , (i) it retrieves e 's geometry from the dictionary, (ii) identifies the cell c whereto e belongs, and (iii) accesses the buckets that correspond to c and its neighboring cells to find candidate entities that can match with e based on their spatial approximations. For each such candidate entity e' , the operator computes the exact distance between e and e' , and outputs the join pair (e, e') if the distance is at most ϵ .

Spatial kNN. The R-tree can also be used to evaluate a spatial kNN predicate in the Filter clause. In this case, the nearest entities are fetched from the R-tree and fed to the operators that follow. Since some of these entities might be filtered out by subsequent operators, we should use an *incremental NN algorithm* for R-trees [Mam11] (an operation often referred to as *distance browsing*). As in the case of spatial selections, the drawback of using this algorithm is that the IDs of the fetched entities are in random order, preventing the use of efficient operators that rely on interesting orders. On the other hand, when the R-tree is not used, the kNN evaluation needs to perform dictionary lookups to fetch the geometries of all entities that qualify the RDF part of the query and keep track using a heap, the k nearest entities. Figure 3.2e and Figure 3.2f depict two possible plans that correspond to the two options above for the query pattern of Figure 3.1d.

3.4 Encoding the Spatial Dimension

We observe that in most RDF engines, the IDs given to resources or literals at the dictionary mapping do not carry any semantics. Instead of assigning random IDs to resources, we propose to encode into the ID of a resource an approximation of the resource's location and geometry that can be used to (i) apply spatial Filter conditions on-the-fly in a query evaluation plan, and (ii) define spatial operators that apply on the approximations.

Figure 3.3b illustrates the *Hilbert* space filling curve, a classic encoding scheme of spatial locations into one-dimensional values. We partition the space using a grid, and order the cells based on the curve. We then divide the ID given to a spatial resource r into two components: (i) the Hilbert order of the cell where r spatially resides occupies the m most significant bits (where $2^{m/2} \times 2^{m/2}$ is the resolution of the grid), and (ii) a *local* identifier which distinguishes r from other resources that reside in the same cell as r . Since the RDF data may also contain resources or literals, which are not spatial, we use a different range of ID values for non-spatial resources with the help of the least significant bit as a flag. In the toy example of Figure 3.3a, the least significant bit (b_0) indicates whether the entity modeled by the ID is spatial ($b_0 = 1$) or non-spatial ($b_0 = 0$), the next 4 bits are used for the local identifier,

3.4. Encoding the Spatial Dimension

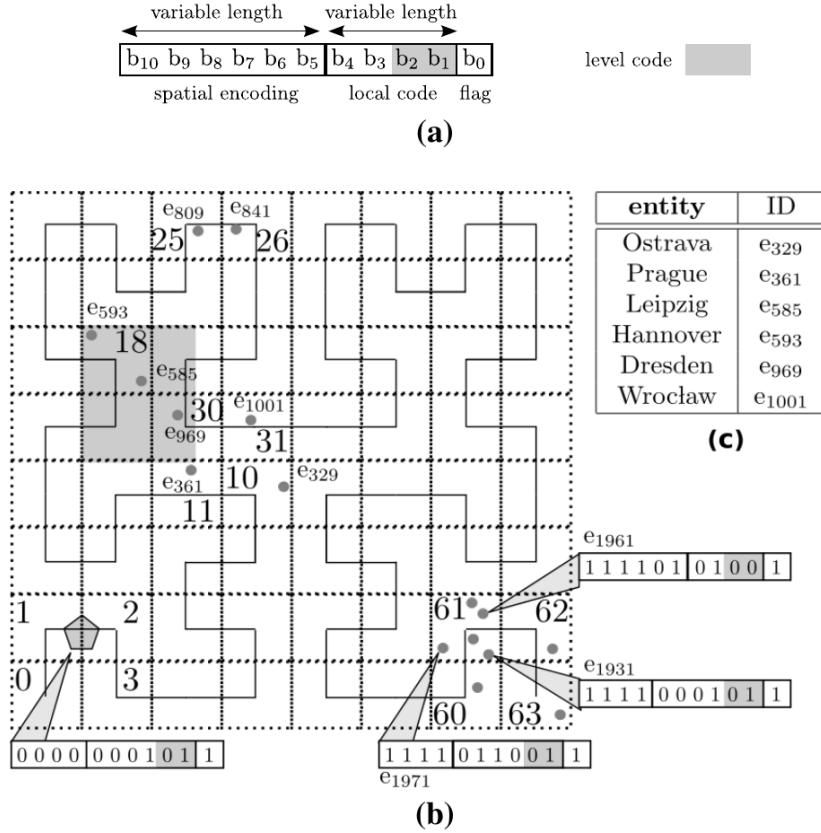


Figure 3.3: Spatial encoding of entity IDs.

and the 6 most significant bits encode the Hilbert order of the cell. For example, in Figure 3.3b, entity e_{1961} is spatial (b_0 is set) and it is located in the cell with Hilbert order 111101 (cell with ID 61), having local code 0100. For a non-spatial resource, bit b_0 would be 0 and the remaining ones would not have any spatial interpretation. Figure 3.3c illustrates which IDs encode the cities of Figure 3.1a.

In the case of a skewed dataset, a cell may overflow, i.e., there could be too many entities falling inside it rendering the available bits for the local codes of entities in it insufficient. In this case, entities that do not fit in a full cell are assigned to the parent of the cell in the hierarchical space decomposition. For instance, consider the data in Figure 3.3b and assume that the cell with ID 61 is full and that the entity e_{1931} cannot be assigned to it. e_{1931} will be assigned to the parent cell, i.e., the square that consists of the cells 60, 61, 62, and 63. This cell's encoding has 4 bits, that is, 2 bits less than its children cells. These 2 bits are now used for the local encoding of entities in it. Intuitively, as we go up in the hierarchy of the grid, each cell can accommodate more entities. An entity that must be assigned to an overflowed cell ends in the first non-full ancestor of that cell as we go up in the hierarchy. The $\lceil \log_2(m/2) \rceil$ least significant bits of the local code area are reserved to encode the level of the spatially-encoded

cell in the ID (the most detailed level being 0). Here, $m = 6$, hence, 2 bits of the local code are used to denote the level of the cell that approximates each entity.

The encoding we described is also used for arbitrary geometries that may overlap with more than one cells of the bottom level. For example, the polygon at the lower left corner of the grid of Figure 3.3b spans across cells with IDs 1 and 2, thus, it will be assigned to their parent cell, which has a spatial encoding 0000. Due to the variable number of bits given to the spatial approximations, the encoding is also suitable for dynamic data (i.e., inserted entities that fall into overflowed cells are given less accurate approximations).

The most important benefit of the spatial encoding is that the (approximate) evaluation of spatial predicates can be seamlessly combined with the evaluation of non-spatial patterns in SPARQL. For example, spatial Filter conditions included in a query which are bound to entity variables (for example, `?s hasGeometry ?g`, `Filter WITHIN (?g, "POLYGON(...)"")`) can be evaluated on-the-fly at any place in the evaluation plan where the entity variable (e.g., `?s`) has been instantiated, by decoding the IDs of the instances. Note that the spatial mapping is only approximate (based on the conservative grid approximation of the spatial locations); by applying a spatial predicate on the approximations (i.e., cells) of the entities, false hits may be included in the results, which need to be verified. Still, for many entities, the spatial approximation suffices to confirm that they are definitely included (or not) in the query result. This way, random accesses for retrieving their exact geometries are avoided.

A side-benefit of using a Hilbert-encoded grid to approximate the object geometries is that by counting the number of resources in each cell (counting is already performed by the mapping scheme), we can have a spatial histogram to be used for selectivity estimation in query optimization (this issue will be discussed in detail in Section 3.6). SRX uses the encoding we described to accelerate queries with spatial predicates as shown in the next section.

3.5 Query Evaluation

We now show how the encoding scheme of SRX further extends the basic framework presented in Section 3.3 to apply efficient spatial filters directly on the entities IDs and reduce the number of dictionary lookups as well as the number of expensive spatial operations on the actual geometries.

All operators we describe in this section evaluate the spatial predicates in two phases: first, by applying the spatial predicate on the IDs of the entities (filtering phase) and, second, by fetching the actual geometries only for the results that could not be verified in the first phase. In general, the sooner we apply the on-the-fly filtering

the better because it does not incur any I/O cost and its CPU cost is negligible³. For spatial range and join predicates, the on-the-fly filtering can be done early: after each non-spatial operator that instantiates entity variables, which also appear in a WITHIN or DISTANCE predicate, the condition is applied to the spatially encoded IDs of the entities. In such cases, after applying the filter, we also append a *verification bit* (or vbit) to the tuples that pass the filter. This bit is used in the second phase as follows: if, for a tuple, the verification bit is 1, the tuple is guaranteed to qualify the corresponding spatial predicate (no verification is required). Yet, if the bit is 0, it is unknown at this point whether the exact geometries of the entities in the tuple qualify the spatial predicate and so they cannot be pruned based on their spatial approximations encoded in their IDs. By the end of processing all non-spatial query components, for tuples having their vbits 0, the system fetches the exact geometries of the involved entities and perform verification of the spatial Filter conditions.

3.5.1 Spatial Range Filtering

Spatial range queries bind a pattern variable to geometries that are spatially restricted by a range. As an example, consider again the query depicted in Figure 3.1b. Our encoding scheme allows the filtering phase of the spatial range query to be performed on-the-fly while scanning the indices, as illustrated by the evaluation plan of Figure 3.4. The plan searches the OPS and PSO indexes in order to fetch and merge-join ($?s = ?s'$) the two lists that qualify patterns $?s$ cityOf Germany, $?s'$ hosted $?o$, i.e., the plan follows the logic of the plan shown in Figure 3.2b. Taking advantage of the spatial encoding, before the merge-join, the plan of Figure 3.4 applies the spatial filter for $(?s \text{ hasGeometry } ?g, \text{WITHIN}(?g, \text{"POLYGON (...)"}))$ on the instances of $?s$ that arrive from scanning the OPS and PSO indexes; a vbit is appended to each survived tuple, to be used by the next operators. In this example, assume that the spatial entities and the spatial range (i.e., “POLYGON (...)”) are the points and the shadowed range, respectively, shown in Figure 3.3b. Entities e_{809} and e_{841} are filtered out from the left scan, because they are not within the cells that intersect the query spatial range. Entity e_{969} survives spatial filtering, but we cannot ensure that it qualifies the spatial range predicate either, because its cell-ID is not completely covered by the spatial query range; therefore the vbit for the tuples that involve e_{969} is 0. On the other hand, the vbit for tuples containing e_{585} or e_{593} is 1 as their cell-ID is completely covered by the spatial range. Therefore, after the merge-join, we only have to fetch and verify the geometry of e_{969} . Range filtering is applied at the bottom of query plans, after each index scan that contains a respective spatial variable.

³Most spatial predicates, when translated to the grid-based approximations of the encoding, involve distance computations and/or cheap geometry intersection tests.

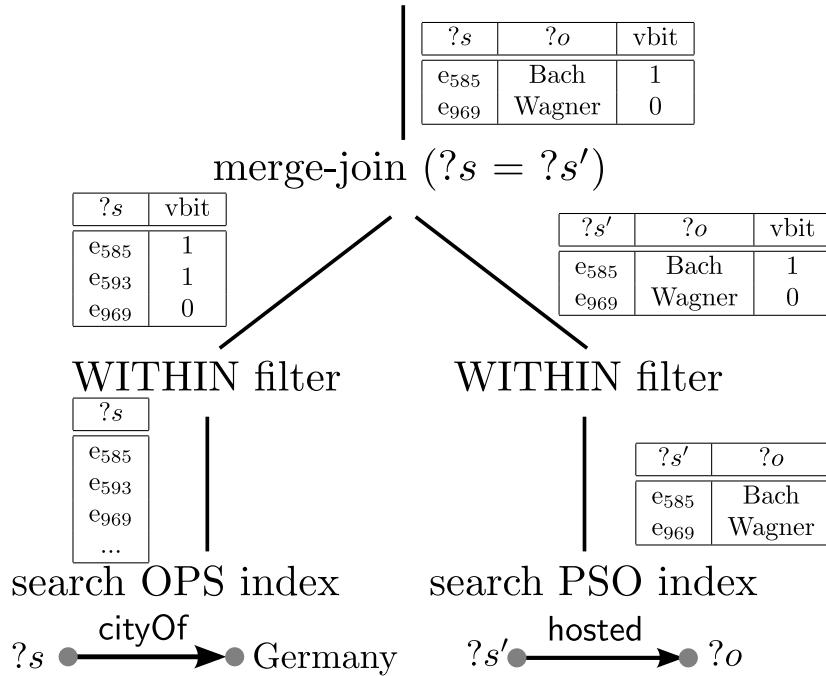


Figure 3.4: Plan for the query of Fig. 3.1b.

3.5.2 Spatial Join Filtering

Similar to spatial range selections, the filtering phase for binary spatial join predicates can also be applied on-the-fly, as soon as the IDs of candidate entity pairs are available. As an example, consider the join query depicted in Figure 3.1c. A possible query evaluation subplan is given in Figure 3.5, which follows the flow of the plan shown in Figure 3.2d; however, the plan of Figure 3.5 applies the spatial join filter (i.e., the distance filter) early. By the time the candidate pairs $(?s'_1, ?s_2)$ are fetched by the index scan on PSO, the filter is applied so that only the pairs of entities that cannot be spatially pruned are passed to the next operator. Assume that the pairs that qualify $?s'_1$ `sisterCityOf` $?s_2$ are as shown at the right-bottom side of Figure 3.5, above the search PSO index operator. Assume that the distance threshold (i.e., 300km) corresponds to the length of the diagonal of each cell in Figure 3.3. After applying the distance spatial filter on all $(?s'_1, ?s_2)$ pairs produced by the PSO index scan, the pairs that survive are (e_{585}, e_{593}) , (e_{969}, e_{1001}) and (e_{969}, e_{329}) . However, only entities e_{585} and e_{593} are guaranteed to be within ϵ distance as they belong to same cell; thus, the vbit for pair (e_{585}, e_{593}) is 1. When the pairs are merge-joined ($?s_1 = ?s'_1$) with the results of the OPS index-scan on the left (for $?s_1$ `cityOf` Germany), the vbits of qualifying tuples are carried forward.

In contrast to the range filter that always appears at the bottom level of the operator tree, distance join filtering can be applied on any intermediate relation that

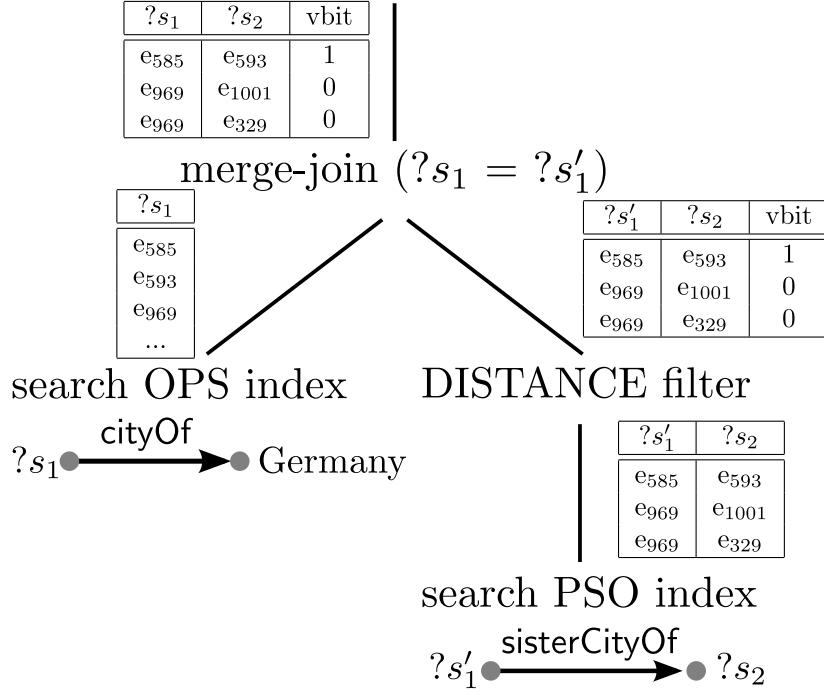


Figure 3.5: Plan for the query of Fig. 3.1c.

contains two joined spatial variables. This case is possible when two relations are first joined on attributes other than the spatial entities. In Section 3.6.1, we show how the query optimizer can identify all pairs of spatially joined variables in a query, for which distance join filtering can be applied; here, we only gave an example with a pair coming from an index scan.

3.5.3 Spatial Merge Join on Encoded Entities

In this section, we propose a *spatial merge join* (SMJ) operator that applies directly on the spatial encodings (i.e., the IDs) of the entities from the two join inputs. SMJ assumes that both its inputs are sorted by the IDs of the spatial entities to be joined. Like the spatial filters discussed above, this algorithm only produces pairs of entities for which the exact geometries are likely to qualify the spatial join predicate (typically, a DISTANCE filter). Again, a verification bit is used to indicate whether the join condition is definitely qualified by a pair. Besides using the spatially encoded IDs of the entities, SMJ takes advantage of and preserves the ID-based sorting of its inputs. Thus, the algorithm does not break the pipeline within the operator tree, as any other spatial join algorithm would. Note that SMJ is a binary join algorithm that takes two inputs, while the filtering technique discussed in Section 3.5.2 takes a single input of candidate join pairs and merely applies the join condition on the entity-ID pairs on-the-fly.

Similarly to a classic merge join algorithm, SMJ uses a buffer B_R to cache the streaming tuples from its right input R . For each entity e_l read from the left input L , SMJ uses the ID of e_l to compute the minimum and maximum cell-IDs that could include entities e_r from R , which could possibly pair with e_l in the join result, based on the given DISTANCE filter. SMJ then keeps reading tuples from input R and buffering them into B_R , as long as they are likely to join with e_l . As soon as B_R is guaranteed to contain all possible entities that may pair with e_l , SMJ computes all join results for e_l and discards e_l (and potentially tuples from B_R).

We now provide the details of SMJ. The algorithm is based on the (on-the-fly and on-demand) computation of four cell IDs for each entity e based on e 's ID. First, minNeighborID and maxNeighborID are the minimum and maximum cell-IDs that could include entities that pair with e in the join result, respectively. To compute these cells, we have to expand e 's cell based on the distance join threshold and find the minimum and maximum cell-ID that intersects the resulting range. For example, consider entity e_{841} contained in cell with ID 26 in Figure 3.3b and assume that the join distance threshold equals the diagonal length of a cell. For this entity, $\text{minNeighborID}=18$ and $\text{maxNeighborID}=39$. Second, minChildID and maxChildID correspond to the minimum and maximum cell-IDs that have a common non-empty ancestor (in the hierarchical Hilbert space decomposition) with the cell of e . For entity e_{841} which has only empty ancestors, the minChildID and maxChildID are both 26, that is, the cell ID of e_{841} . For e_{1931} , the minChildID and maxChildID are 60 and 63 respectively because e_{1931} is assigned to a cell at the first level of the grid.

At each step, the distance join is performed between the current entity e_l from the left input and all entries in B_R . After reading e_l , SMJ reads entries e_r and buffers them into B_R and stops as soon as e_r 's minChildID is greater than the maxNeighborID of e_l ; then we know that we can join e_l and all entities in B_R and then discard e_l , because any unseen tuples from R cannot be included within the required distance from e_l .⁴ For example, consider the buffered inputs of Figure 3.6 that have to be joined. The maxNeighborID of the first entity e_{585} on the left is smaller than the minChildID of entry e_{1931} , therefore e_{585} cannot be paired with entries after e_{1931} (that are guaranteed to have minChildID greater than the maxNeighborID of e_{585}).⁵ Thus, for any e_l , we only need to consider all entities in R before the first entity having minChildID greater than the maxNeighborID of e_l .

After e_l has been joined, it is discarded. At that point we also check if buffered

⁴Recall that the inputs are sorted by ID and that entities may be encoded at different granularities due to data skew or geometry extents. Therefore, using the cell-ID of e_r alone is not sufficient and we have to use the minChildID of e_r .

⁵The fact that the entities arrive from the inputs sorted by their IDs guarantees that they are also sorted based on their minChildIDs .

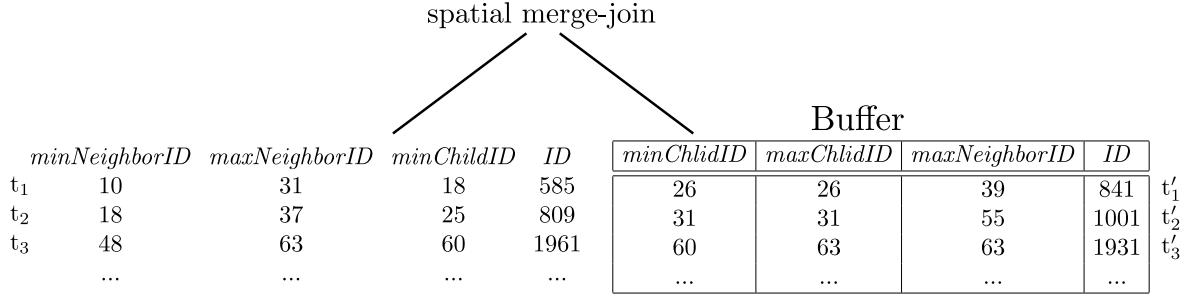


Figure 3.6: Example of SMJ.

tuples in B_R can also be removed. In order to decide this, we use maxNeighborID of each entity on the right. In case this is smaller than the minChildID of the next entity in L , then the right entry can be safely removed from the buffer without losing any qualifying pairs. Below, we give a pseudocode for SMJ.

Algorithm: SMJ

Input : Two join inputs L and R ; a distance threshold ϵ
Output : Grid-based spatial distance join of L and R

- 1 Initialize (empty) buffer B_R ;
- 2 $e_r = R.\text{get_next}();$ add e_r to B_R ;
- 3 **while** $e_l = L.\text{get_next}()$ **do**
- 4 Prune from B_R all tuples e_r such that $e_r.\text{maxNeighborID} < e_l.\text{minChildID}$;
- 5 **while** $e_l.\text{maxNeighborID} \geq e_r.\text{maxChildID}$ **do**
- 6 $e_r = R.\text{get_next}();$ add e_r to B_R ;
- 7 join e_l with all tuples in B_R and output results to the next operator;

We now discuss some implementation details. First, the required min/maxNeighborID and min/maxChildID for the entries are computed fast on-the-fly by simple operations. In particular, min/maxChildIDs are computed by shifting (or masking) bits to keep only those most significant bits that encode the cell ID at a particular level of the grid (cf. Section 3.4). For the neighbor IDs, we rely on the id-to-offset and offset-to-id Hilbert transformations as follows. First, we use e 's ID and the (normalized) distance threshold ϵ to identify the offsets of the bottom-level cells that must be examined. Then, we transform the offsets back to the corresponding cell IDs, and we use the latter to compute the minimum and maximum entity IDs by masking bits in the local code (i.e., the least significant bits - cf. Section 3.4). Second, for joining an entity e_l from L , we scan through the qualifying entities of B_R and compute their grid-based distances to e_l , but only for entities whose $\text{minChildID}-\text{maxChildID}$ range overlaps with the $\text{minNeighborID}-\text{maxNeighborID}$ range of e_l ; this is a cheap filter

used to avoid grid-based distance computations. Finally, we buffer all tuples that have the same entity ID (in either input). For such a buffer, we perform the join only once but generate all join pairs.

3.5.4 Spatial Hash Join on Encoded Entities

If either of the two inputs of a spatial join is not ordered with respect to the joined entities, SMJ is not applicable. In this case we can still use the IDs of the joined entities to perform the filter step of the spatial join. The idea is to apply a *spatial hash join* (SHJ-ID) algorithm (similar to that proposed in [LR96]) using the approximate geometries of the entities taken from their IDs.⁶ SHJ-ID simply uses the existing assignment of the entities to the cells of the grid (as encoded in their IDs) and considers each such cell as a distinct bucket. The only difference from a typical spatial hash join algorithm is that in the bucket-to-bucket join phase, we have to consider all levels of the encoding scheme. So, each bucket from the left input, corresponding to a cell c , is joined with all buckets from the right input which correspond to all cells that satisfy the DISTANCE filter with c . The output of SHJ-ID is verified as soon as the geometries of the candidate pairs are retrieved from disk.

3.5.5 Spatial kNN on Encoded Entities

kNN predicates are evaluated differently from WITHIN and DISTANCE predicates in that no early spatial filtering or verification bits are utilized. We introduce two kNN operators that make use of the encoding: one for handling entities whose IDs come from the previous operator in a random order (Section 3.5.5.1), and a second one that exploits ordering (Section 3.5.5.2) and, thus, can be used efficiently in combination with other order-preserving operators. Both operators are applied in a pipelined fashion at the root of the operator tree (i.e., on the output of the previous operators) and are inspired by the work in [MHP05]. The difference compared to previous kNN operators is the integration with the multi-level encoding scheme of Section 3.4. This integration enables us to (i) compute approximate distances using arithmetic operations on the entity IDs, and (ii) leverage the interesting orders preserved by previous operators in the query plan to reduce random I/Os and improve performance. Random I/Os are common in index-based kNN algorithms from Section 3.3, which we compare with our approach in Section 3.8.2.

⁶Recall that the actual geometries of the entities have not been retrieved yet; otherwise, SHJ [LR96] would be used (see Section 3.3).

3.5.5.1 kNN on Unsorted Entity IDs

The logic of first kNN operator is given in Algorithm KNN-Unsorted-Input. The operator takes as input a point p (the one specified in the Filter clause of the query) along with an iterator I on the tuples coming from the previous operator in the query plan. Let t be a tuple in I and e be the ID of the spatial entity in t that is used in the evaluation of the kNN predicate. The operator uses two priority queues Q_1 and Q_2 to keep tuples ordered in ascending Euclidean distance of e from p 's actual geometry: in the former queue, the distance has been calculated based on e 's cell whereas in the latter based on e 's actual geometry.

Algorithm: KNN-UNSORTED-INPUT

Input : Input I from the previous operator; a point p

Output : k tuples from I that satisfy the kNN predicate

Param. : The number k

```

1 let  $Q_1, Q_2$  be two priority queues;
2  $lastDist = \infty$ ;
3 while  $t = I.get\_next()$  do
4    $\quad$  POPULATE- $Q_1(t, p)$ ;
5 while  $Q_1$  is not empty do
6    $\quad$   $(t, minDist) = Q_1.pop()$ ;
7   if  $minDist \geq lastDist$  then
8      $\quad$  break;
9    $\quad$  POPULATE- $Q_2(t, p, lastDist, k)$ ;
10 return all  $t$  tuples in  $Q_2$ ;
```

Function: POPULATE- $Q_1(t: \text{TUPLE}, p: \text{POINT})$

```

1 let  $e$  be the ID of the spatial entity in tuple  $t$ ;
2 let  $c$  be the grid cell  $e$  belongs to;           //extracted from  $e$ 
3 if  $c$  is the last-level cell then
4    $\quad$   $minDist = 0$ ;
5 else if  $c$  is an upper-level cell then
6    $\quad$  find the bottom-level child cell of  $c$ , let  $c_b$ , which is the
7    $\quad$  nearest to the given point  $p$ ;
8    $\quad$  set  $minDist$  to the minimum distance of  $c_b$  from  $p$ ;
8 else
9    $\quad$  // $c$  is a bottom-level cell
10   $\quad$  set  $minDist$  to the minimum distance of  $c$  from  $p$ ;
10  $Q_1.push((t, minDist))$ ;           //keep in ascending  $minDist$ 
```

Function: POPULATE- $Q_2(t: \text{TUPLE}, p: \text{POINT}, lastDist:$

FLOAT, $k: \text{INTEGER}$)

```

1 let  $e$  be the ID of the spatial entity in tuple  $t$ ;
2 retrieve  $e$ 's geometry from dictionary and compute the exact
   distance between  $e$  and  $p$ , let  $exactDist$ ;
3  $Q_2.push((t, exactDist))$ ;           //keep in ascending  $exactDist$ 
4 if  $|Q_2| = k$  then
5    $\quad$  set  $lastDist$  equal to  $exactDist$  of the  $k$ -th entry in  $Q_2$ ;
```

The evaluation proceeds in two phases. First, the operator pulls all tuples from the previous operator in the query plan and populates Q_1 (lines 3-4). The function `Populate- Q_1` uses the multi-level encoding scheme to compute the minimum distance between e 's cell and p ($minDist$) and keeps entries in ascending $minDist$. Note that $minDist$ is an approximation of the exact distance between the entity e and the point p ; the latter is computed only in the second phase of the algorithm (lines 5-9) where the operator starts draining Q_1 to populate Q_2 . Specifically, each time an entry is popped from Q_1 , the exact geometry of e is retrieved via a dictionary lookup and the tuple t is pushed into Q_2 using now the exact distance between e and p ($exactDist$ in function `Populate- Q_2`). The draining of Q_1 stops when the algorithm pops an entity e whose minimum possible distance from p is at least equal to the current exact distance of the k -th element in Q_2 (lines 7-8 in `KNN-Unsorted-Input`).

In contrast to Q_2 that holds at most k tuples from the input I , Q_1 is populated with all tuples from I in the first phase of `KNN-Unsorted-Input`. The intuition behind this strategy is to sort the entities based on their cells and use this ordering to minimize the expensive geometry lookups in the second phase. Since the IDs of the spatial entities come out of order and each next entity may fall anywhere in the grid, Q_1 must store all input tuples from I . This increases the memory footprint (and the latency) of `KNN-Unsorted-Input` significantly when the RDF part of the query is not selective. When the spatial entities come in order, we can tackle this problem with the kNN operator we describe next.

3.5.5.2 kNN on Sorted Entity IDs

The second kNN operator we introduce uses an adaptation of the CPM technique from [MHP05] and its logic is given in Algorithm `KNN-Sorted-Input`. The core idea here is to exploit the ordering of entities and avoid draining the iterator I , i.e., pulling the whole output from the previous operator in the query plan. To do so, the evaluation proceeds in “zones” starting from the (bottom-level) cell of the point p in the Filter condition. Each such zone consists of four rectangles (*up*, *down*, *left*, *right*), which in turn consist of bottom-level grid cells and form a “circular” area around p 's cell, as shown in Figure 3.7a. The operator follows the same steps as in CPM and extends the original technique to (i) work with our multi-level encoding scheme, and (ii) pull tuples from the input gradually, as it examines the zones.

First, the operator identifies the bottom-level cell c_p that contains the given point p (line 3). It then computes the maximum ID among all spatial entities that might fall in c_p (line 4). This is done in function `Compute-Limit`, which simply returns the maximum spatially encoded ID that exists in the database and falls either in c_p or in

Algorithm: KNN-SORTED-INPUT

Input : Input I from the previous operator; a point p
Output : k tuples from I that satisfy the kNN predicate

Param. : The number k

```

1 let  $Q_1, Q_2$  be two priority queues;
2  $lastDist = \infty$ ;
3 let  $c_p$  be the bottom-level cell that contains  $p$ ;
4  $limit = prevLimit = COMPUTE\_LIMIT(c_p)$ ;
   //load first round of input data into  $Q_1$ 
5 READ_NEXT( $I, limit, p$ );
6 for each rectangle  $r$  in the first zone around  $c_p$  do
7   compute the minimum distance of  $r$  from  $p$ , let  $minDist$ ;
8    $Q_1.push((r, minDist))$ ; //keep in ascending  $minDist$ 
9 while  $Q_1$  is not empty do
10    $(entry, minDist) = Q_1.pop()$ ;
11   if  $minDist \geq lastDist$  then
12     break;
13   if  $entry$  is a tuple  $t$  with a spatial entity then
14     POPULATE- $Q_2(t, p, lastDist, k)$ ;
15   else
16     //entry is a rectangle  $r$ 
17     find the maximum bottom-level cell ID  $c_m$  falling in  $r$ ;
18      $limit = COMPUTE\_LIMIT(c_m)$ ;
19     if  $limit > prevLimit$  then
20        $prevLimit = limit$ ;
          //load next round of input data
21       READ_NEXT( $I, limit, p$ );
22     let  $r'$  be the next zone rectangle in the direction of  $r$ ;
23     set  $minDist$  to the minimum distance of  $r'$  from  $p$ ;
24      $Q_1.push((r', minDist))$ ; //in ascending  $minDist$ 
25 return all  $t$  tuples in  $Q_2$ ;

```

Function: READ_NEXT(I : ITERATOR, $limit$: INTEGER, p : POINT)

```

1 while  $t = I.peek()$  do
2   let  $e$  be the ID of the spatial entity in tuple  $t$ ;
3   if  $e \leqslant limit$  then
4      $t = I.get\_next()$ ; //Pull happens at this point
5     POPULATE- $Q_1(t, p)$ ;
6   else break;

```

Function: COMPUTE_LIMIT(c_{id} : BOTTOM-LEVEL CELL)

```

1 let  $c_i$  be the parent cell of  $c_{id}$  at the  $i$ -th grid level; //  $c_0 \equiv c_{id}$ 
2 let  $m_i$  be the maximum encoded spatial entity ID in  $c_i$ ;
3 return  $\max_{0 \leq i \leq 13} m_i$ ; //14 grid levels with 32-bit IDs

```

a parent cell of c_p ⁷. Compute-Limit is a very cheap function that requires only a few lookups in the grid statistics kept in memory. The returned ID serves as an upper limit to bound the number of tuples pulled from I when populating Q_1 in function Read-Next. At each step of the algorithm, only the tuples of the current examined zone (initially p 's cell) must be pulled from the input. To do so, the operator first peeks into I (line 1 in Read-Next) to check the entity ID e of the next tuple and decide if this ID is at most equal to the $limit$; if so, this means that e 's actual geometry might fall in the examined zone, thus, the tuple is pulled from I (line 4 in Read-Next) and the algorithm continues with peeking the next tuple; otherwise none of the following entities fall in the examined zone, thus, the algorithm exits the loop (line 6 in Read-Next), computes the distance of each rectangle in the first zone from p , as in original CPM, and adds the respective entry to Q_1 (lines 6-8 in KNN-Sorted-Input).

Then, the operator continues similarly to KNN-Unsorted-Input, i.e. it starts pulling from Q_1 (line 9) to populate Q_2 with the exact distances. The termination condition in lines 11-12 is the same as in KNN-Unsorted-Input. The only difference here is that, whenever the algorithm encounters a new rectangle r in Q_1 , the latter is used to update (i.e. increase) the $limit$ and pull the required additional tuples (if any) from the input I (lines 16-20). After that, the algorithm also expands the search space to the next zone (lines 21-23) by adding to Q_1 the rectangle of the next zone that is in the same direction (*up*, *down*, *left*, *right*) as r with respect to p 's cell. This is CPM's actual control flow and the correctness of the computation relies on the correctness of the original method (cf. Lemma 3.1 in [MHP05]). As a final comment, KNN-Sorted-Input is designed to pull as few tuples from I as possible (it exhausts I only in the worst case, i.e. when $limit$ is greater than all spatial IDs in I) and, thus, tends to perform much better than KNN-Unsorted-Input, as we show in Section 3.8.

Example 1. Consider the grid of Figure 3.7 where a_i denotes a spatial entity encoded at the bottom level and b_i denotes a spatial entity encoded at the exact next level. For simplicity, assume that there are no entities at higher levels. Assume also that each bottom-level cell has a side of 1 metric unit. Consider a query point p falling in cell 28 and let $k = 2$. Algorithm KNN-Sorted-Input first pulls from the input I and inserts into Q_1 all tuples with spatial entities that may fall in p 's bottom-level cell, i.e. all tuples from I before a tuple with a spatial entity ID greater than $b_2 = 907$ (recall that tuples in I are in ascending spatial entity ID order). Then, the algorithm proceeds with the insertion of the first zone rectangles L_1, R_1, U_1, D_1 resulting in a priority queue $Q_1 = \{(a_4, 0), (b_1, 0), (b_2, 0), (a_3, 0.1), (U_1, 0.1), (R_1, 0.2), (L_1, 0.8), (D_1, 0.9), (a_2, 2.8), (a_1, 4.9)\}$. Numbers in Q_1 depict the Euclidean distance of the respective entry (grid cell or zone

⁷In case there are no spatial entities in the database falling in c_p or one of its parent cells, then as $limit$ we use the first free (i.e. the minimum) spatial ID for an entity in c_p .

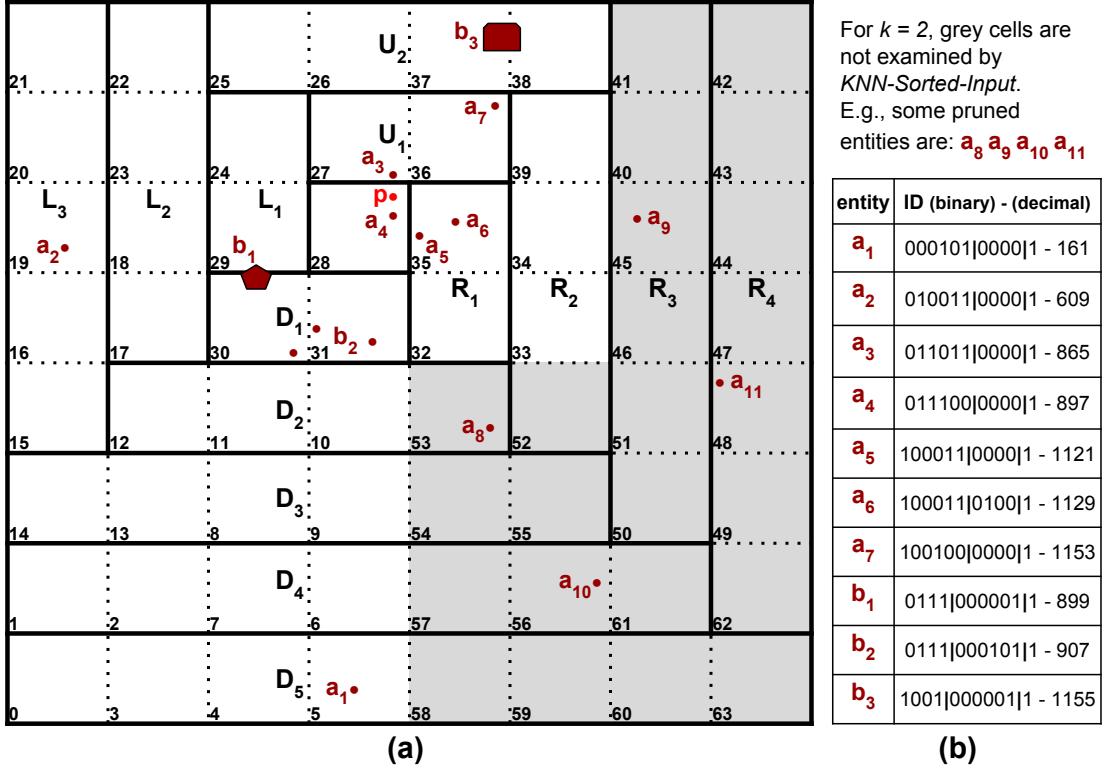


Figure 3.7: An example grid (a) with 64 cells at the bottom level (11-bit encoding) ordered according to the Hilbert curve and organized in CPM zones (L_i , R_i , U_i , D_i) around a query point p in cell 28. The entity IDs are shown on the right (b) in binary and decimal format.

rectangle) from p 's geometry. At the next step, the algorithm starts pulling entries from Q_1 to populate Q_2 . When it reaches the first rectangle entry U_1 , it computes the new limit = $b_3 = 1155$. At that point, we have $Q_1 = \{(R_1, 0.2), (a_5, 0.2), (a_6, 0.2), (a_7, \sqrt{0.05}), (b_3, \sqrt{0.05}), (L_1, 0.8), (D_1, 0.9), (U_2, 1.1), (a_2, 2.8), (a_1, 4.9)\}$ and $Q_2 = \{(a_3, 0.12), (a_4, 0.21)\}$. Distances in Q_2 have now been computed using the Euclidean distance between the entry's actual geometry and the point p . The algorithm then pops entry R_1 , which results in updating Q_1 only with $(R_2, 1.2)$, and terminates when it pops a_7 whose $\text{minDist} = \sqrt{0.05}$ is less than the $\text{lastDist} = 0.21$ of the previous entry a_4 popped from Q_1 . So, $Q_2 = \{(a_3, 0.12), (a_4, 0.21)\}$ and the entries a_3, a_4 are returned.

3.6 Query Optimization

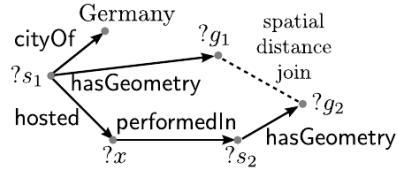
In this section, we describe our extensions to the query optimizer of RDF-3X, so as to take into consideration (i) the R-tree index and the query evaluation plans that involve it (Section 3.3) (ii) the query evaluation techniques described in Section 3.5 for spatial range and join queries. The encoding-based kNN operators (Section 3.5)

Select $?s_1 ?s_2$

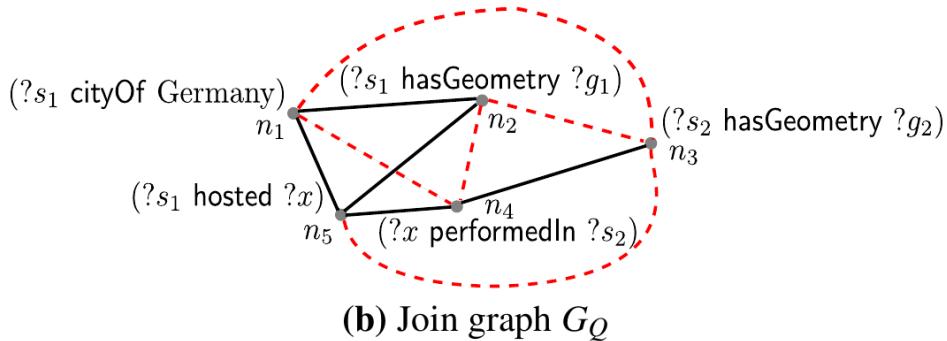
Where

$?s_1 \text{ cityOf Germany} .$
 $?s_1 \text{ hosted } ?x .$
 $?x \text{ performedIn } ?s_2 .$
 $?s_1 \text{ hasGeometry } ?g_1 .$
 $?s_2 \text{ hasGeometry } ?g_2 .$

Filter DISTANCE($?g_1, ?g_2$) < “200km”;



(a) RDF query



(b) Join graph G_Q

Figure 3.8: Augmenting a query graph.

do not affect query optimization as they are applied after the RDF part.

3.6.1 Augmenting the Query Graph

Consider the query depicted in Figure 3.8a. This query includes a spatial distance join between the geometries $?g_1$ and $?g_2$. The filtering phase of the spatial distance join can also be applied on the variables $?s_1$ and $?s_2$, using their IDs, as explained in Section 3.5.3. We call such variables *spatial variables*:

Definition 1. (Spatial Variable) A variable $?s_i$ at the subject position of a triple pattern $?s_i \text{ hasGeometry } ?g_i$ that appears in the *Where* clause of a query Q is called a *spatial variable*. We say that two spatial variables $?s_i, ?s_j$ ($i \neq j$) are joined iff $?g_i$ and $?g_j$ appear in the same DISTANCE predicate in the *Filter* clause of Q .

Spatial variables are identified in the beginning of the optimization process and they are used to augment the initial join query graph G_Q with additional join edges that correspond to the filtering step of the spatial operation. For example, the initial G_Q for the RDF query of Figure 3.8a is the graph shown in Figure 3.8b, considering solid lines only as edges; the nodes of G_Q are the triples of the RDF query graph and there is an edge between every pair of nodes that have at least one common variable. An ordering of the edges of G_Q corresponds to a join order evaluation plan.

The procedure of augmenting G_Q is given in Algorithm *Augment*. First, we identify all spatial variables in the query Q ; in our example, $?s_1$ and $?s_2$. Note that a spatial

variable $?s_i$ may also appear either as subject or object in triple patterns, other than $?s_i \text{ hasGeometry } ?g_i$. The second step is to collect all pairs of nodes in G_Q that include at least one spatial variable. In the example of Figure 3.8b, all nodes include one of $?s_1$ and $?s_2$. Then, for each pair of nodes (n_i, n_j) , where $n_i \neq n_j$, such that n_i includes $?s_1$ and n_j includes $?s_2$, we either add a new edge (if no edge exists between n_i and n_j) or we add the spatial join predicate (e.g., $\text{DISTANCE}(n_i.s_i, n_j.s_j) < \text{"200km"}$) in the set of predicates modeled by the edge between these two nodes (these are equality predicates for their common variables). For instance, n_4 and n_5 in the initial G_Q are connected by an edge with predicate $n_4.x = n_5.x$, but after the augmentation the predicates on this edge are $n_4.x = n_5.x$ and $\text{DISTANCE}(n_4.s_2, n_5.s_1) < \text{"200km"}$. This implies that the optimizer will consider two possible subplans for joining n_4 with n_5 . The first one will first perform the equality join on x and then evaluate the distance predicate whereas the second subplan will first perform the filtering phase of the spatial join on (s_1, s_2) and then apply the equality on x . In the augmented G_Q for our example (Figure 3.8b) the additional edges are denoted with dashed lines.

Algorithm: AUGMENT

Input : A query Q and its initial join query graph G_Q
Output : An augmented query graph G_Q for Q

- 1 Identify all triples in Q that include at least one spatial variable in as subject or object. Each such triple corresponds to a node of G_Q ;
- 2 **for** each pair $?s_i, ?s_j$ of joined spatial variables **do**
- 3 **for** each pair of nodes $(n_i, n_j) \in G_Q$, such that n_i includes $?s_i$ and n_j includes $?s_j$ **do**
- 4 **if** there is no edge in G_Q between n_i and n_j **then**
- 5 Add a new edge denoting the filtering phase of the spatial join of $?s_i$ and $?s_j$;
- 6 **else**
- 7 Add the filtering phase of the spatial join predicate of $?s_i$ and $?s_j$ in the predicate list of edge between n_i and n_j ;
- 8 For each spatial variable $?s$ appearing in a WITHIN predicate, add $\text{WITHIN}(?s, \text{GEOMETRY})$ to filtering conditions of Q ;
- 9 For each pair of spatial variables $?s_i, ?s_j$ ($i \neq j$) joined in Q , add $\text{DISTANCE} (?s_i, ?s_j) \text{ Op } \epsilon$ to filtering conditions of Q ;
- 10 **return** G_Q ;

If a query Q also includes WITHIN predicates, in the end of the augmentation procedure and for each spatial variable $?s$ whose geometry $?g$ participates in a WITHIN predicate, we add a condition of the form $\text{WITHIN} (?s, \text{GEOMETRY})$ to the set of filters of Q , so that this filter can be applied in any (intermediate) relation that

Table 3.1: Spatial join scenarios in optimal plan build.

Case	Algorithm(s) to Consider
L and R sorted on entity IDs	SMJ (Section 3.5.3)
L and R results of ($?s_i$ hasGeometry $?g_i$)	SMJ or R-tree Join [BKS93]
L sorted on entity IDs R result of a pattern ($?s_2$ hasGeometry $?g_2$)	SMJ (Section 3.5.3), SISJ [MP03], or Index Nested Loops
L unsorted R result of a pattern ($?s_2$ hasGeometry $?g_2$)	SHJ-ID (Section 3.5.4), SISJ [MP03] or Index Nested Loops
L and R unsorted	SHJ-ID, SHJ [LR96] or Nested Loops

contains the spatial variable $?s$. Note that this condition differs from the existing spatial condition $\text{WITHIN}(\mathbf{?g}, \text{GEOMETRY})$ in that it includes the spatial variable $?s$ and not the geometry variable $?g$. Similarly, for each pair (s_i, s_j) of joined spatial variables, we add the corresponding spatial join condition to Q 's existing filters, so that this filter can be applied on every (intermediate) relation that includes both the spatial variables s_i and s_j . Overall, the final augmented G_Q may include more edges than the initial G_Q , additional predicates in the edges, and general spatial filters for variables or pairs of variables that can be applied on intermediate results of subplans.

3.6.2 Spatial Join Operators

Our plan generator can place a spatial join operation at every level of the operator tree. Table 3.1 summarizes all possible cases of the L and R inputs of a spatial join (if L and R are swapped there is no difference because the join is symmetric). The right column includes the join algorithms, which the plan generator of the optimizer considers in each case.

Depending on whether the inputs of the join are indexed, sorted, or unsorted, there are different algorithms to be considered. If both join inputs come ordered by the IDs of the spatial entities to be joined, then SMJ (Section 3.5.3) is the algorithm of choice. In the special case where both inputs are the results of $?s_i$ hasGeometry $?g_i$ patterns applied on the entire set of triples, besides of applying SMJ on the SPO (or SOP) index, we can apply an R-tree self-join [BKS93] on the R-tree index. When just one of the inputs, e.g., R , is a result of a $?s_i$ hasGeometry $?g_i$ pattern, besides SMJ, we can also apply the SISJ algorithm [MP03]. In this case, we also consider Index Nested Loops join using the R-tree, by applying one spatial range query for each tuple of the other input, e.g., L . This should be cheap only when L is very small. Finally, when either L or R are unsorted, SMJ is not applicable and we can use SHJ-ID on the entity IDs (Section 3.5.4), or either SISJ or SHJ depending on whether one of the inputs is a direct result of a $?s_i$ hasGeometry $?g_i$ pattern or not. We also consider Index Nested Loops or Nested Loops, if one of the inputs is too small.

3.6.3 Spatial Query Optimization

We extend the query optimizer of RDF-3X to consider all possible spatial join cases and algorithms outlined in Section 3.6.2. In addition, the optimizer considers the case of performing a spatial selection Filter using the R-tree (see Section 3.3). The optimizer also considers any spatial selection and join filter conditions that are applied on-the-fly; i.e., in plans where the non-spatial query pattern components are evaluated first, our optimizer uses spatial query selectivity statistics to estimate the output size of these components after the spatial filter is applied on them. Consider for example, the plan of Figure 3.4. The estimated output of the $?s$ hosted $?o$ pattern is further refined to consider the spatial WITHIN filter that follows. In other words, the cardinality of the right input to the merge-join algorithm that follows is estimated using both RDF-3X statistics on the selectivity of $?s$ hosted $?o$ and spatial statistics for the selectivity of $\text{WITHIN}(?g, \text{"POLYGON (...)"})$.

3.6.4 Selectivity Estimation

For estimating the selectivity of spatial query components, we use grid-based statistics, similar to previous work on spatial query optimization (e.g., see [MP03]). Specifically, we take advantage of statistics that are obtained by the spatial encoding phase of the entity IDs. For each cell of the grid, defined by the Hilbert order, we keep track of the number of spatial entities that fall inside. The spatial join or selection is then applied at the level of the grid, based on uniformity assumptions about the spatial distributions inside the cells. In addition, we assume independence with respect to the other query components. For example, for estimating the input cardinality of the right merge-join input at the plan of Figure 3.4, we multiply the selectivity of the $?s$ hosted $?o$ pattern with that of the $\text{WITHIN} (?g, \text{"POLYGON (...)"})$ filter. In practice, this gives good estimates if the spatial distribution of the entities that instantiate $?s$ is independent to the spatial distribution of all entities.

3.6.5 Runtime Optimizations

RDF-3X uses a lightweight Sideways Information Passing (SIP) mechanism for skipping redundant values when scanning the indexes [NW09]. Consider a merge join, which binds the values of a variable $?s$ coming from two inputs. If the join result is fed to another (upper) merge-join operator that binds $?s$, then the upper operator can use the next value v of its other input to notify the lower operator that $?s$ values less than v need not be computed.

In the case of spatial joins where at least one side comes from a scan in the R-tree (e.g., consider the plan shown in Figure 3.2a), SIP is not applicable since there is

no global order for the geometries in the 2D space. On the other hand, the SMJ algorithm proposed in Section 3.5.3 can use SIP to notify the operators below its left input which is the minimum ID value for the next entity e_l to pair with any entity buffered in B_R . For the spatial hash join, we can also use SIP, by creating a bloom filter for one input, similar to the one RDF-3X constructs for the traditional hash join, and use it to prune tuples from its other input, while scanning the B⁺-tree index. A value is pruned if it is not included in the bloom filter.

3.7 Updates

The proposed encoding scheme requires significant changes in the update mechanism of RDF-3X [NW08, NW10a]. Inserting a new spatial entity is straight-forward and requires generating the appropriate ID based on the entity’s geometry and the occupancy of the grid. On the other hand, removing or updating the geometry of a spatial entity requires additional care as it might trigger the re-encoding of other spatial entities besides the one being updated. Such re-encodings tend to improve the latency of spatial queries, since entities are “moved” to lower levels of the grid, but incur an overhead during updates because they result in additional triples to be removed and re-inserted with new IDs.

Insert and delete commands in RDF-3X (cf. Section 3.2) are given in batches and are processed in two phases. In the first phase, the triples to insert or delete are resolved via lookups in the dictionary, i.e. they are translated into triples of integer IDs used internally by the system. Updates are not applied directly to the database; instead, the affected triples are first resolved in memory for all updates in the batch using *differential indexes*, which are then synchronized with the base indexes in the second phase. This is a common technique in bulk update processing that aims to minimize I/Os and increase the system throughput. RDF-3X maintains six differential indexes (SPO, SOP, OPS, OSP, PSO, POS), one for each full base index, which are synchronized with both the full and the aggregated base indexes. For example, the SPO differential index is synchronized with the base SPO index, the binary aggregated index SP, and the unary aggregated index S.

SRX integrates the original update mechanism of RDF-3X with the encoding scheme of Section 3.4. To do that, it changes only the first update phase, whereas the index synchronization can be used as is. To simplify the presentation, we distinguish two cases: a) only inserts of new triples, i.e., the subject entity s of the input triple $\langle s, p, o \rangle$ does not exist in the dictionary, b) inserts and deletes of triples whose subject entity already exists in the data. The update process takes as input a batch B of triples annotated with *insert* or *delete*, and updates two in-memory sets of triples tI (to insert) and tD (to delete), which are used to build the differential indexes.

Inserts of new entities. The insertion to the triples set tI is performed similarly to the original RDF-3X update process, but IDs are generated using the modified function `Generate-ID`. `Generate-ID` takes as input the entity’s URI and a boolean value, which indicates whether the new ID should be spatial (*true*) or not (*false*), i.e., whether the input triple introduces a geometry for the subject entity. `Generate-ID` is also responsible for updating the dictionary and for tracking the set of *new* IDs for the current batch B . The set *new* (initially empty for a batch) contains all IDs that do not exist in the database and is used in the second part of the update algorithm to avoid expensive lookups in the base indexes, as we explain later on. Since, the insertion of new triples only differs from the original RDF-3X process in the creation of the ID, we omit the pseudocode for the sake of brevity.

Updates on existing entities. The pseudocode for the updates on existing entities is given in Algorithm `Updates on existing entities`. This part handles triples with existing spatial and non-spatial subject entities, and is further split into three sub-parts: one for inserting triples that introduce a geometry for the non-spatial subject (*lines 5-16*), one for inserting triples that do not introduce a geometry (*lines 17-23*), and a last one for deleting triples (*lines 24-34*).

In the first sub-part, when a geometry is introduced for a non-spatial entity, the algorithm generates a new spatial ID s_{new} (*line 8*) and proceeds with updating the in-memory sets tI and tD accordingly. To do so, it first checks if the old subject ID (s_{id}) exists in the set of *new* IDs for the current batch; if yes, it simply updates the set *new* along with tI (*lines 15-16*), otherwise it retrieves all affected triples from the database and updates both tI and tD (*lines 11-14* and *16*). The update algorithm also ensures in this case that each entity is associated with *at most one* geometry but these additional checks are omitted here for the sake of brevity. The last two sub-parts of `Updates on existing entities` follow the original RDF-3X update logic and differ only in the use of *new* in *lines 20* and *28* to avoid expensive lookups in the base indexes; these lookups are only performed as last steps in *lines 22* and *30*.

Spatial re-encoding is the task of re-assigning spatial IDs that get released (after geometry deletions) to spatial entities encoded at higher levels of the grid due to overflow. It is an iterative bottom-up process, from lower to higher levels of the grid, which takes place in *line 34* of `Updates on existing entities`. The re-encoding function receives the spatial ID of an entity whose geometry is being deleted, replaces this ID with a non-spatial one (the next free even ID), and checks if there is a spatial entity from a higher level that can be re-encoded using the recently released spatial ID. If so, the re-encoding of the spatial entity releases another spatial ID, and the process cascades until no more re-encodings are possible.

Algorithm: UPDATES ON EXISTING ENTITIES

Input : Batch B of triples annotated with
 $op = \{insert, delete\}$

Output : Sets of triples tI (to insert) and tD (to delete)

```

1 let  $new$  be the set of new entity IDs for the current batch  $B$ ;
2 while  $t = \langle s, p, o, op \rangle = B.get\_next()$  do
3   if  $s \in \text{dictionary}$  then
4     let  $s_{id}$  be the ID of  $s$  as found in the dictionary;
5     if  $s_{id}$  is not spatial and  $op = insert$  and  $p =$ 
6       "hasGeometry" then
7         //A geometry is given for  $s$ 
8         let  $p_{id} = \text{FETCH\_OR\_GEN\_ID}(p, false)$ ;
9         let  $o_{id} = \text{FETCH\_OR\_GEN\_ID}(o, false)$ ;
10        let  $s_{new} = \text{GENERATE\_ID}(s, true)$ ;
11         $tI = tI \cup \{\langle s_{new}, p_{id}, o_{id} \rangle\}$ ;
12        if  $s_{id} \notin new$  then
13          let  $affected$  be the set of triples in the
14            database containing  $s_{id}$  as subject or object;
15          let  $replace = affected \setminus tD$ ;
16           $tI = tI \cup replace$ ;
17           $tD = tD \cup replace$ ;
18        else  $new = new \setminus \{s_{id}\}$ ;
19        change  $s_{id}$  to  $s_{new}$  in all triples of  $tI$ ;
20      else if  $op = insert$  and  $p \neq "hasGeometry"$  then
21        //Both for spatial and non spatial  $s_{id}$ 
22        let  $p_{id} = \text{FETCH\_OR\_GEN\_ID}(p, false)$ ;
23        let  $o_{id} = \text{FETCH\_OR\_GEN\_ID}(o, false)$ ;
24        if  $\{s_{id}, p_{id}, o_{id}\} \cap new \neq \emptyset$  then
25           $tI = tI \cup \{\langle s_{id}, p_{id}, o_{id} \rangle\}$ ;
26        else if  $\langle s_{id}, p_{id}, o_{id} \rangle \notin \text{database}$  then
27           $tI = tI \cup \{\langle s_{id}, p_{id}, o_{id} \rangle\}$ ;
28      else if  $op = delete$  and ( $s_{id}$  is spatial or  $p \neq$ 
29        "hasGeometry") then
30        //For any valid delete operation
31        if  $p \in \text{dictionary}$  and  $o \in \text{dictionary}$  then
32          let  $p_{id}$  be the ID of  $p$  as found in the
33            dictionary;
34          let  $o_{id}$  be the ID of  $o$  as found in the
35            dictionary;
36          if  $\{s_{id}, p_{id}, o_{id}\} \cap new \neq \emptyset$  then
37             $tI = tI \setminus \{\langle s_{id}, p_{id}, o_{id} \rangle\}$ ;
38          else if  $\langle s_{id}, p_{id}, o_{id} \rangle \in \text{database}$  then
39             $tD = tD \cup \{\langle s_{id}, p_{id}, o_{id} \rangle\}$ ;
40          else continue;
41          if  $s_{id}$  is spatial and  $p = "hasGeometry"$ 
42            then
43              re-encode spatial entities;
44
```

The overall process relies on two thresholds $h_1, h_2 \in [0, 1]$, which define a range $[h_1, h_2]$ on the ratio of assigned to total spatial IDs that a cell can accommodate (fill factor). In particular, a re-encoding process starts when the fill factor of a cell c drops below h_1 , and continues as long as (i) there are entities from higher levels to re-encode in c , and (ii) c 's fill factor remains below h_2 . By the time at least one of these two conditions does not hold, the algorithm continues with examining the parent cells of c (at the next level), and so forth, until it reaches the top level. In the end, all spatial entities have been re-encoded at the lowest possible level such that each cell's fill factor is smaller than h_2 . The two thresholds h_1 and h_2 are used to trade off the frequency of re-encodings with the overhead in processing updates, and we discuss the choice of their values in Section 3.8.

3.8 Experimental Evaluation

We compare SRX with the original RDF-3X system [NW08], the latest version of Strabon [KKK12]; version 3.3.2, and the latest versions of two commercial triple stores with spatial data support, Virtuoso 7.2.5-rc1.3217-pthreads [vir] and GraphDB Free 8.6 [owl] (the successor of OWLIM-SE that we used in [LMBT14]). We implemented SRX in C++ (g++ 8.2.0) and conducted all experiments on a machine with an i7-4930K CPU at 3.40 GHz, a 3.6Tb 7.2K rpm SATA-3 hard disk, and 64GB RAM running Linux Debian (4.18.0-1 amd64). For the Strabon database, we used PostgreSQL 11.2 and PostGIS 2.5 versions, while for the R-tree implementation, we used the open-source SAIL library [HHT05], which we also extended with support for incremental kNN computation (cf. Section 3.3).

3.8.1 Queries Setup

Datasets. We evaluate our system using two real datasets: LinkedGeoData⁸ (LGD) and YAGO2s⁹ (YAGO). LGD contains user-contributed content from OpenStreetMap project. YAGO is an RDF knowledge base, derived from Wikipedia, WordNet and Geonames. Table 3.2 shows statistics about the sizes of the datasets (including the dictionary and indexes) and the number of entities and geometries in them. The sizes of the input triple files are 2.4GB (LGD) and 18.1GB (YAGO). The R-trees (using 4KB nodes) occupy 160MB and 221MB, respectively. The size of the grid in both datasets is 1.5GB (89M cells in total for all levels). Note that, despite the aggressive indexing, in both cases, we end up with a database size having around double the

⁸<https://tinyurl.com/yc4lxqdy>

⁹<https://tinyurl.com/y7ukhge3>

Table 3.2: Characteristics of the real datasets.

Dataset	Triples	Entities	Points	Polygons	Linestrings	Multipoints
LGD (5.1 GB)	15.4M	10.6M	590K	264K	2.6M	0
YAGO (29.8 GB)	205.3M	108.5M	4M	0	0	780K

Table 3.3: Percentage (%) of geometries per grid level.

Level	0 (bottom)	1	2	3	4	5	6	≥ 7
LGD	28.5	21.6	16.9	12.7	8.7	5.4	3.0	3.2
YAGO	50.3	19.2	8.1	4.5	3.0	2.4	1.9	10.6

size of the input files. Regarding the spatial distribution of the entities they include, both datasets are highly skewed, as shown in the Appendix 3.10 of this chapter.

Encoding. We used a grid of $8,192 \times 8,192$ cells at the bottom level, hence, the maximum number of bits used in an entity’s ID to encode its cell-ID is 26. This means that we can have up to 14 levels of spatial approximation. This is the maximum granularity we can achieve when the IDs of the entities are 32-bit integers. Using 64-bit IDs for better spatial approximation is possible, but significantly increases the size of the triple indexes, thus, one should do this only when the total number of entities is greater than 2^{32} . Besides, the grid must be relatively small so that it resides in memory (for selectivity estimation purposes). In our case, the grid size is less than 2GB for both datasets. As shown in Table 3.3, all levels of the grid are used in the encoding, due to data skew and several geometry extents (polygons, linestrings, multipoints); in YAGO many multipoints are not cleaned and have very large MBRs.

Queries. All queries used in our experiments have two parts: (i) an RDF part that can be evaluated by a traditional SPARQL engine and (ii) a spatial part, i.e., a FILTER condition that includes a WITHIN predicate (for spatial range queries), a DISTANCE predicate (for spatial distance joins) or a kNN predicate (for spatial nearest neighbors). The range queries have similar structure to those depicted in Figure 3.1b; we divide them into four classes based on the selectivities of the two parts. Queries belonging to class SL have their RDF part more selective compared to their spatial part and the opposite holds for queries in class LS (S stands for small result, L for large). For queries in classes SS and LL, both parts roughly have the same selectivity. The characteristics of the spatial join queries (denoted by J) and the spatial kNN queries will be discussed in Section 3.8.2. All queries can be found in the Appendix 3.10 of this chapter.

Comparison Measures. We evaluated each query 5 times (both with cold and warm cache) and report their average runtimes that include the query optimization cost (i.e., the time spent by the optimizer to apply the techniques of Section 3.6) and the time spent in the ID-to-string dictionary lookups for the output variables.

System Parameters. RDF-3X does not have its own data cache for the query results; instead, it relies entirely on the OS caching mechanism. The same architectural principle is also adopted in our implementation.¹⁰ When a query is executed for a second time, its optimization and evaluation is performed from scratch, since there are no logs or cached results as in a typical database system. To illustrate the effect of OS caching in the overall response time of the system, we report query evaluation times on warm and cold caches separately.

3.8.2 Queries Comparison

Results on Range Queries. Table 3.4 shows response times for range queries on the LGD dataset. The first three columns of the table show the number of results of the RDF query component only, the spatial component only and the complete query (combined). We first focus on comparing our approach (Encoding) with the basic extension presented in Section 3.3 (Basic) and the original RDF-3X system (Baseline). Basic uses the R-tree to retrieve the entities falling in the given range only for queries where the spatial component is selective enough (LS, SS); in all other cases, it applies the same plan as Baseline, i.e. it evaluates the RDF part first and then applies the WITHIN filter to the tuples that qualify it. On the other hand, Encoding always chooses to evaluate the RDF part of the queries first and uses the spatial range filtering technique (see Section 3.5.1) to reduce the number of entities that have to be spatially verified. Encoding is superior in all queries. In specific, we avoid fetching a large percentage of exact geometries (96% on average for all range queries in both datasets), which Baseline obtains by random accesses to the dictionary. The cost differences between Encoding and Baseline is small only for SL queries, where the spatial filtering has little effect. In all other cases, Encoding is significantly faster than Baseline and Basic, especially in LS and SS queries and in queries involving entities with non-point geometries, denoted by a star (*), where the difference is up to one order of magnitude. In the case of warm caches, all runtimes are very low, so the cost of Encoding may exceed the cost of Baseline sometimes (e.g., see SL queries) due to the overhead of applying the spatial filter on all accessed entities in the evaluation of the RDF part of the query.

The difference in the optimization times (in parentheses) between warm and cold caches in all alternatives is because the reported numbers include the time spent for parsing the query, resolving the IDs of the URIs/strings in it, and finally building the optimal plan. Hence, when a query is issued for the first time, it requires some dictio-

¹⁰We only included a small separate cache of 40Kb for the R-tree. Since the OS caches R-tree pages, we used a small cache size in order to reduce the effect of double caching by the SAIL library.

Chapter 3. SRX: Efficient Management of Spatial RDF Data

Table 3.4: Spatial range queries on LGD; runtime in *ms* - optimizer time in *()*.

Query	Number of results			Strabon		GraphDB		Virtuoso		SRX					
	RDF	Spatial	Combined	Cold	Warm	Cold	Warm	Cold	Warm	Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding	
										Cold	Warm	Cold	Warm	Cold	Warm
LGD.SL1*	524	2,537,757	411	64,581	12,791	6,960	136	74,645	83	3,880 (99)	35 (1)	3,333 (148)	54 (20)	2,334 (107)	78 (1)
LGD.SL2*	215,355	2,943,209	186,302	168,099	20,129	20,957	14,045	78,568	76	4,495 (97)	272 (1)	3,644 (178)	330 (58)	3,009 (100)	323 (1)
LGD.SL3*	13,090	2,537,757	9,814	160,591	18,081	8,496	822	72,624	36	11,554 (97)	82 (1)	9,891 (147)	104 (20)	3,983 (100)	177 (1)
LGD.LS1*	25,617	9,002	86	63,614	12,866	9,568	2,338	58,757	81	1,913 (95)	84 (1)	856 (124)	62 (15)	222 (109)	14 (1)
LGD.LS2*	191,976	908	3	63,065	12,785	24,702	17,159	46,066	167	2,257 (97)	176 (2)	429 (117)	21 (15)	183 (91)	12 (2)
LGD.LS3*	5,791	908	9	63,251	12,694	16,065	410	45,317	183	14,305 (107)	52 (1)	484 (127)	20 (14)	174 (89)	2 (1)
LGD.SS1*	8,621	9,002	69	63,403	13,271	8,118	829	58,658	51	1,696 (98)	65 (1)	932 (131)	63 (16)	211 (92)	14 (1)
LGD.SS2*	13,090	9,002	120	66,370	12,708	8,488	949	57,745	35	8,181 (97)	75 (1)	1,167 (137)	61 (15)	450 (100)	5 (1)
LGD.SS3*	5,791	9,002	7	66,275	12,750	16,479	410	57,251	22	14,308 (107)	53 (1)	934 (142)	53 (16)	350 (100)	3 (1)
LGD.LL1	191,976	350,405	13,416	89,097	13,900	24,868	17,178	53,714	120	2,694 (95)	183 (1)	2,562 (142)	200 (17)	815 (117)	55 (2)

Table 3.5: Spatial range queries on YAGO; runtime in *ms* - optimizer time in *()*.

Query	Number of results			GraphDB		Virtuoso		SRX					
	RDF	Spatial	Combined	Cold	Warm	Cold	Warm	Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding	
								Cold	Warm	Cold	Warm	Cold	Warm
YAGO.SL1*	11,547	364,992	891	23,689	2,823	66,017	9,053	10,610 (61)	61 (1)	10,342 (62)	61 (1)	6,119 (46)	48 (1)
YAGO.SL2*	6,030	31,260	69	32,091	1,816	59,802	1,318	8,984 (173)	70 (1)	8,654 (175)	70 (1)	3,783 (168)	43 (1)
YAGO.LS1*	2,226	138	0	4,817	232	19,276	19	2,769 (60)	61 (1)	2,661 (79)	64 (15)	772 (56)	37 (1)
YAGO.LS2*	285,613	41,945	4,471	182,118	9,657	77,759	1,274	17,590 (183)	696 (1)	16,471 (178)	697 (1)	8,692 (181)	161 (1)
YAGO.SS1*	6,030	8,440	3	30,249	1,853	51,155	459	8,375 (175)	70 (1)	12,019 (173)	275 (17)	3,201 (170)	41 (1)
YAGO.SS2*	7,074	7,042	2	11,131	547	36,008	325	4,641 (61)	62 (1)	11,286 (85)	245 (15)	2,319 (56)	46 (1)
YAGO.LL1*	285,613	184,743	10,454	188,015	13,465	66,381	5,040	19,882 (184)	701 (1)	18,688 (182)	699 (1)	10,978 (179)	173 (1)
YAGO.LL2*	152,693	107,625	88	82,420	10,880	55,230	3,123	6,262 (61)	2,971 (1)	5,999 (62)	2,993 (1)	5,302 (56)	1,921 (1)

nary lookups for resolving the IDs of the entities. With warm caches, the respective dictionary pages are already cached by the OS, thus, query optimization is always cheaper. Note that, in most cases, the time spent for query optimization by Encoding is similar to that of Baseline, meaning that the overhead of augmenting the query graph and using spatial statistics is negligible compared to the query optimization overhead of the original RDF-3X system. With warm caches, the overhead in query optimization by Encoding and Basic compared to Baseline (due to the use of spatial statistics) is more profound.

Similar results are observed for range queries on the YAGO dataset (see Table 3.5). All queries in this case involve entities that may have multipoint geometries (therefore they are marked by a star). Encoding always chooses to evaluate the RDF part of the queries first, as in LGD. Basic chooses the same plan as Baseline in all cases, except for LS1 and SS queries, where it opts to evaluate the spatial selection using the R-tree; on the other hand, the spatial part of LS2 is not considered very selective by the optimizer, preventing the use of R-tree for that query. Note that for YAGO the cost of Basic is high enough (even higher than Baseline for SS queries). After analysis, we found that this is due to the bad performance of the R-tree on this dataset; the range queries access roughly half of the R-tree nodes. The reason is that many multipoints in YAGO are dirty and have huge MBRs that cover most of the data space. Thus, the non-leaf R-tree entries have extremely large MBRs, causing a random query to access a large percentage of tree nodes.

Overall, with cold caches, the median speedup of Encoding over Baseline (resp. Basic) across all queries is $8.3\times$ (resp. $2.6\times$) for LGD, and $2\times$ (resp. $2\times$) for YAGO. When warm caches are used, the median speedups are $5.3\times$ (resp. $4\times$) for LGD, and $1.6\times$ (resp. $2.8\times$) for YAGO.

3.8. Experimental Evaluation

Table 3.6: Spatial distance join queries on LGD; runtime in *ms* - optimizer time in *()*.

Query	Spatial join threshold ϵ	Number of results		Strabon		GraphDB		SRX					
		RDF	Final	Cold	Warm	Cold	Warm	Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding	
				> 5 min	> 5 min	> 5 min	> 5 min	Cold	Warm	Cold	Warm	Cold	Warm
LGD.J1	0.003	12,145,200	6,831	> 5 min	> 5 min	111,208 (123)	109,295 (1)	3,936 (280)	188 (128)	2,854 (246)	566 (130)		
LGD.J2	0.01	274,576	538	> 5 min	27,194	21,796 (133)	18,476 (2)	4,497 (326)	256 (188)	2,229 (314)	273 (196)		
LGD.J3	0.02	13,423,300	8,742	> 5 min	> 5 min	101,756		5,412 (417)	433 (267)	3,952 (401)	712 (275)		
LGD.J4*	0.05	171,348,000	795,322	> 5 min	> 5 min	> 5 min	> 5 min	106,856 (613)	97,426 (474)	30,841 (589)	21,593 (475)		
LGD.J5*	0.01	15,564,000	2,782	> 5 min	> 5 min	59,468 (142)	49,528 (2)	13,530 (334)	355 (186)	10,976 (310)	1,161 (189)		
LGD.J6.1*	0.0005	20,181,600	7	> 5 min	> 5 min	> 5 min	> 5 min	133,685 (141)	119,677 (2)	15,291 (243)	199 (88)	12,943 (208)	204 (91)
LGD.J6.2*	0.001	20,181,600	22	> 5 min	> 5 min	> 5 min	> 5 min	133,693 (139)	120,050 (2)	15,372 (240)	195 (89)	12,970 (208)	198 (91)
LGD.J6.3*	0.01	20,181,600	743	> 5 min	> 5 min	> 5 min	> 5 min	134,433 (137)	119,668 (2)	17,036 (341)	308 (187)	8,533 (297)	1,497 (189)

Table 3.7: Spatial distance join queries on YAGO; runtime in *ms* - optimizer time in *()*.

Query	Spatial join threshold ϵ	Number of results		GraphDB		SRX							
		RDF	Final	Cold	Warm	Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding			
				> 5 min	> 5 min	8,992 (44)	103,626 (1)	14,738 (204)	445 (129)	11,795 (199)	684 (130)		
YAGO.J1*	0.1	6,245,000	2,635	-	-	118,992 (44)	103,626 (1)	14,738 (204)	445 (129)	11,795 (199)	684 (130)		
YAGO.J2*	0.1	523,815,000	6,799,189	> 5 min	> 5 min	8,994 (57)	137,147 (205)	112,954 (129)	136,114 (275)	115,226 (204)			
YAGO.J3*	0.1	16,528	832	142,545	9,732	8,990 (57)	162 (1)	10,547 (220)	264 (129)	9,425 (203)	279 (131)		
YAGO.J4*	0.1	3,165	451	69,923	1,477	7,990 (52)	124 (1)	8,796 (209)	219 (128)	8,139 (204)	228 (130)		
YAGO.J5*	0.1	565	113	-	-	2,836 (48)	95 (1)	3,547 (199)	164 (129)	3,406 (194)	168 (130)		
YAGO.J6*	0.1	19,814,600	664,613	> 5 min	> 5 min	> 5 min	292,119 (2)	45,004 (375)	21,696 (130)	43,679 (356)	22,945 (205)		
YAGO.J7*	0.1	544,771,000	4,204,184	> 5 min	> 5 min	> 5 min	> 5 min	40,454 (195)	21,169 (129)	16,721 (267)	1,831 (201)		
YAGO.J8.1*	0.001	3,519,380	85,188	-	-	86,075 (48)	77,085 (1)	8,889 (185)	150 (115)	8,381 (189)	150 (115)		
YAGO.J8.2*	0.01	3,519,380	86,222	-	-	87,701 (48)	77,093 (1)	9,117 (190)	201 (115)	8,535 (186)	202 (115)		
YAGO.J8.3*	0.1	3,519,380	131,828	-	-	86,068 (48)	77,152 (1)	9,471 (205)	383 (128)	8,066 (206)	461 (130)		

Results on Spatial Joins. Table 3.6 and Table 3.7 show the costs of spatial distance join queries on LGD and YAGO, respectively. The threshold 0.1 shown in the tables corresponds to a distance around 10km. In LGD, all queries have thresholds greater than the diagonal of a cell in our encoding except queries LGD.J6.1 and LGD.J6.2. In YAGO, threshold 0.1 is greater than the cell diagonal, but 0.01 is not. After performing experiments with various types of queries, we found that the SMJ and SHJ-ID algorithms should only be used when the spatial distance threshold is greater than the diagonal of the grid cell at the bottom level. Otherwise, they do not produce any verified results and, hence, they have similar or slightly worse performance compared to directly applying SHJ (as Basic would). We have added this simple rule of thumb in the optimizer of our system, hence, in all spatial join queries that have a distance threshold less than the cell diagonal, Encoding applies the same plans as Basic using the SHJ operator. In LGD, these queries are J6.1 and J6.2, while in YAGO, the respective queries are J8.1 and J8.2. For this reason, we focus mostly on queries where the distance threshold is greater than the cell diagonal.

All spatial join queries on the LGD dataset (Table 3.6) have a similar pattern: they include two disjoint RDF star-shaped parts with a spatial distance predicate between the geometries of their center nodes. This is the only type of queries we could define here since the LGD dataset includes a rather poor RDF part; besides the POI type, there are very few properties such as “label” and “name” which link the POIs with text attributes. For this type of queries, Baseline can only execute a bushy plan where the two stars are evaluated separately and then joined in a nested-loop fashion, applying the spatial distance filter. On the other hand, Basic may choose to apply an R-tree join first for retrieving the candidate pairs within distance ϵ or to first evaluate the RDF part of the query and follow-up with a spatial

hash join (SHJ) in the end (e.g., see the plans of Figure 3.2c and Figure 3.2d). In all queries we tested, Basic chose the SHJ algorithm and this is quite reasonable; in large datasets, the optimizer would prefer not to perform an expensive spatial self-join over the whole set of points. Encoding can choose between one of the previously mentioned methods and also try the algorithms of Section 3.5.3 and Section 3.5.4 on the augmented query graph. Since we have star-shaped queries and the IDs of the center nodes are coming sorted, SMJ was favored in all queries we present. Although Encoding is much faster than Baseline, we observe that for the case of warm caches, the former does not bring much benefit over Basic for most join queries on LGD. The main reason is that Encoding does not save any geometry lookups due to the particular data distribution; every entity from either of the two spatial join inputs participates in at least one non verified spatial join pair and therefore it cannot be pruned without fetching its geometry. In addition, Basic benefits from the fact that it buffers the complete join inputs before hashing them into the buckets of SHJ, thus, the geometry of each entity is processed only once. On the other hand, SMJ (used by Encoding) produces and verifies the join pair candidates on-the-fly, resulting in the processing of a given geometry multiple times. However, if the size of inputs is very large, Basic can become significantly slower than Encoding (see LGD.J4) because SHJ requires the allocation of a large hash table to accommodate a huge number of buffered geometries. Recall that SHJ, used by Basic, is a blocking operator which requires both inputs to be read as a whole before processing the join and, thus, can become a bottleneck if the inputs are too large.

The results for queries with spatial join components in YAGO are shown in Table 3.7. Depending on the type of the query and the selectivities of the two parts, our encoding-based approach uses either SMJ or SHJ-ID. Specifically, SMJ is used in queries J1 and J8.3, whereas SHJ-ID is used in J2, J6, and J7. In queries J8.1 and J8.2, Encoding follows the same plans as Basic. In the remaining queries (J3, J4 and J5), Basic and our encoding-based approach produced the same plans as Baseline; these queries include a single connected RDF graph pattern with a rather selective RDF part. As a result, the performance of Basic is similar to that of Encoding for queries J3, J4, J5, J8.1, and J8.2. For the remaining queries, Encoding is slightly faster than Basic with cold caches (the two techniques are comparable for warm caches), because Encoding selects a different plan based on the augmented query graph. A notable exception is YAGO.J7, where Encoding performs much better than Basic, because the spatial join inputs have a different spatial distribution and Encoding can prune many tuples using SHJ-ID.

Note that, for large inputs, SHJ might be slower than SMJ. This is reflected in the performance difference between queries LGD.J6.1, LGD.J6.2 (resp. YAGO.J8.1, YAGO.J8.2), where SRX applies the same plan as Basic using SHJ, and queries

LGD.J6.3 (resp. YAGO.J8.3), where SMJ is used.

Overall, with cold caches, the median speedup of Encoding over Baseline across all queries we used is $10.3\times$ and $8.4\times$ for LGD and YAGO. When warm caches are used, the respective median speedup is $136.5\times$ for LGD and $82.1\times$ for YAGO. Regarding spatial distance joins, Encoding and Basic have similar median performance for YAGO. For LGD with cold caches, Encoding has a median $1.3\times$ speedup over Basic, whereas with warm caches Encoding shows a median $0.7\times$ slowdown over Basic.

Results on Spatial kNN Queries. kNN queries in Tables 3.8-3.17 have been made from (and have the same names with) the range queries used in Tables 3.4-3.5. Each kNN query has the same RDF part with the respective range query along with a Filter $\text{kNN}(\text{?}g, \text{"POINT (...)"})$ clause, where “POINT (...)” is the middle point of “RECTANGLE (...)” in the Filter $\text{WITHIN}(\text{?}g, \text{"RECTANGLE (...)"})$ clause of the range query. Tables 3.8-3.17 provide results for $k = 5, 10, 20, 50$, and 100 .

In both datasets, Basic follows the same plan as Baseline with only exceptions LGD.SL1 and YAGO.LS1, where Basic uses the R-tree. In these two queries, Basic employs the incremental kNN operator (cf. Section 3.3) followed by hash joins, which are favored by the optimizer because the RDF part of the query is selective enough and, as a result, the cost of the building phase of the respective hash tables is low. Such a plan achieves much better performance over Baseline in some cases (e.g. for LGD.SL1 and $k = 5$ with cold caches), however, our experiments demonstrate that Baseline is usually a better option than Basic, especially for $k > 20$. We attribute this behavior to the following reasons: (i) Baseline evaluates the RDF part of the query by leveraging efficient sequential scans over the compressed B^+ -trees in combination with very fast merge joins, and (ii) LGD.SL1 and YAGO.LS1 have very selective RDF parts (the most selective for each dataset), therefore, the cost of the final dictionary lookups to fetch the geometries of the entities, as performed by Baseline, is low.

When cold caches are used, Unsorted and Sorted are significantly faster (due to filtering and fewer geometry lookups) than Baseline for all kNN queries except YAGO.LL2; in this case, Baseline is superior to both Unsorted and Sorted but for different reasons. First, Unsorted and Baseline use exactly the same plan to evaluate the RDF part of the query, and their only difference lies in the final verification phase where they must retrieve the actual geometries from the dictionary. We found that the result set of the RDF part of YAGO.LL2 contains many tuples carrying the same geometry, which amplifies the effects of caching and significantly reduces the cost of dictionary lookups. So, Baseline outperforms Unsorted due to the CPU overhead of the latter to maintain its priority queues. Yet, when Sorted is an option, the optimizer chooses a different plan for the RDF part, which leverages merge joins and outputs tuples in ascending order of their spatial IDs but shows poor performance.

Chapter 3. SRX: Efficient Management of Spatial RDF Data

Table 3.8: Spatial kNN queries on LGD for $k = 5$; runtime in ms - optimizer time in (\cdot) .

Query	Number of results (RDF)	Strabon		GraphDB		Virtuoso		SRX							
		Cold		Warm		Cold		Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding (Unsorted)		Encoding (Sorted)	
		Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm
LGD.SL1	524	64,581	12,968	7,336	78	108,284	10	1,173 (86)	33 (1)	548 (121)	47 (12)	466 (69)	57 (1)	380 (89)	18 (1)
LGD.SL2	215,355	95,637	44,738	23,353	16,527	91,761	1,001	1,345 (77)	208 (1)	1,342 (76)	208 (1)	449 (69)	100 (1)	177 (79)	8 (1)
LGD.SL3*	13,090	69,479	15,453	9,070	1,289	N/A	N/A	7,122 (85)	77 (1)	7,103 (83)	77 (1)	701 (69)	65 (1)	564 (83)	31 (1)
LGD.LS1	25,617	68,845	16,722	10,099	2,682	98,572	85	1,296 (77)	87 (1)	1,283 (76)	87 (1)	525 (70)	74 (1)	426 (84)	60 (1)
LGD.LS2	191,976	93,550	41,670	26,857	20,171	95,581	325	1,542 (77)	195 (2)	1,541 (77)	191 (1)	530 (70)	102 (1)	185 (73)	25 (1)
LGD.LS3*	5,791	75,161	14,249	16,138	512	N/A	N/A	12,483 (83)	53 (1)	12,471 (79)	53 (1)	464 (69)	48 (1)	188 (73)	12 (1)
LGD.SS1	8,621	66,576	14,342	8,461	941	99,445	33	1,233 (88)	66 (1)	1,266 (87)	66 (1)	502 (70)	63 (2)	487 (90)	50 (1)
LGD.SS2*	13,090	69,496	15,540	9,015	1,248	N/A	N/A	7,174 (87)	77 (1)	7,122 (87)	77 (1)	599 (69)	65 (1)	511 (73)	52 (1)
LGD.SS3*	5,791	75,341	14,061	16,288	519	N/A	N/A	12,375 (87)	53 (1)	12,509 (87)	53 (1)	525 (68)	48 (1)	447 (82)	40 (1)
LGD.LL1	191,976	94,509	42,433	26,931	20,142	97,280	321	1,570 (77)	192 (1)	1,608 (76)	192 (1)	546 (68)	101 (1)	235 (72)	18 (1)

Table 3.9: Spatial kNN queries on LGD for $k = 10$; runtime in ms - optimizer time in (\cdot) .

Query	Number of results (RDF)	Strabon		GraphDB		Virtuoso		SRX							
		Cold		Warm		Cold		Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding (Unsorted)		Encoding (Sorted)	
		Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm
LGD.SL1	524	64,720	12,941	7,340	80	108,416	12	1,174 (88)	33 (1)	876 (121)	72 (12)	504 (70)	57 (2)	504 (90)	27 (1)
LGD.SL2	215,355	95,608	45,125	23,354	16,536	92,213	1,001	1,327 (77)	208 (1)	1,329 (76)	212 (1)	452 (70)	100 (1)	176 (77)	8 (1)
LGD.SL3*	13,090	69,437	15,487	9,149	1,292	N/A	N/A	7,140 (87)	77 (1)	7,115 (74)	77 (1)	749 (69)	65 (1)	612 (92)	31 (1)
LGD.LS1	25,617	68,666	16,665	10,118	2,711	98,710	85	1,306 (77)	87 (1)	1,273 (76)	87 (1)	509 (70)	74 (1)	470 (84)	60 (1)
LGD.LS2	191,976	93,902	42,130	27,158	20,179	95,801	325	1,577 (77)	192 (1)	1,588 (77)	192 (1)	525 (70)	102 (1)	184 (73)	25 (1)
LGD.LS3*	5,791	75,219	14,073	16,200	519	N/A	N/A	12,483 (87)	53 (1)	12,461 (87)	53 (1)	485 (69)	49 (1)	223 (70)	23 (1)
LGD.SS1	8,621	66,535	14,388	8,473	942	99,528	33	1,217 (87)	66 (1)	1,240 (87)	66 (1)	492 (69)	62 (1)	472 (94)	50 (1)
LGD.SS2*	13,090	69,405	15,523	9,082	1,288	N/A	N/A	7,178 (87)	77 (1)	7,055 (87)	77 (1)	618 (73)	65 (1)	543 (92)	52 (1)
LGD.SS3*	5,791	75,315	13,970	16,302	544	N/A	N/A	12,421 (87)	54 (1)	12,475 (87)	53 (1)	667 (69)	49 (1)	575 (95)	56 (1)
LGD.LL1	191,976	94,205	42,670	27,043	20,148	97,809	322	1,620 (78)	192 (1)	1,634 (77)	192 (1)	544 (73)	101 (1)	252 (73)	19 (1)

Table 3.10: Spatial kNN queries on LGD for $k = 20$; runtime in ms - optimizer time in (\cdot) .

Query	Number of results (RDF)	Strabon		GraphDB		Virtuoso		SRX							
		Cold		Warm		Cold		Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding (Unsorted)		Encoding (Sorted)	
		Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm
LGD.SL1	524	64,510	12,954	7,376	87	108,909	12	1,133 (88)	33 (1)	1,269 (121)	116 (12)	545 (70)	56 (1)	532 (92)	28 (1)
LGD.SL2	215,355	95,644	45,126	23,356	16,540	93,970	1,004	1,363 (79)	209 (1)	1,330 (77)	209 (1)	453 (69)	100 (1)	178 (75)	8 (1)
LGD.SL3*	13,090	69,274	15,584	9,150	1,306	N/A	N/A	7,164 (87)	77 (1)	7,168 (82)	77 (1)	750 (69)	65 (1)	608 (92)	30 (1)
LGD.LS1	25,617	68,805	16,686	10,214	2,725	99,406	85	1,302 (77)	88 (1)	1,281 (77)	87 (1)	519 (70)	74 (1)	472 (92)	60 (1)
LGD.LS2	191,976	93,585	41,657	27,164	20,180	95,925	327	1,598 (77)	192 (1)	1,615 (77)	192 (1)	528 (70)	102 (1)	182 (72)	26 (1)
LGD.LS3*	5,791	75,127	14,025	16,284	546	N/A	N/A	12,437 (87)	54 (1)	12,434 (86)	54 (1)	542 (69)	48 (1)	312 (78)	34 (1)
LGD.SS1	8,621	66,618	14,380	8,533	950	100,382	34	1,223 (88)	66 (1)	1,204 (86)	65 (1)	515 (70)	62 (1)	515 (94)	50 (1)
LGD.SS2*	13,090	69,536	15,693	9,094	1,300	N/A	N/A	7,170 (87)	77 (1)	7,149 (86)	77 (1)	608 (69)	65 (1)	551 (88)	52 (1)
LGD.SS3*	5,791	75,403	14,025	16,306	545	N/A	N/A	12,457 (87)	54 (1)	12,467 (87)	54 (1)	667 (69)	48 (1)	560 (83)	55 (1)
LGD.LL1	191,976	94,252	42,573	27,223	20,150	98,285	322	1,590 (77)	192 (1)	1,646 (77)	192 (1)	588 (69)	101 (1)	266 (72)	19 (1)

Table 3.11: Spatial kNN queries on LGD for $k = 50$; runtime in ms - optimizer time in (\cdot) .

Query	Number of results (RDF)	Strabon		GraphDB		Virtuoso		SRX							
		Cold		Warm		Cold		Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding (Unsorted)		Encoding (Sorted)	
		Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm
LGD.SL1	524	64,803	13,097	7,399	110	110,023	13	1,139 (85)	33 (1)	3,686 (129)	265 (12)	668 (70)	56 (1)	653 (92)	32 (1)
LGD.SL2	215,355	95,515	44,992	23,361	16,676	94,825	1,014	1,335 (77)	208 (1)	1,324 (76)	210 (1)	459 (69)	100 (1)	177 (76)	9 (1)
LGD.SL3*	13,090	69,496	15,557	9,455	1,470	N/A	N/A	7,096 (77)	77 (1)	7,091 (87)	77 (1)	884 (69)	65 (1)	851 (94)	33 (1)
LGD.LS1	25,617	68,860	16,667	10,255	2,729	99,767	86	1,315 (77)	87 (1)	1,304 (77)	88 (2)	564 (71)	75 (1)	501 (84)	61 (1)
LGD.LS2	191,976	93,740	41,659	27,320	20,257	96,761	328	1,611 (77)	193 (2)	1,639 (78)	191 (1)	528 (69)	102 (1)	223 (82)	27 (1)
LGD.LS3*	5,791	75,280	13,946	16,333	592	N/A	N/A	12,423 (87)	54 (1)	12,463 (87)	54 (1)	617 (69)	49 (1)	388 (82)	41 (1)
LGD.SS1	8,621	66,514	14,341	8,556	973	100,768	34	1,222 (87)	66 (1)	1,159 (87)	65 (1)	534 (70)	62 (1)	523 (91)	50 (1)
LGD.SS2*	13,090	69,224	15,513	9,269	1,367	N/A	N/A	7,163 (87)	77 (1)	7,170 (86)	77 (1)	688 (69)	65 (1)	629 (85)	52 (1)
LGD.SS3*	5,791	75,301	14,037	16,517	574	N/A	N/A	12,479 (87)	53 (1)	12,351 (87)	54 (1)	791 (69)	48 (1)	752 (82)	60 (1)
LGD.LL1	191,976	93,971	42,339	27,424	20,281	99,163	324	1,633 (77)	192 (1)	1,573 (77)	192 (1)	638 (70)	101 (1)	303 (80)	22 (2)

Table 3.12: Spatial kNN queries on LGD for $k = 100$; runtime in ms - optimizer time in $(\cdot)</$

3.8. Experimental Evaluation

Table 3.13: Spatial kNN queries on YAGO for $k = 5$; runtime in ms - optimizer time in (\cdot) .

Query	Number of results (RDF)	GraphDB		SRX							
				Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding (Unsorted)		Encoding (Sorted)	
		Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm
YAGO.SL1*	11,547	86,349	3,054	6,108 (51)	59 (1)	6,170 (51)	58 (1)	2,411 (72)	57 (1)	2,111 (86)	43 (1)
YAGO.SL2*	6,030	102,461	1,977	7,553 (120)	69 (1)	7,537 (124)	69 (1)	3,068 (169)	66 (1)	2,864 (228)	43 (1)
YAGO.LS1*	2,226	-	-	2,432 (53)	58 (1)	3,943 (110)	98 (16)	1,349 (80)	60 (1)	1,158 (82)	43 (1)
YAGO.LS2*	285,613	>5 min	137,117	13,340 (129)	715 (1)	13,253 (129)	717 (1)	6,009 (193)	720 (1)	5,567 (211)	676 (1)
YAGO.SS1*	6,030	101,674	1,977	7,530 (111)	69 (1)	7,526 (115)	69 (1)	3,096 (192)	66 (1)	2,803 (210)	40 (1)
YAGO.SS2*	7,074	52,999	683	4,100 (51)	60 (1)	4,106 (50)	60 (1)	2,305 (82)	59 (1)	1,850 (70)	21 (1)
YAGO.LL1*	285,613	>5 min	138,160	13,248 (139)	715 (1)	13,220 (127)	718 (1)	6,071 (185)	719 (1)	5,654 (226)	689 (1)
YAGO.LL2*	152,693	169,217	15,148	4,709 (50)	2,992 (1)	4,721 (51)	2,987 (1)	5,750 (75)	3,016 (1)	13,118 (83)	2,273 (1)

Table 3.14: Spatial kNN queries on YAGO for $k = 10$; runtime in ms - optimizer time in (\cdot) .

Query	Number of results (RDF)	GraphDB		SRX							
				Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding (Unsorted)		Encoding (Sorted)	
		Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm
YAGO.SL1*	11,547	86,351	3,055	6,045 (50)	58 (1)	6,097 (50)	58 (1)	2,395 (66)	57 (1)	2,115 (86)	43 (1)
YAGO.SL2*	6,030	102,464	1,979	7,560 (115)	69 (1)	7,548 (121)	69 (1)	3,117 (190)	66 (1)	2,850 (228)	43 (1)
YAGO.LS1*	2,226	-	-	2,431 (51)	58 (1)	3,922 (91)	97 (16)	1,304 (77)	61 (1)	1,152 (82)	43 (1)
YAGO.LS2*	285,613	>5 min	137,120	13,284 (127)	716 (1)	13,208 (137)	716 (1)	5,983 (185)	721 (1)	5,547 (209)	674 (1)
YAGO.SS1*	6,030	101,675	1,979	7,544 (109)	69 (1)	7,574 (121)	69 (1)	3,112 (195)	67 (1)	2,804 (208)	40 (1)
YAGO.SS2*	7,074	53,002	684	4,117 (53)	59 (1)	4,111 (51)	60 (1)	2,327 (77)	59 (1)	1,844 (66)	22 (1)
YAGO.LL1*	285,613	>5 min	138,161	13,343 (127)	716 (1)	13,347 (131)	714 (1)	6,030 (179)	723 (1)	5,713 (227)	690 (1)
YAGO.LL2*	152,693	169,219	15,149	4,726 (51)	2,987 (1)	4,711 (50)	2,990 (1)	5,811 (82)	3,019 (1)	13,114 (86)	2,275 (1)

Table 3.15: Spatial kNN queries on YAGO for $k = 20$; runtime in ms - optimizer time in (\cdot) .

Query	Number of results (RDF)	GraphDB		SRX							
				Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding (Unsorted)		Encoding (Sorted)	
		Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm
YAGO.SL1*	11,547	86,354	3,057	6,073 (50)	59 (1)	6,117 (50)	59 (1)	2,415 (72)	57 (1)	2,096 (68)	43 (1)
YAGO.SL2*	6,030	102,468	1,982	7,549 (114)	69 (1)	7,577 (122)	68 (1)	3,103 (195)	66 (1)	2,854 (228)	43 (1)
YAGO.LS1*	2,226	-	-	2,449 (51)	57 (1)	3,946 (105)	97 (16)	1,335 (80)	61 (1)	1,141 (78)	43 (1)
YAGO.LS2*	285,613	>5 min	137,122	13,345 (127)	718 (1)	13,308 (125)	716 (1)	6,017 (185)	722 (1)	5,589 (211)	676 (1)
YAGO.SS1*	6,030	101,678	1,980	7,543 (114)	69 (1)	7,542 (113)	69 (1)	3,092 (191)	67 (1)	2,829 (214)	40 (1)
YAGO.SS2*	7,074	53,006	686	4,098 (50)	60 (1)	4,098 (50)	59 (1)	2,293 (82)	59 (1)	1,838 (70)	23 (1)
YAGO.LL1*	285,613	>5 min	138,165	13,261 (123)	718 (1)	13,313 (129)	717 (1)	6,055 (191)	721 (1)	5,657 (226)	688 (1)
YAGO.LL2*	152,693	170,222	15,251	4,720 (55)	2,990 (1)	4,720 (51)	2,983 (1)	5,820 (77)	3,017 (1)	13,084 (86)	2,280 (1)

Table 3.16: Spatial kNN queries on YAGO for $k = 50$; runtime in ms - optimizer time in (\cdot) .

Query	Number of results (RDF)	GraphDB		SRX							
				Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding (Unsorted)		Encoding (Sorted)	
		Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm
YAGO.SL1*	11,547	88,397	3,189	6,088 (51)	59 (1)	6,089 (50)	59 (1)	2,417 (74)	57 (1)	2,096 (68)	43 (1)
YAGO.SL2*	6,030	104,512	2,104	7,549 (114)	69 (1)	7,540 (113)	69 (1)	3,114 (194)	66 (1)	2,879 (230)	44 (1)
YAGO.LS1*	2,226	-	-	2,475 (51)	58 (1)	3,884 (89)	97 (16)	1,371 (82)	61 (1)	1,155 (82)	43 (1)
YAGO.LS2*	285,613	>5 min	137,227	13,337 (125)	716 (1)	13,245 (129)	715 (1)	5,993 (187)	725 (1)	5,589 (208)	677 (1)
YAGO.SS1*	6,030	104,722	1,996	7,543 (116)	69 (1)	7,536 (115)	69 (1)	3,084 (181)	67 (1)	2,831 (210)	40 (1)
YAGO.SS2*	7,074	54,011	753	4,096 (50)	60 (1)	4,096 (50)	60 (1)	2,311 (77)	59 (1)	1,876 (66)	23 (1)
YAGO.LL1*	285,613	>5 min	139,201	13,307 (123)	716 (1)	13,351 (131)	716 (1)	6,029 (177)	725 (1)	5,667 (204)	689 (1)
YAGO.LL2*	152,693	171,984	15,305	4,701 (51)	2,989 (1)	4,728 (51)	2,990 (1)	5,804 (79)	3,012 (1)	13,122 (86)	2,275 (1)

Table 3.17: Spatial kNN queries on YAGO for $k = 100$; runtime in ms -optimizer time in (\cdot) .

Query	Number of results (RDF)	GraphDB		SRX							
				Baseline (RDF-3X)		Basic extension (Sec. 3.3)		Encoding (Unsorted)		Encoding (Sorted)	
		Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm
YAGO.SL1*	11,547	91,800	3,308	6,085 (50)	59 (1)	6,109 (58)	59 (1)	2,429 (74)	57 (1)	2,146 (70)	43 (1)
YAGO.SL2*	6,030	107,319	2,261	7,552 (115)	69 (1)	7,520 (113)	69 (1)	3,149 (195)	66 (1)	2,871 (228)	44 (1)
YAGO.LS1*	2,226	-	-	2,447 (51)	57 (1)	10,788 (98)	320 (16)	1,367 (78)	60 (1)	1,590 (70)	147 (1)
YAGO.LS2*	285,613	>5 min	137,529	13,344 (129)	715 (1)	13,258 (127)	717 (1)	6,009 (189)	725 (1)	5,557 (201)	680 (1)
YAGO.SS1*	6,030	107,657	2,211	7,545 (120)	69 (1)	7,541 (114)	69 (1)	3,135 (193)	66 (1)	2,850 (208)	40 (1)
YAGO.SS2*	7,074	55,145	878	4,118 (50)	59 (1)	4,096 (50)	60 (1)	2,363 (80)	59 (1)	1,879 (68)	24 (1)
YAGO.LL1*	285,613	>5 min	142,926	13,252 (129)	713 (1)	13,374 (137)	718 (1)	6,077 (185)	727 (1)	5,729 (227)	691 (1)
YAGO.LL2*	152,693	175,855	15,543	4,718 (53)	2,989 (1)	4,729 (50)	2,987 (1)	5,783 (80)	3,017 (1)	13,135 (86)	2,274 (1)

tends to perform fewer dictionary lookups than Baseline and, in queries with very selective RDF part, such as LGD.SL1, the benefits of Unsorted over Baseline are

negligible. In YAGO with warm caches, Unsorted performs similarly or slightly worse (e.g. for YAGO.LS2, YAGO.LL1, and YAGO.LL2) than Baseline for the same reason we mentioned before: many entities in the result of the RDF part of these queries have the same geometry, and the cost of the additional geometry lookups in Baseline is mitigated by caching. The only LGD queries where Baseline is slightly better than Sorted are LGD.SL1 for $k = 100$ and LGD.SS3 for all k values except $k = 5$. Note that, although LGD.LS3 has the same RDF part with LGD.SS3, Sorted is constantly better than Baseline for LGD.LS3 because its kNN predicate is different. Regarding YAGO, Sorted is better than Baseline in all queries apart from YAGO.LS1 for $k = 100$.

Finally, Sorted is superior to Unsorted in most cases. For LGD and warm caches, the only queries where Unsorted is better than Sorted are LGD.LS3, LGD.SS1 for $k = 100$, and LGD.SS3 for all k values except $k = 5$. Still, under cold caches, Sorted is better even in these cases. For YAGO, Sorted is better than Unsorted in all queries but YAGO.LS1 for $k = 100$, where Unsorted is superior under both warm and cold caches. Generally, Sorted is preferable over Unsorted and Baseline, as it tends to avoid a significant number of geometry retrievals. Yet, Unsorted is a good alternative since it can overcome the deficiencies of Sorted in the few cases where Baseline is better: LGD.SS3 for all k values except $k = 5$, and YAGO.LS1 for $k = 100$.

Overall, for LGD and with cold caches, Encoding Unsorted (resp. Sorted) has a median $2.9\times$ (resp. $7.9\times$) speedup over Baseline whereas, for YAGO, the median speedup over Baseline is $2.2\times$ for Unsorted and $2.3\times$ for Sorted. For LGD queries with warm caches, the median speedups of Unsorted and Sorted over Baseline are $1.1\times$ and $1.5\times$ respectively. Last, for YAGO with warm caches, Unsorted and Baseline perform similarly whereas Sorted has a median $1.3\times$ speedup over Baseline.

Comparison with Existing Systems. We compared SRX against three popular RDF stores with geospatial data support, namely Strabon, GraphDB and Virtuoso. For Strabon, we only present results with LGD, as it could not load YAGO even after three days (and even when using the bulk loader we obtained from the authors of [KKK12]; this issue is also reported in [PGA14]).

We allowed each system to allocate the whole available memory of the machine and performed the experiments with cold and warm caches just like for our system. Since these systems have their own data caches, experiments with cold caches were conducted by clearing the OS cache and restarting the system. The symbol ‘-’ in Tables 3.7, and 3.13-3.17 denotes empty or incorrect results. N/A (not applicable) is used in Tables 3.8-3.12, and 3.18 for Virtuoso because its internal function `bif:st_distance`, which is used in spatial join and kNN queries (cf. Appendix 3.10), works only for point geometries. In fact, this is why there are no Virtuoso results in Tables 3.6-3.7 and Tables 3.13-3.17. Although queries J1, J2, and J3 in Table 3.6

Table 3.18: Median speedups of SRX over Strabon, GraphDB, Virtuoso for LGD and YAGO across all queries of each type.

Query Type	Strabon (LGD)		GraphDB (LGD)		GraphDB (YAGO)		Virtuoso (LGD)		Virtuoso (YAGO)	
	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm	Cold	Warm
range	168.4	933.4	34.4	151.8	8.9	43.7	145.9	4.7	13.1	9.5
distance	31.2	339.8	31.2	397	8.5	13	N/A	N/A	N/A	N/A
unsorted kNN	130.9	264.6	22	20	33.3	30	192.9	2.1	N/A	N/A
sorted kNN	140.2	437.5	24.9	34.2	36.2	49.5	290	6.9	N/A	N/A

involve only points, they are also not supported by Virtuoso, presumably because it tries to apply the DISTANCE predicate first between all types of geometries. Note that both GraphDB and Virtuoso compute the great-circle distance, whereas Strabon, Encoding, Baseline, and Basic compute the Euclidean distance. This does not prevent us from a fair comparison among GraphDB and Virtuoso since both distance functions have similar CPU cost.

Table 3.18 summarizes the median speedups of SRX over Strabon, GraphDB, and Virtuoso across all queries we used on both datasets. SRX is $8.5\times$ faster (in the worst case) than the competitors in all cases except for the case of Virtuoso on LGD with warm caches, where SRX median speedups are lower. We cannot comment further on the performance of GraphDB and Virtuoso as they are not open-source systems. On the contrary, the performance of Strabon for range (Table 3.4), and kNN (Tables 3.8-3.12) queries is dominated by the time needed to fetch data from disk: 60s (resp. 12s) with cold (resp. warm) caches on average in both range and kNN queries. We also observe that for these types of queries, the query time tends to increase with the size of the result for the RDF part of the query. Finally, distance join query times (Table 3.6) in Strabon are dominated by the CPU rather than the I/O time due to the large number of candidate pairs that need to be examined.

3.8.3 Updates Setup

In the experiments for updates we used the same encoding and R-tree configurations as for queries (cf. Section 3.8.1). Thresholds h_1 and h_2 used to trigger re-encodings (cf. Section 3.7) were set to 0.5 and 0.7 respectively. We also experimented with other thresholds (e.g. 0.35 and 0.5) but we did not notice any significant performance difference.

Datasets. The update benchmark relies on Deltas that we are extracted by calculating the difference of two versions of LGD and YAGO: LGD 2013_04_29 to 2015_11_02¹¹, and YAGO 2.5.3 to 3.0.2¹². Details are presented in the Appendix 3.10 of this chapter. In both datasets, each delta consists of a delete, an update, and an insert

¹¹<https://tinyurl.com/ydbscsxf>

¹²<https://tinyurl.com/y7ukhge3>

workload, all measured in number of triples. The column ‘HasGeo’ shows the number of $\langle s, p, o \rangle$ triples with $p = \text{“hasGeometry”}$ whereas the numbers for the rest of the triples are given in column ‘Other’. For the YAGO dataset, we encountered a small number of ‘HasGeo’ deltas between versions 2.5.3 and 3.0.2; thus, only for this type of triples, we extracted the deltas using the last version of YAGO (3.1) instead of 3.0.2. YAGO 3.1 contains many more ‘Other’ triples we did not consider here since they are not related to any spatial entities and, hence, they do not affect the performance of re-encoding. Finally, LGD and YAGO deltas have been extracted using only types of triples that appear in both the initial and final versions.

Update Workloads. To generate realistic update workloads we split deltas in batches of varying size (in number of triples). For the smaller LGD dataset, we present experiments for batch sizes of 1M, 2M, 4M, and 8M triples whereas, for YAGO, we use batch sizes of 1M, 16M, 32M, and 64M triples. Each batch is marked as *delete*, *update* or *insert* and contains randomly selected triples from the respective delta. For instance, to form the delete batches of a specific batch size b , we first collect all delete triples (‘hasGeometry’ and ‘Other’) included in the delete delta, we shuffle them, and group them in batches of size b each. In all experiments of the next section, the batches are applied in a particular order, simulating the transition from the initial to the final version of the dataset: first all deletions, then all updates, and finally all insertions. The update benchmark can be found in the Appendix [3.10](#) of this chapter.

3.8.4 Updates Comparison

We compare SRX with Strabon, GraphDB, and Virtuoso. Results are given in Figure [3.9](#) and Figure [3.10](#) for LGD and YAGO respectively. Update experiments with Strabon (Figure [3.11](#)) were feasible only on small subsets of LGD and are discussed later on. All reported times for SRX and RDF-3X reflect the total time needed to construct the in-memory differential indexes and synchronize them with the base indexes (i.e. the B⁺-trees on disk).

Deletes. As shown in Figure [3.9](#) and Figure [3.10](#), the latency of processing a delete batch with RDF-3X and SRX shows a slight improvement across consecutive batches of the same size (left-most part of each plot) and tends to increase on average with the batch size. The slight improvement of delete latency over time is more apparent with YAGO and is reasonable since the overall database tends to get smaller. SRX latencies are in general higher than those of RDF-3X, and this is because SRX performs additional dictionary lookups and base index scans when trying to re-encode entities at lower levels of the grid. Overall, the median slowdown of SRX over RDF-3X across all delete batches is small: 0.5× for LGD, and 0.8× for YAGO. The slowdown is smaller for YAGO because the geometries in the delete batches of YAGO are almost

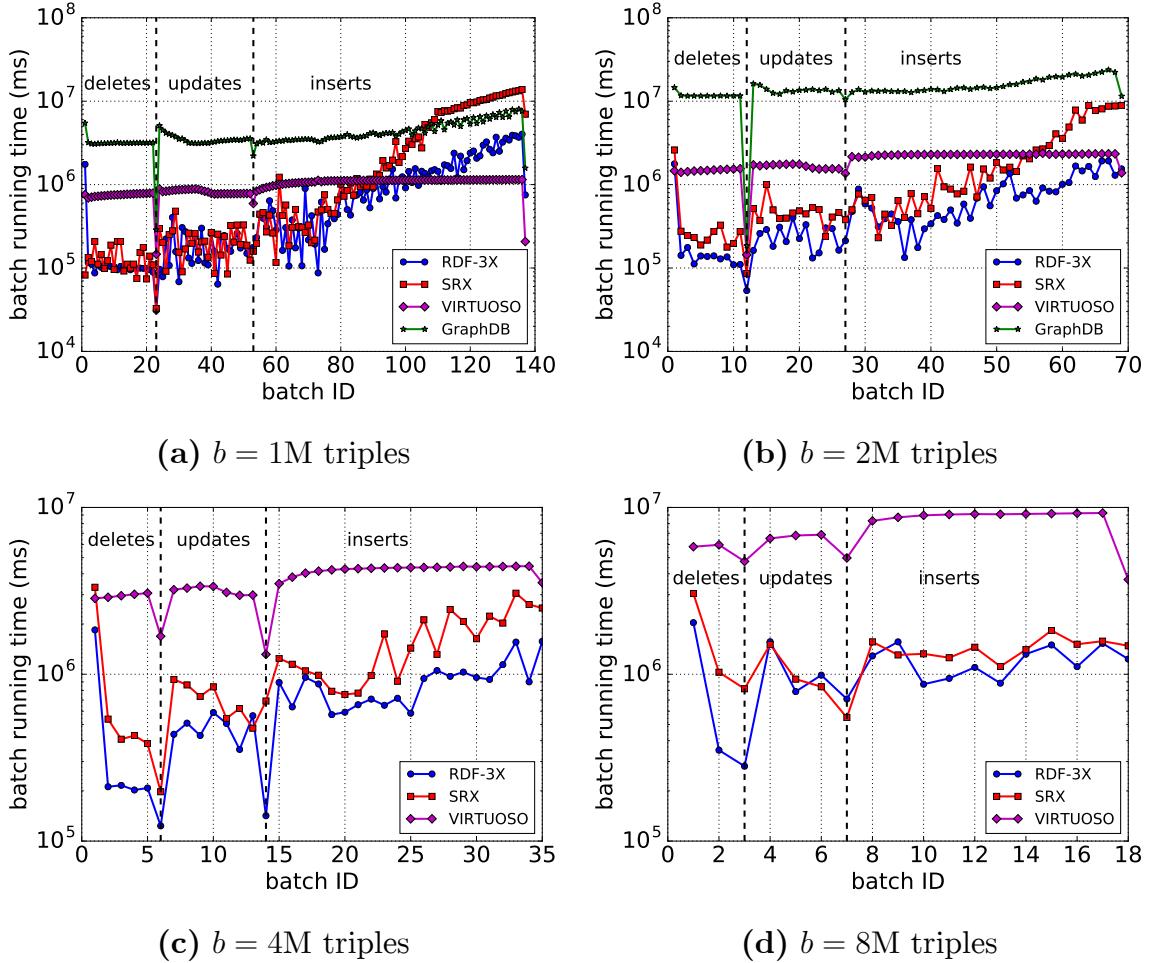


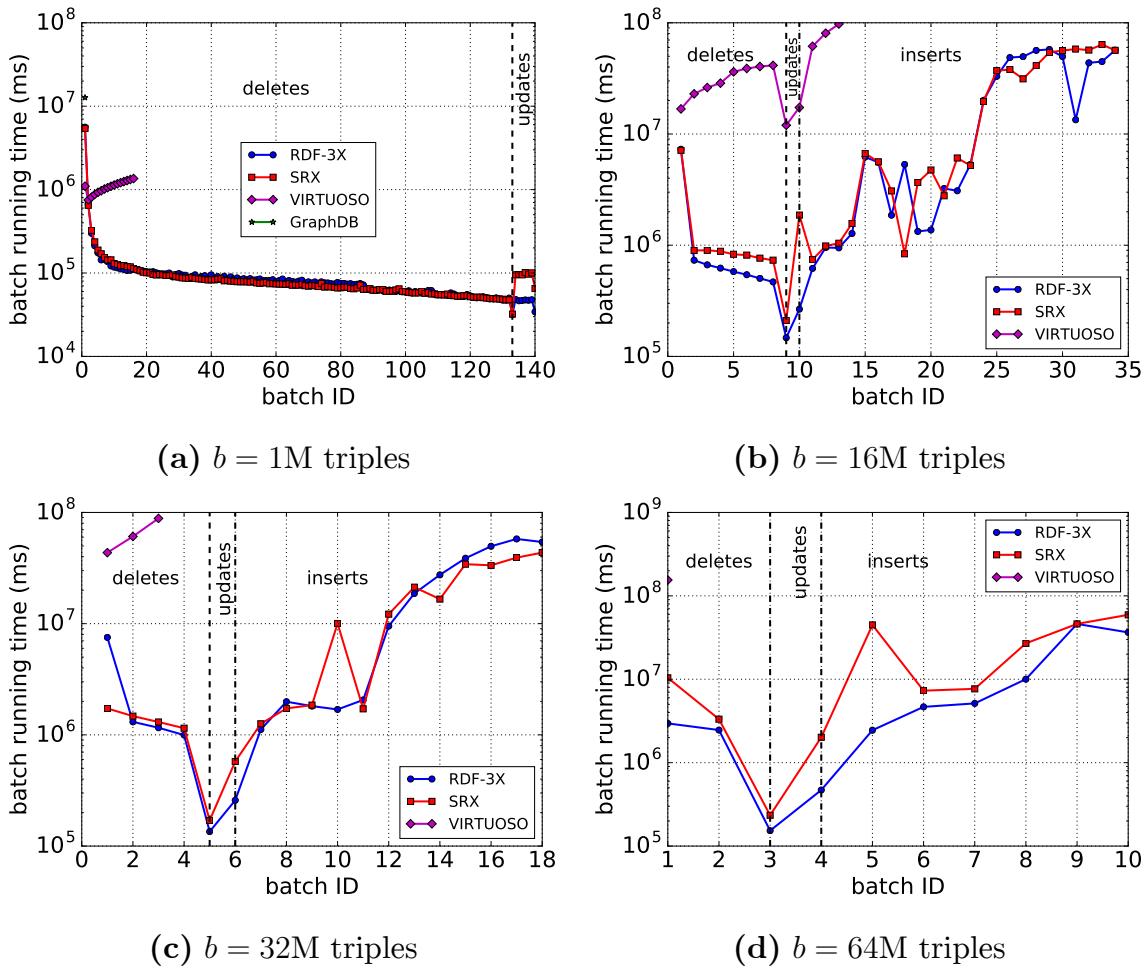
Figure 3.9: Latency (ms) of processing batches of size b on LGD.

half on average compared to LGD, hence, re-encoding is triggered less frequently. Note that the high processing latencies in the first delete batches of each experiment are due to some warm-up issues.

Inserts. The right-most part of each plot in Figure 3.9 and Figure 3.10 (labeled with “inserts”) shows the performance of SRX and RDF-3X for inserts. In contrast to deletes, the latency of processing an insert batch tends to increase over time for both SRX and RDF-3X, although it appears more stable for larger batches, especially for LGD (Figure 3.9d). We attribute this both to the differential index construction and to the index synchronization phases (cf. Section 3.7).

During the differential index construction, SRX and RDF-3X perform a number of database scans to check whether an input triple is new or old¹³. Hence, when the database increases in size as more insert batches are processed, the cost of these scans

¹³This check was not included in the version of RDF-3X we had but we added it for consistency.


 Figure 3.10: Latency (ms) of processing batches of size b on YAGO.

tends to increase. Second, and most important, whenever an index synchronization is performed and a leaf page of a base B^+ -tree overflows, the system generates a new page for the additional triples but avoids tree compaction. This is a common technique that tries to minimize the update latency at the cost of redundant leaf pages, which are expected to be filled by subsequent updates or compacted periodically when the system is idle. The technique achieves better leaf page utilization when used for bulk inserts of large size but does not perform well for small frequent inserts like those in Figure 3.9a-b and Figure 3.10b-c. Applying many small insert batches one after the other results in a large number of almost empty leaf pages (hence, a large increase in I/Os), and the performance of the system degrades significantly. This is in fact the reason we do not present times for insert batches in Figure 3.10a; after some point in this experiment, the processing of each single insert batch started taking almost double the time of the previous batch and we had to stop it. The problem we described is exacerbated due to the extensive use of indexes in RDF-3X (all of

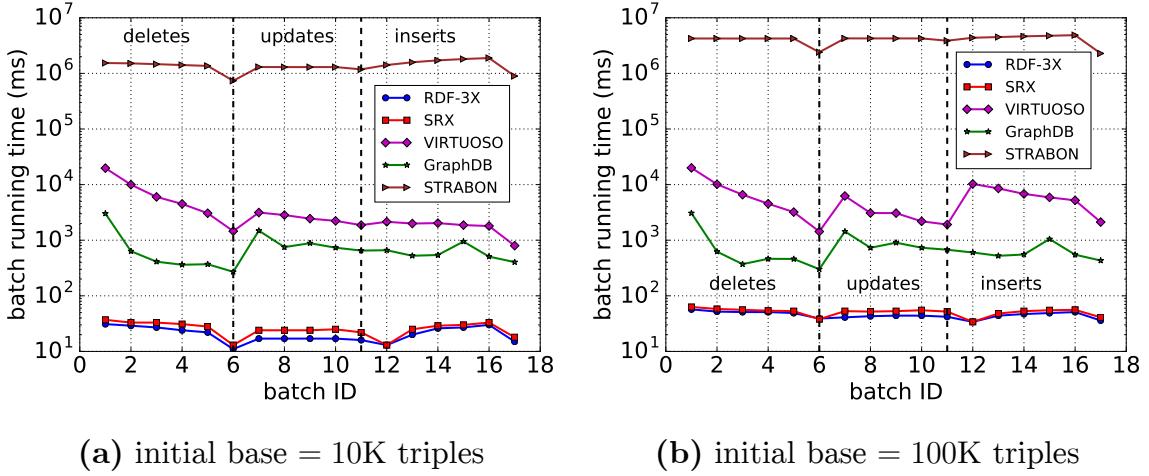


Figure 3.11: Latency (ms) of processing batches of size 1K triples on two different subsets (a) and (b) of LGD; the former dataset is a subset of the latter.

which have to be updated with the new triples) and becomes even more profound in SRX due to the additional re-assignment of IDs when new geometries are introduced for existing entities (*lines 6-16 in Updates on existing entities*). The latter also explains why the difference in performance of inserts between SRX and RDF-3X is amplified over time in Figure 3.9a-c. Overall, the median slowdown of SRX over RDF-3X across all insert batches is $0.6\times$ for both LGD and YAGO.

It should be noted, that SRX inherits its design from the last version of RDF-3X which targets read-intensive workloads and performs well under bulk inserts, but does not focus on Online transaction processing. There is some preliminary work on OLTP in [NW10b], but it has not been integrated to the current RDF-3X. We leave the efficient management of single triples and very small insert as future work.

Updates. The middle part of each plot in Figure 3.9 and Figure 3.10 (labeled with “updates”) shows the performance of SRX and RDF-3X in processing update batches. Here, the latency fluctuates slightly but tends to remain stable in the long term. Since an update in SRX and RDF-3X is implemented as a delete followed by an insert, we attribute the reasons behind this behavior to the combined performance of delete and insert batches, as explained before. Overall, the median slowdown of SRX over RDF-3X is $0.7\times$ for LGD and $0.3\times$ for YAGO.

Comparison with Existing Systems. We also performed experiments with Strabon, GraphDB, and Virtuoso, which we allowed to use the whole available memory of our machine, as for queries. Results are shown in Figure 3.9 for LGD, in Figure 3.10 for YAGO, and in Figure 3.11 for the LGD subsets. In YAGO, both GraphDB and Virtuoso were quite slow, and so we stopped the respective experiment at the point where each system had been running for as long as SRX took to apply all delete,

update, and insert batches. Moreover, GraphDB did not perform updates for batch sizes 4M and 8M in LGD, and 16M, 32M, and 64M in YAGO, due to memory overflow. In the next, we first discuss the update results on LGD and YAGO, and we then explain the results on the LGD subsets that we generated specifically for Strabon.

For LGD and YAGO, SRX performs significantly better compared to GraphDB and Virtuoso in both datasets, especially for deletes and updates, even though it also applies spatial re-encoding of entities. For inserts, SRX is almost an order of magnitude faster (in the worst case) with large batches, and gets worse than GraphDB and Virtuoso only when using batch sizes of 1M and 2M in LGD. We have already explained the reasons behind SRX’s performance degradation for inserts in the previous. Overall, the median speedup of SRX over GraphDB (resp. Virtuoso) for deletes, inserts, and updates is: $35.3\times$, $8.9\times$, and $22.6\times$ (resp. $5.9\times$, $2.4\times$, and $4.1\times$) for LGD. For YAGO, the respective median speedup of SRX over Virtuoso is: $41.2\times$, $82.9\times$, and $9.2\times$ (GraphDB was slow for YAGO and managed to apply only the first delete batch in Figure 3.10a).

The two datasets of Figure 3.11 are selected subsets from LGD, which we describe in the Appendix 3.10 of this chapter. Overall, the median speedup of SRX over GraphDB (resp. Virtuoso) for deletes, inserts, and updates on both these datasets is: $12.4\times$, $16.2\times$, and $22.6\times$ (resp. $132.1\times$, $91.9\times$, and $79.8\times$), while SRX performs very close to RDF-3X. Compared to results we get with LGD and YAGO, it is worth noticing that GraphDB performs better than Virtuoso with smaller datasets. Still, as for the case of queries, we cannot further comment on the performance of GraphDB and Virtuoso over all update experiments because they are not open-source systems.

Strabon is orders of magnitude slower than SRX and at least two orders of magnitude slower than Virtuoso because it does not support bulk updates; instead, each record in the update batch is treated as an individual database transaction. This approach has a high overhead in performance and cannot scale with frequent updates. As a side note, Strabon builds on the Sesame RDF store [BKh01] and extends Sesame’s query engine and optimizer, but not its transaction processing module.

3.9 Conclusion

In this chapter, we presented SRX, a system for spatial RDF data management built on top of the popular RDF-3X system. SRX employs a flexible scheme that encodes approximations of the geometries of the RDF entities into the entities’ IDs. The encoding is based on a hierarchical decomposition of the 2D space and can be effectively exploited in the evaluation of SPARQL queries with various types of spatial filters (ranges, distance joins, kNNs). We did experiments with real datasets showing

that our approach minimizes the evaluation cost of the spatial component in all RDF queries, while incurring a small overhead during updates.

3.10 Appendix

Here, we provide additional information on the datasets and benchmarks used in the experimental evaluation of this chapter. The material is provided for reproducibility purposes and consists of three parts. First, we give the spatial distribution of geometries in the two real datasets we used for query evaluation. Second, we provide the exact queries for each system, including Virtuoso, GraphDB, and Strabon. Third, we give details on the deltas extracted between different versions of the two datasets, which we used to generate update workloads.

To the best of our knowledge, the benchmark we present here is the first one for spatial RDF stores on real data. It includes: (i) a large collection of queries with different spatial predicates and selectivities, and (ii) a configurable update workload based on real changes in the two datasets over time.

3.10.1 Spatial distribution of geometries

Figures 3.12 and 3.13 show the density plots for the spatial distribution of geometries in the following two datasets, which have been used in Section 3.8.2:

1. LinkedGeodata (LGD), as retrieved from <https://tinyurl.com/yc4lxqdv>.
2. YAGO2s 2.5.3 (YAGO), as retrieved from <https://tinyurl.com/y7ukhge3>.

3.10.2 Queries

We describe the exact range, join, and kNN queries used in Section 3.8.2. Each query is uniquely identified by a QueryID, as used in this chapter. All queries can be found at: <https://web.imsi.athenarc.gr/SRX>.

3.10.2.1 Spatial range queries

The spatial range queries for LGD follow the template:

```
Select ?s
Where ?s name ?n . ?s label ?l .
    ?s type [TYPE] . ?s hasGeometry ?g .
Filter WITHIN(?g, "RECTANGLE([MBR])")
```

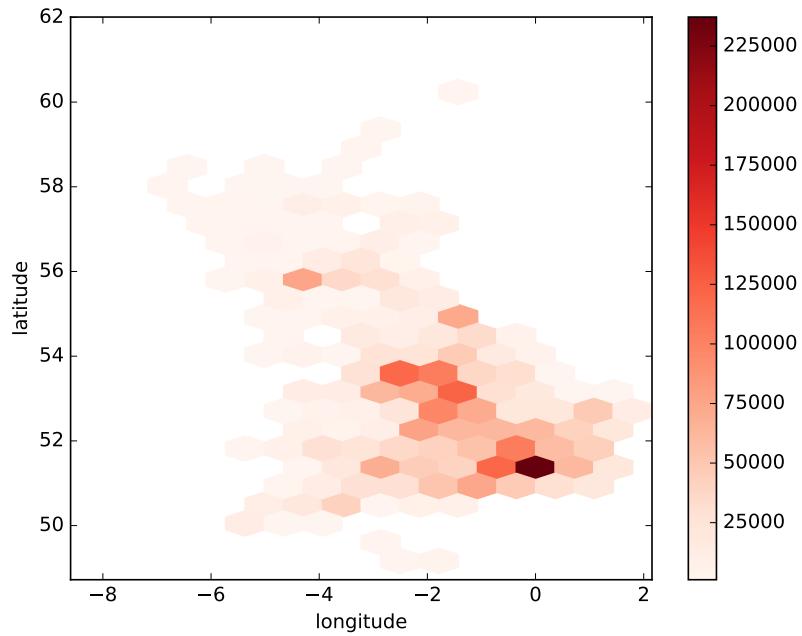


Figure 3.12: Spatial distribution of geometries in LGD. LGD contains geometries for entities in the United Kingdom with highest density in the area around London.

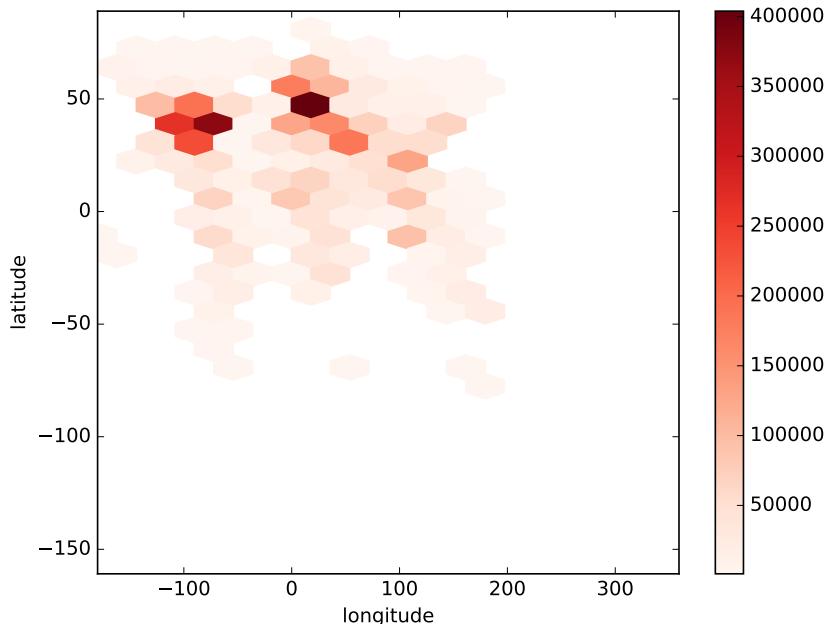


Figure 3.13: Spatial distribution of geometries in YAGO. YAGO's geometries spread all over the globe with highest density in North America and Europe.

where [TYPE] and [MBR] are instantiated for each query as follows:

QueryID	[TYPE]	[MBR]
LGD.SL1	police	-5,50,0,55
LGD.SL2	bus_stop	-10,50,0,60
LGD.SL3*	park	-5,50,0,55
LGD.LS1	pub	-5,45,0,50
LGD.LS2	bus_stop	-10,45,-5,50
LGD.LS3*	road	-10,45,-5,50
LGD.SS1	restaurant	-5,45,0,50
LGD.SS2*	park	-5,45,0,50
LGD.SS3*	road	-5,45,0,50
LGD.LL1	bus_stop	-5,55,0,60

The spatial range queries for YAGO are given in Table 3.19:

Table 3.19: Spatial range queries for YAGO.

YAGO.SL1* Select ?gn ?fn ?pr Where ?p hasGivenName ?gn . ?p hasFamilyName ?fn . ?p hasWonPrize ?pr . ?p diedIn ?c . ?c hasGeometry ?g . Filter WITHIN(?g, “RECTANGLE(-100, 20, -80, 40)”)	YAGO.SL2* Select ?gn ?fn Where ?p hasGivenName ?gn . ?p hasFamilyName ?fn . ?p a Wordnet_scientist_110560637 . ?p wasBornIn ?c . ?c hasGeometry ?g . Filter WITHIN(?g, “RECTANGLE(-95, 40, -90, 45)”)
YAGO.LS1* Select ?p ?w Where ?p hasAcademicAdvisor ?a . ?a worksAt ?w . ?w isLocatedIn ?l . ?l hasGeometry ?g . Filter WITHIN(?g, “RECTANGLE(-160, -50, -150, -40)”)	YAGO.LS2* Select ?e ?c Where ?e happenedIn ?l . ?l a ?c . ?c subClassOf Wordnet_city_108524735 . ?l hasGeometry ?g . Filter WITHIN(?g, “RECTANGLE(-130, 40, -120, 50)”)
YAGO.SS1* Select ?gn ?fn Where ?p hasGivenName ?gn . ?p hasFamilyName ?fn . ?p a Wordnet_scientist_110560637 . ?p wasBornIn ?c . ?c hasGeometry ?g . Filter WITHIN(?g, “RECTANGLE(-105, 45, -100, 50)”)	YAGO.SS2* Select ?p ?w Where ?p graduatedFrom ?u . ?p worksAt ?w . ?u isLocatedIn ?l . ?l hasGeometry ?g . Filter WITHIN(?g, “RECTANGLE(-110, 50, -100, 60)”)
YAGO.LL1* Select ?e ?c Where ?e happenedIn ?l . ?l a ?c . ?c subClassOf Wordnet_city_108524735 . ?l hasGeometry ?g . Filter WITHIN(?g, “RECTANGLE(-90, 30, -80, 40)”)	YAGO.LL2* Select ?p Where ?p hasArea ?a . ?p isLocatedIn ?l . ?l hasGeometry ?g . Filter WITHIN(?g, “RECTANGLE(-100, 30, -90, 40)”)

3.10.2.2 Spatial distance join queries

The spatial distance join queries for LGD are given in Table 3.20:

Table 3.20: Spatial distance join queries for LGD.

LGD.J1 (point-point) Select ?s1 ?s2 Where ?s1 type hotel . ?s1 hasGeometry ?g1 . ?s2 type hotel . ?s2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “0.003”	LGD.J2 (point-point) Select ?s1 ?s2 ?l1 ?l2 Where ?s1 name ?l1 . ?s1 label ?b1 . ?s1 type police . ?s1 hasGeometry ?g1 . ?s2 name ?l2 . ?s2 label ?b2 . ?s2 type police . ?s2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “0.01”
LGD.J3 (point-point) Select ?s1 ?s2 Where ?s1 name ?l1 . ?s1 label ?b1 . ?s1 type pub . ?s1 hasGeometry ?g1 . ?s2 name ?l2 . ?s2 label ?b2 . ?s2 type police . ?s2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “0.02”	LGD.J4* (polygon-polygon) Select ?s1 ?s2 Where ?s1 type park . ?s1 hasGeometry ?g1 . ?s2 type park . ?s2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “0.05”
LGD.J5* (point-polygon) Select ?s1 ?s2 Where ?s1 label ?b1 . ?s1 type police . ?s1 hasGeometry ?g1 . ?s2 label ?b2 . ?s2 type park . ?s2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “0.01”	LGD.J6* (point-line), [EPS] ∈ {0.01,0.001,0.0005} Select ?s1 ?s2 Where ?s1 label ?b1 . ?s1 type hotel . ?s1 hasGeometry ?g1 . ?s2 label ?b2 . ?s2 type road . ?s2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “[EPS]”

The spatial distance join queries for YAGO are given in Table 3.21:

Table 3.21: Spatial distance join queries for YAGO.

YAGO.J1* Select ?c1 ?c2 Where ?a1 hasAirportCode ?c1 . ?a1 hasGeometry ?g1 . ?a2 hasAirportCode ?c2 . ?a2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “0.1”	YAGO.J2* Select ?p1 ?p2 Where ?p1 hasGivenName ?gn1 . ?p1 hasFamilyName ?fn1 . ?p1 hasWonPrize ?pr1 . ?p1 wasBornIn ?c1 . ?c1 hasGeometry ?g1 . ?p2 hasGivenName ?gn2 . ?p2 hasFamilyName ?fn2 . ?p2 hasWonPrize ?pr2 . ?p2 wasBornIn ?c2 . ?c2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “0.1”
YAGO.J3* Select ?p ?c1 ?c2 Where ?p hasGivenName ?gn . ?p hasFamilyName ?fn . ?p actedIn ?m . . ?m isLocatedIn ?c1 . ?c1 hasGeometry ?g1 . . ?p wasBornIn ?c2 . ?c2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “0.1”	YAGO.J4* Select ?p1 ?p2 Where ?p1 hasFamilyName ?fn1 . . ?p1 wasBornIn ?c1 . ?c1 hasGeometry ?g1 . . ?p1 isMarriedTo ?p2 . ?p2 wasBornIn ?c2 . . ?c2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “0.1”
YAGO.J5* Select ?p Where ?p hasFamilyName ?fn . . ?p livesIn ?c1 . ?c1 hasGeometry ?g1 . . ?p worksAt ?c2 . ?c2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “0.1”	YAGO.J6* Select ?p1 ?p2 Where ?p1 hasGivenName ?gn1 . . ?p1 hasFamilyName ?fn1 . ?p1 a Wordnet_scientist_110560637 . . ?p1 wasBornIn ?c1 . ?c1 hasGeometry ?g1 . . ?p2 hasGivenName ?gn2 . ?p2 hasFamilyName ?fn2 . . ?p2 a Wordnet_scientist_110560637 . . ?p2 diedIn ?c2 . ?c2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “0.1”
YAGO.J7* Select ?p1 ?p2 Where ?p1 graduatedFrom ?u1 . . ?u1 hasGeometry ?g1 . ?p2 actedIn ?m2 . . ?m2 isLocatedIn ?l2 . ?l2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “0.1”	YAGO.J8*, [EPS] ∈ {0.1,0.01,0.001} Select ?p1 ?p2 Where ?p1 worksAt ?w1 . . ?w1 hasGeometry ?g1 . ?p2 worksAt ?w2 . . ?w2 hasGeometry ?g2 . Filter DISTANCE(?g1,?g2) < “[EPS]”

3.10.2.3 Spatial kNN queries for Encoding, Baseline, and Basic

The spatial kNN queries for LGD have the following template:

```
Select ?s
Where ?s name ?n . ?s label ?l .
?s type [TYPE] . ?s hasGeometry ?g .
Filter kNN(?g, “POINT([COORDS])”, k)
```

where $k \in \{5, 10, 20, 50, 100\}$, and [TYPE], [COORDS] are instantiated for each query as follows:

QueryID	[TYPE]	[COORDS]
LGD.SL1	police	-2.5, 52.5
LGD.SL2	bus_stop	-5, 55
LGD.SL3*	park	-2.5, 52.5
LGD.LS1	pub	-2.5, 47.5
LGD.LS2	bus_stop	-7.5, 47.5
LGD.LS3*	road	-7.5, 47.5
LGD.SS1	restaurant	-2.5, 47.5
LGD.SS2*	park	-2.5, 47.5
LGD.SS3*	road	-2.5, 47.5
LGD.LL1	bus_stop	-2.5, 57.5

The spatial kNN queries for YAGO, where $k \in \{5, 10, 20, 50, 100\}$, are given in Table 3.22:

Table 3.22: Spatial kNN queries for YAGO.

YAGO.SL1* Select ?gn ?fn ?pr Where ?p hasGivenName ?gn . ?p hasFamilyName ?fn . ?p hasWonPrize ?pr . ?p diedIn ?c . ?c hasGeometry ?g . Filter kNN(?g, “POINT(-90, 30)”, k)	YAGO.SL2* Select ?gn ?fn Where ?p hasGivenName ?gn . ?p hasFamilyName ?fn . ?p a Wordnet_scientist_110560637 . ?p wasBornIn ?c . ?c hasGeometry ?g . Filter kNN(?g, “POINT(-92.5, 42.5)”, k)
YAGO.LS1* Select ?p ?w Where ?p hasAcademicAdvisor ?a . ?a worksAt ?w . ?w isLocatedIn ?l . ?l hasGeometry ?g . Filter kNN(?g, “POINT(-155, -45)”, k)	YAGO.LS2* Select ?e ?c Where ?e happenedIn ?l . ?l a ?c . ?c subClassOf Wordnet_city_108524735 . ?l hasGeometry ?g . Filter kNN(?g, “POINT(-125, 45)”, k)
YAGO.SS1* Select ?gn ?fn Where ?p hasGivenName ?gn . ?p hasFamilyName ?fn . ?p a Wordnet_scientist_110560637 . ?p wasBornIn ?c . ?c hasGeometry ?g . Filter kNN(?g, “POINT(-102.5, 47.5)”, k)	YAGO.SS2* Select ?p ?w Where ?p graduatedFrom ?u . ?p worksAt ?w . ?u isLocatedIn ?l . ?l hasGeometry ?g . Filter kNN(?g, “POINT(-105, 55)”, k)
YAGO.LL1* Select ?e ?c Where ?e happenedIn ?l . ?l a ?c . ?c subClassOf Wordnet_city_108524735 . ?l hasGeometry ?g . Filter kNN(?g, “POINT(-85, 35)”, k)	YAGO.LL2* Select ?p Where ?p hasArea ?a . ?p isLocatedIn ?l . ?l hasGeometry ?g . Filter kNN(?g, “POINT(-95, 35)”, k)

3.10.2.4 Spatial kNN queries for Virtuoso

Virtuoso 7.2.5-rc1.3217-pthreads does not support nearest neighbor queries per se, yet, some of these queries can be expressed using a combination of distance filters and ordering. The spatial kNN queries we used for Virtuoso involve only point queries as other geometries are not supported by the distance function in this version of the system. In Virtuoso’s query language these queries for LGD follow the template:

```
Select ?s DISTANCE(?g, “POINT([COORDS])”)
Where ?s name ?n . ?s label ?l .
    ?s type [TYPE] . ?s hasGeometry ?g .
Filter WITHIN(?g, “POINT([COORDS])”, [RANGE])
Order By ASC 2
Limit k
```

where $k \in \{5, 10, 20, 50, 100\}$, and [TYPE], [COORDS] are instantiated for each query as in Section 3.10.2.3 of this chapter. [RANGE] is measured in kilometers and is the minimum distance so that each query returns 100 results. [RANGE] is set for each query as follows:

QueryID	[RANGE]
LGD.SL1	110
LGD.SL2	20
LGD.LS1	170
LGD.LS2	190
LGD.SS1	180
LGD.LL1	80

3.10.2.5 Spatial kNN queries for GraphDB and Strabon

Similarly to Virtuoso, GraphDB Free 8.6 and Strabon 3.3.2 do not support nearest neighbor queries per se, however, both systems can express them using a combination of distance filters and ordering. The spatial kNN queries for LGD in GraphDB's and Strabon's query language follow the same template:

```

Select ?s
Where ?s name ?n . ?s label ?l .
? s type [TYPE] . ?s hasGeometry ?g .
Order By ASC
DISTANCE(?g, "POINT([COORDS])")
Limit k

```

where $k \in \{5, 10, 20, 50, 100\}$, and **[TYPE]**, **[COORDS]** are instantiated for each query as in Section 3.10.2.3 of this chapter.

The spatial kNN queries for YAGO in GraphDB's query language are given in Table 3.23 with $k \in \{5, 10, 20, 50, 100\}$:

Table 3.23: Spatial kNN queries for YAGO in GraphDB.

YAGO.SL1* Select ?gn ?fn ?pr Where ?p hasGivenName ?gn . ?p hasFamilyName ?fn . ?p hasWonPrize ?pr . ?p diedIn ?c . ?c hasGeometry ?g . Order By ASC DISTANCE(?g, "POINT(-90, 30)") Limit k	YAGO.SL2* Select ?gn ?fn Where ?p hasGivenName ?gn . ?p hasFamilyName ?fn . ?p a Wordnet_scientist_110560637 . ?p wasBornIn ?c . ?c hasGeometry ?g . Order By ASC DISTANCE(?g, "POINT(-92.5, 42.5)") Limit k
YAGO.LS1* Select ?p ?w Where ?p hasAcademicAdvisor ?a . ?a worksAt ?w . ?w isLocatedIn ?l . ?l hasGeometry ?g . Order By ASC DISTANCE(?g, "POINT(-155, -45)") Limit k	YAGO.LS2* Select ?e ?c Where ?e happenedIn ?l . ?l a ?c . ?c subClassOf Wordnet_city_108524735 . ?l hasGeometry ?g . Order By ASC DISTANCE(?g, "POINT(-125, 45)") Limit k
YAGO.SS1* Select ?gn ?fn Where ?p hasGivenName ?gn . ?p hasFamilyName ?fn . ?p a Wordnet_scientist_110560637 . ?p wasBornIn ?c . ?c hasGeometry ?g . Order By ASC DISTANCE(?g, "POINT(-102.5, 47.5)") Limit k	YAGO.SS2* Select ?p ?w Where ?p graduatedFrom ?u . ?p worksAt ?w . ?u isLocatedIn ?l . ?l hasGeometry ?g . Order By ASC DISTANCE(?g, "POINT(-105, 55)") Limit k
YAGO.LL1* Select ?e ?c Where ?e happenedIn ?l . ?l a ?c . ?c subClassOf Wordnet_city_108524735 . ?l hasGeometry ?g . Order By ASC DISTANCE(?g, "POINT(-85, 35)") Limit k	YAGO.LL2* Select ?p Where ?p hasArea ?a . ?p isLocatedIn ?l . ?l hasGeometry ?g . Order By ASC DISTANCE(?g, "POINT(-95, 35)") Limit k

3.10.3 Deltas between different dataset versions

Here, we provide the exact differences (in number of triples) between versions:

1. 2013_04_29 and 2015_11_02 of LGD, as retrieved from <https://tinyurl.com/ydbscsxf>.

3.10. Appendix

2. 2.5.3 and 3.0.2 of YAGO, as retrieved from <https://tinyurl.com/y7ukhge3>.

which we used to generate the update workloads of Section 3.8.4 of this chapter. The workloads can be found at: <https://web.imsi.athenarc.gr/SRX>.

Table 3.24: Deltas between LGD versions 2013_04_29 (14.7GB) and 2015_11_02 (21.6GB) used for the update benchmark.

Category	Deleted triples		Updated triples		Inserted triples	
	HasGeometry	Other	HasGeometry	Other	HasGeometry	Other
AerialwayThing_Nodes	2,265	19,761	12,640	61,878	22,799	198,796
AerialwayThing_Ways	1,126	15,329	9,419	38,345	9,486	94,979
AerowayThing_Nodes	7,860	95,947	7,366	53,177	18,488	174,201
AerowayThing_Ways	7,972	84,099	41,772	131,235	87,970	816,948
Amenity_Nodes	411,396	4,694,204	383,260	3,126,323	1,958,451	20,917,360
Amenity_Ways	168,547	2,381,609	465,889	1,913,576	1,705,501	19,771,641
Craft_Nodes	571	8,322	831	6,477	19,720	220,220
Craft_Ways	333	8,973	1,070	5,709	9,483	134,084
EmergencyThing_Nodes	4,996	107,124	9,631	197,532	369,059	4,031,998
EmergencyThing_Ways	3,002	62,098	9,412	42,782	26,342	442,821
HistoricThing_Nodes	11,783	241,258	20,013	191,571	170,642	1,459,176
HistoricThing_Ways	5,703	98,317	16,870	94,209	89,578	1,211,885
MilitaryThing_Nodes	472	5,968	583	6,345	11,282	147,547
MilitaryThing_Ways	732	12,412	2,155	9,376	10,062	122,245
Place_Nodes	72,941	10,117,063	183,472	5,211,302	930,603	10,494,621
Place_Ways	21,435	337,257	67,907	433,830	169,437	1,932,685
Shop_Nodes	79,362	948,495	115,006	999,194	678,149	7,271,373
Shop_Ways	21,683	1,300,070	59,325	205,847	242,059	906,075
TourismThing_Nodes	47,322	523,433	62,653	466,938	375,669	4,065,277
TourismThing_Ways	14,438	238,691	35,340	178,622	121,136	1,741,557
Total	883,939	21,300,430	1,504,614	13,374,268	7,025,916	76,155,489

Table 3.25: Deltas between YAGO versions 2.5.3 (18.1GB) and 3.0.2 (33.4GB) used for the update benchmark.

Category	Deleted triples		Updated triples		Inserted triples	
	HasGeometry	Other	HasGeometry	Other	HasGeometry	Other
yagoDBpediaClasses	0	381,887	0	0	0	500,640
yagoDBpedialInstances	0	85,949	0	0	0	2,637,748
yagoFacts	0	1,198,399	0	325	0	2,379,436
yagoGeonamesClasses	0	585	0	0	0	586
yagoGeonamesClassIds	0	0	0	0	0	1
yagoGeonamesData	473,999	6,661,873	1,693,055	0	2,973,857	21,919,394
yagoGeonamesEntityIds	0	106,133	0	0	0	117,579
yagoGeonamesGlosses	0	0	0	0	0	1
yagoImportantTypes	0	2,723,628	0	0	0	0
yagoLabels	0	6,941,073	0	1,601,561	0	36,642,854
yagoLiteralFacts	0	2,533,216	0	650	0	1,584,304
yagoMetaFacts	0	820,616	0	0	0	2,295,716
yagoMultilingualClassLabels	0	0	0	0	0	1
yagoMultilingualInstanceLabels	0	8,164,316	0	0	0	0
yagoSchema	0	10	0	24	0	133
yagoSimpleTaxonomy	0	5,670	0	0	0	474,817
yagoSimpleTypes	0	4,334,474	0	0	0	15,824,332
yagoSources	0	61,838,095	0	0	0	190,482,753
yagoStatistics	0	7	0	88	0	5
yagoTaxonomy	0	381,893	0	0	0	500,645
yagoTransitiveType	0	11,145,280	0	0	0	45,085,647
yagoTypes	0	8,274,938	0	0	0	16,182,189
yagoWikipediaInfo	0	16,347,220	0	0	0	36,167,000
yagoWordnetDomains	0	172	0	0	0	1
yagoWordnetIds	0	0	0	0	0	1
Total	473,999	131,945,434	1,693,055	1,602,648	2,973,857	372,795,783

Columns “HasGeometry” in Tables 3.24 and 3.25 show the numbers of $\langle s, p, o \rangle$ triples where $p = \text{“hasGeometry”}$. These triples contain entity geometries as objects. The respective numbers for the rest of the triples are given in columns “Other”.

For the update experiments with Strabon, which we present in Figure 3.11 of chapter, we created two different subsets of LGD (2013_04_29) that we used in Table 3.24. The first one corresponds to the initial base of Figure 3.11a, while the second is a superset of the first and corresponds to the initial base of Figure 3.11b.

To form the initial base (a) in Figure 3.11a, we initially took the first 250 point (Nodes) and 250 linestring (Ways) ‘HasGeometry’ triples, including all their related ‘Other’ triples, from each of the deleted, updated, and inserted LGD deltas in Table 3.24. These are the `deltas_a` to apply on base (a) in Figure 3.11a. Then, we kept from the initial version of LGD (2013_04_29) only the triples related to any spatial entities found in `deltas_a`, and we formed the base (a) that contains 10K triples.

The initial base (b) in Figure 3.11b contains 100K triples, and is created similarly. In this case, we picked the first 2500 point (Nodes) and 2500 linestring (Ways) ‘HasGeometry’ triples, including all their related ‘Other’ triples, from each of the LGD deltas.

3.10. Appendix

Chapter 4

Content Recommendation for Viral Social Influence

How do we create content that will become viral in a whole network after we share it with friends or followers? Significant research activity has been dedicated to the problem of strategically selecting a *seed set* of initial adopters so as to maximize a meme's spread in a network. This line of work assumes that the success of such a campaign depends solely on the choice of a *tunable* seed set of adopters, while the way users perceive the propagated meme is *fixed*. Yet, in many real-world settings, the opposite holds: a meme's propagation depends on users' perceptions of its *tunable* characteristics, while the set of initiators is *fixed*.

In this chapter, we address the natural problem that arises in such circumstances: Suggest content, expressed as a limited set of *attributes*, for a creative promotion campaign that starts out from a given seed set of initiators, so as to maximize its expected spread over a social network. To our knowledge, no previous work addresses this problem. We find that the problem is NP-hard and inapproximable. As a tight approximation guarantee is not admissible, we design an efficient heuristic, **Explore-Update**, as well as a conventional **Greedy** solution. Our experiments show that **Explore-Update** selects near-optimal attribute sets with real data, achieves 30% higher spread than baselines, and runs an order of magnitude faster than the **Greedy** solution.

4.1 Introduction

Online networking offers opportunities for new types of marketing. A prime example of such a new marketing technique is viral marketing, whereby organizations run promotion campaigns through word-of-mouth effects within online social networks. The *Influence Maximization* (IM) problem [KKT03], studied intensively during the last decade, aims to find well-chosen *seed nodes* from which to launch such campaigns so as to achieve good results.

Recent works [dVGL12, CM13] have focused on the parameters that define the popularity of a post, campaign, idea, or *meme* within a network. Such works were the first to study the question of how commercial brand posts engage online social network users, drawing from the theory of Uses & Gratifications [Kat59]; they examine post parameters such as *content type* (e.g., entertaining, informational), *media type* (e.g., vivid, interactive), *posting time* (e.g., workday, peak hours) and *valence of comments* (e.g., positive, negative). Interestingly, such studies have reached some ambivalent conclusions; for instance, [dVGL12] ascertains that entertaining content decreases the number of “likes”, while [CM13] claims the exact opposite.

Concurrent research has studied the problem of *viral product design* [AW11, BB14], which calls for engineering products by incorporating viral attributes so as to generate peer-to-peer influence that encourages adoption within a network. Aral and Walker [AW11] study the question of viral attribute selection under randomized trials only; Barbieri and Bonchi [BB14] allude to the same problem as a complement to the standard IM problem of selecting a set of seed nodes that maximizes influence, but do not investigate it as a stand-alone problem in its own right. Conceptually, both these works pertain to attributes attached to products; they do not investigate the more general problem of choosing content, out of a set of eligible options, for any kind of *meme* spreading in a network, so as to make it viral.

In this chapter, we introduce and study the problem of selecting content that characterizes any type of *meme*, so as to maximize its expected spread through a network, starting out from a fixed set of initial adopters. For instance, an advertisement post may feature aspects such as *topics*, *people*, *locations* and *abstract themes*. We are particularly interested in those content aspects that are associated with specific *online social network pages*; we denote such aspects as **content attributes**. Fittingly, online social network users themselves are associated with such non-personal network pages: they express their preferences for specific brands, topics of interest, public persons, hobbies, or locations by subscribing to or “liking” such pages. Thereby, an attribute’s popularity can be gauged via its number of subscribers or page “likes”. For our purposes, we denote the pages that a user subscribes to or “likes” as **user attributes**. We contend that, the more content and user attributes overlap, the more likely that user is to propagate that post. We envisage an organization that aims to achieve high viral effect of a campaign initiated from its fixed set of subscribers. For example, assume *FlyFast* airways wants to launch a promotion campaign in social media. *FlyFast* already has a social network presence, and its page has a subscribers’ set S fixed at a given moment, while it faces constraints related to its budget and people’s attention span. In their design, *FlyFast* consultants are interested to identify a set of k content attributes, out of a universe of eligible, mutually compatible options, that will maximize the expected network spread of a post starting out from its

subscribers' set S . Assume that, for $k = 4$, the optimal attribute set is {“Best travel Accessories”, “Airline food guide”, “Hipster Europe”, “Backpacker tips”}. Guided by this knowledge, *FlyFast* can infuse its post with complementary content that appeals to users interested in those topics, e.g., promotions to backpackers, references to its hipster audience, and highlights on its food quality. Thereby, it can maximize its promotion's reach.

We are the first to study the *Influence Maximization* problem in which the seed is given and post content is *sought*. Our related contributions are as follows:

Problem Setting. We motivate the Influence Maximization problem in settings where the set of initial adopters is fixed, or even a single point of origin, and the content of a propagated meme can be tuned. We formulate the concept of *digital influence* as a special case of social influence.

Propagation Model. We devise a *content-aware* propagation model, whereby the probability of influence across edges depends on content. We show that, with this model: (i) the problem of choosing content attributes that maximize influence is NP-hard; (ii) the spread function is not submodular, hence no submodularity-based approximation algorithm applies; and (iii) it is NP-hard to approximate the optimal solution within a factor of $n^{1-\varepsilon}$ for $\varepsilon > 0$.

Algorithm. We design a fast algorithm, *Explore-Update*, which achieves higher influence spread than baselines; its effectiveness is based on the iterative estimation of the marginal spread achieved by each attribute, while its efficiency is gained by limiting such computations only to nodes within a probability-based distance threshold θ and attributes potentially affecting such nodes.

Experiments. We compare *Explore-Update* to two baselines and show that it always achieves better propagation results, while it is significantly faster than a naïve Greedy approach; we calculate the optimal solution on a reduced dataset with a small universe of attributes, showing *Explore-Update* can achieve optimality; last, we demonstrate the scalability of *Explore-Update* on seed set size.

4.2 Motivation

4.2.1 Idea Habitats

An *idea habitat* [BH05] is the set of environmental cues that make people think about an idea and pass it along. Regardless of how well an idea is encoded, it will persist and spread only if the environment cues people to retrieve it regularly. In other words, even if an idea *can* be easily recalled, if it is only rarely cued by the environment, it may remain rare and be forgotten. It follows that, for an idea to

spread, it should be not only well encoded and easily recalled, but also regularly retrieved. Promotion campaigns aim to assist the spread of any meme in the same way as idea habitats assist the spread of ideas. For instance, assume that *FlyFast* food is good and therefore memorable. Still, without a viral promotion, *FlyFast* will miss the spotlight, letting other airlines gain public attention.

4.2.2 Digital Influence

Social influence is defined based on peer behavior [Ara11], so as to distinguish it from confounding factors [GM04, dBL01, AMS09, Man93].

Definition 2. Social Influence expresses the extent to which the behavior of one's peers changes the utility one expects to receive from engaging in a certain behavior, and hence the likelihood that one will engage in that behavior.

This definition can be used to make an argument that, if we understand how behaviors spread from person to person, our society will be able to promote agreeable behaviors, such as physical exercise and financial responsibility, and limit disagreeable ones, such as violence and dirty needle sharing. By this definition, peer behaviors may relate to awareness, persuasiveness, imitation and social learning [Ara11]. We propose that *digital influence* can be seen as a case of awareness-related social influence, defined as follows:

Definition 3. Digital Influence expresses the extent to which the content of a commercial digital posting changes the support one wants to provide to that posting, and thus the likelihood that one will propagate it.

In online social networks, posts are propagated from user to user by means of actions such as *like*, *share*, or *repost* [ALB⁺15]. Without loss of generality, we group all these actions under the *like* action. What matters is whether users endorse and promote a post further by making it visible to their friends and followers. In our setting, we emphasize the importance of earning *likes* from users at large.

Nowadays, as online social network users are exposed to a plethora of posts, we can safely assume that they become selective on what they *like*. Therefore, a brand that fails to issue likable posts via its social network pages is unlikely to spread awareness of its activities and products. In the same vein, a brand that can estimate how viral a post will be and create appropriate digital content, stands good chances to succeed. We argue that such estimation can be based on the digital influence of content associated with a meme.

4.2.3 Distinctiveness

We emphasize three distinct elements of this work.

Problem Formulation. While our model takes into consideration the influence with regard to a post exercised by users themselves, we seek to maximize the influence exercised by the appeal and quality of a post’s content within a network. To the best of our knowledge, we are the first to define this problem, which is distinct from, and cannot be treated by methods aiming at, user selection.

Nature of Attributes. In related works, attributes are configured as *general topics*; e.g., in [BB14] authors assign *tags*, conceived of as general topics, to the role of product attributes. By contrast, in our problem formulation and in our experimental study, an attribute corresponds to a topic of interest *identifiable via a non-personal social network page*. Thus, even a page on an abstract theme (e.g., *Psychology of Relations*) is a possible attribute in our setting: it has a specific commercial value (4,719,837 followers) that differentiates it from other pages on similar topics.

Use of Tags. The type of content attributes we investigate can hardly be articulated via tagging. Tags express highly idiosyncratic user impressions, focusing on arbitrary aspects of content; they are volatile as descriptors of content, whereas we need a stable ground truth to represent content. For example, a video post may relate to several specific topics of interest, yet it would be hard to identify these via user tagging. In consequence, past tagging does not offer valuable information in our problem setting.

4.3 Problem Statement

From the preceding discussion, we conclude that any brand would gain by maximizing the expected effectiveness of its product promotion campaigns within an online social network. We assume that there exists a certain set of *subscribers* to the brand’s social network page, and a promotion campaign aims to influence the maximum number of *non-subscribers*; as we discussed, such users are associated with topics expressing their interests.

4.3.1 Content-Aware Cascade Model

We model an online social network as a directed graph $G = (V, E)$, where $V = \{v_1, v_2, \dots, v_n\}$ is a set of nodes, each of which corresponds to an individual user, and $E \subset V \times V$ is a set of directed edges representing social relations among users. Each node v has a set of associated attributes $F_v = \{f_v^1, f_v^2, \dots\}$, from a universe Φ , that define user preferences; we identify these attributes as the non-personal network pages

a user expresses interest in. A meme propagated through the network is associated with a set of attributes $F = \{f_{p_1}, f_{p_2}, \dots, f_{p_K}\} \subseteq \Phi$; these content attributes, along with the user attributes F_v associated with the targeted node v , affect the probability of its propagation across a network edge e_{uv} .

Accordingly, we define the *Content-Aware Cascade* model (CAC) as a variant of the Independent Cascade model (IC), in which edge propagation probabilities depend on content and user attributes. A CAC diffusion process unfolds in steps, starting from an initial seed set of activated nodes. A node u activated at time step t has a single chance to activate its out-neighbors. The process is incremental, as nodes can alternate only from inactive to active states; the diffusion ends when there is no newly activated node at a given step. At any step, a newly activated node u activates its out-neighbor v with probability $p(u, v)$ equal to:

$$\begin{aligned} p_{uv} &= b_{uv} + q_{uv} \cdot h_{uv}(F_v, F), \quad b_{uv}, q_{uv} \in [0, 1] \\ h_{uv}(F_v, F) &= \min \left\{ \frac{1-b_{uv}}{q_{uv}}, |F_v \cap F| \right\} \end{aligned} \tag{4.1}$$

where b_{uv} is a *base probability* on an edge and q_{uv} a *marginal probability* that indicates how much the probability on an edge increases for each selected attribute in F matching a preference of node v , as indicated by the transition function $h_{uv}(F_v, F)$, with a sanity bound of $\frac{1-b_{uv}}{q_{uv}}$. We stress that the marginal probability q_{uv} distinguishes among different user links, albeit not among different attributes for a given link; a more complex model could distinguish among different attributes, or even define a probability distribution function over the set of all attributes [BBM12], to be learned by historical logs. We choose to relegate the problem of defining and learning such probability distribution functions to future work, and now study the problem under the modeling assumption that each attribute has the same independent effect on the probability function. Yet, our simplified model forms a special case of any more complex model in which each attribute would have a different effect on the probability function; i.e., in this special case, such effects are rendered equal. So, our subsequent hardness and inapproximability results hold for any such more complex model as well. Furthermore, parameters q_{uv} and b_{uv} can be obtained from past data, as in [BBM12]; in our setting, we assume that such parameters have been obtained in advance.

Given a seed set S of subscribers, for every set of attributes F , we can obtain the total number of activated nodes after running several trials of the diffusion process from S [KKT03]. The expected number of activated nodes for a given seed set S and a selected set of attributes F is called *influence spread*, denoted as $\sigma(F|S)$, or, as S is fixed in our problem, just $\sigma(F)$. Thus, $\sigma(F)$ is the *expected spread* of the diffusion, which we can calculate using live-edge instances of the graph (i.e., instances

of activated-only edges [KKT03]) as:

$$\sigma(F) = \sum_X Prob[X] \cdot \sigma_X(F) \quad (4.2)$$

where $\sigma_X(F)$ is the influence spread in live-edge instance X .

4.3.2 Content-Aware Influence Maximization

We define the CAIM problem as follows:

Problem 1. Given a directed graph $G = (V, E)$, where each node v is associated with user attributes $F_v = \{f_v^1, f_v^2, \dots\}$ from a universe of eligible attributes Φ , a seed set of adopter nodes S , quantities q_{uv}, b_{uv} for each edge $e_{uv} \in E$, and a transition function $h_{uv}(F_v, F) = \min \left\{ \frac{1-b_{uv}}{q_{uv}}, |F_v \cap F| \right\}$ for edge probabilities, select a set of k attributes $F \subset \Phi$ that maximizes the spread $\sigma(F|S)$ of a diffusion process with content attributes F starting from S .

CAIM is a novel problem that aims to find out how one can maximize the benefits of a network promotion campaign with given points of departure. The motivation derives from the fact that, in the real world, brands want to exploit their own social network pages for marketing purposes. Instead of targeting the most influential initiators for a promotion, as in classical IM, one can judiciously invest in the creation of a post with lucrative content, under fixed initiators, guided by the content attributes provided by the CAIM solution. As promotions can be formed with a wide variety of content attributes, each possible attribute set F corresponds to a different probabilistic graph, on which we can compute the influence spread of the seed set S ; the attribute set F that achieves maximum spread constitutes the CAIM solution. We emphasize that, due to the drastic difference between classical IM and CAIM in the way influence spread is achieved, *the solutions to these two problems cannot be qualitatively compared against each other*.

4.4 Hardness and Inapproximability

We now show the hardness of the CAIM problem and study the properties of influence spread function $\sigma(F)$. To calculate $\sigma(F)$, we first calculate edge probabilities with respect to the selected content attributes F and then estimate the expected spread on the graph starting from the given set of subscribed nodes S .

Theorem 1. *The CAIM problem with the CAC model is NP-hard.*

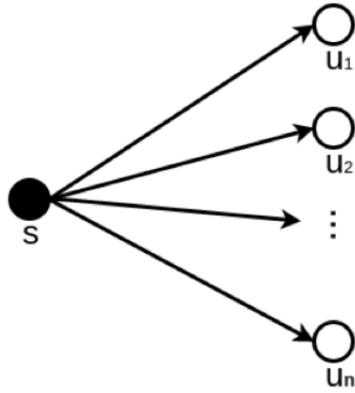


Figure 4.1: A graph instance demonstrating that the CAIM problem is NP-hard.

Proof. Consider an instance of the NP-complete *Set Cover* problem, defined by a collection of subsets S_1, S_2, \dots, S_m , a universe of elements $U = \{u_1, u_2, \dots, u_n\}$ and an integer k . We are asked whether there are k sets that will cover all elements in U . We show that *Set Cover* can be reduced to a *trivial* instance of CAIM as follows: We construct a bipartite graph with one activated node on the left side that connects to n nodes on the right side, as shown in Figure 4.1. We map each member u_i of universe U to a node on the right side and add an attribute f_j to set F_{u_i} if u_i belongs to subset S_j . We set $b_{uv} = 0$ and $q_{uv} = 1$ for all edges $(u, v) \in E$, i.e., a node v is influenced if at least one of its user attributes is selected. In this trivialized version of CAIM, the spread can be computed deterministically; there is no need for expected spread computations. Then, an algorithm that could optimally solve this trivial instance of CAIM, among others, would decide any instance of *Set Cover*: if we can target all nodes in the CAIM instance using k attributes, we can in effect cover all elements in U using k subsets in *Set Cover*. Otherwise, if the optimal spread in CAIM does not reach all nodes, it follows that there is no set of k subsets that covers all elements in *Set Cover*. Thus, by reduction from *Set Cover*, CAIM is at least as hard as any problem in NP. \square

By Theorem 1, there is no polynomial-time algorithm to find an optimal set of attributes F , unless P=NP. We now proceed to study the properties of the influence spread function $\sigma(F)$.

A function $\sigma(F)$ is *submodular* if it follows a *diminishing returns* rule: the marginal gain from adding an element to a set F is at most as high as the marginal gain from adding the same element to a subset of F . That is, $\sigma(F_1 \cup \{f\}) - \sigma(F_1) \geq \sigma(F_2 \cup \{f\}) - \sigma(F_2)$, where $F_1 \subset F_2 \subset \Phi$, for any $f \in \Phi$.

We call a transition function $h_{uv}(F_v, F)$ *monotonic* on F if, for subsets of attributes $F_1 \subset F_2 \subset \Phi$, it holds that $h_{uv}(F_v, F_1) \leq h_{uv}(F_v, F_2)$, for any node v . If the transition function is not monotonic, then the influence spread function is neither

monotonic, nor submodular, because selecting more attributes may reduce probabilities $p(u, v)$ and thereby reduce the total influence spread. We assume that attributes have nonnegative effects on users, rendering the transition function $h_{uv}(\cdot)$ monotonic: edge probabilities can only increase if we add attributes to F , i.e. $h_{uv}(F_v, F) \leq h_{uv}(F_v, F + \{f\})$ for any $f \in \Phi$ and $v \in G$; hence $\sigma(F)$ is monotonic. We now examine whether $\sigma(F)$ is also submodular. This turns out to not be the case, even for a monotonic and submodular transition function, as the next counterexample shows.

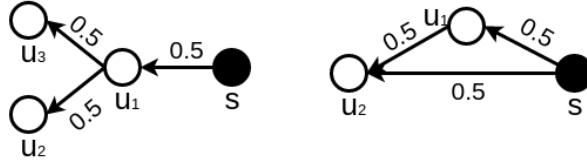


Figure 4.2: Increasing and decreasing marginal returns.

Example 2. Consider the graph on the left-hand side in Figure 4.2, with a universe of attributes $\Phi = \{A, B, C\}$, sets of preferred attributes for each node be $F_{v_1} = \{A\}$ and $F_{v_2} = F_{v_3} = \{A, B, C\}$, $b_{uv} = 0.5$, $q_{uv} = \frac{1}{2|F_v|}$ on all edges, and one active node s . Then, consider two subsets of attributes $F_1 = \emptyset$, $F_2 = \{B, C\}$, where $F_1 \subset F_2$, and a attribute $f = A \in \Phi \setminus F_2$. The achieved spreads for each attributes subset, and the respective marginal gains obtained after adding attribute f to subsets F_1 and F_2 , are calculated as follows. For subset attribute F_1 selected, we have:

$$p_{sv_1} = \frac{1}{2}, \quad p_{v_1v_2} = p_{v_1v_3} = \frac{1}{2}$$

$$\sigma(F_1) = \frac{1}{2} + 2 \cdot \frac{1}{4} = 1$$

whereas when $f = A$ is added to F_1 , we get:

$$p_{sv_1} = 1, \quad p_{v_1v_2} = p_{v_1v_3} = \frac{2}{3}$$

$$\sigma(F_1 + \{A\}) = 1 + 2 \cdot \frac{2}{3} = \frac{7}{3}$$

Hence $\Delta_1 = \sigma(F_1 + \{A\}) - \sigma(F_1) = \frac{7}{3} - 1 = \frac{4}{3}$. Similarly, for F_2 selected, we have:

$$p_{sv_1} = \frac{1}{2}, \quad p_{v_1v_2} = p_{v_1v_3} = \frac{5}{6}$$

$$\sigma(F_2) = \frac{1}{2} + 2 \cdot \frac{5}{12} = \frac{4}{3}$$

while when $f = A$ is added to F_2 , we get:

$$p_{sv_1} = 1, \quad p_{v_1v_2} = p_{v_1v_3} = 1$$

$$\sigma(F_2 + \{A\}) = 3$$

Hence $\Delta_2 = \sigma(F_2 + \{A\}) - \sigma(F_2) = 3 - \frac{4}{3} = \frac{5}{3}$. Since $\Delta_2 > \Delta_1$, the submodularity of $\sigma(F)$ does not hold.

4.4. Hardness and Inapproximability

Given this negative result, the influence function $\sigma(F)$ might have an *increasing returns* property (supermodularity), whereby it would hold that $\sigma(F_1 \cup \{f\}) - \sigma(F_1) \leq \sigma(F_2 \cup \{f\}) - \sigma(F_2)$, for $F_1 \subset F_2 \subset \Phi$ and any attribute $f \in \Phi$. The following counterexample shows that this property does not hold either.

Example 3. Consider the right graph in Figure 4.2, with a universe of attributes $\Phi = \{A, B\}$, sets of preferred attributes per node $F_{v_1} = \{A, B\}$ and $F_{v_2} = \{A\}$, $b_{uv} = 0.5$ and $q_{uv} = \frac{1}{2|F_v|}$ on all edges, and one active node s . Consider two subsets of attributes $F_1 = \emptyset$ and $F_2 = \{B\}$. Then, for subset attribute F_1 selected, we have:

$$p_{sv_1} = \frac{1}{2}, \quad p_{sv_2} = p_{v_1v_2} = \frac{1}{2}$$

$$\sigma(F_1) = \frac{1}{2} + \left(1 - \left(1 - \frac{1}{4}\right)\frac{1}{2}\right) = \frac{9}{8}$$

whereas when $f = A$ is added to F_1 we get:

$$p_{sv_1} = \frac{3}{4}, \quad p_{sv_2} = p_{v_1v_2} = 1$$

$$\sigma(F_1 + \{A\}) = \frac{3}{4} + 1 = \frac{7}{4}$$

Hence $\Delta_1 = \sigma(F_1 + \{A\}) - \sigma(F_1) = \frac{7}{4} - \frac{9}{8} = \frac{5}{8}$. Similarly, for F_2 selected, we have:

$$p_{sv_1} = \frac{3}{4}, \quad p_{sv_2} = p_{v_1v_2} = \frac{1}{2}$$

$$\sigma(F_2) = \frac{3}{4} + \left(1 - \left(1 - \frac{3}{8}\right)\frac{1}{2}\right) = \frac{23}{16}$$

while when $f = A$ is added to F_2 we get:

$$p_{sv_1} = 1, \quad p_{sv_2} = p_{v_1v_2} = 1$$

$$\sigma(F_2 + \{A\}) = 2$$

Hence $\Delta_2 = \sigma(F_2 + \{A\}) - \sigma(F_2) = 2 - \frac{23}{16} = \frac{9}{16}$. Since $\Delta_1 > \Delta_2$, the influence function $\sigma(F)$ is not supermodular either.

Eventually, we have established the following:

Theorem 2. The spread function $\sigma(F)$ with a probability transition function $h_{uv}(F_v, F) = \min\left\{\frac{1-b_{uv}}{q_{uv}}, |F_v \cap F|\right\}$ is neither submodular nor supermodular.

By Theorem 2, it follows that we cannot use a greedy algorithm with an approximation guarantee based on submodularity, as in [KKT03]. Moreover, in the following we show that it is NP-hard to approximate the optimal solution to CAIM.

Theorem 3. It is NP-hard to approximate the optimal solution to the CAIM problem with the Content-Aware Cascade model within a factor $n^{1-\varepsilon}$ for any $\varepsilon > 0$.

Proof. Consider an instance of the *Set Cover* problem, in which we need to decide whether we can cover all elements of a universe $U = \{u_1, u_2, \dots, u_n\}$ by selecting at most k subsets out of a collection of $S_1, S_2, \dots, S_m \subset U$.

We then construct a graph G for the CAIM problem with a single subscriber node s and nodes u_1, u_2, \dots, u_n corresponding to elements in U , connected so that u_{i-1} points towards u_i for all $i = 2 \dots n$, and s is connected to u_1 , and, for every subset S_j an element u_i belongs to, we add a attribute f_j to the preferred attributes of u_i . Next, for some integer c we add $\eta = n^c - n - 1$ more nodes x_1, x_2, \dots, x_η such that u_n has outgoing edges to them and each x_i has the same preferred attributes as u_n . Graph G , shown in Figure 4.3, has $N = n^c$ nodes. We set $b_{uv} = 0$ and $q_{uv} = 1$ for all edges, so that an edge becomes active if at least one of the attributes associated with its target node is selected. Then, if it is possible to select k subsets that cover all elements of universe U , we can also have $N = n^c$ activated nodes. Conversely, if there is no selection of k subsets that covers all U , then there is at least one node u_i that does not get activated, precluding influence spread to nodes x_1, x_2, \dots, x_η . We can then only target at most n out of n^c nodes, a fraction of $n^{1-c} = N^{\frac{1}{c}-1}$. Thus, if we had a polynomial-time algorithm that approximated the optimal solution to CAIM within a factor of $N^{1-\varepsilon}$ for any $\varepsilon > 0$, then it would suffice to set $c = \lceil \frac{1}{\varepsilon} \rceil$ and use that algorithm so as to decisively distinguish between a case that accepts a solution activating all N nodes and one that does not, and thereby also decide *Set Cover*. Thus, by reduction from *Set Cover*, we have shown that it is NP-hard to approximate the optimal solution to CAIM within a reasonable factor. \square

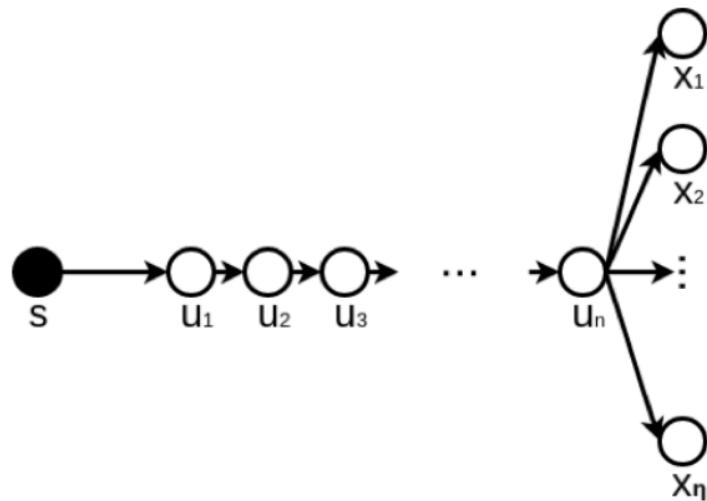


Figure 4.3: A graph instance demonstrating that it is NP-hard to approximate the optimal solution to the CAIM problem.

4.5 The Explore-Update Algorithm

As it is NP-hard to approximate the CAIM solution within a factor of $n^{1-\varepsilon}$ with the Content-Aware Cascade model, we proceed to design heuristic solutions therefor. We structure our exposition as follows: we first present a simple, yet time-consuming greedy heuristic; then, through a sequence of simplifying assumptions, we will generate a much more efficient algorithm called Explore-Update.

Our first proposal is a baseline greedy algorithm that selects the attribute of highest marginal gain to add at each iteration, shown in Algorithm Greedy. This is an adaptation of the *Local Update* algorithm in [BB14] to our problem. Intuitively, it is reasonable to greedily select the locally best attribute in each iteration, especially for small values of k . This kind of algorithm has been shown to achieve better quality than others in classical Influence Maximization [CWW10, CLC13, LKG⁺07].

Algorithm 1: Greedy(G, S, k)

```

1  $F = \emptyset;$ 
2 while  $|F| < k$  do
3   | for every  $f \in \Phi \setminus F$  do
4     |   | calculate  $\sigma(F + \{f\})$  using Monte Carlo simulations
5     |   |  $F = F \cup \operatorname{argmax}_f \{\sigma(F + \{f\})\}$ 
6 return  $F$ 
```

Though simple and effective, Greedy is inefficient due to its calculation of influence spread by MC simulations. In a manner reminiscent of [CWW10], we can improve efficiency by considering *maximum influence paths* between nodes and the seed set. We call a path $P_{max} = \langle u = u_0, u_1, u_2, \dots, v = u_m \rangle$ between vertices $u \in S$ and $v \in G$ *maximum influence path* (MIP) if this path is the most probable among all paths between u and v : $P_{max} = \operatorname{argmax}_P \prod_{i=0}^{m-1} \operatorname{prob}(u_i, u_{i+1})$. Under the *simplifying assumption* that influence is propagated only through MIPs, we can estimate influence spread in polynomial time as follows: For a threshold θ and a node v , we build a tree structure called *in-arborescence* $A_{in}(v)$, which includes all MIPs of probability higher than θ from any node to v : $A_{in}(v) = \{\text{MIP}(u, v) \mid \operatorname{prob}(\text{MIP}(u, v)) > \theta, u \in G\}$. Then, given a node u , the seed set S , and an arborescence $A_{in}(v)$, Algorithm calculateAP recursively estimates the probability that u is activated in $A_{in}(v)$, i.e., its *activation probability* $ap(u, A_{in}(v))$.

Based on these calculations, for all nodes $u \in G$, we can calculate the influence spread $\sigma(F)$ as follows:

$$\sigma(F) = \sum_{u \in G} ap(u, A_{in}(u)) \tag{4.3}$$

Algorithm 2: calculateAP($u, A_{in}(v), S$)

```

1 if  $u \in S$  then
2   |    $ap(u, A_{in}(v)) = 1$ 
3 else if  $u$  has no in-neighbors in  $A_{in}(v)$  then
4   |    $ap(u, A_{in}(v)) = 0$ 
5 else
6   |    $ap(u, A_{in}(v)) = 1 - \prod_{\omega \in N_{in}(u)} (1 - ap(\omega, A_{in}(v))prob(\omega, u))$ 
7 return  $ap(u, A_{in}(v))$ 

```

We can then employ Equation 4.3 so as to estimate influence spread in Algorithm Greedy, in lieu of MC simulations, deriving Algorithm Arb; at each iteration, we compute the in-arborescence of node u for a given threshold θ by converting each probability p_e on an edge e to $-\log p_e$ and employing an efficient implementation of Dijkstra's algorithm. If computing an arborescence takes time t , then Lines 4-6 take nt and the total time is $O(k|\Phi|nt)$.

Algorithm 3: Arb(G, S, θ, k)

```

1  $F = \emptyset;$ 
2 while  $|F| < k$  do
3   |   for every  $f \in \Phi \setminus F$  do
4     |     |   for every  $u \in G$  do
5       |       |     |   compute  $A_{in}(u)$  with threshold  $\theta$ 
6       |       |     |    $ap(u, A_{in}(u)) = \text{calculateAP}(u, A_{in}(u), S)$ 
7       |       |   calculate  $\sigma(F + f) = \sum_{u \in G} ap(u, A_{in}(u))$ 
8     |   |    $F = F \cup \arg\max_f \{\sigma(F + \{f\})\}$ 
9 return  $F$ 

```

We further reduce the runtime of Algorithm Arb by eschewing redundant iterations of the loops over nodes u and attributes f . First, we limit the calculation of in-arborescences and activation probabilities only to nodes whose in-arborescence under threshold θ reaches at least one node in S ; only such nodes can yield non-zero estimated activation probability. To find out these nodes, we compute the out-arborescence of all nodes in S , $A_{out}(S)$, consisting of all MIPs of probability higher than θ from a node $v \in S$ to other nodes in G . Nodes in $A_{out}(S)$ yield non-zero activation probability estimates. Yet, the set of paths in $A_{out}(S)$ may contain directed loops, hence we cannot apply a recursive algorithm like Algorithm calculateAP directly on $A_{out}(S)$; we still need to obtain the in-arborescence $A_{in}(u)$ of each $u \in A_{out}(S)$; we do so while building $A_{out}(S)$, by adding $MIP(v, u)$ to $A_{in}(u)$ for each $u \in A_{out}(v)$. Algorithm Explore illustrates this explore process.

Algorithm 4: Explore(G, F, S, θ)

```

1  $A_{out}(S) = \emptyset$ 
2  $A_{in}(u) = \emptyset$  for every  $u$  in  $G$ 
3 for every  $v \in S$  do
4   | compute  $A_{out}(v)$  for given  $\theta$  and  $F$ 
5   | update  $A_{in}(u)$  for each  $u \in A_{out}(v)$ 
6 return  $\{A_{in}(u) \neq \emptyset\}$  for  $u \in G$ 

```

Then we can calculate influence spread $\sigma(F)$ using the union of such in-arborescences, A_{in} , by Algorithm Update.

Algorithm 5: Update(A_{in}, S)

```

1 for every  $u \in A_{in}$  do
2   |  $ap(u) = \text{CalculateAP}(u, A_{in}(u), S)$ 
3   |  $\sigma(F) = \sum_{u \in A_{in}} ap(u)$ 
4 return  $\sigma(F)$ 

```

Second, we limit the calculation of marginal gain in Algorithm Arb only to those attributes that can affect the influence spread. We call an edge in G *participating*, if at least one of its endpoints are in $A_{out}(S)$. Figure 4.4 presents a graph for a seed set S (and selected attributes set F) in the green area; the yellow area includes nodes in $A_{out}(S)$; the set of participating edges Π is shown in solid and dotted lines; dotted edges have only one endpoint in $A_{out}(S)$; non-participating edges are shown in dashed lines, in the gray area.

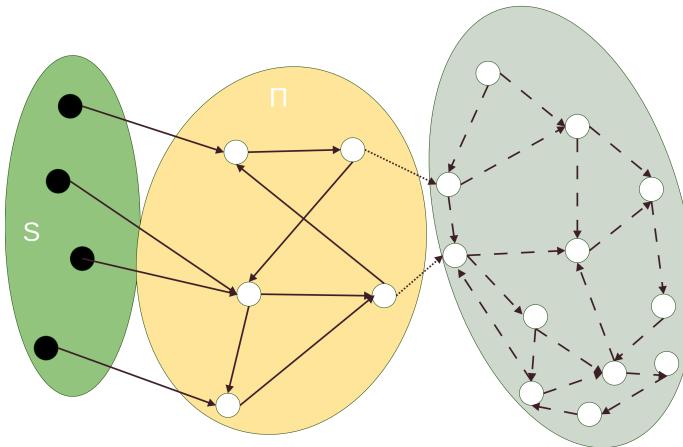


Figure 4.4: Participating and non-participating edges.

Non-participating edges cannot increase influence spread, regardless whether their probability is increased; only participating edges have such potential. We limit the attributes Algorithm **Arb** considers based on this observation. Let $E(f)$ be the set of edges that include attribute f among their preferred attributes, hence their probability is affected when adding f to F . Then, at any iteration, if none of the edges in $E(f)$ is a participating edge, i.e., $E(f) \cap \Pi = \emptyset$, then attribute f need not be examined as a candidate to be added to F ; it bears no effect to influence function $\sigma(F + \{f\})$.

Putting together our enhancements to Algorithm **Arb**, we design the polynomial-time Algorithm **Explore-Update**. In a nutshell, at each iteration, **Explore-Update** selects the hitherto unselected attribute f affecting participating edges that brings about the largest increase of influence spread, using the **Explore** procedure for calculating in-arborescences and the **Update** procedure for calculating influence spread, while updating the set of participating edges Π at each iteration and using it to determine which attributes need to be examined at the next iteration.

Algorithm 6: $\text{Explore-Update}(G, S, k, \theta)$

```

1  $F = \emptyset$ 
2  $A_{in} = \text{Explore}(G, F, S, \theta)$ 
3  $\Pi = \{(u, v) \in G \mid u \in A_{in} \text{ or } v \in A_{in}\}$ 
4 while  $|F| < k$  do
5   for  $f \in \Phi \setminus F$  do
6     if  $E(f) \cap \Pi \neq \emptyset$  then
7        $A_{in} = \text{Explore}(G, F + \{f\}, S, \theta)$ 
8        $\Pi_f = \{(u, v) \in G \mid u \in A_{in} \text{ or } v \in A_{in}\}$ 
9        $\sigma(F + \{f\}) = \text{Update}(A_{in}, S)$ 
10       $f_{max} = \text{argmax}_f \{\sigma(F + \{f\})\}$ 
11       $F = F \cup f_{max}$ 
12       $\Pi = \Pi_{f_{max}}$ 
13 return  $F$ 

```

Let the time complexity to calculate an out-arborescence for node in S be $t_{out\theta}$, then the **Explore** procedure takes $|S|t_{out\theta}$ and the **Update** procedure takes $O(n_{in\theta}n_{out\theta})$ time, where $n_{in\theta}$ is the number of nodes in in-arborescences, and $n_{out\theta}$ is the number of nodes in out-arborescence of S . Therefore, if we perform κ calculations of A_{in} per iteration, the total runtime is $O(k\kappa(|S|t_{out\theta} + n_{in\theta}n_{out\theta}))$. In effect, the **Explore-Update** algorithm is expected to perform well when the size of arborescences is small, and the number of updates κ per iteration is smaller than $|\Phi|$. As propagation probabilities on edges are usually small in real networks, the size of arborescences is indeed expected to be small. The number of updates depends on the structure of the network. In a large-diameter network where multiple hops are required to reach most nodes from S via a MIP, there is a good chance to reduce the number of computations significantly.

4.6 Experimental Study

This section presents an experimental study on the Greedy and Explore-Update algorithms we have introduced. All experiments were run on a 32GB Intel Core i5-2450M CPU machine @ 2.50GHz, while algorithms were implemented in C++.

As there is no previous work on the CAIM problem, we compare to basic baselines. Still, as we discussed, the previous work that comes closest to our problem is that by Barbieri and Bonchi [BB14]; yet that work solves primarily the problem of selecting a set of seed nodes, and secondarily a set of product attributes, so as to maximize product influence in a network. The best-performing algorithm for updating an attribute set in [BB14], *Local Update*, performs one addition or removal of an attribute to/from the current attribute set at each iteration; in effect, our *Greedy* algorithm can be considered as an adaptation of *Local Update* to our problem, where only additions of attributes are needed. Therefore, to the extent that a comparison to [BB14] is possible, we conduct it via the comparison to the *Greedy* algorithm itself. Another method for updating an attribute set proposed in [BB14], *Generic Update*, is a hard-to-tune genetic algorithm, which may lead to an unpredictable number of output attributes. Besides, as the experimental study in [BB14] shows, *Genetic Update* offers no qualitative advantage while it is much slower than *Local Update*, which is already by far the most time-consuming algorithm in our study. Therefore, we do not consider a genetic algorithm in our experimental study.

Diffusion models. In the Content-Aware Cascade model the probability on edge (u, v) is a linear function of product and base probabilities q_{uv} and b_{uv} . To assign these probabilities we use two techniques prevalent in previous work [CWW10].

- **Weighted Cascade model:** probability $\frac{1}{d_v}$ is assigned to edge (u, v) , where d_v is the in-degree of node v . We use this model for the sake of compatibility with previous works, even while it may fit less to our problem setting.
- **Multivalency model:** the probability for edge (u, v) is drawn uniformly at random from a set of probabilities. We choose that set to be $[0.02, 0.04, 0.08]$.

Further, we calculate b_{uv} for every edge (u, v) , and set $q_{uv} = \frac{b_{uv}}{|F_v|}$.

Algorithms. We compare the Explore-Update algorithm under different threshold θ values to three other algorithms:

- **Greedy:** This is Algorithm Greedy in this chapter, which is effectively an adaptation of *Local Update*, the best algorithm in [BB14]. A similar algorithm has been used extensively in the context of the Influence Maximization problem, and always demonstrated top performance in terms of spread, while being slower than other heuristics [CLC13]; it requires specifying the number of Monte-Carlo simulations to calculate influence spread, as we do in the following.

Table 4.1: Data characteristics.

Dataset	Gnutella	VK
Nodes	10,876	7,420
Edges	39,994	57,638
Average Clustering Coefficient	0.0062	0.28
Number of Triangles	934	168,284
Diameter	9	16
Attributes/Seed sets	151	3,882
Default Seed Size	34	15

- **Top-Nodes:** This algorithm measures each attribute’s frequency among node preferences and selects the k most frequent ones.
- **Top-Edges:** This algorithm assigns to each edge $e = (u, v)$ the attribute preferences of node v , F_v , and selects the k most frequent attributes across all edges.
- **Brute-Force:** This algorithm finds all possible sets of attributes of size k , computes each one’s influence spread using Monte-Carlo simulations, and opts for the best. Because the solution space is exponential, we use this method on reduced datasets.

Datasets. We run experiments in two real-world networks. The first network is a peer-to-peer file sharing directed network Gnutella¹, where nodes represent hosts and edges represent connections between the Gnutella hosts. Our second network is extracted by crawling the social network VK²; nodes are users and edges are friendships among them. Statistics are presented in Table 4.1.

Attribute assignment and seed selection. We utilize one general and one ad-hoc method for attribute preference assignment. In Gnutella, to assign an attribute preferences set F_v to node v , we find the block partitioning that minimizes the description length of the network by stochastic blockmodel ensemble; this technique is used to discover the block structure of empirical networks and results to block memberships for each node [KN11, Pei15]. We allow nodes to have overlapping memberships to different blocks. Each block β_i is associated with a distinct attribute f_i . The attribute preference set of a node v_j , F_{v_j} is the set of attributes of the blocks v_j belongs to. The returned partitioning consists of 151 blocks; the default seed set S is one of the blocks, of size 34. For VK, the data comes along with annotations of *groups* and *pages*, which allow us to derive both node attributes and seed sets. A *group* or *page* is a community of users that share content with each other and communicate about a topic of interest (e.g., football clubs or TV series). We use these group memberships to derive both node attributes and seed sets, consistently to our motivation. There are 3882 such groups; the default seed size is 15. Unless otherwise indicated, in our experiments we use the default seeds.

¹<https://snap.stanford.edu/data/p2p-Gnutella04.html>
²<https://vk.com/>

4.6. Experimental Study

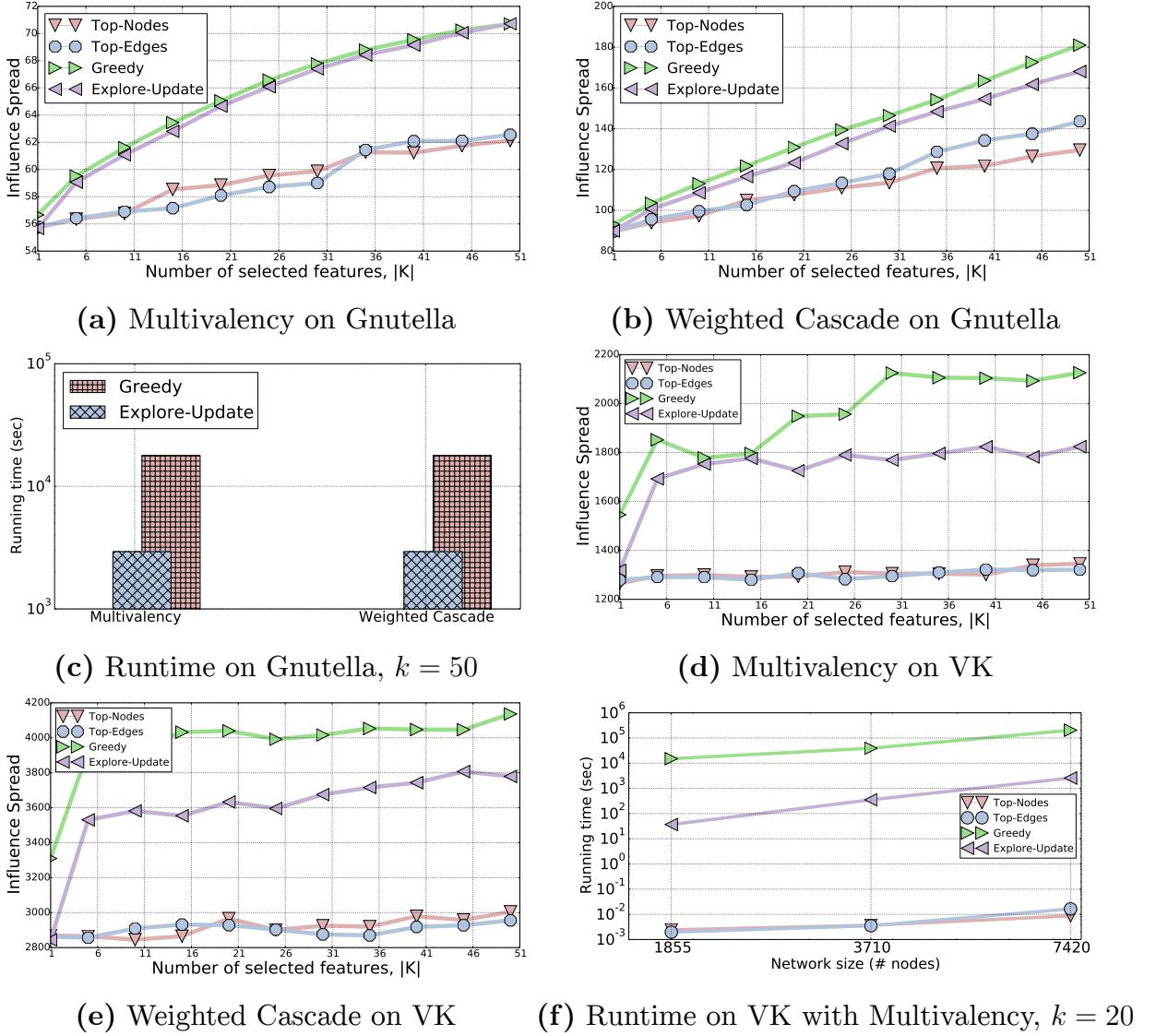


Figure 4.5: Influence spread and runtime results.

4.6.1 Influence spread

Figures 4.5a and 4.5b present our results on competing algorithms' influence spread³ on the Gnutella network, varying number of selected attributes k from 1 to 51. We used 10000 MC simulations for Greedy, and $\theta = 1/320$ for Explore-Update. We observe that Explore-Update arrives just 1% and 5% below the performance of Greedy with the Multivalency and Weighted Cascade model, respectively. On the other hand, the Top-Edges and Top-Nodes algorithms reach only 88% and 85% of the spread of Explore-Update. Figures 4.5d and 4.5e present influence spread in VK network. Now Greedy used with just 500 Monte-Carlo simulations comfortably achieves 15%, 37%,

³We use 10000 Monte-Carlo simulations to compute the final spread of all solutions.

and 48% higher spread than Explore-Update with $\theta = 1/40$, Top-Nodes, and Top-Edges, respectively, in MV model. The picture is similar with the WC model, where Greedy achieves spread 9%, 38%, 40% higher than Explore-Update, Top-Nodes, and Top-Edges. Overall, our results confirm that Explore-Update achieves high influence spread for networks where the local neighborhood of the seed set has structure amenable to long distance arborescences.

4.6.2 Runtime

We now compare algorithms in terms of runtime. Figure 4.5c presents the results with Gnutella for $k = 50$; Explore-Update ($\theta = 1/320$) runs an order of magnitude faster than Greedy (10000 simulations); Top-Edges and Top-Nodes output a selected set in less than a second, hence we do not include them. Next, we investigate how the algorithms scale with increasing network size. We extract subnetworks of VK consisting of 1855, 3710, and 7420 nodes of the original network (i.e., 1/4, 1/2, and full network) and proportional edge density to the full network. In all cases, we compute the runtime on a seed set S of size 15, with the Multivalency model for $k = 20$, for Greedy (10000 simulations), Explore-Update ($\theta = 1/40$), and the Top-Edges and Top-Nodes heuristics. Figure 4.5f shows that runtime scales linearly in network size in all cases. Moreover, we ascertain that while Explore-Update fares no better than Greedy in terms of influence spread, it is much faster.

4.6.3 Effect of Seed Size

We now test the performance of Explore-Update for different sizes of the seed set S . We select different seed sets from size 21 (minimal size for the current block partition) to 101 with step 10 on Gnutella. Figures 4.6a and 4.6b present the influence spread for Explore-Update and Top-Edges for $k = 50$, as well as the runtime of Explore-Update, whereas Greedy is orders of magnitude slower for this setup, and Top-Nodes performs worse than Top-Edges. We note that Explore-Update always achieves better influence spread than Top-Edges. Interestingly, influence spread and runtime do not always grow with $|S|$. This is explicable by the fact that different seed sets induce different local structures.

4.6.4 Effect of θ

Next, we study the effect of the θ threshold, which controls the size of arborescences and thereby the influence spread achievable from seed set S . Figure 4.6c presents the influence spread and runtime with the Gnutella network for θ in $\{\frac{1}{10}, \frac{1}{20}, \dots, \frac{1}{320}\}$, with the WC model for $k = 50$. The runtime of Explore-Update grows linearly in the

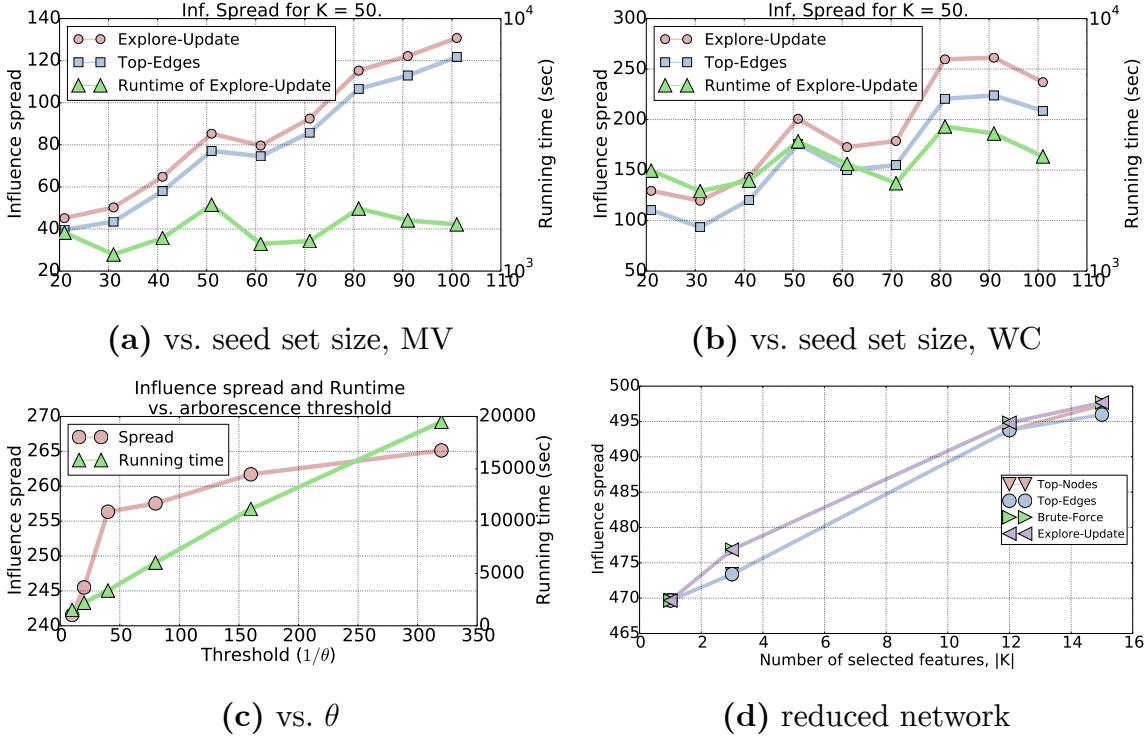


Figure 4.6: Influence spread and runtime vs. seed set size and θ on Gnutella in (a), (b), and (c). Influence spread on reduced network in (d).

inverse threshold θ , while influence spread grows logarithmically in it. A good tradeoff between quality and runtime is found at the knee point in the influence spread curve for $\theta = \frac{1}{40}$.

4.6.5 Comparison to the Optimal Solution

By Theorem 3, we proved it is NP-hard to approximate the optimal solution to CAIM. Now, we compare the results of heuristics to the optimal solution obtained by brute force; we reduce the total number of attributes to 16 and use a reduced Gnutella network by selecting 2K nodes, yielding similar degree distribution properties to the original. Figure 4.6d shows the influence spread results, with the Multivalency model, for a random seed set of size 10. Remarkably, Explore-Update finds the optimal set of attributes with varying k .

Next, we select $k = 10$, yielding $\binom{16}{10} = 8008$ possible attribute sets, and calculate, with a new random seed set of size 428, the rank of each algorithm's solution among all possible attribute sets: for each attribute set, we compute its influence spread using 10000 MC simulations; we sort sets by their spread values, and identify the rank of the solution returned by each heuristic. Table 4.2 presents those ranks. Explore-Update selects the optimal solution, while Greedy with 500 parsimonious MC simulations

Table 4.2: Algorithm ranking with regards to optimal solution.

Algorithm	Rank	Spread
Explore-Update	1	34.592
Greedy	5	34.114
Top-Edges	113	33.592
Top-Nodes	113	33.592
...		
—	8008	27.82

yields the fifth best attribute set. The selected attribute sets differ from each other in 2 out of 10 attributes. We obtained similar results for other values of k , with Explore-Update always returning the optimal attribute set.

4.6.6 Real-World Examples

Last, we looked into the actual results - seed sets and selected attribute sets of our experiments, with special attention to the VK data set with the multivalency model, and inspected our results. One interesting observation was that those attributes that are liked by seed set users were rarely among the ones selected in the final solution; this fact indicates that our problem makes good practical sense, while a straightforward naive solution of sticking to what is liked by seed nodes does not yield good results. Nevertheless, selected attributes exhibited a remote, yet unpredictable, resemblance to the attributes liked by seed set nodes. For example, with a group titled “La vie et l’amour” as seed, the selected attributes in our VK network sample included “Home Comfort | Design | Interior Design | Style”. With “Psychology of Relations” as seed, the selected attribute set included “Philosophy of Life”. Such analogies between seed set and selected attributes, while retrospectively intuitive, would not be derived otherwise; they depend on the way nodes of diverse interests interact within the overall network structure. Such results vindicate our problem motivation.

We also checked how result sets change when we vary k . For example, we select 100 out of 431,374 subscribers of “Esoterica YOGA MEDITATION” as seed set. With $k = 3$, the selected attributes are {“MODA”, “La vie et l’amour”, “Blog for Men”}. As “Esoterica YOGA MEDITATION” targets primarily women, results such as “MODA” and “La vie et l’amour” are unsurprising. Nevertheless, interestingly, both Explore-Update and Greedy also return “Blog for Men” as a selected attribute, whereas the simple Top-Nodes and Top-Edges heuristics do not. This result shows that our algorithm can select non-trivial attributes.

4.7 Conclusion

In this chapter, we proposed the problem of content-aware influence maximization (CAIM). The goal is to select k attributes that characterize a propagated meme's content, such that its spread across a network from fixed points of departure is maximized, whereby different attribute sets yield different propagation probabilities across network edges. To our knowledge, there is no previous work on this problem. We formulated a content-aware cascade model and showed that the problem is NP-hard and inapproximable, while the influence function is neither submodular, nor supermodular. We developed an efficient algorithm for CAIM using bounded local arborescences to calculate influence spread. Our experimental study demonstrates that this **Explore-Update** algorithm selects topic sets that achieve high spread and is orders of magnitude faster than a conventional **Greedy** solution resembling algorithms developed for related problems. We also provide evidence that **Explore-Update** can achieve the optimal solution when the number of selected topics is small.

Chapter 5

Adaptive Content-Aware Influence Maximization through Online Learning To Rank

How can we adapt the composition of a post by a *brand* agent over a series of rounds to make it more appealing in a social network? Techniques that progressively learn how to make a fixed post more influential over several rounds have been studied in the context of the *Influence Maximization* (IM) problem, which seeks a set of *seed users* that maximize a post's influence. However, to our knowledge, there is no work on progressively learning how a post's *features* affect its influence. In this chapter, we propose and study the problem of *Adaptive Content-Aware Influence Maximization* (ACAIM), which calls to find k features to form a post in each round so as to maximize the cumulative influence of those posts in a social network over all rounds. We solve ACAIM by applying, for the first time, an *Online Learning To Rank* (OLR) framework to IM purposes. We introduce the *Content-Aware TopRank Influence Dissemination* (CATRID) model, which expresses how posts disseminate in a social network using click probabilities and post visibility criteria, and develop a simulator that runs CATRID via a training-testing scheme based on VKontakte (VK) posts, so as to realistically represent the learning environment. We deploy three distinct learners that solve ACAIM in an online manner. We experimentally ascertain the practical suitability of our solutions via exhaustive experiments on diverse VK datasets, while the learner that achieves best performance is one that leverages user feedback along with a *transitive* data structure that determines the relative importance of post features to users.

5.1 Introduction

Consider stakeholders (henceforward, *brands*) that maintain social network pages¹ for advertising purposes and social network users who choose to *follow* pages they are interested in. After a user follows a page, she gets notified of relative posts and the page is added to her set of *user features* [ITTK17]. A brand may aim to build a *post* around a set of appealing *content features*, so that it spreads well in the network when shared with subscribers. This problem is called *Content-Aware Influence Maximization* (CAIM) [ITTK17]. Contrary to the problem of *Influence Maximization* (IM) [KKT03, LFWT18], CAIM seeks a set of features of a post propagated from a known set of initial adopters, rather than a set of initial adopters to promote a known post (or idea, belief, legend, behavior, etc.). Other works also utilize content to achieve influence in a social network [AW11, BBM12, BB14, CFL⁺15, LZT15], yet differ in terms of their targets, content type, and initial adopters.

A variation on IM is *adaptive* IM, whereby a brand diffuses a given post in many rounds and selects initial adapters in each round utilizing previous feedback, aiming to maximize the cumulative influence over all rounds. Some works study this problem under known network parameters [GK11, CK13, YT17, SHYC18, HTH⁺20], while others solve it aiming to simultaneously learn latent parameters [LMM⁺15, CWYW16, VLS16, VKW⁺17, WKVV17, WLW⁺19]. To our knowledge, no prior work addresses multi-round CAIM.

In this chapter, we study *Adaptive Content-Aware Influence Maximization* (ACAIM), which seeks content of a sequence of posts over several rounds, each first reaching a brand’s subscriber set, aiming to maximize cumulative influence. We apply the TopRank Model (TRM), used to solve the *Online Learning To Rank* (OLR) problem [LKLS18], to model click probabilities per user and define a novel propagation model, *Content-Aware TopRank Influence Dissemination* (CATRID). We presume that users respond to specific post features; once a sufficient number of a user’s friends likes a post, that post becomes visible to the user; user clicks on features of a propagated post denote the reasons for a user to like the post. We deploy a *simulator* that runs CATRID based on VK² data to obtain realistic feedback per round.

ACAIM addresses a real-world need; brands share everyday posts in their pages and wish to have appealing content. For instance, we solved ACAIM for a brand on “Olympic Games”; after feedback rounds of various content we found that the three most influential features for that brand are the VK pages “Anonymous revelation stories”, “Show business news”, and “Funny & ridiculous jokes”. These pages serve as

¹We consider that each *feature* corresponds to a specific social network page.

²<https://vk.com/>

a repository for an advertiser to check posts shared therein and get ideas on how to form posts combining “Olympic Games” with the discovered features. The abundance of repository posts helps the advertiser create a different post each time; our simulator considers post uniqueness.

We summarize our main contributions as follows:

- **Problem.** We introduce the ACAIM problem, which seeks influential features in an adaptive setting, and solve it via Online Learning to Rank with the CATRID propagation model.
- **Simulator.** We deploy a simulator that runs CATRID to generate realistic feedback guided by real VK posts.
- **Learners.** We develop three learners for ACAIM: `RANDOM`, a baseline with non-trivial application semantics; *TopRank IM Classic* (TRIM_C), integrating `TOPRANK` [LKLS18]; and *TopRank IM Eliminator* (TRIM_E), which uses the same TRM click model as TRIM_C , but leverages clicks differently: TRIM_E eliminates features over rounds during *exploration* and further ranks them during *exploitation*, with impressive results.
- **Experiments.** We conduct a thorough experimental study to assess our solutions on multiple brands and VK datasets.

5.2 Problem Statement

We define ACAIM on a network $G = (V, E)$, where V is a set of nodes (users) and $E \subseteq V \times V$ is a set of directed edges (user relations).

5.2.1 TRM Click Model in Social Networks

Let L be the set of *features*, k the number of ranking positions with $|L| \geq k$, where each user $v \in V$ is associated with a feature set F_v capturing user preferences [ITTK17]. The set of $\frac{|L|!}{(|L|-k)!}$ k -permutations of L , \mathcal{A} , is a set of rankings, each corresponding to a *post* A_t , and defines the action space of a *learner* aiming to maximize the expected number of *likes* [ITTK17] over n rounds. By the learning process of [LKLS18], in each round t , the learner chooses a post $A_t \in \mathcal{A}$ and observes binary random variables $C_{t1}^v, \dots, C_{tk}^v$ for each user $v \in V$, where $C_{tf}^v = 1$ if v clicked on feature f appearing in position i_f of A_t . Given feature f in position i_f in post A_t , the probability that user v clicks on f by the TRM click model [LKLS18] is:

$$v(A_t, i_f) = \mathbb{P}(C_{tf}^v = 1 | A_t) \quad (5.1)$$

We apply TRM as a position-based click model, where each A_t and each set F_v is a weighted feature vector of size $|L|$; the weight of feature f in A_t , $A_t(f)$, expresses the portion degree of f in A_t , while the weight of f in F_v , $F_v(f)$, expresses the preference degree of f in F_v ; the sum of weights over features is equal to 1 in both A_t and F_v , while A_t has exactly k non-zero weights. We define:

$$\mathbb{P}(C_{tf}^v = 1 \mid A_t) = A_t(f) \cdot F_v(f) \quad (5.2)$$

By Equation 5.2, a user v clicks on feature f in A_t with probability equal to the inner vector product $A_t(f) \cdot F_v(f)$, which corresponds to the sum operand of a PBM click model [RDR07], where $A_t(f)$ holds *position-dependent examination* probabilities and $F_v(f)$ *item-dependent attraction* probabilities. We contend that if user v clicks on a feature f of A_t (i.e., if $C_{tf}^v = 1$), she is influenced by $A_t(f) \cdot F_v(f)$ value, otherwise (i.e., if $C_{tf}^v = 0$), f does not influence v . We argue that v ends up to *like* post A_t if she is sufficiently influenced by it. Semantically, each click of v on a feature f of A_t is interpreted as a positive impression of f on v that contributes to v liking A_t , and hence making it visible to her relations in G . For a post A_t to be visible to v , several friends of v should like it. Otherwise, v does not see A_t , hence she is not influenced by it. Thus, if a feature f of A_t is clicked by many users, it opens up many influence paths for A_t . We rewrite Equation 5.2 as:

$$\mathbb{P}(C_{tf}^v = 1 \mid A_t \text{ is visible to } v) = A_t(f) \cdot F_v(f) \quad (5.3)$$

We integrate TRM (henceforward, depicted by Equation 5.3) in the CATRID propagation model, which we describe next.

5.2.2 CATRID Propagation Model

Parameters. The CATRID propagation model has four parameters: (i) post feature vector, A_t , (ii) user feature vector, F_v , (iii) user like threshold, lt_v , and (iv) post visibility fraction, vis_v . A_t and F_v define a user's TRM click model (Equation 5.3); lt_v defines how much a user v should be influenced to like A_t ; and vis_v defines how many in-degree friends of v should like A_t to make it visible to v .

Execution. By CATRID, a post A_t propagates in discrete time steps. In the first step, all subscribers of the posting brand see A_t , apply TRM to possibly click on its features, and are influenced by the clicked features. A user v likes A_t when the *total influence* (the sum of separate influence incurred by each clicked feature) on v by A_t is not lower than lt_v . Activated subscribers comprise the first-step activated users. In each subsequent step, new non-subscribers are activated by A_t , where the vis_v of a user v is utilized to check whether A_t is visible to v and apply TRM in accordance

Table 5.1: The fixed parameters of CATRID, A_t and lt_v . Each X (Y) depicts the weight (participation units) of feature f in A_t .

k	3	4	5	Description
$A_t(f_1)$	0.5 (15)	0.4 (12)	0.33 (10)	f_1 denotes the 1st-ranked feature in A_t
$A_t(f_2)$	0.33 (10)	0.3 (9)	0.26 (8)	f_2 denotes the 2nd-ranked feature in A_t
$A_t(f_3)$	0.16 (5)	0.2 (6)	0.2 (6)	f_3 denotes the 3rd-ranked feature in A_t
$A_t(f_4)$	N/A	0.1 (3)	0.13 (4)	f_4 denotes the 4th-ranked feature in A_t
$A_t(f_5)$	N/A	N/A	0.06 (2)	f_5 denotes the 5th-ranked feature in A_t
lt_v	[0, 0.5]	[0, 0.4]	[0, 0.33]	lt_v is randomly selected from each range

with lt_v to determine whether v likes A_t . The propagation of A_t stops when no new user likes A_t . The next section provides more details.

Training. To tune the parameters of CATRID we introduce a training³ phase that considers only posting *brands* as features, hence handles single-feature posts (*sp*). This training phase tunes the F_v and vis_v parameters of CATRID. We calculate $F_v(f)$ as $\frac{\text{clicks_on_}f}{\text{sum_of_clicks}}$, where the numerator depicts the number of times that v clicks on f and the denominator the total clicks by v , and vis_v as the average percentage of in-degree friends of v who liked *sp* before v liked it. The training process initializes values in each F_v uniformly to $\frac{1}{|L|}$ and also sets each vis_v to 10%. For each examined post *sp*, if user v liked *sp*, we consider that v clicked on *brand*, hence update counters for F_v and vis_v accordingly. We set parameters A_t and lt_v for each k as in Table 5.1. The weights in A_t relate to the *simulator* we present in next section; we select lt_v values independently at random from ranges based on the maximum value in A_t .

Knowledge. The training phase yields a *trainedCATRID* known solely to our *simulator*, while each *learner* only knows that user clicks abide by the TRM click model. In the testing phase, a learner starts its execution knowing the same A_t and F_v as the simulator. We evaluate learners under real-world settings, where *partial knowledge* (i.e., the TRM click model) is available, but *complete knowledge* (i.e., the CATRID propagation model) is not. To enhance the realism of the testing phase, we apply the simulator’s TRM using VK posts.

5.2.3 ACAIM Problem

Given a social network $G = (V, E)$, a feature universe L , a weighted feature set F_v of size $|L|$ capturing the preferences of each user v , a budget k , a *brand* corresponding to a feature in L and having a subscriber set S , a *simulator* representing the feedback environment based on a known *trainedCATRID*, a set of testing posts, and a number of rounds n , build a *learner* that, selecting a post A_t of k features to be propagated from S in each round t and evaluating its known TRM by random

³In our experiments, the training phase covers the years 2010–2017 of VK (80% of data).

sampling, maximizes the cumulative influence spread (i.e., number of total likes) I_n in G over n rounds:

$$I_n = \max \sum_{t=1}^n \text{spread}(A_t | S) \quad (5.4)$$

From now on, we use the expressions *likes*, *is influenced*, or *is activated* interchangeably. As F_v is updated differently by simulator and learner in the testing phase, we depict their respective variants as F_v^s for the simulator's variant and F_v^l for the learner's variant. The set S is the starting point of the influence process, but is not taken for granted: A_t activates *some* users in S , and *brand* is always the top-ranked feature in A_t ; we search for the remaining $k - 1$ features.

5.3 CATRID Simulator

Our simulator receives a post A_t , a *brand*'s subscriber set S , a budget k , the social network G , the posts of the testing phase *testingPosts*, and the CATRID model tuned up to round t , and outputs the set of users \mathcal{I}_t activated by A_t along with the CATRID model updated by processing of A_t . Function *SimCATRID* presents the details.

SimCATRID runs CATRID to produce realistic output when called by the learners, to be discussed in next section. To do so, the simulator considers complete knowledge of any post *sp* published during testing⁴ phase involving any feature in L ; *testingPosts*[*f*] contains all *sp* posts by posting *brand* *f* in increasing chronological order; to avoid the ambiguities of post tagging [ITTK17] we consider only *sp* posts, as in training. For each feature in A_t , the simulator first loads (Lines 1–2) a number of posts equal to the proportional *participation units* of *f* in A_t (Table 5.1). For instance, if *SimCATRID* is called for first time with $k = 4$ and *f* is the 3rd feature in A_t , then *clickPosts*[*f*] includes the first 6 posts published by *f* in the testing phase. We load posts progressively over calls to *SimCATRID*; in the rare case that no more posts exist for some feature, we randomly pick one over all testing years.

After loading posts, the simulator initializes (Line 3) the set of users influenced by A_t in round t , \mathcal{I}_t , and that of users influenced in a discrete time step within t , \mathcal{I}_τ , and the map of users marked to be examined (for which we check whether A_t is visible to), *toExmV*, and that of eventually examined users, *hasExmV* (who saw A_t), along with the number of their in-degree friends who liked A_t .

Thereafter, the simulator finds which subscribers get influenced by A_t and stores them in \mathcal{I}_τ (Lines 4–10). To that end, it applies the TRM click model on each user $v \in S$ based on the posts *sp* associated to the features in A_t and computes the *total influence* $I_{\tau v}$ of A_t on v (Lines 4–8); if that is not less than *lt_v*, then v is influenced

⁴In our experiments, the testing phase covers the years 2018–2019 of VK (20% of data).

Function SimCATRID

```

Input      :  $A_t, S, k, G, testingPosts, \text{CATRID\_}_t$ 
Output     :  $\mathcal{I}_t, \text{CATRID\_}_t$ 
// Compute clickPosts given  $A_t$ 
1 for each feature  $f \in A_t$  do
2    $clickPosts[f] = testingPosts[f].getNext(i_f, k);$  // use  $\text{getRandom}(i_f, k)$  if
     $getNext(i_f, k)$  is over
3  $\mathcal{I}_t, \mathcal{I}_\tau = \emptyset; hasExmV, toExmV = map(\emptyset, 0);$  // init
  // Find the subscribers who get influenced by  $A_t$ 
4 for each user  $v \in S$  do // all users in  $S$  see  $A_t$ 
  // Compute the total influence of  $A_t$  on  $v$  to find if  $A_t$  influences  $v$ 
   $I_{\tau v} = 0;$  // init the total influence
  for each feature  $f \in A_t$  do // TRM applicability
    if  $v$  liked at least 1 post  $sp \in clickPosts[f]$  then //  $v$  clicked on  $f$ 
       $I_{\tau v} = I_{\tau v} + A_t(f) \cdot F_v^S(f);$ 
      Update  $F_v^S(f)$  due to  $v$  clicked on  $f$ ; // CATRID $_t$  updated
  if  $I_{\tau v} \geq l_{\tau v}$  then  $\mathcal{I}_\tau.insert(v);$  //  $v$  gets influenced by  $A_t$ 
  // Find the non-subscribers who get influenced by  $A_t$ 
11 while  $\mathcal{I}_\tau \neq \emptyset$  do // influence propagation in  $G$  is not over
  // Collect user examination info for next  $\tau$ 
12   for each user  $v \in \mathcal{I}_\tau$  do // influenced  $v$  in current  $\tau$ 
13     for each user  $z \in G.outEdges(v) \wedge z \notin S$  do
14       if  $z \notin hasExmV$  then  $toExmV[z]++;$ 
15       else  $hasExmV[z]++;$ 
16    $\mathcal{I}_t = \mathcal{I}_t \cup \mathcal{I}_\tau;$  // update  $\mathcal{I}_t$  with current  $\mathcal{I}_\tau$ 
17    $\mathcal{I}_\tau = \emptyset;$  // current  $\tau$  now equals to  $\tau + 1$ 
  // Possibly examine and influence users in current  $\tau$ 
18   for each user  $v \in toExmV$  do //  $vis_v$  applicability
19      $nf_v = vis_v * |G.inEdges(v)|;$  // number of in-degree friends of  $v$  that should
      like  $A_t$  so as it can be visible to  $v$ 
20     if  $toExmV[v] \geq nf_v$  then //  $A_t$  gets visible to  $v$ 
21        $hasExmV[v] = toExmV[v]; toExmV.erase(v);$ 
22       Repeat lines 5-10 to may insert  $v$  to  $\mathcal{I}_\tau;$ 
  // Update  $vis_v$  based on  $hasExmV$  for non-subscribers
23 for each user  $v \in hasExmV \wedge v \notin S$  do //  $A_t$  was visible to  $v$ 
24    $obs\_vis_v = hasExmV[v] \div |G.inEdges(v)|;$ 
25    $vis_v = (vis_v + obs\_vis_v) \div 2;$  // CATRID $_t$  updated
26 return  $\mathcal{I}_t, \text{CATRID\_}_t;$ 

```

by A_t (Line 10); A_t updates CATRID $_t$ when F_v^S changes (Line 9). The simulator in effect follows a TRM click pattern by the condition in Line 7, since the more prominently a feature f appears in a post (higher $A_t(f)$) and the more it attracts the interest of user v (higher $F_v^S(f)$), the more likely it is that v clicks on at least one of the posts associated with f . To correctly update $I_{\tau v}$ (Line 8), the simulator requires

a good tuning of A_t and F_v , ensured in training. Besides A_t and F_v , a user’s reaction to a post depends on latent factors (e.g., how popular the post’s features are at that time, how often a user likes posts) that our simulator implicitly considers.

Once it has found the influenced subscribers, the simulator proceeds to detect non-subscribers influenced by A_t (Lines 11–22). In each time step, it updates $hasExmV$ and $toExmV$ for out-degree connections of users influenced in the previous step, to keep track of their relations who liked A_t (Lines 12–15). Guided by these maps, it checks which users see, and are possibly influenced by A_t in the current step (Lines 18–22). To do so, it utilizes the vis_v of each candidate v (Lines 18–19); when it finds that v is to be examined (Line 20), it removes v from examination candidates (Line 21) and applies TRM on v to ascertain whether A_t influences v (Line 22).

When no more users get influenced (Line 11), the simulator updates the vis_v of each v who saw A_t (Lines 23–25); such updates are redundant for subscribers, since they always see each A_t . We update vis_v as in our training phase, with the difference that now vis_v is also updated for users who saw a post but did not like it; such visibility info is not given by VK, hence is unavailable in training.

5.4 ACAIM Learners

In this section, we present our ACAIM learners, which utilize a known input (G , L , F_v^l) and an unknown input (*trainedCATRID*, *testingPosts*), leveraged by the simulator. Associated with the known input are the brand feature *brand*, the subscriber set S , the number of features per post k , and the number of rounds n ; TRIM_C has an additional clicks threshold parameter θ . The output of all learners is the cumulative influence spread I_n and the modified CATRID (CATRID_n) over n rounds of learning. For all learners, the top-ranked feature for each A_t is *brand*, hence we consider *brand* as chosen by default.

5.4.1 The Learner **R**ANDOM

The Algorithm **R**ANDOM presents a baseline to solve ACAIM. In each round t , **R**ANDOM forms the post A_t by uniformly at random selecting $k - 1$ features and defining the rank of each feature in A_t (Lines 4–6). Thereafter, it uses **SimCATRID** to find influenced users (\mathcal{I}_t) to update I_n and also updates CATRID (Lines 7–8). After processing all n rounds, it returns the cumulative influence spread I_n and the associated CATRID_n (Line 9). Despite the trivial operation of **R**ANDOM, its randomly picked $k - 1$ features for each A_t could express an arbitrary human selection in the real world. So, **R**ANDOM has non-trivial application semantics.

Algorithm RANDOM

Input	$: G, L, F_v^L, trainedCATRID, testingPosts$
Output	$: I_n, CATRID_n$ // cumulative influence spread and modified CATRID over n rounds in ACAIM
Param.	$: brand, S, k, n$

```

1  $A_t[1] = brand$ ; // top-ranked feature for each  $A_t$  is  $brand$ 
2  $I_n = 0$ ;  $CATRID\_n = trainedCATRID$ ; // init
3 for  $t = 1, \dots, n$  do
    // Form  $A_t$  based on uniformly at random operations
    4 Assign  $k - 1$  randomly selected features of  $L$  to  $self$ ;
    5 Randomly define the rank of each  $f \in self$ ;
    6 for  $i_f = 2, \dots, k$  do  $A_t[i_f] = self[i_f]$ ;
        // Compute  $\mathcal{I}_t$  and update  $CATRID\_n$  given  $A_t$ 
    7  $\mathcal{I}_t, CATRID\_n = SimCATRID(A_t, S, k, G, testingPosts, CATRID\_n)$ ;
    8  $I_n = I_n + |\mathcal{I}_t|$ ; // update  $I_n$  with current  $|\mathcal{I}_t|$ 
9 return  $I_n, CATRID\_n$ ;

```

5.4.2 The Learner T_{RIM_C}

In a nutshell, the *classic* learner T_{RIM_C} maintains a set of feature pairs across rounds; in each pair, the first feature has been found to be less influential than the second. In each round, T_{RIM_C} forms a post informed by this knowledge of the dominance of some features over others, while randomizing its decisions regarding features among which no dominance relationship has been found.

We introduce notations relative to the OLR problem [LKLS18]. Given a relation $R \subseteq L^2$ where $L^2 = \{(i, j) | i, j \in L\}$, for a set $X \subseteq L$, let $\min_R(X) = \{i \in X : (i, j) \notin R \forall j \in X\}$, and B_{t1}, \dots, B_{td} be a *partition* of L in round t , where each subset, B_{tc} for $c \leq d$, corresponds to a *block* and the union of the disjoint blocks forms L . The action space \mathcal{A} of T_{RIM_C} in round t is:

$$\mathcal{A}(B_{t1}, \dots, B_{td}) = \{b_{t1} \| \dots \| b_{td}, \forall b_{tc} \text{ of } B_{tc} \mid c \leq d\} \quad (5.5)$$

where b_{tc} denotes a permutation of block B_{tc} and $\forall b_{tc}$ denotes all permutations of that block; the symbol $\|$ denotes concatenation. Thus, $\mathcal{A}(B_{t1}, \dots, B_{td})$ represents the set of all the block permutation concatenations of the partition B_{t1}, \dots, B_{td} . As a learner selects $k - 1$ out of L features for its posts, for the last block B_{td} in Equation 5.5, $\forall b_{td}$ denotes the set of k' -permutations of B_{td} with cardinality $\frac{|B_{td}|!}{(|B_{td}| - k')!}$ where $k' = (k - 1) - |\cup_{c < d} B_{tc}|$.

The Algorithm T_{RIM_C} presents the learner T_{RIM_C} . In each round t , T_{RIM_C} partitions L by creating the blocks B_{t1}, \dots, B_{td} , based on relation R_t updated in the previous round, until the partition blocks altogether contain at least $k - 1$ features (Lines 4–8). After partitioning L , T_{RIM_C} proceeds depending on the value of k ; for

Algorithm TRIM_C

Input : $G, L, F_v^L, \text{trainedCATRID}, \text{testingPosts}$
Output : I_n, CATRID_n // cumulative influence spread and modified CATRID over n rounds in ACAIM
Param. : $\text{brand}, S, k, n, \theta$

```

1  $A_t[1] = \text{brand};$  // top-ranked feature for each  $A_t$  is  $\text{brand}$ 
2  $I_n = 0; \text{CATRID}_n = \text{trainedCATRID}; R_1 = \emptyset;$  // init
3 for  $t = 1, \dots, n$  do
    // Compute the partition  $B_{t1}, \dots, B_{td}$ ; if  $R_t$  was not updated in  $t - 1$  then the
    // partition remains the same and is not computed again (skip lines 4-8)
    4  $d = 0;$ 
    5 while  $L \setminus \bigcup_{c=1}^d B_{tc} \neq \emptyset$  do
        6  $d = d + 1;$ 
        7  $B_{td} = \min_{R_t} (L \setminus \bigcup_{c=1}^{d-1} B_{tc});$ 
        8 if  $\left| \bigcup_{c=1}^d B_{tc} \right| \geq k - 1$  then break;
    // Form  $A_t$  based on  $B_{t1}, \dots, B_{td}$  partition
    9 if  $k < 5$  then // computation of  $\mathcal{A}$  is practical
        // If the partition remains the same then  $\mathcal{A}$  is also the same and not
        // computed again (skip line 10)
        10 Compute  $\mathcal{A}(B_{t1}, \dots, B_{td})$  as denoted in Eq.(5);
        11 Form  $A_t$  by uniformly at random selecting a post from  $\mathcal{A}(B_{t1}, \dots, B_{td})$ 
            such that  $|A_t| = k$ ;
    12 else // computation of  $\mathcal{A}$  costs too much and avoided
        // This process is similar to lines 5-6 of RANDOM
        13 Uniformly at random define the permutations  $b_{t1}, \dots, b_{td}$ ;
        14 Form  $A_t$  by concatenating  $b_{t1}, \dots, b_{td}$  such as  $|A_t| = k$ ;
    15 Repeat lines 7-8 of RANDOM to compute  $I_t$  of  $A_t$  to update  $I_n$  and  $\text{CATRID}_n$ ;
    16  $\forall$  user  $v \in I_t$ , guess if  $C_{tf}^v \in \{0, 1\}$  via random sampling (Eq. (3))  $\forall$  feature  $f \in A_t$ ;
    17 Update  $F_v^L(f)$  when  $C_{tf}^v = 1$ ; //  $v$  clicked on  $f$ 
    18 Sum all guessed  $C_{tf}^v$  and store the result to  $C_{tf}$ ;
    19 for each feature pair  $(f_1, f_2) \in A_t : f_1, f_2 \neq \text{brand}$  do
        20 if  $f_1, f_2 \in B_{tc} | c \leq d$  then // else  $\Delta_{tf_1f_2} = 0$ 
            21  $\Delta_{tf_1f_2} = C_{tf_1} - C_{tf_2};$ 
            22  $\Omega_{tf_1f_2} = \sum_{s=1}^t \Delta_{sf_1f_2};$ 
            23  $R_{t+1} = R_t \cup \{(f_2, f_1) : \Omega_{tf_1f_2} \geq \theta\};$ 
    24 return  $I_n, \text{CATRID}_n$ ;

```

$k < 5$, the action space \mathcal{A} is computed in reasonable time and the learner forms A_t by selecting a post from \mathcal{A} uniformly at random (Lines 9–11); note that if R_t was not updated in the previous round, then \mathcal{A} remains the same. For larger k , the computation of \mathcal{A} is too expensive, so the learner forms A_t by uniformly at random defining a permutation for each block and concatenating the features appearing in those permutations (Lines 12–14). After creating A_t , TRIM_C uses SimCATRID to com-

pute \mathcal{I}_t and update I_n and the A_t -associated CATRID (Line 15). In a second step (Lines 16–18), the feature preference set of each influenced user (F_v^l) is updated based on the observed clicks of that user on the features of A_t ; \mathcal{C}_{tf} aggregates all clicks by influenced users on the feature f of A_t . Based on \mathcal{C}_{tf} values, TRIM_C computes the *clicks difference* ($\Delta_{tf_1f_2}$) of each pair of features (f_1, f_2) in the same block, adding to the difference computations of previous rounds (Lines 19–22). A feature pair (f_2, f_1) is added to relation R_t if the aggregated clicks difference $\Omega_{tf_1f_2}$ is no less than the threshold θ (Line 23), as TRIM_C has collected enough evidence up to round t to conclude that f_2 is less influential than f_1 .

TRIM_C has an *exploration* and *exploitation* stage. In *exploration*, the relation R is filled to maintain an order among the partition blocks of L . This order comprises levels corresponding to blocks; features residing in higher levels compete to participate in higher-ranked positions of A_t than features in lower levels. Adding a feature pair (f_2, f_1) to R_t denotes that f_1, f_2 were previously in the same level, yet henceforward f_2 is to be located at a lower level than f_1 , as the selection in Line 7 ensures. As learning rounds proceed, TRIM_C applies relation R to find a winner feature for each rank in A_t . The *exploration* ends when A_t comprises a winner feature for each rank apart from the last rank, for which no further clicks difference computations are possible since each b_{td} will contain only one feature. Then the *exploitation* stage starts, in which all formed posts differ only on the last feature, whose choice depends on the uniformly at random selection of b_{td} ; during *exploitation* there are no updates on R , hence no new partitions of L .

5.4.3 The Learner TRIM_E

The *eliminator* learner TRIM_E follows a policy of double-checking elimination. When it finds evidence that a feature is less influential than another, TRIM_E records the former as suspect; upon receiving a second piece of evidence corroborating the suspicion (either by full repetition or by providing an alternative inference path), it eliminates the suspect. Yet, it may also render a once-suspect feature non-suspect upon contrary evidence. Figure 5.1 presents an execution flow of TRIM_E .

TRIM_E leverages a *transitive structure* \mathcal{T} and a *participation structure* \mathcal{P} . The transitive structure \mathcal{T} stores *suspicion paths* where a path's source feature is *suspect to be less influential than* (henceforward, *slit*) its destination feature. Each path encodes a *suspicion* among its source and destination features. For instance, consider features f_1, f_2, f_3 ; if f_1 is *slit* f_2 and f_2 is *slit* f_3 then \mathcal{T} includes the paths $\{\langle f_1, f_2 \rangle, \langle f_2, f_3 \rangle, \langle f_1, f_2, f_3 \rangle\}$. A feature may be connected with any other feature in \mathcal{T} via only one path; in case a new path⁵ arises among connected features, then the

⁵A new path is either repetitive (step 9 in Figure 5.1) or alternative (step 7 in Figure 5.1).

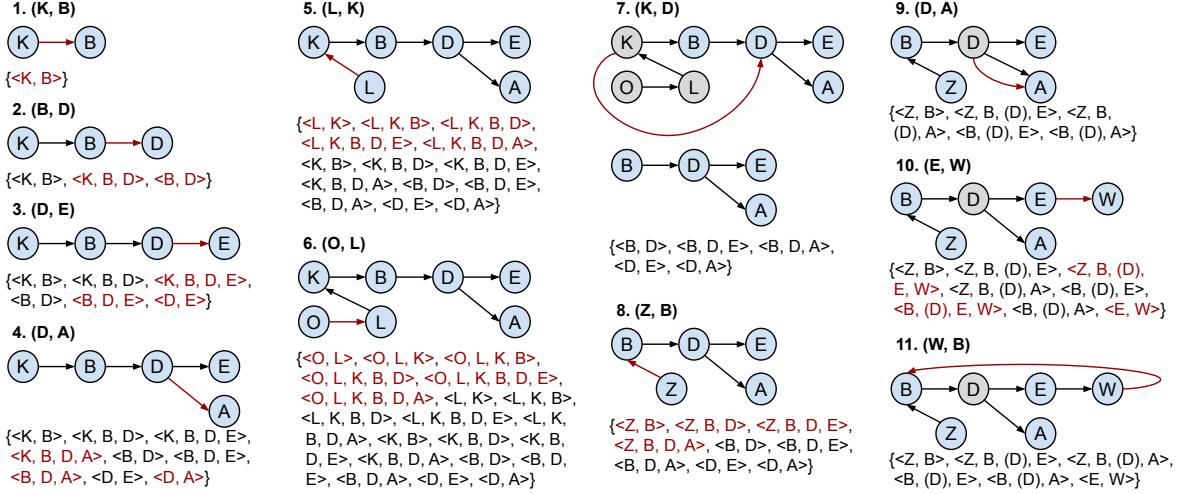


Figure 5.1: Flow of TRIM_E . Each step digests new evidence (feature path, red); in step 7, (K, D) eliminates features in grey; in step 9, the eliminated D remains as intermediate; in step 11, the cycle caused by (W, B) cancels paths containing subpath $\langle B, (D), E, W \rangle$.

source feature is eliminated. We discuss technical details about \mathcal{T} later. The participation structure \mathcal{P} stores the average independent contribution of each feature that participates in posts. The *independent contribution (score)* of a feature f is the difference of the clicks achieved by f and the total clicks achieved by other features in a propagated post A_t .

The Algorithm TRIM_E presents learner TRIM_E . In each round, TRIM_E may eliminate features of L utilizing \mathcal{T} . It stores all features eliminated over rounds in elimF and keeps track of features not yet eliminated in liveF , computed as the set difference between L and elimF (Line 5). Feature elimination stops when the size of liveF is $k - 1$; whence the *exploitation* stage of TRIM_E commences, whereas preceding rounds constitute its *exploration* stage. During *exploration*, TRIM_E forms a post A_t as RANDOM, using liveF instead of L (Lines 6–7). During *exploitation*, it applies the majority rule of \mathcal{P} to form A_t (Lines 8–9). After creating A_t , it computes \mathcal{I}_t via SimCATRID to update I_n , CATRID $_n$, and $F_v^l(f)$ and forms \mathcal{C}_{tf} , as in TRIM_C (Line 10).

In *exploration*, for each ranked feature pair (f'_1, f'_2) (Line 11), TRIM_E computes its normal range (NR) and clicks $ratio$ using the posts and clicks, respectively, associated with those features (Lines 12–14). If $ratio$ is no less than the upper limit of NR , then it calls function ElimWithT to potentially eliminate features due to f'_2 being slit f'_1 (Lines 15–16). On the other hand, if $ratio$ is no more than the lower limit of NR , then ElimWithT induces eliminations due to f'_1 being slit f'_2 (Lines 17–18). Lastly, in both *exploration* and *exploitation*, A_t updates \mathcal{P} considering the \mathcal{C}_{tf} values and the features eliminated in round t (Lines 19–23).

ElimWithT takes as input two features f_1, f_2 , where f_1 is slit f_2 without intermediate features, and, based on this new suspicion (not present in \mathcal{T}) and previous

Algorithm TRIM_E

Input : $G, L, F_v^l, trainedCATRID, testingPosts$
Output : $I_n, CATRID_n$ // cumulative influence spread and modified CATRID over n rounds in ACAIM
Param. : $brand, S, k, n$

- 1 $A_t[1] = brand$; // top-ranked feature for each A_t is $brand$
- 2 $I_n = 0; CATRID_n = trainedCATRID$; // init
- 3 $\mathcal{T}, elimF = \emptyset; \mathcal{P}[i_f][f] = 0, \forall f \in L \wedge i_f = 2, \dots, k$; // init
- 4 **for** $t = 1, \dots, n$ **do**
 - // Form A_t based on $liveF$
 - 5 $liveF = L \setminus elimF$; // non-eliminated features
 - 6 **if** $|liveF| > k - 1$ **then** // elimination is not over
 - 7 Repeat lines 4-6 of RANDOM for $liveF$ instead of L ;
 - 8 **else if** $|liveF| = k - 1$ **then** // elimination is over
 - 9 // Apply the majority rule of \mathcal{P} to form A_t
 - A feature $f \in liveF$ is ranked higher in A_t than another feature $f' \in liveF$ when for $\mathcal{P}[i_f][f], \mathcal{P}[i_f][f'] \neq 0$ it holds that $\mathcal{P}[i_f][f] > \mathcal{P}[i_f][f']$ in most ranking positions $i_f = 2, \dots, k$. In case of a tie ($\mathcal{P}[i_f][f] > \mathcal{P}[i_f][f']$ and $\mathcal{P}[i_f][f] < \mathcal{P}[i_f][f']$ equal number of times over all i_f), as higher-ranked in A_t it is considered the feature with the greater sum over all $\mathcal{P}[i_f]$ values;
 - 10 Given A_t , repeat lines 15-18 of TRIM_C to update I_n and $CATRID_n$, update $F_v^l(f)$ and compute C_{tf} ;
 - // Eliminate features based on C_{tf} and \mathcal{T} (the **loop** in line 11 is executed only when elimination is not over)
 - 11 **for** each feature pair $(f'_1, f'_2) \in A_t : f'_1, f'_2 \neq brand \wedge rank(f'_1) < rank(f'_2)$ **do**
 - 12 $nxtf'_1P = getTheNumberOfspPosts(rank(f'_1) - 1, k);$
 - 13 $f'_2P = getTheNumberOfspPosts(rank(f'_2), k);$
 - 14 $NR = range(1, nxtf'_1P \div f'_2P); ratio = C_{tf'_1} \div C_{tf'_2};$
 - 15 **if** $ratio \geq upper limit of NR$ **then** // f'_2 is slit f'_1
 - 16 $\mathcal{T}, elimF = ElimWithT(f'_2, f'_1, \mathcal{T}, elimF);$
 - 17 **else if** $ratio \leq lower limit of NR$ **then** // f'_1 is slit f'_2
 - 18 $\mathcal{T}, elimF = ElimWithT(f'_1, f'_2, \mathcal{T}, elimF);$
 - 19 // Update \mathcal{P} based on C_{tf} and eliminated features in t
 - Let E_t have the features that were eliminated in t ;
 - 20 **for** each feature $f \in E_t$ **do** Delete all its \mathcal{P} entries;
 - 21 **for** each feature $f \in A_t \wedge f \neq brand \wedge f \notin E_t$ **do**
 - 22 $score = C_{tf} - \sum_{f' \neq f} C_{tf'}; i_f = rank(f);$
 - // Avg is the average of $score$ additions to $\mathcal{P}[i_f][f]$
 - $\mathcal{P}[i_f][f] = Avg(\mathcal{P}[i_f][f] + score);$
 - 24 **return** $I_n, CATRID_n;$

suspicious in \mathcal{T} , it updates \mathcal{T} and $elimF$. $ElimWithT$ uses the auxiliary functions $FwdCheck$ and $BwdCheck$, where the former checks whether f_1 is already slit f_2 , while the latter checks the opposite. Both functions update \mathcal{T} ; $FwdCheck$ also updates $elimF$,

Function FwdCheck

Input :*newPath*, *multiElim*, \mathcal{T} , *elimF*
Output :1 (satisfied) or 0 (not satisfied)

```

1  $\phi_1 = \text{newPath}.first(); \phi_2 = \text{newPath}.last();$ 
2 if  $\mathcal{T}.\text{hasPath}(\phi_1, \phi_2)$  then //  $\phi_1$  is already slit  $\phi_2$ 
    $// \phi_1$  is 2-times slit  $\phi_2$ , so  $\phi_1$  will be eliminated
3   if multiElim = 1 then
    $// \phi_1$  elimination may incur elimination domino (ed)
4     existingPath =  $\mathcal{T}.\text{getPath}(\phi_1, \phi_2);$ 
5     if existingPath  $\neq$  newPath then // ed happens
6       for each feature prvφ :  $\mathcal{T}.\text{hasPath}(\text{prv}\phi, \phi_1) \wedge \mathcal{T}.\text{hasPath}(\text{prv}\phi, \phi_2)$ 
7         do
8            $\text{elimF.insert}(\text{prv}\phi); // eliminate } \text{prv}\phi$ 
           $\mathcal{T}.\text{erase}(\mathcal{T}.\text{outPaths}(\text{prv}\phi)); \mathcal{T}.\text{erase}(\mathcal{T}.\text{inPaths}(\text{prv}\phi));$ 
9
10      elimF.insert( $\phi_1$ ); // eliminate  $\phi_1$ 
11       $\mathcal{T}.\text{erase}(\mathcal{T}.\text{outPaths}(\phi_1)); \mathcal{T}.\text{erase}(\mathcal{T}.\text{inPaths}(\phi_1));$ 
12      return 1;
13
14 else return 0;
    
```

Function BwdCheck

Input :*newPath*, \mathcal{T}
Output :1 (satisfied) or 0 (not satisfied)

```

1  $\phi_1 = \text{newPath}.first(); \phi_2 = \text{newPath}.last();$ 
2 if  $\mathcal{T}.\text{hasPath}(\phi_2, \phi_1)$  then //  $\phi_2$  is already slit  $\phi_1$ 
    $// A cycle occurs among  $\phi_1$  and  $\phi_2$ ; the newPath cancels the existingPath and$ 
    $all the superpaths of newPath and existingPath from  $\mathcal{T}$$ 
3   existingPath =  $\mathcal{T}.\text{getPath}(\phi_2, \phi_1); \mathcal{T}.\text{erase}(\text{existingPath});$ 
4    $\mathcal{T}.\text{erase}(\mathcal{T}.\text{subPath}(\text{existingPath})); \mathcal{T}.\text{erase}(\mathcal{T}.\text{subPath}(\text{newPath}));$ 
5   return 1;
6 else return 0;
    
```

but for brevity, we do not include \mathcal{T} and *elimF* in the output. Instead, we let the output of both functions be a flag that denotes whether the respective condition is satisfied. Besides \mathcal{T} and *elimF*, the input of FwdCheck comprises a suspicion path *newPath*, which carries suspicion knowledge not existing in \mathcal{T} , and a flag *multiElim* that dictates whether the elimination of a feature may incur further feature eliminations; BwdCheck takes as input \mathcal{T} and *newPath*, which is not stored in \mathcal{T} .

FwdCheck operates as follows. Let ϕ_1 be the source feature and ϕ_2 the destination feature of *newPath* (Line 1). If a suspicion path from ϕ_1 to ϕ_2 already exists (Line 2), the consideration of *newPath* denotes that ϕ_1 is twice slit ϕ_2 , hence ϕ_1 should be eliminated by policy. The eliminated ϕ_1 is added to *elimF* and all paths where ϕ_1 is source or destination are deleted from \mathcal{T} (Lines 9–10). Since *multiElim* may be activated (Line 3), to manage a possible feature elimination domino (ed) induced by the elimination of ϕ_1 , FwdCheck needs the info of ϕ_1 in \mathcal{T} , hence eliminates ϕ_1 at

the end of *ed*. An *ed* happens when the existing path from ϕ_1 to ϕ_2 is different from *newPath* (Lines 4–5), hence two different paths to ϕ_2 are also trodden by each other feature $prv\phi$ that already has a path to ϕ_1 and ϕ_2 , hence each such $prv\phi$ is eliminated (Lines 6–8). In *BwdCheck*, ϕ_1 and ϕ_2 are still the source and destination features of *newPath* (Line 1), but now if there exists a path from ϕ_2 to ϕ_1 , then *newPath* induces a cycle (Line 2). By policy, TRIM_E resolves a suspicion cycle by canceling the two paths that form the cycle and deleting from \mathcal{T} any path containing them (Lines 3–4).

Let us resume our focus on *ElimWithT*. This function sequentially executes three steps for given features f_1, f_2 . In the first step, it checks the early termination cases (Lines 1–2). If *FwdCheck* finds that f_1 is already *slit* f_2 , then it updates *elimF* and deletes from \mathcal{T} all relative *elimination* info; *multiElim* is activated in this call as the elimination of f_1 in this case may induce an *ed*. Yet, early termination also happens if *BwdCheck* finds that f_2 is already *slit* f_1 , hence deletes from \mathcal{T} all relative *cycle* info. If no early termination happens, *ElimWithT* updates \mathcal{T} with the suspicion that f_1 is *slit* f_2 (Line 3). In the second step, *ElimWithT* checks the outpaths of f_2 to update \mathcal{T} (Line 4). For each destination feature $nxtf_2$ of f_2 , a *path* is formed from f_1 to $nxtf_2$ to search for new suspicion knowledge among f_1 and $nxtf_2$ to insert to \mathcal{T} (Lines 5–6). In case f_1 is eliminated through *FwdCheck* (*multiElim* remains activated), *ElimWithT* stops checking the outpaths of f_2 , as f_1 is permanently deleted from \mathcal{T} (Line 7). Otherwise, if *FwdCheck* is not satisfied, *BwdCheck* is called to examine whether *path* yields a cycle; if so, *path* is invalidated, hence we proceed with the next outpath of f_2 (Line 8), otherwise insert *path* to \mathcal{T} (Line 9). Last, in the third step, *ElimWithT* checks the inpaths of f_1 to update \mathcal{T} (Line 10). For each source feature $prvf_1$ of f_1 , a path *prefix* is formed from $prvf_1$ to f_2 to gauge new suspicions (Lines 11–12); an elimination of $prvf_1$ via *FwdCheck* does not affect the examination of other inpaths of f_1 , since f_1 itself is not eliminated (Line 13). The *multiElim* flag is not activated when the source feature of *newPath* in *FwdCheck* is $prvf_1$, since the possible elimination of a feature depending on the elimination of $prvf_1$ is already considered by examining each $prvf_1$ in Line 11. Back to the third step, if *FwdCheck* is not satisfied in Line 13, *BwdCheck* checks if *prefix* creates a cycle, to invalidate it and proceed with the next inpath of f_1 (Line 14); if no such cycle exists, we insert *prefix* to \mathcal{T} (Line 15). Lastly, a *path* is formed from $prvf_1$ to $nxtf_2$ to update \mathcal{T} with the suspicion knowledge derived from the second and third steps of *ElimWithT* (Line 16).

Technical Details of \mathcal{T} . We implement \mathcal{T} using indexes *src* and *dst*. For a given path, *src* maps the path source to path destination, whereas *dst* maps the path destination to the whole path ignoring its destination. E.g., if \mathcal{T} includes the paths $\{\langle f_1, f_2 \rangle, \langle f_2, f_3 \rangle, \langle f_1, f_2, f_3 \rangle\}$, then $src[f_1] = \{f_2, f_3\}$ and $src[f_2] = \{f_3\}$, while $dst[f_2] = \{\langle f_1 \rangle\}$ and $dst[f_3] = \{\langle f_2 \rangle, \langle f_1, f_2 \rangle\}$. As only one path exists in \mathcal{T} among any con-

Function ElimWithT

```

Input      :  $f_1, f_2, \mathcal{T}, elimF$ 
Output     :  $\mathcal{T}, elimF$ 
    // Check if  $f_1$  is already slit  $f_2$ ; If yes, update elimF and delete from  $\mathcal{T}$  all elimination info
1 if FwdCheck( $\langle f_1, f_2 \rangle$ , 1,  $\mathcal{T}, elimF$ ) = 1 then return  $\mathcal{T}, elimF$ ;
    // Check if  $f_2$  is already slit  $f_1$ ; If yes, delete from  $\mathcal{T}$  all cycle info
2 if BwdCheck( $\langle f_1, f_2 \rangle, \mathcal{T}$ ) = 1 then return  $\mathcal{T}, elimF$ ;
3  $\mathcal{T}.\text{insert}(\langle f_1, f_2 \rangle)$ ; // insert to  $\mathcal{T}$  the fact that  $f_1$  is slit  $f_2$ 
    // Update  $\mathcal{T}$  with all slit knowledge derived from  $f_2$  and delete elimination and cycle info
4 if  $\mathcal{T}.\text{outPaths}(f_2) \neq \emptyset$  then
5   for each feature  $nxtf_2 : \mathcal{T}.\text{hasPath}(f_2, nxtf_2)$  do
6      $path = \langle f_1 \rangle \parallel \mathcal{T}.\text{getPath}(f_2, nxtf_2)$ ;
7     if FwdCheck( $path$ , 1,  $\mathcal{T}, elimF$ ) = 1 then break;
8     if BwdCheck( $path$ ,  $\mathcal{T}$ ) = 1 then continue;
9      $\mathcal{T}.\text{insert}(path)$ ;
    // Update  $\mathcal{T}$  with all slit knowledge derived from  $f_1$  and delete elimination and cycle info
10 if  $\mathcal{T}.\text{inPaths}(f_1) \neq \emptyset$  then
11   for each feature  $provf_1 : \mathcal{T}.\text{hasPath}(provf_1, f_1)$  do
12      $prefix = \mathcal{T}.\text{getPath}(provf_1, f_1) \parallel \langle f_2 \rangle$ ;
13     if FwdCheck( $prefix$ , 0,  $\mathcal{T}, elimF$ ) = 1 then continue;
14     if BwdCheck( $prefix$ ,  $\mathcal{T}$ ) = 1 then continue;
15      $\mathcal{T}.\text{insert}(prefix)$ ;
16     Repeat lines 4-9 with only difference that now
         $path = prefix.\text{allButLast}() \parallel \mathcal{T}.\text{getPath}(f_2, nxtf_2)$  and FwdCheck( $path$ , 0,
           $\mathcal{T}, elimF$ );
17 return  $\mathcal{T}, elimF$ ;

```

nected features, \mathcal{T} is a space saving structure that provides efficient data access to its associated functions *hasPath*, *getPath*, *outPaths*, *inPaths*, *erase*, *insert*, and *subPath*. We design \mathcal{T} so that it retains suspicion knowledge despite eliminations or cycles, to facilitate the *exploration* stage of **TRIM_E**, as such retention causes beneficial *eds*. Thus, when a feature is eliminated, its outpaths and inpaths are deleted from \mathcal{T} , yet that feature may exist as a path intermediate feature in \mathcal{T} . Besides, when two paths form a cycle and are canceled, all their superpaths are deleted from \mathcal{T} , yet their subpaths remain in \mathcal{T} . Lastly, \mathcal{T} is scalable and memory efficient, as it is gradually constructed via size increases and decreases, but in the long term its size diminishes.

5.5 Experimental Evaluation

We wrote code in C++ and ran experiments on an Intel Xeon CPU E7-4830 @ 2.13 GHz machine with 252GB RAM running Linux Debian (4.9.0-13-amd64). For graph operations we used the open-source graph library Lemon [DJK11].

5.5.1 Setup

Datasets. To evaluate our learners for the ACAIM problem, we created two big VK data that we often call data \mathbb{A} and data \mathbb{B} . Since VK has in total 27 categories, by selecting the 10 (\mathbb{A}) and 20 (\mathbb{B}) most popular pages from each category, we formed a feature set $|L| = 270$ and $|L| = 540$ in each case; the popularity of a page was measured by the number of its subscribers. In particular, both \mathbb{A} and \mathbb{B} operate as a cluster that includes three, scalable to network size, datasets that we call A1, A2, and A3 in \mathbb{A} and B1, B2, and B3 in \mathbb{B} . Our experimental evaluation is done on these six datasets. The requested preprocessing to form the mentioned datasets consists of two stages; Tables 5.2 and 5.3 present the first and second stage respectively. In first stage, we subsequently (i) collected all the VK users until August 2019, (ii) removed the blocked users (deactivated, private, or empty profile), (iii) applied a *feature-pruning* to filter out any user who did not like at least one post published from a feature in L , as that user does not contribute at all neither in training (years 2010-2017) nor in testing (years 2018-2019) phases of ACAIM for our selected L , and (iv) applied a *testing-pruning* to further exclude any user who did not like at least one post of L in testing phase, as the evaluation of learners depends on a feedback during testing. In second stage, we created the described six datasets with a top-to-bottom process as presented in Table 5.3. The logic here is first to find the average out-degree ($hop1_out_avg$) of previous 43.8M and 48.3M users, and to increase it by a different factor for each dataset so as to create a network size scalability. Then, each user that has an out-degree equal or higher than $increased_hop1_out_avg$ is considered an *influential_user*, while each user that can be directly influenced by an *influential_user* is considered an *influenced_user*. Further, to enhance the probability that a propagation reaches an *influential_user*, we also considered users who contribute to that via one-hops or two-hops. The union of *influential_users*, *influenced_users*, *hop1_in_influential_users*, and *hop2_in_influential_users* yields the nodes of respective dataset along with their accompanied edges. We stress that our preprocessing approach gives as maximum a realistic enough three-hop influence connectivity for an *influenced_user* that represents the target for IM problems, while any pruned user incurred unnecessary memory overhead for the proof-of-concept purposes of our work. Last, we note that in training phase we used 9.5M (resp. 18.4M) sp posts for \mathbb{A} (resp. \mathbb{B}) and the total available sp posts to simulator in testing phase are 2.8M for \mathbb{A} and 5.6M for \mathbb{B} .

Metrics. We use the metrics of *reliability* and *scalability* to measure the performance of our learners. In ACAIM, the most *reliable* learner is the learner that consistently achieves the highest influence in most cases for separate brands in different datasets,

Table 5.2: First Preprocessing to prune VK users for $|L| = 270$ and $|L| = 540$.

VK Users	After Removing Blocked Users	Feature-Pruning		Testing-Pruning	
		$ L = 270$	$ L = 540$	$ L = 270$	$ L = 540$
500M	348.4M	91.1M	99.9M	43.8M	48.3M

Table 5.3: Second Preprocessing to create three, scalable to network size, VK datasets respectively for data \mathbb{A} ($|L| = 270$) and data \mathbb{B} ($|L| = 540$).

Dataset	$\mathbb{A}:A1$	$\mathbb{A}:A2$	$\mathbb{A}:A3$	$\mathbb{B}:B1$	$\mathbb{B}:B2$	$\mathbb{B}:B3$
hop1_out_avg	94.48	94.48	94.48	98.76	98.76	98.76
increase factor (%)	200K	150K	100K	200K	150K	100K
increased_hop1_out_avg	189K	141.8K	94.5K	197.6K	148.2K	98.8K
influential_users	10	20	39	16	31	53
influenced_users	1.9M	2.7M	3.7M	2.8M	3.9M	4.8M
hop1_in_influential_users	6.6K	7.7K	15.7K	7.1K	9.5K	17.9K
hop2_in_influential_users	1.5M	1.8M	3M	1.7M	2.3M	3.6M
Nodes	3.3M	4.4M	6.2M	4.4M	5.9M	7.8M
Edges	320.8M	417.2M	669.7M	434.6M	596.5M	879M

while the most *scalable* learner is the learner that remains most valuable for a *brand* as the network size grows. We highlight that in this work we are not interested to measure the scalability of an unreliable learner.

Runs. All *reliability* results are averaged over 10 runs, while all *scalability* results are stemmed from 1 run. The former experiments relate with ordinary brands where their execution time is affordable and so we used 10 runs to reduce randomness, while the latter experiments mention to highly popular brands and their execution time is as expected very high to allow multiple runs. For that, we used nine different brands for scalability experiments to derive a general conclusion for scalability from 9 runs instead of just 1 run. Note also that each run relates with a different lt_v for CATRID.

Notation. We use the symbols \mathcal{E}_1 for *exploration* and \mathcal{E}_2 for *exploitation*. Also, in all our figures, llr means last learning round of \mathcal{E}_1 , fte means remaining features to eliminate, *influence spread* is the cumulative influence spread over rounds, and *min* measures the total execution time over rounds in minutes excluding the input loading time of each round. The number of considered rounds is logically set to 2000, and we depict each *brand* by using its ID in VK (see Table 5.7 in Section 5.5.4).

5.5.2 Reliability

To measure the *reliability* of ACAIM learners, we select six brands belonging to different categories for maximum diversification; three brands for datasets A1, A2, and A3 and three brands for datasets B1, B2, and B3. In all cases, we selected the *brand* whose number of subscribers is closest to the average number of subscribers

Chapter 5. Adaptive Content-Aware Influence Maximization through Online Learning To Rank

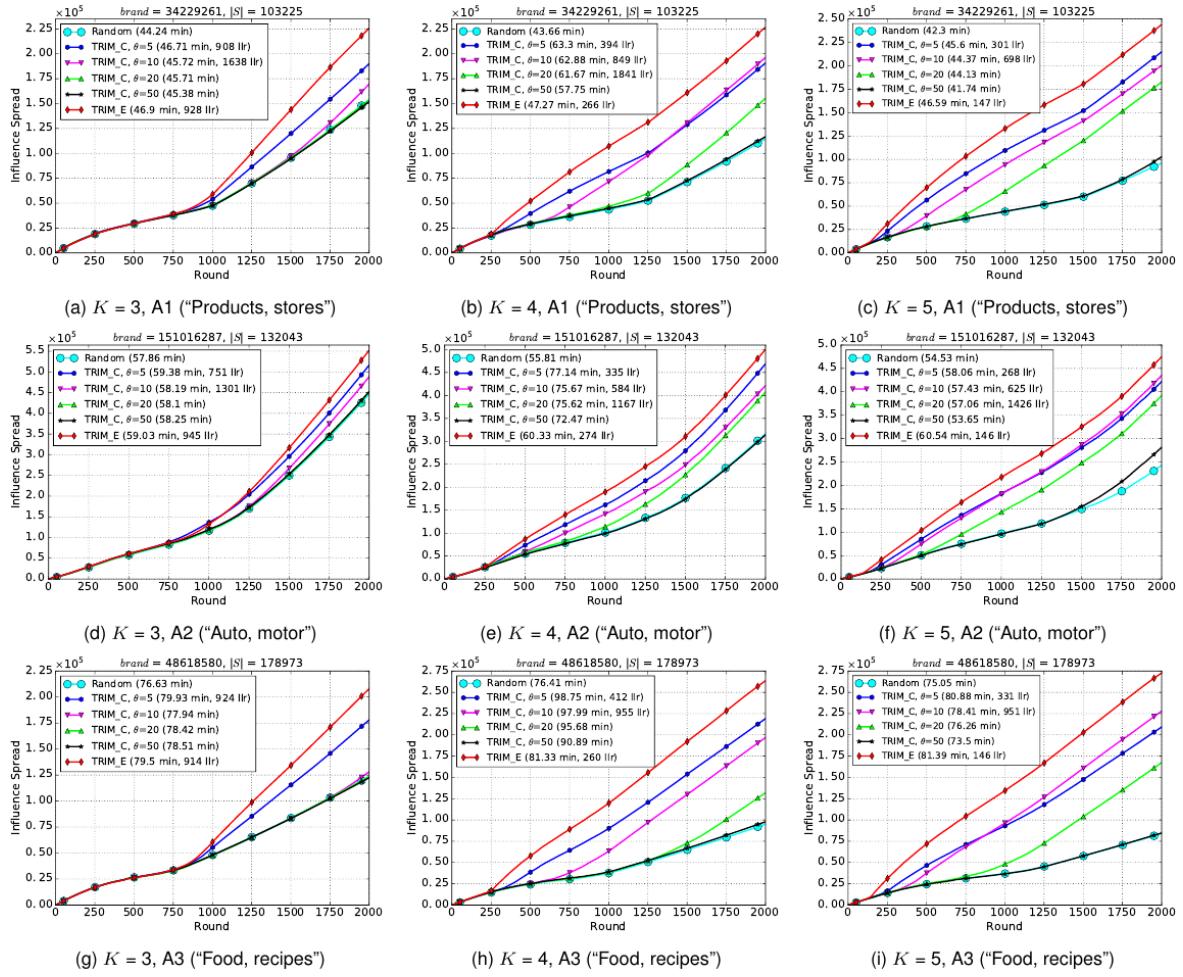


Figure 5.2: Reliability results of RANDOM, TRIM_C, and TRIM_E in datasets A1, A2, and A3 for $k = 3, 4$, and 5 .

over all brands for respective dataset; we argue that this is a fair selection policy that expresses the majority of brands in VK in terms of their subscription numbers. Figures 5.2 and 5.3 present the reliability results for data A and data B respectively.

In Figure 5.2, the first three plots relative to A1 depict the influence spread of a *brand* belonging to “Products, stores” category for $k = 3, 4$, and 5 ; the next three plots relative to A2 of a *brand* belonging to “Auto, motor” category and the last three plots relative to A3 of a *brand* belonging to “Food, recipes” category. In all cases, TRIM_E achieves the best performance due to its faster \mathcal{E}_1 that also enables a really influential \mathcal{E}_2 ; TRIM_E has the second fastest \mathcal{E}_1 only for $k = 3$ in Figures 5.2a and 5.2d, but even in these two cases, its derived \mathcal{E}_2 is more effective than others. TRIM_C with $\theta = 5$ is the second best solution, while a choice of $\theta = 10$ yields a slightly better performance in Figures 5.2b, 5.2f, and 5.2i; these figures denote that a slower \mathcal{E}_1 of a higher θ can induce a more influential \mathcal{E}_2 compared to a lower θ . This is a sound result as a stricter θ is theoretically more reliable. Yet, overall, we observe in practice

5.5. Experimental Evaluation

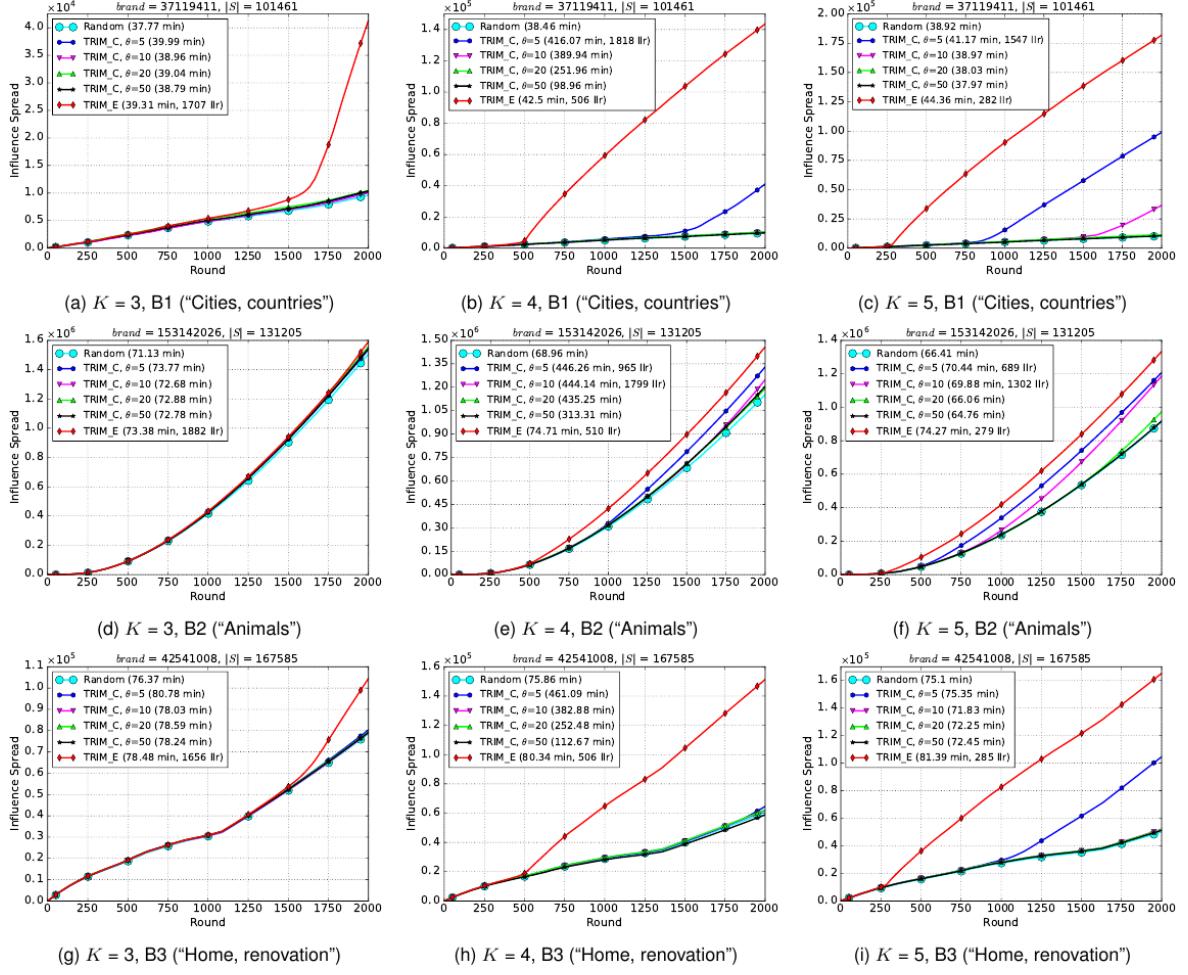


Figure 5.3: Reliability results of RANDOM, TRIM_C, and TRIM_E in datasets B1, B2, and B3 for $k = 3, 4$, and 5 .

that the more TRIM_C increases its θ the more close it performs with RANDOM, which is the worst learner of all. The reason is that the clicks collected over a logical number of rounds do not easily permit an aggregated computed clicks delta among features to early surpass a high θ , and that incurs a slow \mathcal{E}_1 for such θ cases. Note that in real world and as our CATRID simulator also successfully captures, the subscribers of an ordinary *brand* do not like so often its posts and that usually prevents a high propagation collecting a large number of clicks. So, a lower θ is more beneficial for TRIM_C, which is a deficiency in its operation due to the loose ranking of features. Yet, there is no other alternative, especially if one notice that TRIM_C has high θ cases that incur a too much slow \mathcal{E}_1 (for $\theta = 10$ in Figure 5.2a and $\theta = 20$ in Figure 5.2b) or an unfinished \mathcal{E}_1 (for $\theta = 10$ in Figure 5.2g, $\theta = 20$ in Figures 5.2a, 5.2c, 5.2d, 5.2g, 5.2h, and 5.2i, and $\theta = 50$ in all plots of Figure 5.2). In the worst-case scenario, where $\theta = 50$, TRIM_C performs similarly to RANDOM since during \mathcal{E}_1 it scarcely manages to surpass θ , and so it selects posts under very common settings with RANDOM.

In Figure 5.3, the first three plots relative to B1 depict the influence spread of a *brand* belonging to “Cities, countries” category for $k = 3, 4$, and 5 ; the next three plots relative to B2 of a *brand* belonging to “Animals” category and the last three plots relative to B3 of a *brand* belonging to “Home, renovation” category. In all cases, TRIM_E outperforms TRIM_C and RANDOM , it is also the only learner that always has a finite \mathcal{E}_1 , while its superiority is really impressive in several cases (Figures 5.3a, 5.3b, 5.3c, 5.3h, and 5.3i). Again, RANDOM achieves the worst performance, whereas TRIM_C is remarkably less competitive in data \mathbb{B} compared to data \mathbb{A} due to the much bigger effect of θ on its performance, even when θ is low. That is explained by the larger L in data \mathbb{B} which entails a significantly broader distribution of clicks on its features. This deficiency of TRIM_C shows that its efficacy is not scalable to the feature size.

Moreover, in both Figures 5.2 and 5.3 the \mathcal{E}_1 of TRIM_E and TRIM_C finishes sooner as k increases with some exceptions (for $\theta = 10$ in Figure 5.2f and $\theta = 20$ in Figures 5.2c and 5.2f). This happens because a larger k enables a more frequent participation of a feature in propagated posts, and so more chances for it to be compared with other features based on its collected clicks. Yet, a faster \mathcal{E}_1 of a larger k may lead to a lower influence for both TRIM_E and TRIM_C as it is shown in Figures 5.2d, 5.2e, and 5.2f for A2 and in Figures 5.3d, 5.3e, and 5.3f for B2. How k affects the influence of a *brand* depends on how loyal are its subscribers. Namely, if the subscribers of a *brand* like often its posts, then they create the chances for a good influence in the network, and so the learner should choose a small k to give more focus on the *brand*. Yet, there is a loyalty tradeoff as we observe in the next section, since a highly adopted *brand* may perform better with a larger k instead of a smaller one. Regarding RANDOM , we notice that its performance constantly deteriorates as k increases except for B1 (Figures 5.3a, 5.3b, and 5.3c) where it hardly improves. The reason is that the power of RANDOM lies mostly on *brand*, so by giving a less focus on it with a larger k , we boost the uncertain influence of arbitrary features that accompany the *brand*.

Overall, for both data \mathbb{A} and data \mathbb{B} , Table 5.4 presents the influence comparison of TRIM_C and TRIM_E with RANDOM , Table 5.5 compares the influence among TRIM_E and TRIM_C , while Table 5.6 compares the learning speed of previous learners; for TRIM_C we selected its most influential θ version in each case. In all Tables, each single column after *Algorithm* and before *Average* includes the results for each dataset that stems from the intersection with the respective row. For instance, Table 5.4 denotes that TRIM_C is 123.9% influence better than RANDOM for $k = 5$ in A1 and TRIM_E is 148.9% influence better than RANDOM for $k = 4$ in B3.

We observe in *Average* column of Table 5.4 that both TRIM_C and TRIM_E outperform more RANDOM as k increases, that difference in performance becomes more intense in data \mathbb{B} , and also that RANDOM is obviously more competitive to TRIM_C than TRIM_E . As previously explained for ordinary brands, RANDOM performs better with

5.5. Experimental Evaluation

Table 5.4: Percentages (%) depicting how much *influence better* TRIM_C (with most influential θ) and TRIM_E are over RANDOM for data \mathbb{A} and data \mathbb{B} .

Algorithm	RANDOM (A1, B1)			RANDOM (A2, B2)			RANDOM (A3, B3)			Average		
	$k = 3$	$k = 4$	$k = 5$	$k = 3$	$k = 4$	$k = 5$	$k = 3$	$k = 4$	$k = 5$	$k = 3$	$k = 4$	$k = 5$
$\text{TRIM_C}(\mathbb{A})$	23.3	71.2	123.9	15.6	48.1	78.9	43.7	130.5	171.1	27.5	83.2	124.6
$\text{TRIM_C}(\mathbb{B})$	8.6	320.9	849.7	3.3	15.3	32	2.1	6.2	108.6	4.6	114.1	330.1
$\text{TRIM_E}(\mathbb{A})$	46.5	97.9	154.3	23.6	58.1	95.4	67.9	177.3	224.4	46	111.1	158
$\text{TRIM_E}(\mathbb{B})$	329	1372.9	1646.9	5.6	26.5	45.8	32.9	148.9	229.4	122.5	516.1	640.7

Table 5.5: Percentages (%) depicting how much *influence better* TRIM_E is over TRIM_C (with most influential θ) for data \mathbb{A} and data \mathbb{B} .

Algorithm	$\text{TRIM_C}(\mathbb{A}, \mathbb{B}_1)$			$\text{TRIM_C}(\mathbb{A}, \mathbb{B}_2)$			$\text{TRIM_C}(\mathbb{A}, \mathbb{B}_3)$			Average		
	$k = 3$	$k = 4$	$k = 5$	$k = 3$	$k = 4$	$k = 5$	$k = 3$	$k = 4$	$k = 5$	$k = 3$	$k = 4$	$k = 5$
$\text{TRIM_E}(\mathbb{A})$	18.8	15.5	13.5	6.9	6.7	9.2	16.8	20.3	19.6	14.1	14.1	14.1
$\text{TRIM_E}(\mathbb{B})$	294.7	249.9	83.9	2.1	9.6	10.4	30.1	134.3	57.8	108.9	131.2	50.7

Table 5.6: Percentages (%) depicting how much *learning faster* TRIM_E is over TRIM_C (with most influential θ) for data \mathbb{A} and data \mathbb{B} .

Algorithm	$\text{TRIM_C}(\mathbb{A}, \mathbb{B}_1)$			$\text{TRIM_C}(\mathbb{A}, \mathbb{B}_2)$			$\text{TRIM_C}(\mathbb{A}, \mathbb{B}_3)$			Average		
	$k = 3$	$k = 4$	$k = 5$	$k = 3$	$k = 4$	$k = 5$	$k = 3$	$k = 4$	$k = 5$	$k = 3$	$k = 4$	$k = 5$
$\text{TRIM_E}(\mathbb{A})$	-2.2	219.1	104.7	-25.8	22.2	328	1.1	58.4	551.3	-8.9	99.9	328
$\text{TRIM_E}(\mathbb{B})$	> 17.1	259.2	448.5	> 6.2	89.2	146.9	> 20.7	> 295.2	> 601.7	> 14.6	> 214.5	> 399

a small k while the other two learners are often more influential with a larger k . In addition, the more features that exist in data \mathbb{B} aggravate further the performance of RANDOM as its arbitrary selections are increased. Yet, the interesting point here is that although TRIM_C has severe scalability issues in data \mathbb{B} (e.g., TRIM_C is just 4.6% influence better than RANDOM for $k = 3$ in data \mathbb{B}), it manages to amplify its average influence over RANDOM in regards to data \mathbb{A} for $k = 4$ and $k = 5$.

The Average column of Table 5.5 shows that TRIM_E is 14.1% influence better than TRIM_C for each k in data \mathbb{A} , while TRIM_E is 108.9%, 131.2%, and 50.7% influence better than TRIM_C for $k = 3, 4$, and 5 in data \mathbb{B} . Since TRIM_C is not scalable to the feature size, it needs a large enough k to be competitive to TRIM_E and slightly achieves that for $k = 5$. Additionally to the influence comparison, in Average column of Table 5.6 we remark that TRIM_E is -8.9%, 99.9%, and 328% learning faster than TRIM_C for $k = 3, 4$, and 5 in data \mathbb{A} , whereas TRIM_E is at least 14.6%, 214.5%, and 399% learning faster than TRIM_C for $k = 3, 4$, and 5 in data \mathbb{B} . The \mathcal{E}_1 of TRIM_C is faster than the one of TRIM_E only for $k = 3$ in A1 and A2, while in all other cases it is slower and actually emphatically slower in most times. Both Figures 5.2 and 5.3 demonstrate that the learning speed of a learner crucially defines its influence spread, and that connection becomes even more dependent when that speed is accompanied by an effective learning. This connection is highly satisfied by TRIM_E and that explains its remarkable superiority over TRIM_C , especially in data \mathbb{B} .

Chapter 5. Adaptive Content-Aware Influence Maximization through Online Learning To Rank

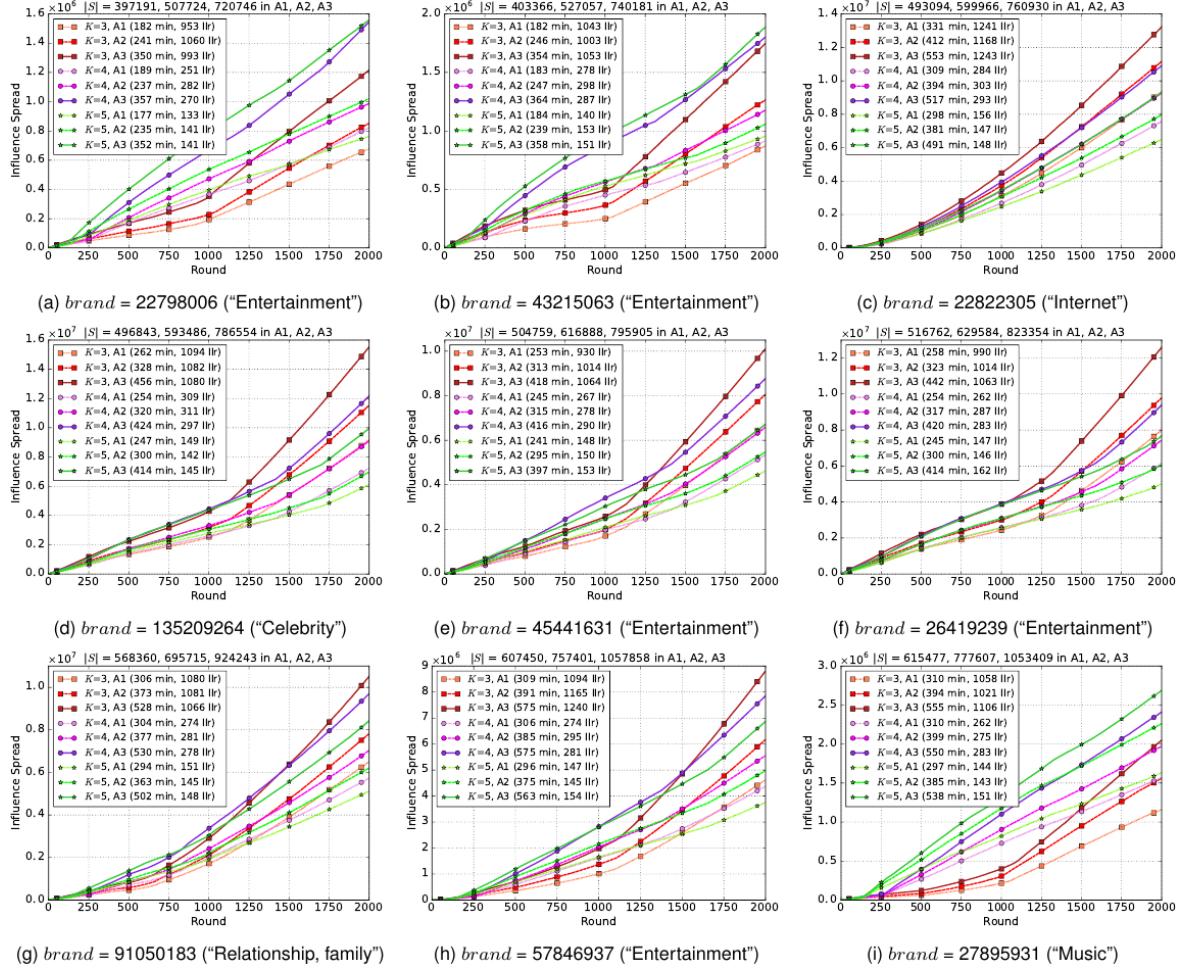


Figure 5.4: Scalability results of TRIM_E for the nine most popular VK brands (depicted in increasing $|S|$) in datasets A1, A2, and A3 for $k = 3, 4$, and 5.

The most representative case is for $k = 4$ in data \mathbb{B} where TRIM_E is at least 214.5% learning faster and 131.2% influence better than TRIM_C.

Regarding the running time of learners, RANDOM is the most fast solution in data \mathbb{A} and data \mathbb{B} for $k = 3$ and $k = 4$, while it usually comes second after TRIM_C with $\theta = 50$ for $k = 5$; in the latter case TRIM_C is a little faster due to a gradually smaller randomized feature set induced after exceeding θ . These results are expected due to the low overhead and influence spread of mentioned learners. The noteworthy point for TRIM_E is that besides its much higher influence than RANDOM, its execution time remains very close to RANDOM in all cases, which highlights the efficient, flexible, and compact implementation of \mathcal{T} . Last, note the remarkably costly execution of TRIM_C for $k = 4$, especially in data \mathbb{B} , which relates with the expensive computation of \mathcal{A} ; for $k = 3$ the cost of \mathcal{A} is much more affordable, whereas for $k = 5$ the computation of \mathcal{A} is avoided.

5.5. Experimental Evaluation

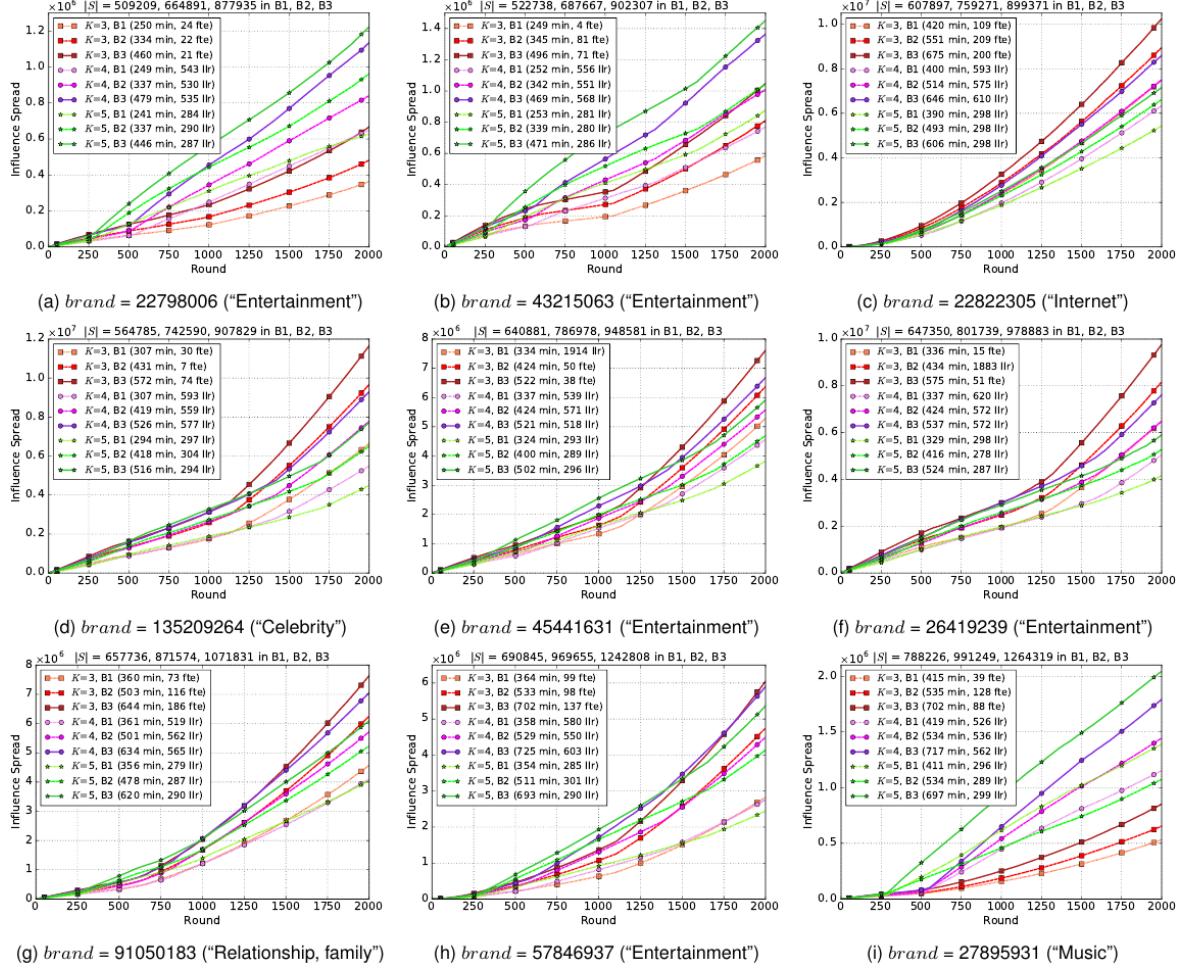


Figure 5.5: Scalability results of TRIM_E for the nine most popular VK brands (depicted in increasing $|S|$) in datasets B1, B2, and B3 for $k = 3, 4$, and 5 .

5.5.3 Scalability

In previous section, we have experimentally proved the superiority of TRIM_E over RANDOM and TRIM_C by evaluating the *reliability* of all learners for separate brands in different datasets. In this section, we check if TRIM_E also enables the *scalability* of a brand to network size; the other two learners are ignored due to their much inferior reliability. To measure the scalable performance of TRIM_E under the most demanding settings, we selected the nine most popular brands of VK, where the popularity is interpreted by the number of subscribers. In Figure 5.4, we present the scalability results of selected brands for $k = 3, 4$, and 5 in datasets A1, A2, and A3, while Figure 5.5 shows the respective results in datasets B1, B2, and B3.

In both Figures 5.4 and 5.5, we observe that for all brands, TRIM_E is highly scalable to network size since under any fixed k and as the dataset grows, TRIM_E not only manages to complete its learning but also it achieves a constantly higher

influence accompanied by logically higher execution times. Regarding influence, the only exception takes place in Figure 5.5i where TRIM_E is more influential for $k = 5$ in B1 compared to $k = 5$ in B2, while regarding incomplete learning, the sole exceptions happen for $k = 3$ in Figure 5.5 when fte replaces llr . The fte cases show an obvious difficulty of TRIM_E to provide a finite \mathcal{E}_1 , but even in these special cases, its efficacy remains highly scalable to network size.

As in previous section, we verify that TRIM_E still learns faster as k grows, as also that the choice of the proper k relates with the subscription loyalty of a *brand*. In the most cases of both Figures 5.4 and 5.5, we remark that a smaller k is more beneficial for a highly popular *brand*, yet there is a loyalty tradeoff that clearly demonstrates the opposite for some brands (Figures 5.4a, 5.4b, 5.4i, 5.5a, 5.5b, and 5.5i).

5.5.4 Business Applicability

In Table 5.7 we present all the features (L) we used in this work according to the 27 VK categories they belong to.

The Tables 5.8 and 5.9 enhance the *business applicability* of this work by presenting the ACAIM results of TRIM_E for thirty worldwide-familiar VK brands for $k = 4$ and $k = 5$ respectively; for illustration, we selected the higher k values and the biggest dataset of chapter (B3) to keep up more with most common application scenarios in social networks.

We observe that in all cases TRIM_E manages to early finish its learning (depicted by llr) and significantly improve the influence spread per round among *exploration* and *exploitation* that also accordingly increases the execution time per round with retaining the real-time nature of ACAIM.

Regarding the found features, we generally remark that a feature relative to “revelation” or “humor” is very popular in VK as it is part of most solutions. Further, we observe that in several cases there is an interesting *semantic connection* among brands and found features that makes sense in different extents under various ways; it can be characterized as (i) *expected* (e.g., “E-squire magazine” with “A celebrity magazine” for $k = 4$ and “A taxi order service” with “Taxi TaxovichkoF” for $k = 4$), (ii) *non-trivial* (e.g., “Great Britain” with “Football Europe” for $k = 5$ and “Shakira” with “Between us girls” for $k = 4$), (iii) *profound* (e.g., “Tools & garden equipment” with “Kontinental Hockey League (KHL)” for $k = 5$ and “Doctor Komarovsky” with “Believe Orthodoxy” for $k = 4$ and 5), (iv) *opposite* (e.g., “Real Madrid CF” with “FC Barcelona” for $k = 4$ and 5, and “Moto” with “Auto” for $k = 5$), and (v) *unexpected* (e.g., “Pharmaceutical products” with “Dating jokes” for $k = 5$ and “Helsinki airport” with “Modern mom” for $k = 5$).

5.5. Experimental Evaluation

Table 5.7: The IDs of brands (features in L) belonging to 27 categories of VK we considered in experiments. The first part of each category presents its 10 most popular brands in VK (depicted in decreasing $|S|$) and by aggregating all such first parts we form the $|L| = 270$ of data A. Similarly, both parts of each category correspond to the 20 most popular brands in VK and totally yield the $|L| = 540$ of data B. The VK social network page of each brand is publicly available to registered users just by using the ID of brand as follows: <https://vk.com/publicID>. In case of visit, we recommend using the Google Chrome browser so as to translate the content of pages since they mostly use the Russian language.

Animals	Auto, motor	Beauty, health	Celebrity	Cities, countries	Communication Services	Consumer Services	Culture, art	Education
32015300	23783750	39728801	135209264	20845272	34243323	170175796	147845620	1264877
110065937	151016287	45064245	45588871	127925490	44345402	173437170	38634441	25346844
65320054	29196806	28627911	35930308	127925061	52602834	171825683	142754704	33338722
34964358	38472666	34757875	104052041	27042171	24862387	35356518	99464023	149500863
33621085	148411246	28646177	41623203	37119411	18970929	49688236	19589874	34305040
35806476	39236729	46509740	139740824	31516466	149499917	49054793	143826157	36959676
32375815	31128331	23245066	41734504	69518720	28722213	41155241	66687279	44759043
29188817	40679923	35486195	26211015	66232651	41676683	166344567	180103132	149653328
23530818	133712630	36085261	134533652	34274053	33341280	46664792	31554488	33842750
59227775	51362846	23164653	225666	69547083	47786809	103075639	102090470	42307767
153142026	159415165	126100310	123675921	36338110	133310139	25198495	148230634	74595166
33798093	38894284	32922940	23546772	33156709	274672	39659800	40137153	61912980
36168102	90060110	107019848	174163184	88229325	20194367	98749490	47551578	28938560
30262661	125649823	16945011	42457435	40901026	49381792	81115275	61744518	72985232
82141367	43387535	125770269	23482802	36086936	38875560	41637433	122195630	40167434
43228812	23983339	49591829	3113588	144260965	114022615	25811852	74641828	29599237
35850395	163406855	34981365	95470601	32258596	1590042	66220245	37537193	45688121
105229582	24444828	126100569	46688098	37169648	126999658	168454811	26270763	103027469
159858626	153344099	29686754	30314549	32182751	66177799	82417135	26610299	46603834
50336153	37735548	26614831	94334686	33025155	161848270	96734258	96794207	142977738

Entertainment	Finance and Insurance	Food, recipes	Hobbies	Home, renovation	Internet	Job search	Media	Medicine	Music	Products, stores
57846937	22529055	43879004	55662720	36184135	22822305	12248221	29246653	133726577	27895931	36941068
26419239	124002407	39009769	29559271	32439535	28477986	31154183	48512305	55122354	45703770	24098496
45441631	34638472	48946342	31976785	38379853	2158488	29046529	29573241	29716377	34384434	34229261
43215063	84244100	32194285	34118551	42541008	162729615	44490261	22079806	27885374	35983383	30559917
22798006	43978738	18464856	24713873	32228890	35145657	78800961	18496184	149732802	22866546	28556858
58170807	166850908	46117626	79483347	41002749	23688663	48020228	9693056	43540875	28905875	35114569
12382740	38182092	40020627	43688579	90730773	79525017	34116496	101982925	49119584	29078047	80345103
31836774	20225241	48618580	34215577	137793489	139732885	56292539	15755094	51431740	29258893	39513007
23064236	73260851	47118092	23390361	123474518	72982321	23315499	104864	47591566	23108464	34483558
56160344	102218499	34451036	41883468	1792796	550910	85115964	29534144	42852777	105999460	10362317
40567146	140864751	83935640	76314525	36326284	31279395	36188056	24199209	160966228	48713061	40020304
26750264	144847188	42092461	43772432	64392368	41154660	4052789	73247559	45377300	34001496	96457590
460389	34980638	42025607	26776509	70275937	153766591	127560216	25386026	149401625	64977560	28673372
43776215	79995162	32509740	25397178	106277494	22884714	117282915	152992737	64964821	4335937	23616160
36164349	181526312	138816943	68229174	134655935	26456494	4808406	20035339	45251830	74653596	165599758
30179569	68349832	32231484	147720339	124764538	442	53809539	36792820	37875518	52599708	48210134
26669118	154346076	49694199	56048543	66630509	72378974	72355306	55264762	121816651	42440233	12530384
48319873	35144827	94216909	47679753	124303122	72866026	127738547	89493	49517102	26515827	34168005
38683579	104669972	28565318	23170931	42689002	40182105	63740509	24565142	488383801	27338836	24698811
33769500	75905130	34889014	23337480	136958443	43001537	82020695	16775977	36456996	33898099	49840023

Professional Services	Relationship, family	Restaurants	Social and public organizations	Sport	Tourism and Leisure	Transportation Services
111499611	91050183	30637940	71729358	71474813	346191	74611457
19542789	40498005	130641709	36166073	137451337	14897324	79459310
31577711	28890647	165062392	32194500	128350290	133668394	108040038
1163441	68114884	172149046	133180305	48940689	34543413	70228347
5425797	32432151	123287849	26307864	30428125	150802579	142153191
49690338	68895020	104616476	34137285	23693281	25117353	96196967
61796383	28293246	166652899	27794994	22746750	22558194	70270286
50512447	27470044	172231135	68016956	51812607	38363244	104141529
130698013	32651025	61879718	42701798	15326149	33445697	59131740
25276999	36318299	170513366	40587282	8722610	42763042	40738724
33301261	23758942	25300817	33382046	16202769	69580110	151316183
182438388	3555977	164409488	46987089	86224065	41053835	91708214
354372	20249656	53484080	31976441	12637219	21352492	26690472
27570276	34378420	59415411	54391852	23403635	63731512	170878251
57950162	55074079	9926079	38630769	64098698	30975621	53639653
57409266	24036559	49440926	36606428	30111136	133799113	153050613
87610407	24722253	87945351	39881081	23471538	26127512	66775477
31424891	68519692	78995926	31727306	28639294	54365037	82297106
30786950	44786979	164782094	25539323	138649060	146744120	45078595
38000521	24985591	127265072	50406378	40586683	35806721	93511310

Chapter 5. Adaptive Content-Aware Influence Maximization through Online Learning To Rank

Table 5.8: The ACAIM results of TRIM_E in regards to 30 selected brands for $k = 4$ in dataset B3 of data \mathbb{B} . The first table presents the found most influential features for each brand when exploration of TRIM_E ends; the second table shows the influence spread and execution time per round for each brand before and after exploration as also their cumulative values. All results are derived from 1 run of TRIM_E over 2000 rounds for each brand.

brand (id category name)	2nd-ranked feature (id category name)	3rd-ranked feature (id category name)
354372 Professional Serv. "A recruiting platform"	34215577 Hobbies "Anonymous human revelation"	29534144 Media "Humoristic information blog"
4th-ranked feature (id category name)	12382740 Entertainment "Epicenter of humor"	
16202769 Sport "Kontinental Hockey League (KHL)"	26419239 Entertainment "Laugh to tears"	12382740 Entertainment "Epicenter of humor"
4th-ranked feature (id category name)	135209264 Celebrity "The best Bot on VK"	
21352492 Tourism and Leisure "Great Britain"	34215577 Hobbies "Anonymous human revelation"	56106344 Entertainment "Anonymous revelation stories"
4th-ranked feature (id category name)	22746750 Sport "FC Barcelona"	
22522055 Finance and Insurance "Sberbank"	26419239 Entertainment "Laugh to tears"	135209264 Celebrity "The best Bot on VK"
4th-ranked feature (id category name)	12382740 Entertainment "Epicenter of humor"	
22558194 Tourism and Leisure "USA"	56106344 Entertainment "Anonymous revelation stories"	34215577 Hobbies "Anonymous human revelation"
4th-ranked feature (id category name)	135209264 Celebrity "The best Bot on VK"	
23403635 Sport "Real Madrid CF"	22746750 Sport "FC Barcelona"	12382740 Entertainment "Epicenter of humor"
4th-ranked feature (id category name)	26419239 Entertainment "Laugh to tears"	
24098496 Products, stores "E-squire magazine"	34215577 Hobbies "Anonymous human revelation"	66687279 Culture, art "A celebrity magazine"
4th-ranked feature (id category name)	40498005 Relationship, family "Relationship psychology"	
24565142 Media "National Geographic"	34215577 Hobbies "Anonymous human revelation"	12382740 Entertainment "Epicenter of humor"
4th-ranked feature (id category name)	66687279 Culture, art "A celebrity magazine"	
27570276 Professional Serv. "Tool & garden equipment"	79525017 Internet "Sweepstakes"	31976785 Hobbies "Science & Technology"
4th-ranked feature (id category name)	123474518 Home, renovation "Builder"	
30559917 Products, stores "Business Strategy"	34215577 Hobbies "Anonymous human revelation"	29246653 Media "Sarcastic and funny news"
4th-ranked feature (id category name)	56106344 Entertainment "Anonymous revelation stories"	
36184135 Home, renovation "Interior design ideas"	34215577 Hobbies "Anonymous human revelation"	56106344 Entertainment "Anonymous revelation stories"
4th-ranked feature (id category name)	460389 Entertainment "Fresh memes uncensored"	
36941068 Products, stores "AliExpress"	34215577 Hobbies "Anonymous human revelation"	460389 Entertainment "Fresh memes uncensored"
4th-ranked feature (id category name)	12382740 Entertainment "Epicenter of humor"	
37119411 Cities, countries "Moscow"	34215577 Hobbies "Anonymous human revelation"	460389 Entertainment "Fresh memes uncensored"
4th-ranked feature (id category name)	56106344 Entertainment "Anonymous revelation stories"	
37537193 Culture, art "Couple relationships"	29599237 Education "Believe Orthodoxy"	56106344 Entertainment "Anonymous revelation stories"
4th-ranked feature (id category name)	29246653 Media "Sarcastic and funny news"	
40679923 Auto, motor "Moto"	26419239 Entertainment "Laugh to tears"	135209264 Celebrity "The best Bot on VK"
4th-ranked feature (id category name)	12382740 Entertainment "Epicenter of humor"	
43001537 Internet "Apple"	12382740 Entertainment "Epicenter of humor"	26419239 Entertainment "Laugh to tears"
4th-ranked feature (id category name)	22746750 Sport "FC Barcelona"	
44257435 Celebrity "Shakira"	22746750 Sport "FC Barcelona"	34215577 Hobbies "Anonymous human revelation"
4th-ranked feature (id category name)	36166073 Social and public organizations "Between us girls"	
48383801 Medicine "Pharmaceutical products"	79525017 Internet "Sweepstakes"	34215577 Hobbies "Anonymous human revelation"
4th-ranked feature (id category name)	30179569 Entertainment "Slaughter humor"	
48940689 Sport "Olympic Games"	56106344 Entertainment "Anonymous revelation stories"	73247559 Media "Show business news"
4th-ranked feature (id category name)	45441631 Entertainment "Funny & ridiculous jokes"	
49119584 Medicine "Doctor Komarovsky"	34215577 Hobbies "Anonymous human revelation"	56106344 Entertainment "Anonymous revelation stories"
4th-ranked feature (id category name)	29599237 Education "Believe Orthodoxy"	
53484080 Restaurants "Dodo Pizza"	26419239 Entertainment "Laugh to tears"	56106344 Entertainment "Anonymous revelation stories"
4th-ranked feature (id category name)	34215577 Hobbies "Anonymous human revelation"	
53639653 Transportation Serv. "A suburban passenger company"	135209264 Celebrity "The best Bot on VK"	89493 Media "A dance radio station"
4th-ranked feature (id category name)	34215577 Hobbies "Anonymous human revelation"	
66775477 Transportation Serv. "Helsinki airport"	34215577 Hobbies "Anonymous human revelation"	29534144 Media "Humoristic information blog"
4th-ranked feature (id category name)	32015300 Animals "A paradise for cat lovers"	
74611457 Transportation Serv. "A taxi order service"	12382740 Entertainment "Epicenter of humor"	79525017 Internet "Sweepstakes"
4th-ranked feature (id category name)	70270286 Transportation Serv. "Taxi TaxivichkoF"	
103075639 Consumer Serv. "A clothing & footwear care product"	29534144 Media "Humoristic information blog"	79525017 Internet "Sweepstakes"
4th-ranked feature (id category name)	66687279 Culture, art "A celebrity magazine"	
104052041 Celebrity "Natalia Oreiro"	26419239 Entertainment "Laugh to tears"	34215577 Hobbies "Anonymous human revelation"
4th-ranked feature (id category name)	12382740 Entertainment "Epicenter of humor"	
126100569 Beauty, health "Hairstyles & Haircuts"	56106344 Entertainment "Anonymous revelation stories"	34215577 Hobbies "Anonymous human revelation"
4th-ranked feature (id category name)	32015300 Animals "A paradise for cat lovers"	
139732885 Internet "Joom"	73247559 Media "Show business news"	56106344 Entertainment "Anonymous revelation stories"
4th-ranked feature (id category name)	26419239 Entertainment "Laugh to tears"	
151316183 Transportation Serv. "Lufthansa"	34215577 Hobbies "Anonymous human revelation"	29534144 Media "Humoristic information blog"
4th-ranked feature (id category name)	24098496 Products, stores "E-squire magazine"	
165599758 Products, stores "Xiaomi"	26419239 Entertainment "Laugh to tears"	460389 Entertainment "Fresh memes uncensored"
4th-ranked feature (id category name)	56106344 Entertainment "Anonymous revelation stories"	

brand (id S)	llr	avg. spread per round until llr	avg. spread per round after llr	spread	avg. time per round until llr	avg. time per round after llr	time
354372 59821	508	4.96	51.19	78909	0.83 sec	1 sec	32.28 min
16202769 138344	501	105.36	385.14	630119	1.95 sec	2.62 sec	82.68 min
21352492 20038	474	11.02	42.26	69724	0.27 sec	0.34 sec	11.11 min
22522055 222795	547	60.61	369.24	569672	3.43 sec	3.94 sec	127.9 min
22558194 53809	513	45.29	157.87	257993	0.79 sec	1.1 sec	34.4 min
23403635 107830	504	253.57	390.34	711751	1.7 sec	1.99 sec	64.57 min
24098496 317643	535	443.39	780.91	1381252	4.88 sec	5.68 sec	184.2 min
24565142 139134	505	56.28	170.92	283960	1.9 sec	2.32 sec	74.72 min
27570276 44052	474	9.72	86.79	137056	0.63 sec	0.87 sec	27.49 min
30559917 201038	544	34.99	99.6	164063	2.68 sec	3.08 sec	100.01 min
36184135 449805	536	75.14	314.6	500855	6.1 sec	7.21 sec	232.57 min
36941068 304871	535	21.17	160.86	246994	4.15 sec	4.87 sec	157.6 min
37119411 175994	517	18.22	193.26	296033	2.19 sec	2.46 sec	80.7 min
37537193 50057	471	14.89	38.17	65381	0.65 sec	0.82 sec	26.5 min
40679922 89837	519	99.91	161.59	291184	1.24 sec	1.34 sec	44.51 min
43001537 128777	523	22.73	79.77	129725	1.69 sec	1.93 sec	62.99 min
44257435 33375	502	112.88	326.16	545261	0.56 sec	0.76 sec	24.14 min
48383801 22067	450	6.34	129.87	204158	0.29 sec	0.57 sec	17.2 min
48940689 213802	542	133.42	418.75	682860	3.38 sec	3.99 sec	128.67 min
49119584 49144	444	16.45	62.18	104060	0.72 sec	0.83 sec	27.33 min
53484080 69284	492	29.26	154.05	246718	1.07 sec	1.22 sec	39.89 min
53639653 3077	409	1.99	10.61	17707	0.04 sec	0.05 sec	1.93 min
66775477 4121	409	3.92	16.52	27892	0.06 sec	0.08 sec	2.71 min
74611457 28455	457	14.18	77.89	126671	0.4 sec	0.62 sec	19.23 min
103075639 9314	425	5.77	39.28	64324	0.12 sec	0.2 sec	6.51 min
104052041 43913	522	222.62	620.62	1033487	0.98 sec	1.35 sec	42.47 min
126100569 245088	536	95.61	207.92	355646	3.28 sec	3.76 sec	122.11 min
139732885 204755	518	42.4	218.15	345276	2.96 sec	3.29 sec	107.88 min
151316183 6271	450	13.58	51.4	85795	0.09 sec	0.15 sec	4.81 min
165599758 312288	504	94.89	530.24	841075	4.85 sec	5.37 sec	176.24 min

5.5. Experimental Evaluation

Table 5.9: The ACAIM results of TRIM_E in regards to 30 selected brands for $k = 5$ in dataset B3 of data \mathbb{B} . The first table presents the found most influential features for each brand when exploration of TRIM_E ends; the second table shows the influence spread and execution time per round for each brand before and after exploration as also their cumulative values. All results are derived from 1 run of TRIM_E over 2000 rounds for each brand.

brand (id category name)	2nd-ranked feature (id category name)	3rd-ranked feature (id category name)
354372 Professional Serv. "A recruiting platform"	34215577 Hobbies "Anonymous human revelation"	460389 Entertainment "Fresh memes uncensored"
4th-ranked, 5th-ranked feature (id category name)	28477986 Internet "Unbelievable humoristic news"	12382740 Entertainment "Epicenter of humor"
16202769 Sport "Kontinental Hockey League (KHL)"	26419239 Entertainment "Laugh to tears"	22746750 Sport "FC Barcelona"
4th-ranked, 5th-ranked feature (id category name)	12382740 Entertainment "Epicenter of humor"	45441631 Entertainment "Funny & ridiculous jokes"
21352492 Tourism and Leisure "Great Britain"	73247559 Media "Show business news"	22746750 Sport "FC Barcelona"
4th-ranked, 5th-ranked feature (id category name)	23693281 Sport "Football Europe"	31976785 Hobbies "Science & Technology"
22520255 Finance and Insurance "Sberbank"	26419239 Entertainment "Laugh to tears"	45441631 Entertainment "Funny & ridiculous jokes"
4th-ranked, 5th-ranked feature (id category name)	56106344 Entertainment "Anonymous revelation stories"	135209264 Celebrity "The best Bot on VK"
22558194 Tourism and Leisure "USA"	56106344 Entertainment "Anonymous revelation stories"	22746750 Sport "FC Barcelona"
4th-ranked, 5th-ranked feature (id category name)	66687279 Culture, art "A celebrity magazine"	135209264 Celebrity "The best Bot on VK"
23403635 Sport "Real Madrid CF"	22746750 Sport "FC Barcelona"	12382740 Entertainment "Epicenter of humor"
4th-ranked, 5th-ranked feature (id category name)	31836774 Entertainment "Cool tricks for fun"	12637219 Sport "European Football"
24098496 Products, stores "Esquire magazine"	34215577 Hobbies "Anonymous human revelation"	40498005 Relationship, family "Relationship psychology"
4th-ranked, 5th-ranked feature (id category name)	26307864 Social and public organizations "The art of reality"	28477986 Internet "Unbelievable humoristic news"
24565142 Media "National Geographic"	34215577 Hobbies "Anonymous human revelation"	56106344 Entertainment "Anonymous revelation stories"
4th-ranked, 5th-ranked feature (id category name)	29559271 Hobbies "Science"	66687279 Culture, art "A celebrity magazine"
27570276 Professional Serv. "Tools & garden equipment"	25397178 Hobbies "Male thoughts"	12382740 Entertainment "Epicenter of humor"
4th-ranked, 5th-ranked feature (id category name)	31976785 Hobbies "Science & Technology"	16202769 Sport "Kontinental Hockey League (KHL)"
30559917 Products, stores "Business Strategy"	34215577 Hobbies "Anonymous human revelation"	12382740 Entertainment "Epicenter of humor"
4th-ranked, 5th-ranked feature (id category name)	56106344 Entertainment "Anonymous revelation stories"	460389 Entertainment "Fresh memes uncensored"
36184135 Home, renovation "Interior design ideas"	34215577 Hobbies "Anonymous human revelation"	460389 Entertainment "Fresh memes uncensored"
4th-ranked, 5th-ranked feature (id category name)	56106344 Entertainment "Anonymous revelation stories"	36164349 Entertainment "Satirical humor"
36941068 Products, stores "AliExpress"	34215577 Hobbies "Anonymous human revelation"	26419239 Entertainment "Laugh to tears"
4th-ranked, 5th-ranked feature (id category name)	135209264 Celebrity "The best Bot on VK"	79525017 Internet "Sweepstakes"
37119411 Cities, countries "Moscow"	34215577 Hobbies "Anonymous human revelation"	91050183 Relationship, family "A dating Bot"
4th-ranked, 5th-ranked feature (id category name)	66687279 Culture, art "A celebrity magazine"	29534144 Media "Humoristic information blog"
37537193 Culture, art "Couple relationships"	29599237 Education "Believe Orthodoxy"	56106344 Entertainment "Anonymous revelation stories"
4th-ranked, 5th-ranked feature (id category name)	460389 Entertainment "Fresh memes uncensored"	3338722 Education "Vocabulary"
40679923 Auto, motor "Moto"	135209264 Celebrity "The best Bot on VK"	133712630 Auto, motor "Auto Blog"
4th-ranked, 5th-ranked feature (id category name)	26419239 Entertainment "Laugh to tears"	23783750 Auto, motor "Auto"
43001537 Internet "Apple"	73247559 Media "Show business news"	29573241 Media "NR.Music"
4th-ranked, 5th-ranked feature (id category name)	12382740 Entertainment "Epicenter of humor"	26419239 Entertainment "Laugh to tears"
44257435 Celebrity "Shakira"	56106344 Entertainment "Anonymous revelation stories"	34215577 Hobbies "Anonymous human revelation"
4th-ranked, 5th-ranked feature (id category name)	22746750 Sport "FC Barcelona"	73247559 Media "Show business news"
48383801 Medicine "Pharmaceutical products"	79525017 Internet "Sweepstakes"	34215577 Hobbies "Anonymous human revelation"
4th-ranked, 5th-ranked feature (id category name)	28477986 Internet "Unbelievable humoristic news"	44786979 Relationship, family "Dating jokes"
48940689 Sport "Olympic Games"	12382740 Entertainment "Epicenter of humor"	56106344 Entertainment "Anonymous revelation stories"
4th-ranked, 5th-ranked feature (id category name)	34215577 Hobbies "Anonymous human revelation"	31836774 Entertainment "Cool tricks for fun"
49119584 Medicine "Doctor Komarovsky"	34215577 Hobbies "Anonymous human revelation"	56106344 Entertainment "Anonymous revelation stories"
4th-ranked, 5th-ranked feature (id category name)	29599237 Education "Believe Orthodoxy"	29246653 Media "Sarcastic and funny news"
53484080 Restaurant "Dodo Pizza"	26419239 Entertainment "Laugh to tears"	45441631 Entertainment "Funny & ridiculous jokes"
4th-ranked, 5th-ranked feature (id category name)	135209264 Celebrity "The best Bot on VK"	31976785 Hobbies "Science & Technology"
53639653 Transportation Serv. "A suburban passenger company"	43540875 Medicine "Humoristic medicine"	135209264 Celebrity "The best Bot on VK"
4th-ranked, 5th-ranked feature (id category name)	460389 Entertainment "Fresh memes uncensored"	133799113 Tourism and Leisure "Civilization cemetery"
66775477 Transportation Serv. "Helsinki airport"	79525017 Internet "Sweepstakes"	36959676 Education "Interesting facts"
4th-ranked, 5th-ranked feature (id category name)	135209264 Celebrity "The best Bot on VK"	55074079 Relationship, family "Modern mom"
74611457 Transportation Serv. "A taxi order service"	79525017 Internet "Sweepstakes"	12382740 Entertainment "Epicenter of humor"
4th-ranked, 5th-ranked feature (id category name)	29246653 Media "Sarcastic and funny news"	23783750 Auto, motor "Auto"
103075639 Consumer Serv. "A clothing & footwear care product"	135209264 Celebrity "The best Bot on VK"	29534144 Media "Humoristic information blog"
4th-ranked, 5th-ranked feature (id category name)	29573241 Media "NR.Music"	34215577 Hobbies "Anonymous human revelation"
104052041 Celebrity "Natalia Oreiro"	29246653 Media "Sarcastic and funny news"	26419239 Entertainment "Laugh to tears"
4th-ranked, 5th-ranked feature (id category name)	56106344 Entertainment "Anonymous revelation stories"	460389 Entertainment "Fresh memes uncensored"
126100569 Beauty, health "Hairstyles & Haircuts"	56106344 Entertainment "Anonymous revelation stories"	34215577 Hobbies "Anonymous human revelation"
4th-ranked, 5th-ranked feature (id category name)	26419239 Entertainment "Laugh to tears"	31836774 Entertainment "Cool tricks for fun"
139732885 Internet "Joom"	26419239 Entertainment "Laugh to tears"	23848202 Celebrity "Egor Kreed"
4th-ranked, 5th-ranked feature (id category name)	45441631 Entertainment "Funny & ridiculous jokes"	12382740 Entertainment "Epicenter of humor"
151316183 Transportation Serv. "Lufthansa"	34215577 Hobbies "Anonymous human revelation"	40567146 Entertainment "Worldwide entertaining news"
4th-ranked, 5th-ranked feature (id category name)	29534144 Media "Humoristic information blog"	26419239 Entertainment "Laugh to tears"
165599758 Products, stores "Xiaomi"	135209264 Celebrity "The best Bot on VK"	12382740 Entertainment "Epicenter of humor"
4th-ranked, 5th-ranked feature (id category name)	26419239 Entertainment "Laugh to tears"	56106344 Entertainment "Anonymous revelation stories"

brand (id S)	llr	avg. spread per round until llr	avg. spread per round after llr	spread	avg. time per round until llr	avg. time per round after llr	time
354372 59821	285	5.87	60.57	105556	0.76 sec	1 sec	32.55 min
16202769 138344	289	89.29	320.32	573875	1.95 sec	2.54 sec	82.73 min
21352492 20038	267	11.64	35.01	63791	0.25 sec	0.33 sec	10.97 min
22520255 222795	283	51.92	363.06	638083	3.24 sec	3.84 sec	126.32 min
22558194 53809	297	42	127.44	229517	0.76 sec	1.01 sec	32.93 min
23403635 107830	290	227	331.61	632901	1.64 sec	1.82 sec	60.59 min
24098496 317643	301	447.29	620.99	1189713	4.91 sec	5.56 sec	183.99 min
24565142 139134	291	52.68	167.02	300769	1.83 sec	2.33 sec	76.04 min
27570276 44052	263	8.96	32.15	58218	0.58 sec	0.72 sec	23.74 min
30559917 201038	305	36.29	117.61	210425	2.68 sec	2.98 sec	98.86 min
36184135 449805	312	99.02	339.02	603175	6.04 sec	7.1 sec	233.3 min
36941068 304871	300	25.23	178.22	310548	3.96 sec	4.99 sec	162.78 min
37119411 175994	285	20.77	190.02	331822	2.14 sec	2.53 sec	83.44 min
37537193 50057	265	15.32	32.06	59688	0.64 sec	0.79 sec	26.01 min
40679923 89837	277	100.12	154.91	294655	1.22 sec	1.42 sec	47.02 min
43001537 128777	282	28.66	105.29	188976	1.7 sec	1.93 sec	64.06 min
44257435 33375	287	57.79	265.5	471388	0.48 sec	0.71 sec	23.16 min
48383801 22067	257	7.48	143.44	251944	0.27 sec	0.55 sec	17.6 min
48940689 213802	290	76.34	353.2	626113	3.04 sec	3.64 sec	119.58 min
49119584 49144	268	12.88	52.47	94346	0.67 sec	0.82 sec	26.94 min
53484080 69289	295	20.68	171.54	298579	1 sec	1.22 sec	39.98 min
53639653 3077	239	0.64	7.43	13247	0.04 sec	0.05 sec	1.86 min
66775477 4121	241	1.22	12.65	22550	0.05 sec	0.08 sec	2.97 min
74611457 28455	267	10.1	82.24	145232	0.36 sec	0.63 sec	20.06 min
103075639 9314	241	2.95	32.13	57240	0.11 sec	0.17 sec	5.85 min
104052041 43913	305	142.09	524.15	931774	0.79 sec	1.3 sec	41.55 min
126100569 245088	298	103.14	203.05	376331	3.25 sec	3.64 sec	120.7 min
139732885 204755	292	32.59	199.6	350442	2.85 sec	3.34 sec	110.06 min
151316183 6271	264	9.35	43.6	78166	0.1 sec	0.16 sec	5.25 min
165599758 312288	281	73.75	528.8	929742	4.63 sec	5.39 sec	177.87 min

5.6 Conclusion

In this chapter, we proposed the ACAIM problem and realistically solve it on several VK datasets. In particular, we utilized for first time an OLR framework for IM purposes, we introduced the CATRID propagation model to enable OLR for ACAIM, we deployed a simulator to express a real feedback environment based on VK posts, we developed three learners to solve ACAIM, and we presented a thorough experimental evaluation that illustrates the importance of ACAIM to social network industry. A detailed business applicability study is also included which, by considering worldwide known brands, it further stresses the suitability of ACAIM in real world.

5.6. Conclusion

Chapter 6

A Content Recommendation Policy for Gaining Subscribers

How can we recommend content for a *brand* agent to use over a series of rounds so as to gain new subscribers to its social network page? The *Influence Maximization* (IM) problem seeks a set of k users, and its content-aware variants seek a set of k post features, that achieve, in both cases, an objective of expected influence in a social network. However, apart from raw influence, it is also relevant to study gain in subscribers, as long-term success rests on the subscribers of a brand page; classic IM may select k users from the subscriber set, and content-aware IM starts the post's propagation from that subscriber set. In this chapter, we propose a novel *content recommendation policy* to a brand agent for *Gaining Subscribers by Messaging* (GSM) over many rounds. In each round, the brand agent messages a fixed number of social network users and invites them to visit the brand page aiming to gain their subscription, while its most recently published content consists of features that intensely attract the preferences of the invited users. To solve GSM, we find, in each round, which content features to publish and which users to notify aiming to maximize the cumulative subscription gain over all rounds. We deploy three GSM solvers, named **RANDOM**, **SCAN**, and **SUBSTITUTE**, and we experimentally evaluate their performance based on VKontakte (VK) posts by considering different user sets and feature sets. Our experimental results show that **SUBSTITUTE** provides the best solution, as it is significantly more efficient than **SCAN** with a minor loss of efficacy and clearly more efficacious than **RANDOM** with competitive efficiency.

6.1 Introduction

The problem of *Influence Maximization* (IM) [LFWT18] is relevant and useful to stakeholders (henceforward, *brands*) that pursue viral marketing campaigns in social

networks. The classic IM [KKT03] seeks k users that maximize the influence of a *fixed* post in a network; the inverse variant of IM [ITTK17] seeks k content features¹ to form a viral post that starts its diffusion from a *fixed* set of initial adopters.

Nowadays, most brands maintain social network pages for advertising purposes, since social network users choose to *follow* pages they are interested in; these followers are the *subscribers* of a brand. Yet, in the IM literature, subscribers are usually taken for granted [ITTK17, KLK20] or ignored [KKT03, LFWT18]. This neglect limits the applicability of IM, as new brands have no concrete way to gain their first subscribers, while established brands ignore the impact of their subscribers on influence diffusion. Further, in the real world, the network topology is usually not known on its whole [SS13, HS15, LCCM19], whereas subscribers are always known, even if that knowledge requires explicit on-demand retrieval. Therefore, the need arises to focus on subscribers and study how brands can gain subscribers, so as to apply existing IM techniques in conventional or new ways (e.g., classic IM could select k users from the set of subscribers based on their popularity and loyalty).

In this chapter, we propose a novel multi-round *content recommendation policy* that a brand agent/advertiser can use to *Gain Subscribers by Messaging* (GSM). As the GSM process takes place over many rounds, we deploy three algorithms that solve GSM in a non-adaptive way (beforehand) for all rounds. Our solutions recommend to the advertiser, in each round, which k content features to publish and which m non-subscriber users to notify of those k features so as to maximize chances to gain the subscription of those m users. The notification is done by messaging (e.g., a short message acting like an invitation to visit the brand page), and each user is notified once; that user is never notified again. The best GSM solver is the one that achieves the maximum *subscription gain* over all rounds. We define subscription gain (SG) as a weighted sum depicting the aggregate preference of m users for k features. For any two (k, m) solutions that achieve similar SG , we consider the cumulative (no duplicates allowed) *reach* of their respective m users to select the (k, m) solution with the maximum reach; the reach (R) of a user is equal to her out-degree.

GSM naturally applies to social networks, such as VK². We consider real VK posts to fine-tune user preferences for our experiments. In particular, VK strictly allows 20 messages per 12 hours to any user who has a VK account and wishes to send a message to any other non-friend VK user. Thus, each message is valuable for the advertiser and constitutes a single chance to attract the attention of notified user. By solving GSM, the advertiser prioritizes the publishing (as each round has priority over the next round) of the right k -size content for the right m users to maximize the gain of subscribers. If, alternatively, the advertiser were to apply a random messaging

¹We consider that each *feature* corresponds to a specific social network page.

²<https://vk.com/>

policy, then she would gain subscribers at a lower pace, and also face the danger of losing access to her page for some period due to spam reports sent by notified users to the VK company; a user invited to visit a page of no interest may report spamming.

We summarize our contributions as follows: (1) we propose the GSM problem; (2) we deploy three GSM solvers, named **RANDOM**, **SCAN**, and **SUBSTITUTE**; and (3) we provide a rich experimental evaluation that verifies the superiority of **SUBSTITUTE** over other solvers; to the best of our knowledge, the problem of gaining subscribers by using content has not been studied previously.

6.2 GSM Solvers

We define the GSM problem as follows: Given a social network $G = (V, E)$ with $|V|$ users and $|E|$ edges, a feature universe L , a weighted feature set F_v of size $|L|$ capturing the preferences of each user v , a budget k , a limited number of m notification messages, a similarity threshold d , and a number of rounds n , find in each round t what k content features to publish and which m users to notify so as to maximize the cumulative subscription gain SG over n rounds:

$$SG = \max \sum_{t=1}^n SG_t(k, m)$$

6.2.1 The Solver **RANDOM**

Algorithm **RANDOM** presents a baseline that solves GSM. In each round t , **RANDOM** uniformly at random selects m users from V not selected in previous rounds and focuses on selecting features (Lines 3–5). In more detail, **RANDOM** searches for the k -size feature set that yields the maximum gain in round t (*roundBest.SG*) with regard to the selected m users, and increases the cumulative SG and reach R (Lines 6–10). Lastly, it keeps track of all notified users (Line 11) to avoid messaging them again.

6.2.2 The Solver **SCAN**

The Algorithm **SCAN** presents the solver **SCAN**. In a nutshell, **SCAN** computes and stores in each round t the best solution for each k -size feature combination by examining all m -size user combinations, so as to find the *roundBest* for t , and skips in next rounds any feature combinations that do not need to be processed.

We indicate the algorithm's workflow by marking three execution steps with bold numbers above Line 7, Line 20, and Line 23. In the first step (Lines 7–19), if all users notified in the previous round are *not* contained in best current feature set c_f , then we use c_f to possibly update *roundBest* using similarity threshold d . If the first step

Algorithm RANDOM

```

Input      :  $G, L, F_v$ 
Output     :  $SG, R$  // cumulative subscription gain and reach over  $n$  rounds in GSM
Param.    :  $k, m, d, n$  //  $d$  is not used here
1  $SG = 0; R = 0; deletedV = \emptyset;$ 
2 for  $t = 1, \dots, n$  do
3    $roundBest.users = \emptyset; roundBest.features = \emptyset; roundBest.SG = 0;$ 
    $roundBest.R = 0$ ; // initialize the best solution for round  $t$ 
4   Randomly select  $m$  users from  $G.V$  to form  $roundBest.users$  such as
    $|roundBest.users| = m$  and none of these  $m$  users is included in  $deletedV$ ;
5    $roundBest.R = \text{COMPUTEREACH}(roundBest.users, G);$ 
6   for each feature combination  $c_f$  from  $k$  feature combinations of  $L$  do
7      $c_f.SG = \text{COMPUTESG}(roundBest.users, c_f, F_v);$ 
8     if  $c_f.SG > roundBest.SG$  then
9        $roundBest.SG = c_f.SG; roundBest.features = c_f;$ 
10     $SG = SG + roundBest.SG; R = R + roundBest.R;$ 
11    for each user  $v \in roundBest.users$  do  $deletedV.insert(v);$ 
12 return  $SG, R;$ 

```

is not executed (condition in Line 7 is false), then we move to the second step (Lines 20–22). This step is the most costly part of **SCAN** as it examines all sets of m users sequentially to find the best match for feature set c_f . After this processing, the third step (Line 23) possibly updates $roundBest$ based on the found feature set c_f .

6.2.3 The Solver SUBSTITUTE

The Algorithm **SUBSTITUTE** presents the solver **SUBSTITUTE**. This solver addresses the main bottleneck of **SCAN**, which is the examination of all user combinations in its second step. In particular, **SUBSTITUTE** adds an intermediate step (Lines 10–32) that creates only the necessary m -size user combinations to find the best match for feature set c_f . To do that, it utilizes the structure $selV$ (initialized in Lines 2–4) that stores for each c_f a set of pairs corresponding to users associated with their achieved SG in c_f , organized in descending order by their SG values.

Specifically, the first m entries of $selV[c_f]$ form the first user combination (Line 10). Then, we compare each entry $e2$ (Line 12) having a position greater than m in $selV[c_f]$ with each previous entry $e1$ of $selV[c_f]$ (Line 14) to check whether $e2$ can substitute $e1$. If a substitution happens (Line 23), there may exist a best solution for c_f having $e2$ in place of $e1$. Otherwise, if $e2$ cannot substitute the first compared $e1$, then $e2$ cannot substitute any $e1$, and also this holds for any next $e2$, resulting in early termination (Line 30) of $e2$ -processing. This termination is beneficial as it ends the generation of new user combinations for c_f , and so it incurs less overhead in third step (Line 33). To further enhance that termination, we add a condition in Line 28 that counts instances of *invisibility*. A case of invisibility occurs when the current

Algorithm SCAN

```

Input      : $G, L, F_v$ 
Output     : $SG, R$  // cumulative subscription gain and reach over  $n$  rounds in GSM
Param.    : $k, m, d, n$ 

1 for each feature combination  $c_f$  from  $k$  feature combinations of  $L$  do  $fC[c_f] = \emptyset$ ;
2 for each user combination  $c_u$  from  $m$  user combinations of  $V$  do  $uC[c_u] = -1$ ;
3  $SG = 0; R = 0; deletedV = \emptyset$ ;
4 for  $t = 1, \dots, n$  do
5    $roundBest.users = \emptyset; roundBest.features = \emptyset; roundBest.SG = 0$ ;
    $roundBest.R = 0$ ; // initialize the best solution for round  $t$ 
6   for each feature combination  $c_f \in fC$  do
7     // 1. Check to may skip computation of  $fC[c_f]$  and Update  $roundBest$ 
8     if all users in  $deletedV$  are not contained in  $fC[c_f]$  then
9        $c_f.SG = COMPUTE(SG(fC[c_f], c_f, F_v))$ ;
10      if  $roundBest.SG \geq c_f.SG$  then
11         $diff = (roundBest.SG - c_f.SG) \div c_f.SG$ ;
           if  $diff > d$  then continue;
12      else
13         $diff = (c_f.SG - roundBest.SG) \div roundBest.SG$ ;
14        if  $diff > d$  then
15          if  $uC[fC[c_f]] = -1$  then
16             $uC[fC[c_f]] = COMPUTEReach(fC[c_f], G)$ ;
17             $roundBest.users = fC[c_f]$ ;
             $roundBest.features = c_f$ ;  $roundBest.SG = c_f.SG$ ;
             $roundBest.R = uC[fC[c_f]]$ ; continue;
18
19     Repeat lines 15-16 to compute  $uC[fC[c_f]]$ ;
20     if  $uC[fC[c_f]] > roundBest.R$  or ( $uC[fC[c_f]] = roundBest.R$ 
21     and  $c_f.SG > roundBest.SG$ ) then Update  $roundBest$  as in line 17;
           // continue is executed either the condition in if is true or false
22
23     // 2. Compute  $fC[c_f]$  with the examination of each  $c_u \in uC$  (loop here!)
24     Let  $crnt\_cu$  be the current best user combination stored in  $fC[c_f]$ ;
25     Let  $next\_cu$  be the next  $c_u \in uC$  for examination to may update  $fC[c_f]$ ;
26     Compare  $crnt\_cu$  with  $next\_cu$  by following a similar process to lines 8-19
           so as to may update  $fC[c_f]$  and  $crnt\_cu$  with  $next\_cu$ ;
27     // 3. Update  $roundBest$ 
28     Repeat lines 8-19 to update  $roundBest$ ;
29
30      $SG = SG + roundBest.SG; R = R + roundBest.R$ ;
31      $deletedV = \emptyset$ ; for each user  $v \in roundBest.users$  do  $deletedV.insert(v)$ ;
32     Delete each  $c_u \in uC$  that contains at least one user  $v \in deletedV$ ;
33
34 return  $SG, R$ ;

```

combination that includes e_1 has no less reach than the same combination with e_2 in place of e_1 , and also the atomic reach of e_1 is no less than the atomic reach of e_2 ; then, in most cases e_1 is a better option than e_2 . If such evidence is observed for all the e_1 to which we compare e_2 , then we mark e_2 as invisible henceforward (Line 31). Intuitively, the invisibility of e_2 means that it is very likely that no user combination derived by replacing any e_1 with e_2 , would constitute the best solution for c_f , so we opt to ignore them.

Algorithm SUBSTITUTE

Input : G, L, F_v
Output : SG, R // cumulative subscription gain and reach over n rounds in GSM
Param. : k, m, d, n

```

1 for each feature combination  $c_f$  from  $k$  feature combinations of  $L$  do  $fC[c_f] = \emptyset$ ;  

2 for each feature combination  $c_f \in fC$  do  

3    $v.SG = \text{COMPUTESG}(v, c_f, F_v)$ ;  $allV[c_f].insert((v, v.SG))$ ; //  $v.SG$  desc sort  

4   Form  $selV[c_f]$  by taking the first  $m * n$  pairs of  $allV[c_f]$ ; // plus  $d$ -extensions  

5    $SG = 0; R = 0; deletedV = \emptyset$ ;  

6   for  $t = 1, \dots, n$  do  

7      $roundBest.users = \emptyset; roundBest.features = \emptyset; roundBest.SG = 0$ ;  

      $roundBest.R = 0$ ; // initialize the best solution for round  $t$   

8     for each feature combination  $c_f \in fC$  do  

      // 1. Check to may skip computation of  $fC[c_f]$  and Update  $roundBest$   

9      Repeat lines 7-19 of SCAN with only difference that now  $c_f.R =$   

        $\text{COMPUTEREACH}(fC[c_f], G)$  is used and so  $c_f.R$  replaces  $uC[fC[c_f]]$ ;  

      // 2. Create only the necessary  $m$ -size user combinations ( $uC$ ) for  $c_f$   

10     Form the first  $c_u$  by taking the first  $m$  pairs of  $selV[c_f]$  and set  

        $uC[c_u] = -1$ ; // each  $c_u$  is a set of pairs  $(v, v.SG)$  here  

11      $offset = m; c_f.invisible = \emptyset$ ; // it contains each invisible  $v \in selV[c_f]$   

12     for  $e2 = selV[c_f].get(m + 1), \dots, selV[c_f].last()$  do  

13        $e2\_out = G.outEdges(e2.v); subs = \emptyset; invis = 0; travs = 0$ ;  

14       for  $e1 = selV[c_f].get(offset), \dots, selV[c_f].first()$  do  

15         if  $e1.v \in c_f.invisible$  then {  $invis += 1; travs += 1$ ; continue }  

16          $e1\_comb = \emptyset; e2\_comb = \emptyset; denom.SG = e2.v.SG$ ;  

17         for  $ei = selV[c_f].first(), \dots, selV[c_f].get(m)$  do  

18           if  $ei = e1$  then continue;  

19            $e1\_comb.insert(ei.v); e2\_comb.insert(ei.v)$ ;  

20            $denom.SG = denom.SG + ei.v.SG$ ;  

21           After  $m - 1$  loop executions break;  

22            $diff = (e1.v.SG - e2.v.SG) / denom.SG$ ;  

23           if  $diff > d$  then break; else  $subs.insert(e1)$ ; //  $e2$  can sbst  $e1$   

24            $e1\_comb.insert(e1.v); e1.R = \text{COMPUTEREACH}(e1\_comb, G)$ ;  

25            $e2\_comb.insert(e2.v); e2.R = \text{COMPUTEREACH}(e2\_comb, G)$ ;  

26            $e1\_out = G.outEdges(e1.v); cmn.V = \emptyset; e1.V = \emptyset; e2.V = \emptyset$ ;  

27           Remove from  $e1\_out, e2\_out$  the  $cmn.V = e1\_out \cap e2\_out$ ;  

28           if  $(e1.R \geq e2.R \text{ and } |e1\_out| \geq |e2\_out|)$  then  $invis += 1$ ;  

29            $travs += 1$ ; // traversal of  $selV[c_f]$  proceeds with its previous  $e1$   

30           if  $|subs| = 0$  then break; // termination; no more  $uC$  are created  

31            $offset += 1$ ; if  $travs = invis$  then  $c_f.invisible.insert(e2.v)$  and  

            continue; //  $e2$  marked as invisible for all traversed  $e1$ ; no  $uC$  for  $e2$   

32           For each  $e1 \in subs$  and for each  $c_u \in uC$  where  $e1 \in c_u$ , replace  $e1$   

             with  $e2$  to form  $c_{u'}$  and set  $uC[c_{u'}] = -1$ ; // each  $c_{u'}$  is a new  $uC$   

33           // 3. Compute  $fC[c_f]$  with the examination of each  $c_u \in uC$  (loop here!)  

34           Repeat lines 20-22 of SCAN; //  $c_f.R$  in place of  $uC[fC[c_f]]$  as in line 9  

35           // 4. Update  $roundBest$   

36           Repeat line 23 of SCAN; //  $c_f.R$  in place of  $uC[fC[c_f]]$  as in line 9  

37            $SG = SG + roundBest.SG; R = R + roundBest.R$ ;  

38            $deletedV = \emptyset$ ; for each user  $v \in roundBest.users$  do  $deletedV.insert(v)$ ;  

             Delete from each  $selV[c_f]$  all pairs that contain a user  $v \in deletedV$ ;  

39           return  $SG, R$ ;
    
```

In effect, for an entry $e2$ to generate winner-candidate user combinations for c_f in Line 32, it must substitute at least one $e1$, and it must not be marked as invisible. The performance of `SUBSTITUTE` relies on the fact that only a few $e2$ pass the two mentioned checks, as well as on the occurrence of early termination (Line 30). Since it makes sense to compare reach only for really similar (k, m) candidates (d is small), any false misses due to the condition in Line 28 would have a negative impact on efficacy only in case early termination takes too long to take effect.

6.3 Experimental Evaluation

We wrote code in C++ and ran experiments on an AMD Ryzen 5 4600U CPU @ 2.1 GHz machine with 16GB RAM running Linux Ubuntu 20.04.3 LTS 64-bit. For graph operations we used the open-source graph library Lemon [DJK11].

6.3.1 Setup

Datasets. As VK comprises 27 categories, we select the 1 and 2 most popular pages from each category, to form sets of 27 and 54 features. We uniformly at random select 10 different groups of 80 users for $|L| = 27$ and $|L| = 54$ (same groups for both L), and 10 different groups of 100 users for $|L| = 27$ and $|L| = 54$ (same groups for both L); there is no relation among groups of 80 and 100 users. So, all the experimental results in our figures are averaged by 10.

Tuning. For each selected user, we realistically tune her $F_v(f)$ value for each feature $f \in L$ by taking the average of her *like* responses in all posts of the years 2010-2017 of VK; as f we considered only the *brand* that published the relative post where user liked it. The feature weight sum of each $F_v(f)$ equals to 1, and for any f where no *like* found, we initialized $F_v(f)$ with a dummy value.

Parameters. The number of content features (k) is examined until $k = 3$, as higher k values were costly for `SCAN`. The other parameters are fixed; the number of m invited users is 3, the similarity threshold d is 0.001, and the number of n rounds is 20.

6.3.2 Results

Figure 6.1 presents the subscription gain (SG) and running time (measured in seconds) results per round of `RANDOM`, `SCAN`, and `SUBSTITUTE` for $k = 1, 2$, and 3, and for $|L| = 27, |V| = 80$; Figure 6.2 for $|L| = 27, |V| = 100$, Figure 6.3 for $|L| = 54, |V| = 80$, and Figure 6.4 for $|L| = 54, |V| = 100$. Besides the per round results, in each SG figure there is in legend for each solver a pair (X, Y) , where X is the cumulative SG and Y

6.3. Experimental Evaluation

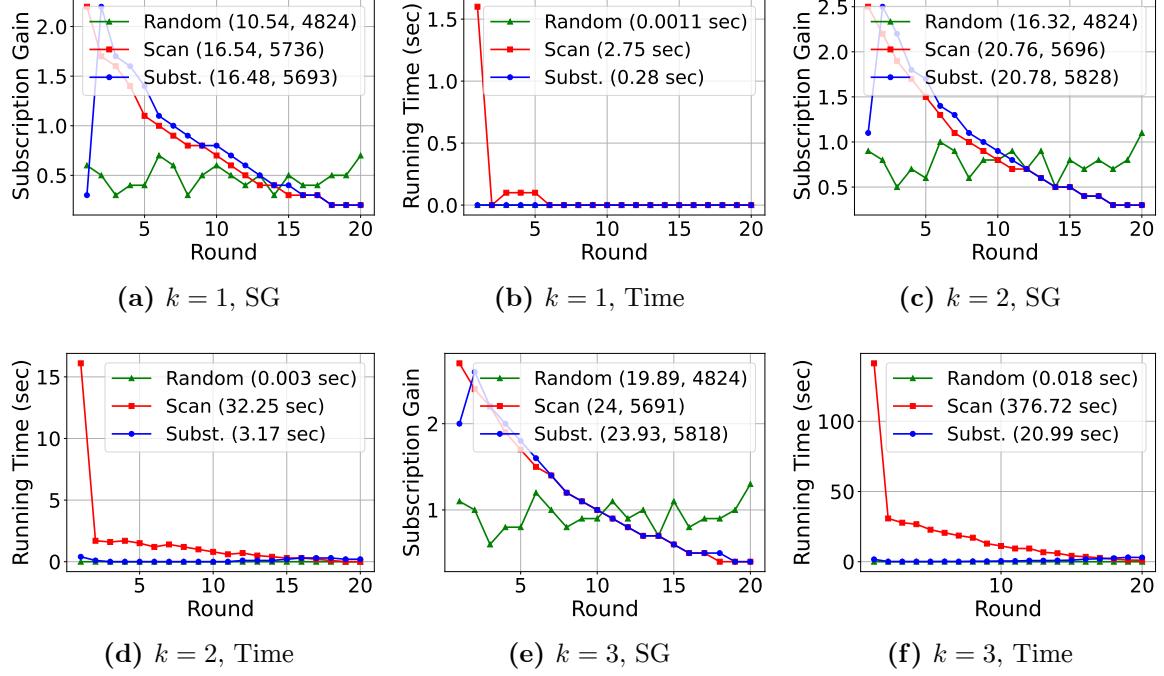


Figure 6.1: SG per round and Time per round results of RANDOM, SCAN, and SUBSTITUTE for $|L| = 27$, $|V| = 80$, and $k = 1, 2$, and 3.

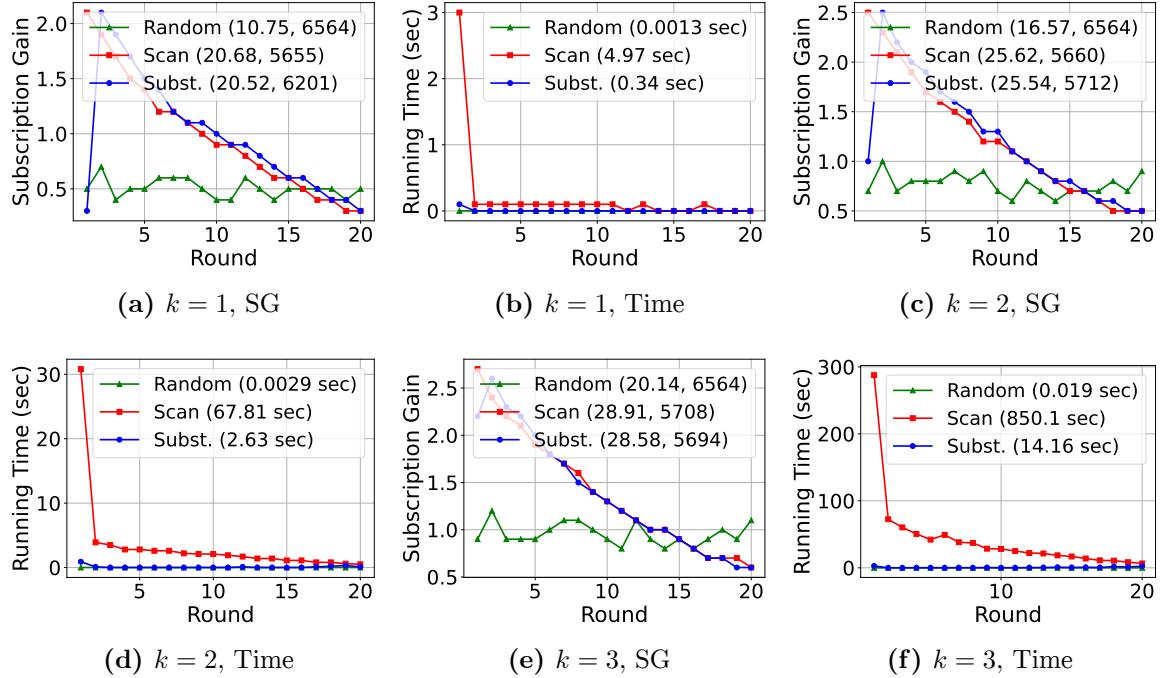


Figure 6.2: SG per round and Time per round results of RANDOM, SCAN, and SUBSTITUTE for $|L| = 27$, $|V| = 100$, and $k = 1, 2$, and 3.

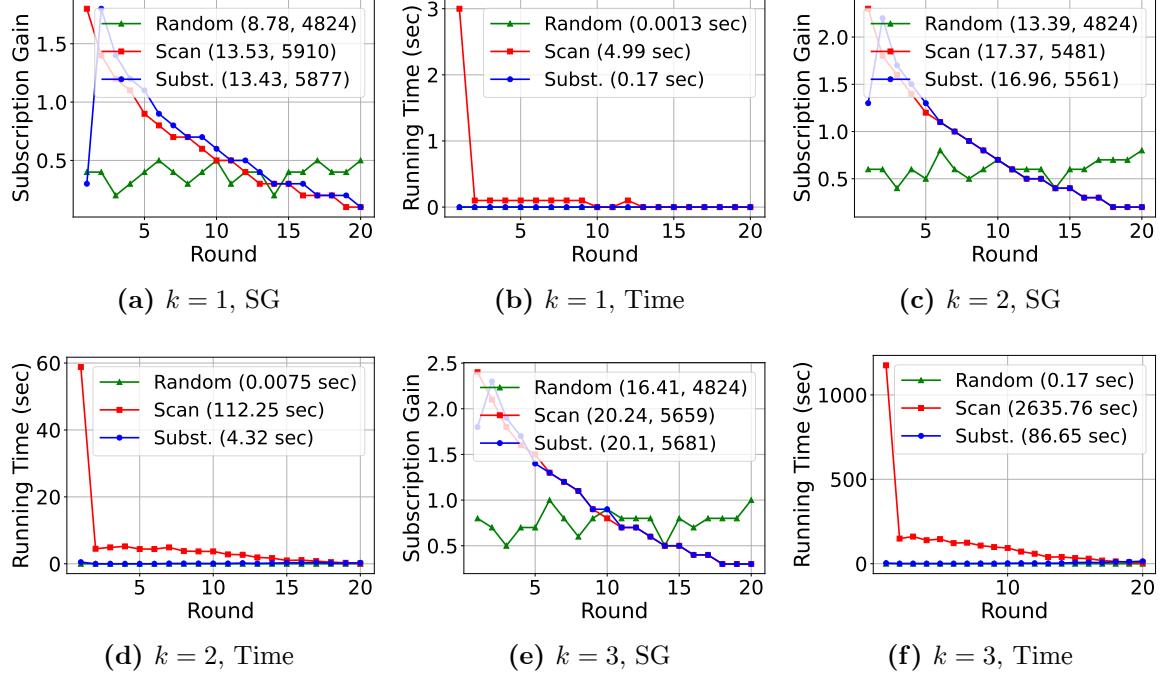


Figure 6.3: SG per round and Time per round results of RANDOM, SCAN, and SUBSTITUTE for $|L| = 54$, $|V| = 80$, and $k = 1, 2$, and 3.

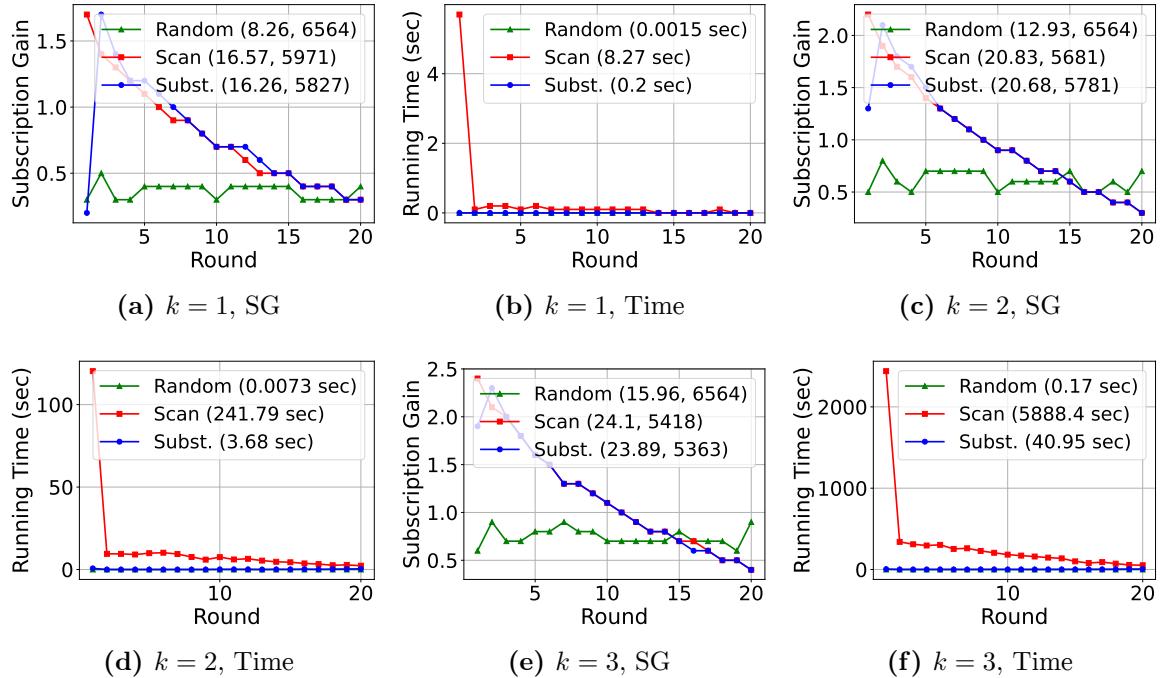


Figure 6.4: SG per round and Time per round results of RANDOM, SCAN, and SUBSTITUTE for $|L| = 54$, $|V| = 100$, and $k = 1, 2$, and 3.

6.3. Experimental Evaluation

is the cumulative reach (R)³ over all rounds. Similarly, in each time figure, the legend mentions for each solver the cumulative running time over all rounds.

We observe that the general trend is the same in all figures; **SUBSTITUTE** is almost equally effective to **SCAN** and competitively efficient to **RANDOM**. It is evident that **SUBSTITUTE** achieves a slightly less cumulative SG in regards to **SCAN** due to the poor SG result it achieves in first round; that behavior depends on aggressive false misses derived from true evaluation of condition in Line 28, but as rounds evolve such false misses reduce due to skipping of each c_f having an unaffected best solution (Line 9). Except for first round, the SG of both solvers is very close and diminishes as rounds grow because promising subscription candidates are messaged from the beginning. Yet, the SG of **RANDOM** is the lowest and can be seen as fixed across rounds, as it does not apply messaging with priority as previous solvers. Regarding running time, **SUBSTITUTE** is slower than **RANDOM** but constant, while **SCAN** performs better as rounds evolve but has a clear scalability issue when any of V , L , or k increases.

Moreover, we remark that in all figures as k grows all solvers achieve higher efficacy (SG) and lower efficiency (time). This is logical as a higher k yields a higher feature weight sum of m users to k features, but also it incurs more feature combinations for processing. Another interesting result is to see what happens when V increases over a fixed L , and what happens when L increases over a fixed V . In the former case, by comparing the Figures 6.1 and 6.2, as also the Figures 6.3 and 6.4, we note that the SG of **RANDOM** is constant, while the SG of other two solvers clearly improves; a larger selection pool of users is always more beneficial for prioritized solvers. Yet, with more candidate user combinations present for processing, only **SUBSTITUTE** improves its performance on running time (e.g., see Figures 6.1f, 6.2f, and Figures 6.3f, 6.4f), whereas **SCAN** heavily deteriorates its performance; more users offer more chances for early termination cases to occur in Line 30 of **SUBSTITUTE** since it is more likely that some users clearly separate over others. In the latter case, by comparing Figures 6.1 and 6.3, as well as Figures 6.2 and 6.4, we observe that both the efficacy and efficiency of all solvers worsen. The SG deteriorates as a heavier feature weight segmentation in F_v (due to larger L) decreases the feature weights of each user and so incurs lower feature weight sums of m users to k features. Further, a more intense segmentation strengthens the similarity of feature weights in F_v , and so increases the (k, m) candidates not filtered by threshold d . Lastly, the inferior running time is expected due to processing a larger pool of candidate feature combinations.

³We present R results for the sake of completeness; their analysis is not critical for GSM.

6.4 Conclusion

In this chapter, we proposed that brands can use a *content recommendation policy* to gain subscribers to their social network pages via messaging. We deployed three algorithms, **RANDOM**, **SCAN**, and **SUBSTITUTE** to this task using a realistic tuning of VK posts. Our thorough experimental study on different user and feature sets verified that **SUBSTITUTE** outperforms other solvers. To our knowledge, this is the first work to study how brands can gain subscribers using content.

6.4. Conclusion

Chapter 7

Conclusion

7.1 Summary

In this thesis, we studied important social and spatial marketing problems applied on Web. In spatial domain, we deployed the SRX system that supports spatial RDF data management operations, while in social domain we introduced three novel problems of content-aware influence maximization applied to social networks.

To efficiently solve spatial RDF problems, we built the SRX system by extending the popular RDF-3X store [NW08] with a novel spatial encoding grid-scheme to support WITHIN, DISTANCE, and kNN queries, as also Updates of spatial RDF entities; the grid-scheme approximates the geometries of the spatial RDF entities inside their integer IDs and is used along with several introduced operators and optimizations. We evaluated SRX on LinkedGeoData (LGD) [LGD] and YAGO [YAG] RDF datasets and compared its performance with the latest versions of Virtuoso [vir], GraphDB [owl], and Strabon [KKK12]. The results show SRX's superior performance over the competitors; in regards to RDF-3X, SRX improves its performance for queries with spatial predicates while incurring little overhead during updates.

The first problem we proposed and studied in social domain was the *Content-Aware Influence Maximization* (CAIM) problem. CAIM asks for the k features (or attributes) that can form a post so as to make it viral in a social network, starting its diffusion from the subscribers of the brand's social network page. We proved that the CAIM problem is NP-hard and inapproximable, and so we deployed an efficient heuristic, Explore-Update, which uses bounded local arborescences to calculate influence spread. Our experimental results on Gnutella¹ and VK² datasets show that Explore-Update selects near-optimal feature sets, achieves 30% higher spread than baselines, and runs an order of magnitude faster than the classic Greedy solution.

¹<https://snap.stanford.edu/data/p2p-Gnutella04.html>

²<https://vk.com/>

We also studied CAIM under adaptive settings, and so we proposed the *Adaptive Content-Aware Influence Maximization* (ACAIM) problem for online applicability to social networks. ACAIM seeks for k features to form a post in each round so as to maximize the cumulative influence of those posts over all rounds. To solve ACAIM, we utilized an *Online Learning To Rank* (OLR) framework for IM purposes, we introduced the *Content-Aware TopRank Influence Dissemination* (CATRID) propagation model to enable OLR for ACAIM, we deployed a simulator to express a real feedback environment for learners based on VK posts, and we developed three ACAIM learners evaluated on several VK datasets. Our experimental results show the practical value and suitability of ACAIM to social network industry.

The last problem we studied around social networks was to propose a *content recommendation policy* to a brand agent/advertiser for *Gaining Subscribers by Messaging* (GSM) over many rounds. To solve GSM, we find in each round, what content features to publish and which users to notify of those features aiming to maximize the cumulative subscription gain over all rounds. We deployed three GSM solvers, named **RANDOM**, **SCAN**, and **SUBSTITUTE** and tested their performance on VK datasets. Our experimental evaluation show that **SUBSTITUTE** clearly outperforms other solvers.

7.2 Future Work

In regards to SRX work, we plan to extend our update mechanism to support online updates in the spirit of [NW10b] and extend our query optimizer to consider the spatial distribution of entities that support a *characteristic set* [NM11]. A promising research direction that we also plan to pursue is to investigate how our encoding-based techniques can be adapted to distributed spatial analytics systems, such as those in [PKNK18], to improve their performance.

In social domain around content-aware recommendation, for CAIM we plan to study other propagation models and investigate the parallelization of Explore-Update, for ACAIM our main target is to extend it further for *loyalty marketing* purposes (e.g., gaining new subscribers over rounds), and for GSM we intend to apply our solutions on larger user and feature sets and develop better solvers for such cases.

Also, there is a number of new spatio-social problems that we can propose and study based on the contributions of this thesis. For instance, we can solve ACAIM focusing on brands that have physical stores (e.g., Starbucks, Zara), having as target to maximize the influence over users for which we estimate that move around those stores. The same spatial dimension can also be considered for the GSM problem where the nearby users to the stores of brand will be notified earlier than more distant ones.

Lastly, another interesting direction is how to diversify the content we recommend to brands so as to solve ACAIM and GSM with more fairness and naturalness.

Bibliography

- [ABBY14] C. Aslay, N. Barbieri, F. Bonchi, and R. A. Baeza-Yates. Online topic-aware influence maximization queries. In *EDBT*, 2014.
- [ACZH10] Medha Atre, Vineet Chaoji, Mohammed J. Zaki, and James A. Hendler. Matrix “Bit” loaded: A scalable lightweight join query processor for RDF data. In *WWW*, 2010.
- [AGR17] Akhil Arora, Sainyam Galhotra, and Sayan Ranu. Debunking the myths of influence maximization: An in-depth benchmarking study. In *SIGMOD*, pages 651–666, 2017.
- [ALB⁺15] C. Aslay, W. Lu, F. Bonchi, A. Goyal, and Laks V.S. Lakshmanan. Viral marketing meets social advertising: Ad allocation with minimum regret. *Proc. VLDB Endow.*, 8(7):814–825, 2015.
- [AMMH07] Daniel J. Abadi, Adam Marcus, Samuel Madden, and Katherine J. Hollenbach. Scalable semantic web data management using vertical partitioning. In *VLDB*, 2007.
- [AMS09] S. Aral, L. Muchnik, and A. Sundararajan. Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks. *Proc. of the National Academy of Sciences of the U.S.A.*, 106(51):21544–21549, 2009.
- [Ara11] S. Aral. Commentaty-Identifying social influence: A comment on opinion leadership and social contagion in new product diffusion. *Marketing Science*, 30(2):217–223, 2011.
- [ATOR] Christopher R. Aberger, Susan Tu, Kunle Olukotun, and Christopher Ré. Old techniques for new join algorithms: A case study in RDF processing. In *ICDE Workshops*, 2016.

- [ATOR16] Christopher R. Aberger, Susan Tu, Kunle Olukotun, and Christopher Ré. Emptyheaded: A relational engine for graph processing. In *SIGMOD*, 2016.
- [AW11] S. Aral and D. Walker. Creating social contagion through viral product design: A randomized trial of peer influence in networks. *Management Science*, 57(9):1623–1639, 2011.
- [BB14] N. Barbieri and F. Bonchi. Influence maximization with viral product design. In *SDM*, 2014.
- [BBCL14] C. Borgs, M. Brautbar, J. T. Chayes, and B. Lucier. Maximizing social influence in nearly optimal time. In *SODA*, 2014.
- [BBM12] N. Barbieri, F. Bonchi, and G. Manco. Topic-aware social influence propagation models. In *ICDM*, 2012.
- [BDK⁺13] Mihaela A. Bornea, Julian Dolby, Anastasios Kementsietsidis, Kavitha Srinivas, Patrick Dantressangle, Octavian Udrea, and Bishwaranjan Bhattacharjee. Building an efficient RDF store over a relational database. In *SIGMOD*, 2013.
- [BH05] J. A. Berger and C. Heath. Idea Habitats: How the prevalence of environmental cues influences the success of ideas. *Cognitive Science*, 29(2):195–221, 2005.
- [BK12] R. Battle and Dave Kolas. Enabling the geospatial semantic web with parliament and geosparql. *Semantic Web*, 3(4):355–370, 2012.
- [BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient processing of spatial joins using R-trees. In *SIGMOD*, 1993.
- [BKvH01] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: An architecture for storing and querying RDF data and schema information. In *Semantics for the WWW*. MIT Press, 2001.
- [BNM10] Andreas Brodt, Daniela Nicklas, and Bernhard Mitschang. Deep integration of spatial query processing into native RDF triple stores. In *GIS*, 2010.
- [CDES05] Eugene Inseok Chong, Souripriya Das, George Eadon, and Jagannathan Srinivasan. An efficient SQL-based RDF querying scheme. In *VLDB*, 2005.

Bibliography

- [CFL⁺15] S. Chen, J. Fan, G. Li, J. Feng, K.-L. Tan, and J. Tang. Online topic-aware influence maximization. *Proc. VLDB Endow.*, 8(6):666–677, 2015.
- [CK13] Yuxin Chen and Andreas Krause. Near-optimal batch mode active learning and adaptive submodular optimization. In *ICML*, pages 160–168, 2013.
- [CLC13] Wei Chen, Laks V. S. Lakshmanan, and Carlos Castillo. *Information and Influence Propagation in Social Networks*. Morgan & Claypool Publishers, 2013.
- [CLY14] W. Chen, T. Lin, and C. Yang. Efficient topic-aware influence maximization using preprocessing. *CoRR*, abs/1403.0057, 2014.
- [CM13] I. P. Cvijikj and F. Michahelles. Online engagement factors on facebook brand pages. *Social Network Analysis and Mining*, 3(4):843–861, 2013.
- [CMdR15] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. *Click Models for Web Search*. Morgan and Claypool Publishers, 2015.
- [CMPL15] Richard Combes, Stefan Magureanu, Alexandre Proutière, and Cyrille Laroche. Learning to rank: Regret lower bounds and efficient algorithms. In *SIGMETRICS*, pages 231–244, 2015.
- [CPL12] Y. -C. Chen, W. -C. Peng, and S. -Y. Lee. Efficient algorithms for influence maximization in social networks. *Knowl. Inf. Syst.*, 33(3):577–601, 2012.
- [CSH⁺14] S. Cheng, H. Shen, J. Huang, W. Chen, and X. Cheng. IMRank: influence maximization via finding self-consistent ranking. In *SIGIR*, 2014.
- [CWW10] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *KDD*, 2010.
- [CWY13] Wei Chen, Yajun Wang, and Yang Yuan. Combinatorial multi-armed bandit: General framework, results and applications. In *ICML*, pages 151–159, 2013.
- [CWYW16] Wei Chen, Yajun Wang, Yang Yuan, and Qinshi Wang. Combinatorial multi-armed bandit and its extension to probabilistically triggered arms. *JMLR*, 17(1):1746–1778, 2016.
- [CYZ10] W. Chen, Y. Yuan, and L. Zhang. Scalable influence maximization in social networks under the linear threshold model. In *ICDM*, 2010.

- [CZTR08] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *WSDM*, pages 87–94, 2008.
- [dBBL01] C. Van den Bulte and G. L. Lilien. Medical innovation revisited: Social contagion versus marketing effort. *Am. J. Sociol.*, 106(5):1409–1435, 2001.
- [DJK11] Balázs Dezső, Alpár Jüttner, and Péter Kovács. Lemon – an open source c++ graph template library. *ENTCS*, 264(5):23–45, 2011.
- [DR01] P. Domingos and M. Richardson. Mining the network value of customers. In *KDD*, 2001.
- [dVGL12] L. de Vries, S. Gensler, and Peter S.H. Leeflang. Popularity of brand posts on brand fan pages: An investigation of the effects of social media marketing. *Journal of Interactive Marketing*, 26(2):83–91, 2012.
- [EM16] Ahmed Eldawy and Mohamed F. Mokbel. The era of big spatial data: A survey. *Foundations and Trends in Databases*, 6(3-4):163–273, 2016.
- [GBL11] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan. A data-based approach to social influence maximization. *PVLDB*, 5(1):73–84, 2011.
- [GCM⁺15] Artem Grotov, Aleksandr Chuklin, Ilya Markov, Luka Stout, Finde Xu-mara, and Maarten de Rijke. A comparative study of click models for web search. In *CLEF*, pages 78–90, 2015.
- [GK11] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *JAIR*, 42(1):427–486, 2011.
- [GM04] D. Godes and D. Mayzlin. Using online conversations to study word-of-mouth communication. *Marketing Science*, 23(4):545–560, 2004.
- [Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.
- [HD10] L. Hong and B. D. Davison. Empirical study of topic modeling in twitter. In *SOMA*, 2010.
- [HHT05] Marios Hadjieleftheriou, Erik G. Hoel, and Vassilis J. Tsotras. Sail: A spatial index library for efficient application integration. *GeoInformatica*, 9(4):367–389, 2005.

Bibliography

- [HS15] Thibaut Horel and Yaron Singer. Scalable methods for adaptively seeding a social network. In *WWW*, pages 441–451, 2015.
- [HTH⁺20] Keke Huang, Jing Tang, Kai Han, Xiaokui Xiao, Wei Chen, Aixin Sun, Xueyan Tang, and Andrew Lim. Efficient approximation algorithms for adaptive influence maximization. *VLDBJ*, 29(6):1385–1406, 2020.
- [ITTK17] Sergei Ivanov, Konstantinos Theocharidis, Manolis Terrovitis, and Panagiotis Karras. Content recommendation for viral social influence. In *SIGIR*, pages 565–574, 2017.
- [Kat59] E. Katz. Mass communications research and the study of popular culture: An editorial note on a possible future of this journal. *Studies in Public Communication*, 2:1–6, 1959.
- [KK10] Manolis Koubarakis and Kostis Kyzirakos. Modeling and querying metadata in the semantic sensor web: The model stRDF and the query language stSPARQL. In *ESWC*, 2010.
- [KKK12] Kostis Kyzirakos, Manos Karpathiotakis, and Manolis Koubarakis. Strabon: A semantic geospatial DBMS. In *ISWC*, 2012.
- [KKSW16] Sumeet Katariya, Branislav Kveton, Csaba Szepesvári, and Zheng Wen. DCM bandits: Learning to rank with multiple clicks. In *ICML*, pages 1215–1224, 2016.
- [KKT03] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *KDD*, 2003.
- [KLK20] Ansh Khurana, Alvis Logins, and Panagiotis Karras. Selecting influential features by a learnable content-aware linear threshold model. In *CIKM*, page 635–644, 2020.
- [KN11] Brian Karrer and M. E. J. Newman. Stochastic blockmodels and community structure in networks. *Phys. Rev. E*, 83:016107, Jan 2011.
- [KSWA15] Branislav Kveton, Csaba Szepesvári, Zheng Wen, and Azin Ashkan. Cascading bandits: Learning to rank in the cascade model. In *ICML*, pages 767–776, 2015.
- [KWAS15] Branislav Kveton, Zheng Wen, Azin Ashkan, and Csaba Szepesvári. Combinatorial cascading bandits. In *NeurIPS*, pages 1450–1458, 2015.

- [LCCM19] Paul Lagrée, Olivier Cappé, Bogdan Cautis, and Silviu Maniu. Algorithms for online influencer marketing. *TKDD*, 13(1):1–30, 2019.
- [LFWT18] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. Influence maximization on social graphs: A survey. *TKDE*, 30(10):1852–1872, 2018.
- [LGD] Ledgeodata. <http://ledgeodata.org/> About.
- [LKG⁺07] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *KDD*, 2007.
- [LKLS18] Tor Lattimore, Branislav Kveton, Shuai Li, and Csaba Szepesvári. TopRank: A practical algorithm for online stochastic ranking. In *NeurIPS*, pages 3949–3958, 2018.
- [LLS19] Shuai Li, Tor Lattimore, and Csaba Szepesvári. Online learning to rank with features. In *ICML*, pages 3856–3865, 2019.
- [LMBT14] John Liagouris, Nikos Mamoulis, Panagiotis Bouros, and Manolis Terrovitis. An effective encoding scheme for spatial rdf data. *Proc. VLDB Endow.*, 7(12):1271–1282, 2014.
- [LMM⁺15] Siyu Lei, Silviu Maniu, Luyi Mo, Reynold Cheng, and Pierre Senellart. Online influence maximization. In *KDD*, pages 645–654, 2015.
- [LR96] Ming-Ling Lo and Chinya V. Ravishankar. Spatial hash-joins. In *SIGMOD*, 1996.
- [LVC16] Paul Lagrée, Claire Vernade, and Olivier Cappé. Multiple-play bandits in the position-based model. In *NeurIPS*, pages 1605–1613, 2016.
- [LWZC16] Shuai Li, Baoxiang Wang, Shengyu Zhang, and Wei Chen. Contextual combinatorial cascading bandits. In *ICML*, pages 1245–1253, 2016.
- [LZT15] Y. Li, D. Zhang, and K.-L. Tan. Real-time targeted influence maximization for online advertisements. *Proc. VLDB Endow.*, 8(10):1070–1081, 2015.
- [Mam11] Nikos Mamoulis. *Spatial Data Management*. Morgan & Claypool Publishers, 2011.
- [Man93] C. F. Manski. Identification of endogenous social effects: The reflection problem. *The Review of Economic Studies*, 60(3):531–542, 1993.

Bibliography

- [MHP05] Kyriakos Mouratidis, Marios Hadjieleftheriou, and Dimitris Papadias. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In *SIGMOD*, 2005.
- [MP03] Nikos Mamoulis and Dimitris Papadias. Slot index spatial join. *TKDE*, 15(1):211–231, 2003.
- [NM11] Thomas Neumann and Guido Moerkotte. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins. In *ICDE*, 2011.
- [NVDV18] Panagiotis Nikitopoulos, Akrivi Vlachou, Christos Doulkeridis, and George A. Vouros. DiStRDF: Distributed spatio-temporal RDF queries on Spark. In *EDBT/ICDT*, 2018.
- [NW08] Thomas Neumann and Gerhard Weikum. RDF-3X: A RISC-style engine for RDF. *Proc. VLDB Endow.*, 1(1):647–659, 2008.
- [NW09] Thomas Neumann and Gerhard Weikum. Scalable join processing on very large RDF graphs. In *SIGMOD*, 2009.
- [NW10a] Thomas Neumann and Gerhard Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB J.*, 19(1):91–113, 2010.
- [NW10b] Thomas Neumann and Gerhard Weikum. x-RDF-3X: Fast querying, high update rates, and consistency for RDF databases. *Proc. VLDB Endow.*, 3(1-2):256–263, 2010.
- [Ö16] M. Tamer Özsu. A survey of RDF data management systems. *Front. Comput. Sci.*, 10(3):418–432, 2016.
- [owl] GraphDB. <http://graphdb.ontotext.com>.
- [par] Parliament. <http://parliament.semwebcentral.org>.
- [Pei15] Tiago P. Peixoto. Model selection and hypothesis testing for large-scale network models with overlapping groups. *Phys. Rev. X*, 5, 2015.
- [PGA14] Kostas Patroumpas, Giorgos Giannopoulos, and Spiros Athanasiou. Towards geospatial semantic data management: Strengths, weaknesses, and challenges ahead. In *GIS*, 2014.
- [PKNK18] Varun Pandey, Andreas Kipf, Thomas Neumann, and Alfons Kemper. How good are modern spatial analytics systems? *Proc. VLDB Endow.*, 11(11):1661–1673, 2018.

- [RD02] M. Richardson and P. Domingos. Mining knowledge-sharing sites for viral marketing. In *KDD*, 2002.
- [RDR07] Matthew Richardson, Ewa Dominowska, and Robert Ragno. Predicting clicks: Estimating the click-through rate for new ads. In *WWW*, pages 521–530, 2007.
- [SHYC18] Lichao Sun, Weiran Huang, Philip S. Yu, and Wei Chen. Multi-round influence maximization. In *KDD*, pages 2249–2258, 2018.
- [SS13] Lior Seeman and Yaron Singer. Adaptive seeding in social networks. In *FOCS*, pages 459–468, 2013.
- [TXS14] Y. Tang, X. Xiao, and Y. Shi. Influence maximization: near-optimal time complexity meets practical efficiency. In *SIGMOD*, 2014.
- [vir] Virtuoso. <http://virtuoso.openlinksw.com>.
- [VKW⁺17] Sharan Vaswani, Branislav Kveton, Zheng Wen, Mohammad Ghavamzadeh, Laks V.S. Lakshmanan, and Mark Schmidt. Model-independent online learning for influence maximization. In *ICML*, pages 3530–3539, 2017.
- [VLS16] Sharan Vaswani, Laks V.S. Lakshmanan, and Mark Schmidt. Influence maximization with bandits, 2016.
- [WKB08] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore: Sextuple indexing for semantic web data management. *Proc. VLDB Endow.*, 1(1):1008–1019, 2008.
- [WKC12] Chih-Jye Wang, Wei-Shinn Ku, and Haiquan Chen. Geo-store: A spatially-augmented sparql query evaluation system. In *GIS*, 2012.
- [WKVV17] Zheng Wen, Branislav Kveton, Michal Valko, and Sharan Vaswani. Online influence maximization under independent cascade model with semi-bandit feedback. In *NeurIPS*, pages 1–24, 2017.
- [WLW⁺19] Qingyun Wu, Zhige Li, Huazheng Wang, Wei Chen, and Hongning Wang. Factorization bandits for online influence maximization. In *KDD*, pages 636–646, 2019.
- [WSKR03] Kevin Wilkinson, Craig Sayers, Harumi A. Kuno, and Dave Reynolds. Efficient RDF storage and retrieval in Jena2. In *SWDB*, 2003.

Bibliography

- [WZF⁺13] Dong Wang, Lei Zou, Yansong Feng, Xuchuan Shen, Jilei Tian, and Dongyan Zhao. S-store: An engine for large RDF graph integrating spatial information. In *DASFAA*, 2013.
- [YAG] YAGO. [https://en.wikipedia.org/wiki/YAGO_\(database\)](https://en.wikipedia.org/wiki/YAGO_(database)).
- [YLW⁺13] Pingpeng Yuan, Pu Liu, Buwen Wu, Hai Jin, Wenya Zhang, and Ling Liu. TripleBit: A fast and compact system for large scale RDF data. *Proc. VLDB Endow.*, 6(7):517–528, 2013.
- [YT17] Jing Yuan and Shaojie Tang. No time to observe: Adaptive influence maximization with partial feedback. In *IJCAI*, page 3908–3914, 2017.
- [YWZ⁺09] Ying Yan, Chen Wang, Aoying Zhou, Weining Qian, Li Ma, and Yue Pan. Efficient indices using graph partitioning in RDF triple stores. In *ICDE*, 2009.
- [ZMC⁺11] Lei Zou, Jinghui Mo, Lei Chen, M. Tamer Özsu, and Dongyan Zhao. gStore: Answering SPARQL queries via subgraph matching. *Proc. VLDB Endow.*, 4(8):482–493, 2011.
- [ZNS⁺16] Shi Zong, Hao Ni, Kenny Sung, Nan Rosemary Ke, Zheng Wen, and Branislav Kveton. Cascading bandits for large-scale recommendation problems. In *UAI*, pages 835–844, 2016.
- [ZTG⁺17] Masrour Zoghi, Tomas Tunys, Mohammad Ghavamzadeh, Branislav Kveton, Csaba Szepesvári, and Zheng Wen. Online learning to rank in stochastic click models. In *ICML*, pages 4199–4208, 2017.
- [ZYW⁺13] Kai Zeng, Jiacheng Yang, Haixun Wang, Bin Shao, and Zhongyuan Wang. A distributed graph engine for web scale RDF data. *Proc. VLDB Endow.*, 6(4):265–276, 2013.