Query-driven Event Search in News Information Network

by

Shanshan Chen

B.Eng., Beijing University of Posts and Telecommunications, 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The Faculty of Graduate and Postdoctoral Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

August 2014

Abstract

Traditional search focuses on keyword matching and document ranking, thus users will only get a overload of related news articles or videos, with little semantics aggregation, when they input keywords among which they are interested in exploring potential connections. To capture these connections, one good way is to model news articles into a complex network of events, where an event itself is a complex network of interrelated *actions* (or assertions). Event search enables users to get a big picture of essential actions involved without going through overwhelming documents. Further on, Query-driven event search, in which users can access key actions that occurred in various events with respect to input query and construct a new event capturing corresponding actions and connections between them, is able to inherently revolutionize search from its traditional keyword matching to a higher level of abstraction. Thus, we propose a natural and useful paradigm, *Query-driven event search in News Information Network*, to allow users to search through news articles and obtain events as answers. We construct the news information network with nodes representing actions and edges indicating relatedness between nodes. We define a good answer as a structurally densely linked and semantically related subgraph, describing a concise and informative event. Thus our objective function combines both the linkage structure of graph and semantic content relatedness. We then formally define our problem to build a query-driven event search engine that processes users query and generates a subgraph satisfying following conditions: 1) covering all keywords but without noisy nodes 2) maximizes the objective function. We prove this problem is NP-complete and propose heuristics algorithms. Our experimental evaluation on real-life news datasets demonstrates algorithms efficiency and meaningful solutions we obtain.

Preface

This dissertation, and the project it covers, is fully original and unpublished. I came up with the idea of building an event search engine that determines a densely linked, semantically related and query-covering subgraph with respect to keywords users are interested in exploring with the help of Professor Laks V.S. Lakshmanan and Pei Li. I prepared all datasets needed, did extensive experiments, analyzed results and wrote this dissertation by myself. Professor Laks V.S. Lakshmanan and Pei Li were involved in the whole process of brainstorming the idea, formally defining the problem, proposing heuristics algorithms and evaluating algorithms' performance and results' quality.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank all people who give me help on this thesis. I am most grateful to my supervisor, Professor Laks V.S. Lakshmanan, for his supervision, inspiration, trust and support in the past two years. His passion in research, keen insight on solving problems and diligence set up a great example to me. I am also grateful to Professor George Tsiknis, the second reader of my thesis, for his editorial and technical suggestions. I would also like to express my gratitude to Pei Li for spending generous time on helping me brainstorming and giving insightful discussions in meetings. I also sincerely thank Min Xie for his valuable suggestions on problem definition and experiments. Finally and above all, I want to thank my friends and family for their unconditional love and support.

Chapter 1

Introduction

In the current social web page, various kinds of information from news or microblogs is pouring to us everyday, and thus information overload has been a serious problem. To mitigate this problem, traditional search focuses on keyword matching and document ranking to help users find the best answer easier. However, traditional search has little semantics aggregation, thus there are still quite a few cases where users have to read numerous documents especially when the query contains multiple keywords and users intend to explore the connections between them. To capture these connections, one good way is to model news articles into a complex network of events, where an event itself is a complex network of interrelated actions (or assertions).

Event search enables users to get a big picture of essential actions involved without going through overwhelming documents. State-of-the-art keyword search in social stream can group related posts into events[25] to avoid reading duplicated information. Yan and Kong et al. [41] investigate the problem of generating news timelines, consisting individual but correlated summaries on each date, from massive data on the Internet to provide users a high-level summary of event. However, these existing approaches either neglect the connections between input keywords within events or lack the interface for users to retrieve information with respect to queries they are interested in. We thus propose a natural and useful paradigm, Query-driven event search, in which users can access key actions that occurred in various events with respect to input query and construct a new event capturing corresponding actions and connections between them. This new approach is able to inherently revolutionize search from its traditional keyword matching to a higher level of abstraction.

Query-driven event search in News Information Network allows users to search through news articles and obtain events as answers. For example, for query (*Snowden, Julian Assange, Hong Kong, Russia*), top ten results returned by traditional search are news articles covering only part of queries and "Edward Snowden" Wikipedia page, and users won't be able to understand clearly the relationships between keywords without reading all results, while our event search engine is able to return an event shown in figure 1.1

in which all keywords are covered by the union of *assertions* (represented as rectangles) and connections between them are specified. In figure 1.1, each node contains assertion (action it represents) , timestamp this action happened, and frequency of this assertion appearing in the news corpus; edge weight measures the degree of relatedness between two nodes. Our result is more interesting in following reasons: 1) it provides an concise and informative event with respect to input query; 2) connections between actions are specified; 3) query expansion is possible when two query nodes (node containing any keyword in query) need intermediate nodes (not query node) to connect; and 4) it can access key actions from various static events to construct a new event.

Keyword Search in Graph attracts increasing attention as many real life datasets can be represented by trees or graphs. Possible outputs are either a spanning tree [15, 17, 19] or a subgraph with various constraints [4, 7, 13, 20] covering all input keywords. For instance, it can be used to find connections between nodes in social network [13] to explore deeper relationships between people, and in a World Wide Web dataset, it detects relevance between web pages [14]; Hulgeri et al. [17] enables users to extract information from a *relational database* by just typing a few keywords, following hyper links, and interacting with controls on the displayed results, without writing any complex query language. Therefore, we adapt *Keyword Search in Graph* technique to build the query-driven event search engine in News information Network.

For *Keyword Search in Graph* problem, there are always two basic questions to ask: first, what is the semantics of keyword search in the graph; second, what constitutes a good answer[1]. Some previous research [21, 38] seek to find a subgraph that contains query nodes, and is densely-connected. Yet, they rank candidate results by either degree or distance. Neither of them have combined both the linkage structure of subgraph and the semantic content relatedness between nodes. In our news information network, each node is an assertion representing one specific action (rectangle in figure 1.1), links between them stating that those actions are related with each other and edge weight denotes the importance (details in section 3). Thus, in our problem, both the size of graph and semantic relatedness between nodes matter because the former feature guarantees *conciseness* and the latter provides *informativeness*. To provide such event search engine which can generate group of densely-linked, semantic-related nodes which also satisfy particular keyword-based queries, we propose a new ranking technique combining both structure density of graph and semantic relatedness between nodes to rank candidate results.

As a densely-linked and closely-related subgraph can provide a good summary of query-covering actions, this problem can be applied to many real-life applications: 1) **For columnist** who wants to analyze historical or streaming events, especially trying to explore connections between a few key persons or organizations, our query-driven event search can output a subgraph connecting input keywords in the format of actions. 2) **For politician** who has a campaign to launch, our framework can help analyze rival's actions towards certain policies or specific proposal, and prepare him or her better. 3) It enables **Ordinary users** to figure out how interested keywords construct an event and the to explore connections between them in a short time without all related news articles.

Therefore, we define our problem as follows: given a formally written news corpus, we build a query-driven event search engine which is able to process user's query and generate *actions* covering input keywords and to provide *relationships* between them. To achieve this, we first build an undirected and weighted news information network from input news corpus and then seek to find a structurally dense, semantically related and connected subgraph which covers all input keywords. In this paper, we will detailedly describe how to build a news information network from raw news data, formally define the problem, propose a few heuristics algorithms to solve it, and experiment on three real-life datasets to evaluate our approaches.

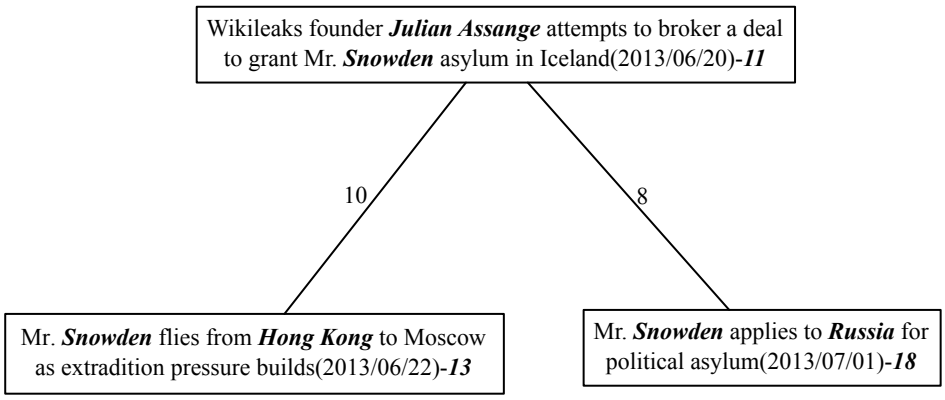


Figure 1.1: A graph example for specified query

1.1 Challenges

There are several challenges for this work.

- how to build a news information network from raw news data, specifically what qualifies as a node (semantics content unit) and a connection between nodes.
- how to define a *good* answer for input queries, and design efficient algorithms to generate it
- how to evaluate algorithms by comparing with existing work

1.2 Key contributions

In a summary, our contributions are as follows:

- To our best knowledge, we are the first to build a news information network from raw news article, and study the problem of query-driven event search on this graph
- We formally define our problem and prove it to be NP-complete
- We propose a few heuristics algorithms to solve the problem, and experiment with real-life dataset showing our algorithm can output meaningful answers

1.3 Roadmap

The outline of this thesis is arranged as follows: The next chapter will list this problem's related work. In the third chapter, we introduce the methodology to process raw news articles to news information network. Then we explain details in our data model in the fourth chapter. The following chapter formally defines our problem, proves the hardness, and proposes a few heuristic algorithms. Then extensive experiments on real-life datasets and evaluations are presented. In the end, we conclude our work and discuss available future work.

Chapter 2

Related Work

2.1 Graph-based text summarization

Text summarization generally refers to the process of automatically generating a compressed, yet with preserved important information, version of the given text[9]. It broadly consists of two approaches: extractive and abstractive summarization. Graph-based methods belong to extractive approach, and are proposed to rank units based on "votes" between each other. For example, Erkan et al. [11] introduces a stochastic graph-based method, similar to PageRank and HITS, to compute sentence's importance based on the *centrality* concept in graphs. Mani et al. [28] uses a graph, with a single word as node and several kinds of links, to represent one document, aiming to find similarities and dissimilarities in pairs of documents. This work is related to our work in terms of using a graph to process raw text (news articles), building connections between text units (nodes) and adopting graph attributes to measure nodes' prestige. The difference is that all the above papers focus on selecting exact words or sentences in text to form a compressed and coherent summary of the input text, while our work treats *assertion* as text unit, introduces the notion of *action*, and seeks to find an *event* composed by actions with respect to our query keywords.

2.2 Connectivity subgraph

Faloutsos and McCurley[13] addresses the problem of extracting a *connection subgraph* to best capture the relationship between *two* nodes in a social network based on the measure of *electrical-current flows*. Tong et al. [39] uses the related notion of *random walks* to find nodes and subgraph that have strong connections to all or most of input query nodes. They share the task to extract subgraph that is not only near to query nodes based on their proximity measure, but also connects query nodes. Cheng et al. [8] then put the *object connection discovery* problem in a *context-aware* situation , where object has a *context*. They fisrt partition the original graph

into a set of communities based on the concept of *modularity*, and study object connection in *intra-community* and *inter-community*. Kasneci and Elbassuoni [20] set this in an entity-relationship diagram, and focuses on extracting informative subgraphs with respect to query nodes.

Our approach differs from above research in following reasons: 1) we are more interested to extract the best connection graph that connects our query nodes by adopting graph diameter and edge weight features; 2) The fundamental difference between our news information network structure(edge weight semantic meaning etc.) and theirs allows us to formalize our problem in a different way.

2.3 Dense subgraph

Angel et al. [3] maintains dense subgraph under streaming edge weight updates and outputs dense subgraph periodically. This relates to our work in the way that they output groups of tightly-coupled entities as a story, while we treat a group of *actions* as an *event*. Dourisboure and Geraci [10] focuses on extracting a dense subgraph as *cyber-communities*(sets of cites and pages sharing a common interest) from World Wide Web (WWW). They also come up with new heuristic algorithms which demonstrate high scalability and effectiveness. [14] presents a new algorithm, based on a recursive application of fingerprinting via shingles, to charaterize large and dense subgraphs of a graph showing connections between hosts on WWW. Their algorithm is extremely efficient, capable of handling graphs with tens of billions of edges on a single machine with modests resources. Quite a few researches put constraint on subgraphs. Asahiro et al. [4] aims to find a $k - vertex$ ($k \geq 3$) subgraph with maximum weight and derives tight bounds for k in different ranges. Charikar [7] studies the problems of finding subgraphs maximizing notions of density for undirected and directed graphs. Andersen et al. [2] specified lower or upper bounds on the number of vertices in the subgrah, gave a $\frac{1}{3}$-approximation algorithm for the former problem, and proved there is no good approximation algorithm for the latter one.

Our work also outputs a dense subgraph which, however, needs to cover all input query keywords. Moreover, we define our own notion of density, which combines structural density and semantic relatedness between nodes, instead of only using average node degree [7], diameter [21] or maximum weight [4].

2.4 Keyword search in graph

Unlike traditional graph search focusing more on graph's structure rather than the semantic content, keyword search aims to extract a group of densely linked nodes in the graph which satisfy particular keyword-based query[1]. Hulgeri et al. [17] enables keyword-based search on relational databases, and outputs rooted trees connecting tuples that match individual keywords in the query as answers. Hao et al. [15] focuses on implementing efficient ranked keyword searches on schemaless node-labeled graphs, and offers order-of-magnitude performance improvement over existing approaches. Our project also enables users to extract information without any knowledge of schema, yet we output compact subgraphs instead of rooted trees as answers. Sozio and Gionis [38] seek to find a *subgraph* that contains all query nodes and is densely connected. They measure density based on minimum degree and distance constraints, and propose corresponding heuristic algorithms. Our work not only constrains the structure density of generated graph, but also considers the semantic relatedness between nodes. In addition, there are multiple nodes corresponding to one query in our work, which is a challenge in searching efficiency.

2.5 Answering relationship queries

Luo and Tang [27] seeks to find relationships between two entities(people, places, institutes or documents etc.) on the Web. To filter massive noise in the web pages without losing useful information, they identify potential connecting terms capturing relationships between entities by assigning weights to terms. It presents users with top ranked web page pairs based on the likelihood of two web pages mentioning the relationship between entities and highlights connecting terms to clarify relationships between query entities. Our work relates to this paper in the motivation of connecting *relationship queries*, but is different than this paper in the following aspects: first, we decompose raw documents into *triples*(assertions), build a news information network to model connections(relatedness) between them, and output subgraph to connect input queries; secondly, this paper only deals with two queries, while our work has no upper limit on the number of input queries.

2.6 Team formation

Lappas et al. [21] solves the problem of team formation: given an undirected edge weighted graph where each node is labeled with a set of skills (text) and edges represent communication cost between nodes they connect, the task is to find a subgraph which covers all skills required and minimizes the communication cost at the same time. Our problem is most related to this paper, and we actually prove the hardness of our problem(see Section 5.1) by reducing to their problem. Yet, our edge weight has the exact opposite semantics, and we aim to combine the structural density and semantic relatedness between nodes, which leads to a clearly different problem than theirs.

Chapter 3

Data Preprocessing

3.1 Overview

This chapter will introduce approaches we take to process raw news text into a new information network. The general work flow is shown in figure 3.1, and each part's motivation and methodology will be introduced separately and detailedly in each section. Before getting into processing details, we define our worked-on text unit *triple* as follows:

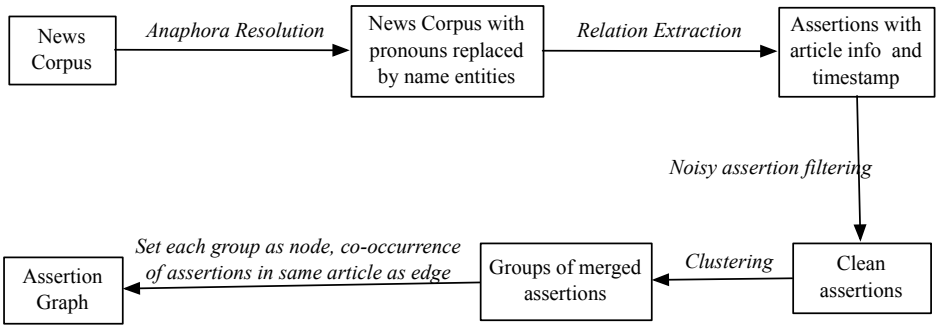


Figure 3.1: Dataset processing workflow

Definition 1. *(triple) A triple* Γ *consists of three parts of meta data: subject* s*, predicate* p*, object* o*, denotes as:*

$$\Gamma = (s; p; o)$$

s *and* o *are entities, and* p *denotes the relationship between them. Each triple* Γ *represents an action. Each triple* Γ *has following features:*

- *(news_id)* Γ_{id} *means the id of news from which this triple is generated*
- *(timestamp)* Γ_t *means the timestamp of this triple(or news).*
- *(description)* Γ_d *describes the fact this triple represents.*

Previous work mostly refer to sentence [11, 35, 41] as text unit to work on , others use entities([3, 27]) and relationships between them to explore interesting connections from the web. We extract *triple* from text for the following reasons:

- we aim to provide a subgraph(an event) with multiple actions, which are able to tell users what happened between those queries specified. Sentences in news articles are too long, while entity without relationships convey little information. *Triple* instead can present a fact clearly in the shortest length.
- with text volume increasing, number of sentences will increase linearly, while *triple* volume will increase in the beginning and gets stable at some point because of the fact that a triple transmits over web more easily than a sentence and later-detected triple has been found out already.

Thanks to above reasoning, we refer to *triple* as text unit. However, not all triples convey enough information to represent an *action*. For instance, triple *(she; ate; one apple)* provides little information to us as we won't be able to know who *she* is without context. Thus, we define *wise triple* as below:

Definition 2. *(wise triple) A wise triple* Γ *needs to satisfy following conditions:*

- *both subject s and object o are not pronouns*

To get more wise triples from raw text, we first adopt *Anaphora Resolution* to replace pronouns to their equal entities, and then use *Relation Extraction* to extract triples.

3.2 Anaphora resolution

Halliday & Hasan 1976 defines *anaphora* as cohesion which points back to previous item. It refers to the interpretation of one expression depends on its *antecedent*, the entity being pointed back to. The process of identifying an anaphora's antecedent is then called *anaphora resolution*[33](aka. coreference resolution). For example:

Russia says it will give Ukraine a new tranche of a $15bn aid package[1]

[1] All examples in the thesis are from real-life news

In this example, *it* is the anaphora, and *Russia* is the antecedent.

In formal written news articles, it is quite common to have *anaphora* situations, and the antecedent-search scope is usually located at the current and preceding sentence. The reason we need *anaphor* in processing the news article is because quite a few tuples we get from Relation Extraction 3.3 starts with pronouns like *it, she, he*, and it is impossible to get what these pronouns refer to without context. For example:

Russia had hinted it would freeze the aid until a new government is formed after Ukraine's PM quit last month.

From relation extraction introduced in section 3.3, generated triples are Γ_1 = *(it; would freeze; the aid[enabler=until a new government is formed after Ukraines PM quit last month])* and Γ_2 = *(Ukraines PM; quit in; last month)*. Without reading the whole sentence, we would have no idea *who* did the action of freezing the aid, and Γ_1 will be tossed later because it technically provides no information. To avoid getting rid of valuable triples, we apply *anaphora resolution* to news articles so that all pronouns are replaced with their antecedents before relation extraction. Then the previous sentence becomes the following:

Russia had hinted Russia would freeze the aid until a new government is formed after Ukraines PM quit last month.

Then generated triples become Γ_1 =*(Russia; would freeze; the aid[enabler=until a new government is formed after Ukraines PM quit last month])* and Γ_2 = *(Ukraines PM; quit in; last month)*, both of which are *wise triples*.

In our project, we apply Standford Deterministic Coreference Resolution System[23], which automatically identifies coreferring entities in text, as tool to solve this problem. Its system incorporates lexical, syntactic, semantic and discourse information in coreference resolution models[24], and ranked first at the CoNLL-2011 shared task[2]. It first identifies all mentions of entities in the text, and then cluster them into equivalence classes. Based on generated equivalence classes, pronouns among them can be replaced by name entities, which provides more information, as introduced before. Take the following text for example:

President Barack Obama and his Democrats must agree to delay implementation of Obama's health care law.

[2]http://conll.cemantix.org/2011/

This system generates equaivalence classes as follows:
1 CHAIN1-["Barack Obama" in sentence 1, "his" in sentence 1, "Obama 's" in sentence 1]
3 CHAIN3-["President Barack Obama and his Democrats" in sentence 1]
4 CHAIN4-["President Barack Obama" in sentence 1]
5 CHAIN5-["his Democrats" in sentence 1]
7 CHAIN7-["implementation of Obama 's health care law" in sentence 1]
8 CHAIN8-["Obama 's health care law" in sentence 1]

From CHAIN1, where all entities belong to the same class, we are able to replace pronoun "his" as "Obama 's". Therefore, with each news article as input, we can get all equivalence classes containing multiple entities, and replace pronouns with corresponding name entity.

3.3 Relation extraction

Relation Extraction is the task of extracting semantic relationships between a set of entities from text[3]. In this project, triples in the format of $(s; p; o)$ are needed to extracted from news articles. Generally, two kinds of Information Extraction(IE) systems are available for this; a traditional IE system needs relations of interest to be specified in advance, while open IE ones can generate relations between entities without providing domain-specific knowledge or target of relations.

We refer to Ollie [30], a state-of-art open IE system, to generate binary relationships from news text. There are other advanced open IE systems available, such as SONEX [31], TreeKernel [40]and EXEMPLAR[32]. The reasons that we choose Ollie over these toos are as following:

- EXEMPLAR and SONEX only extract relationships between name entities(people, organization etc.), thus fewer than five tuples can be generated from one news article, while Ollie can extract more than hundred tuples.
- TreeKernel is supervised approach and the lacking of training examples makes it hard to use for our work.
- Ollie has extra features compared with other tools, which are:
  - it calculates confidence score for each triple it generates, which shows how correct this triple could be.

[3] http://en.wikipedia.org/wiki/Relationship_extraction

– it captures enabling condition, a condition that needs to be met so the tuple is true[4]. For example, for sentence

She will move out if rent keeps rising

Triple *(She; will; move out)* will be true when condition *if rent keeps rising* is present.

– it captures attribution clause which specifies an entity that asserted an extraction and a verb that specifies the expression [30]. A sentence like below:

Russia has miscalculated over Crimea, says Hague

get result: *(Russia; has miscalculated over; Crimea)[attrib=Hague says]*

– When some relations are expressed without verb, like:

Edward Snowden is the leaker of NSA

Ollie is able to get triple *(Edward Snowden; be the leaker of; NSA).*

3.4 Filtering

Each news article can averagely generate one hundred triples; however, not every triple is *wise triple*. Take following sentence as example:

Mr Cameron urged Mr Putin to support the formation of a contact group that could lead to direct talks between the governments of Russia and Ukraine.

Generated triples are:

- Γ_1 = *(Mr Cameron; urged Mr Putin to support; the formation of a contact group)*; *confidence score: 0.8*
- Γ_2 = *(Mr Cameron; urged; Mr Putin)*; *confidence score: 0.75*
- Γ_3 = *(direct; talks between; the governments of Russia)*; *confidence score:0.262*

[4]https://github.com/knowitall/ollie

Γ_1 and Γ_2 are able to deliver enough information without other context, while Γ_3 can't. To ensure high quality of the news information network, here are a few filtering criteria:

- Confidence Score: Ollie[30] provides confidence score for each triple it generates. We observe that any triple with score under 0.7 is of low quality, so we get rid of those triples.
- Predicate filtering: Triple with "said in", "said at" only contains date or location information, which is not enough to describe one fact
- Not a wise triple

3.5 Similarity between triples

After generating triples from news articles, we observe that different news articles may contain the exact same or very similar triples, such as *(Edward Snowden; be the leaker of; NSA)* and *(Snowden; be the leaker of; National Security Agency)*. These two triples are presenting the same *fact* and thus need to be merged as one triple. Similarity in this thesis has two flavors, one is textual similarity, which measures the ratio of exact common words between triples, and the other one is semantic similarity with the idea of comparing the likeness of triples' meaning[5].

3.5.1 Preprocessing

A triple averagely consists of ten words, thus noisy words, if any, will bring huge bias on similarity score. Common words, like "the", "so", "and", are the most frequent words in a document[36]. Even though they help convey the meaning, they don't have much importance compared with other representative words. Treating the whole triple as a bag of words [29] will blur this triple's focus and get lower accuracy. To get accurate result, we refer to [34] to focus only on entity words. We obtain entities from a triple using a Part-Of-Speech Tagger [6], and we change the noun to single form when it is plural(POS tag "NNS" or "NNPS"). Also, if there are duplicate entities in the tuple text(very rare), just count one. In the news corpus dataset we used in experiments (see chapter 6), each triple contains on the average 6.2 entities. For further reference, Γ^L will denote as the list of entities triple Γ's text contains.

[5] http://en.wikipedia.org/wiki/Semantic_similarity
[6] POS Tagger, http://nlp.stanford.edu/software/tagger.shtml

3.5.2 Textual similarity

For triple Γ_1 and Γ_2, *Jaccard Similarity* measures similarity between these two finite sets ($Sim(\Gamma_1, \Gamma_2)$) by the ratio of the size of intersection of Γ_1 and Γ_2 to the size of their union[36], that is:

$$Sim(\Gamma_1, \Gamma_2) = \frac{|\Gamma_1^L \cap \Gamma_2^L|}{|\Gamma_1^L \cup \Gamma_2^L|} \tag{3.1}$$

Local Comparison

Each triple consists of *Subject*, *Predicate*, *Object*. *Local Comparison* is the average of the similarities of three parts of the triple. Then:

$$Sim(\Gamma_1, \Gamma_2)_{local} = \frac{Sim_s + Sim_p + Sim_o}{3} \tag{3.2}$$

Global Comparison

Global Comparison is to regard this triple as a whole sentence rather than three parts and then compare. The reason behind this is because there are quite a few passive-format triples, like "the agreement; is signed by ; Putin and Obama". If compared with "Putin and Obama; signed; this agreement", *local comparsion* will conclude that they are not similar at all, while in fact they are talking about the same subevent. To prevent miscalculating these tuples, we choose global similarity over local one as `Jaccard Similarity` score.

3.5.3 Semantic similarity

To measure the degree of *semantic similarity*, or more generally *relatedness*, between two lexically expressed concepts is a problem involved in many computational linguistics tasks [6]. *Semantic relatedness* covers more aspects than *semantic similarity* in the sense that the latter often only measures the likeness between entities, while the former regards antonymy (cellphone-battery), antonymy (love-hate) and some other relationships as relatedness metrics. As we are only interested in clustering similar triples when they represent the same fact, *semantic similarity* provides more accurate results.

We refer to WordNet [7] as knowledge source to calculate semantic similarity score between triples. A few measures are available for this, such

[7] http://wordnet.princeton.edu

as HirstSt-Onge [16], LeacockChodorow[22], Resnik[37], JiangConrath[18], Lin[26]. Among these five, Hirst-St-Onge measures *semantic relatedness* as it uses all relationships in WordNet, and remaining four ways calculate *semantic similarity* in that they only use the hyponymy relation. [6] experimentally compared these five measures, and conclude that JiangConrath performs the best overall, followed by Lin, Leacok and Chodorow; Resnik is seriously under-related, and Hirst-St-Onge seriouly over-related. Lin uses the same elements as Jiang-Conrath, which calculates *semantic distance* instead of *semantic similarity*. As Lin provides more promising score than Jiang-Conrath in our dataset[8], we choose Lin for simpler combination with textual similarity.

Lin [26] defines similarity function as follows:

$$sim(x_1, x_2) = \frac{2 \times logP(C_0)}{logP(C_1) + logP(C_2)} \tag{3.3}$$

where C is a synset, word $x_1 \in C_1$, $x_2 \in C_2$, C_0 is the most specific class that subsumes both C_1 and C_2, and $P(C)$ is the probability of encountering an instance of a synset C in specific corpus.

3.5.4 Similarity combination

To combine textual and semantic similarity, we propose the following similarity function to calculate the final similarity score of triple Γ_1 and Γ_2:

$$S(\Gamma_1, \Gamma_2) = \frac{\sum\limits_{w_2 \in \Gamma_2^L} \max\limits_{w_1 \in \Gamma_1^L} sim(w_1, w_2)}{|\Gamma_1^L| + |\Gamma_2^L| - |\Gamma_1^L \cup \Gamma_2^L|} \tag{3.4}$$

where Γ^L denotes the entity list of triple Γ, $|\Gamma^L|$ is the length of the list, $sim(w_1, w_2)$ is the semantic similarity score between words w_1 and w_2 (When $w_1 = w_2$, $sim(w_1, w_2) = 1$), and $|\Gamma_2| \leq |\Gamma_1|$.

Intuition behind this formula is a combination of extractive and abstractive approaches in text summarization. Extractive approaches focuses on extracting *exact* words, phrases, sentences or even paragraphs from text, and abstractive approach needs to rephrase the text using NLP techniques. Wordnet takes semantic co-relations between words into consideration, which lifts the similarity score between triples which are telling the same fact while telling it in different ways.

[8] Check and compare manually

3.5.5 Computation

We generate more than around four million triples from news corpus using existed NLP packages(see section 3.3) [9]. Pair-wise computation complexity is $O(n^2)$, which is extremely expensive in this scale. Also, it does not make much sense to compare triple Γ_1 from event A with triple Γ_2 from event B. However, we are not able to differentiate them before the news information network is created, then using existing techniques is impossible to locate which triples to compare based on the fact that they belong to one event.

As explained in section 3.5, each triple is represented by a set of entities. We then borrow the *Linkage Search* idea from [34] where the unit for computation can also be represented with entities. We first construct a triple-entity bipartite graph and then perform a two-step walk to locate which triples to compute. Take one uncomputed triple Γ for example, the first step is to choose an entity e_1 in Γ^L, and then find all triples connected to e_1, whose count ranges from 1 to 87228. The union set of neighbors for all entities in Γ form the triple group Γ should be compared with, averagely numbering 68599, which is still very expensive.

To reduce computation cost, we propose the following algorithm(line 4 to line 9 in algorithm 1): suppose the similarity score threshold is τ, $|\Gamma_1^L| = l_1$, $|\Gamma_2^L| = l_2$, and denote $|\Gamma_1^L \cap \Gamma_2^L| = |com|$. Then we ideally want that if Γ_1 and Γ_2 are similar to satisfy the inequality::

$$\frac{|com|}{|\Gamma_1^L| + |\Gamma_2^L| - |com|} \geq \tau \tag{3.5}$$

Suppose we fix Γ_1, we are not able to know $|\Gamma_2^L|$ as we are trying to avoid check every triple $|\Gamma_1^L|$ connects. Thus we can tighten equation 3.5.5 based on $(|\Gamma_1^L| + |\Gamma_2^L| - |com|) \geq |\Gamma_1^L|$, and we get equation:

$$\frac{|com|}{|\Gamma_1^L|} \geq \tau \tag{3.6}$$

Therefore, $|com| \geq \tau\ |\Gamma_1^L|$ indicates that at least $\tau\ |\Gamma_1^L|$ entities in Γ_1^L has to be contained by $\Gamma_1^L \cap \Gamma_2^L$. In this case, when checking if $S(\Gamma_1, \Gamma_2)$ is at least τ, instead of picking $\lceil \tau |\Gamma_1^L| \rceil$ entities in Γ_1 to see if they also exist in Γ_2, we can do the opposite by checking the remaining $|\Gamma_1^L| - \lceil \tau |\Gamma_1^L| \rceil + 1$ entities. If these entities are not in $|\Gamma_2|$, then $S(\Gamma_1, \Gamma_2)$ will definitely be less

[9] Some news articles generate more than one hundred triples, and also the headline will be counted three times

than τ. Based on this theorem, we pick $|\Gamma_1^L| - \lceil \tau |\Gamma_1^L| \rceil + 1$ entities with the least size of support set(a set of triples connecting to this entity) from Γ_1^L. The union of these entities' support set is Γ_1's candidate set. The average size for each triple is 2553.

Algorithm 1: Similarity Score Computation

input : $\{\Gamma^L\}$, τ
output: Each Γ's similarity score with potential similar triples

Build one triple-entity bipartite graph Bi;
for *each* Γ^L **do**
    // Generate candidate triples;
    for *each entity e* **do**
        $S(e) \leftarrow$ triples connected to e in graph Bi
    rank S(e) based on set increasing size ;
    $\Psi(\Gamma) \leftarrow \phi$;
    for *each first* $|\Gamma^L| - \lceil \tau |\Gamma_1^L| \rceil + 1$ *entities e* **do**
        $\Psi \leftarrow S(e)$
    // Calculate;
    for *each triple* Γ' *in* Ψ **do**
        use equation 3.5.4 to calculate $S(\Gamma, \Gamma')$

3.6 Cluster

3.6.1 Introduction

Generated triples represent various *actions* . As the same *action* is referred to multiple times in one news article or different ones, such as *Edward Snowden is the leaker of NSA* has been brought up for more than 1,000 times in our dataset, clustering similar instances into groups is necessary.

To fix context and aid in understanding, we assume each triple is a data point belonging to the same space, and we examine this collection of points, then group them into clusters based on the similarity score (see section 3.5) between each other. What we want to achieve from clustering is that triples in each group have high similarity between each other and are able to represent the same action, and different groups are talking about different

actions.

Clustering techniques can be generally divided into *hierarchical* and *partitioning* way. Hierarchical algorithms, as its name suggests, seeks to build a hierarchy of clusters. It consists of "bottom up" approach, starting with its own cluster and paring up clusters as one to move up the hierarchy; and "Divisive" approach, which starts from one cluster and moves down the hierarchy as recursively splitting clusters[10]. This process stops until some criterion is made, such as the number of clusters requested. Partition algorithms focuses on discovering dense connected components of data, so they are less sensitive to outliers and quite flexible in terms of clusters' shape.

As *hierarchical* technique requires some termination criteria which are difficult to establish in our domain, we choose cluster tuples with partition way by which we don't need to specify the number of clusters beforehand.

3.6.2 Density-based partitioning

The intuition behind Density-Based Partitioning lies in the fact that an open set in the Euclidean space can be divided into multiple connected components[5]. One connected component is defined as one cluster, and can grow in any direction the density leads. Thus, Density-Based algorithms are able to discover clusters with arbitrary shapes, and also the stopping criteria is quite clear that as long as dense components are properly connected.

The algorithm DBSCAN (Density Based Spatial Clustering of Applications with Noise) [12] is one of the most common Density-Based algorithms. It requires two parameters, a given distance ϵ and the minimum points needed to form a dense region(minPts). It arbitrarily starts with an unvisited point, retrives its ϵ-neighhood points, and starts a cluster if minPts is satisfied. All points' ϵ-neighhood belong to the same cluster, and this cluster grows until all density-connected points are found. Then start with a new unvisited point, repeat the process until all points are visited. DBSCAN's running time complexity is $O(n^2)$ without using an accelerating index structure, and it needs $O(n^2)$ memory. Also DBSCAN's clustering quality relies much on the minPt-ϵ combination, which may be not appropriate for all clusters.

To reduce parameter's overall control and algorithm's complexity, we apply a more straightforward density-based algorithm to cluster. As we measure similarity between triples(introduced in 3.5), we set similarity threshold τ instead of ϵ between points as *Connectivity* parameter. Each point,

[10] http://en.wikipedia.org/wiki/Hierarchical_clustering

a triple, is isolated in the Euclidean space first, and then an edge is added between two points if their similarity score is greater than τ. After checking all edges, connected components are clusters. *Density* parameter minPts in our algorithm is not used to set the minimum number of nodes in a neighborhood to grow forward, but the number of nodes to present as a cluster at the end. We only store point pairs with similarity score greater than τ in memory, thus it costs $O(|E|)$ memory, and $O(|E|)$ running time($|E|$ is the number of edges of this graph). Pseudo Code of this algorithm is in Algorithm 2 line 1 to line 5(see section 4.3):

Line 6 in Algorithm 2 removes a triple with only one degree. This step gets rid of noisy nodes that connects two not-that-relevant clusters together.

To aid explain how to build news information network, we define *triple cluster* as follows:

Definition 3. *(triple cluster) A triple cluster* Δ *is a group of similar triples representing the same action.* $\Delta = \{\Gamma_1, \Gamma_2, \Gamma_3 .... \Gamma_m\}$.

3.7 Processing running time

Most of the dataset processing time is determined by *Anaphora Resolution* in section 3.2 and *Relation Extraction* in section 3.3. Even though we adopted multi-threading processing, it still took around seven hours to fully process WordWide dataset(210,206 news articles). As we rely on external packages in this step, there is little we can do to speed up the process. This process is regarded as off-line preparation for the *assertion graph* on which we issue query keywords and determine densely linked and semantically related subgraphs.

Chapter 4

Data Model

Definition 4. *(Assertion Graph) Given n* triple clusters *with each triple within cluster labeled with its news_id, timestamp and description, co-occurrence threshold τ_c, the graph created based on the following rules is called an Assertion Graph:*

- *Each triple cluster is regarded as one node.*
- *If node μ and ν share more than τ_c triples labeled with the same news_Id, add one edge between node μ and ν.*

4.1 Node

Node-related features are listed as follows:

- *Node/vertices:* For each triple cluster Δ we generated from clustering in section 3.6, the graph has a corresponding node ν_Δ. We also refer to it as an *action*. Each node has a specific id.

- *Node weights:* We assign a weight $N(\nu_\Delta)$ to each node ν_Δ in the graph. It shows the *popularity* or *prestige* of this node. The value is equal to the frequency this triple has appeared among all raw text, which is also the size of this triple cluster Δ. The higher the weight is, the more popular or important this node is.

- *Node description:* We choose the best description $D(\nu_\Delta)$ for node ν_Δ. One cluster contains various similar descriptions. We pick the one with the highest frequency as this node's description.

- *Node timestamp:* A *triple cluster* contains multiple triples, and so multiple timestamps. We assume that the earliest one among them is the time this *fact* happens, and others are just referring to it in following news articles. We denote ν_t to specify the timestamp.

4.2 Edge

Edge-related features are listed in the following:

- *Edge:* For each pair of nodes μ and ν, if they share more than τ_c [11]triples labeled with same news_Ids(the number of co-occurrences in news), we create an edge between them.

- *Edge Weight:* We use ω_{uv} to denote the edge weight between node μ and ν. $\omega_{\mu\nu}$ is equal to how many triples with same news_Id are shared between μ and ν. We assume that when two *facts* are referred in the same news, they are *semantically related* to each other. The greater the edge weight is, the more semantically related these nodes are.

- *Time-sensed Edge Weight:* The time difference between node μ and ν represents how much time lapses between two *facts*. If fact μ happens a few months later than ν, we say they are not as close as the pair in which one fact happens right after another one. Therefore, the less time-lapse between two nodes, the closer they are. To combine time feature into *Edge Weight*, we define *Time-sensed Edge Weight* for nodes μ and ν as follows:

$$\omega_t(\mu,\nu) = \begin{cases} \omega(\mu,\nu) & |\mu_t - \nu_t| \leq 2 \\ \frac{\omega(\mu,\nu)}{log_2(|\mu_t - \nu_t|)} & otherwise \end{cases} \tag{4.1}$$

4.3 Graph overview

We build the assertion graph following Algorithm 2. Line 1 to 5 explains the way to cluster single triple into clusters. Line 6 removes noisy triples. Then we treat each triple cluster as one node, and connect edges between nodes if they co-occure more than τ_c times(line 8 to 14). Figure 4.1 is a subgraph extracted from the assertion graph we build from dataset 1 (see table 6.1). As explained, each rectangle represents one node with feature *description*, *timestamp* and *frequency*. The number on the edge shows the frequency they co-occur in one article.

[11] τ_c is determined experimentally

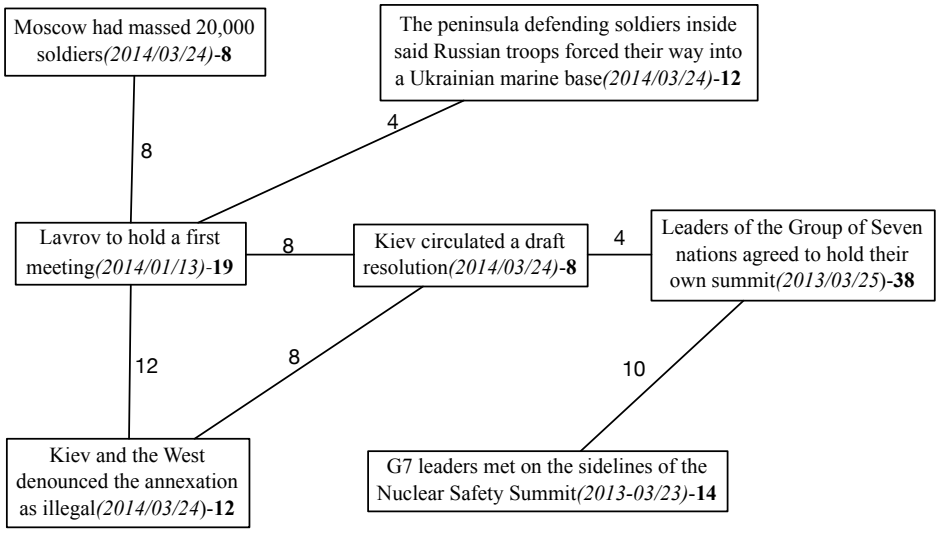


Figure 4.1: Graph overview

Algorithm 2: Building assertion Graph

input : $\{\Gamma\}$ with corresponding similarity scores, τ_s, τ_c
output: Assertion Graph G

Initiate a graph G_c with isolated Γ;
for *each* Γ *in* $\{\Gamma\}$ **do**
    for Γ' *in* Γ*'s candidates* **do**
        if $S(\Gamma, \Gamma') \geq \tau_s$ **then**
            connects one edge between Γ and Γ'

remove Γ *with only fewer than three degree*;
Each connected subgraph is one group with multiple similar triples, denoted as Δ*(triple cluster)*

Initiate a graph G with each Δ as one node;
for *each node* μ *in* G **do**
    for *each node* $\nu \neq \mu$ **do**
        $c = \sum_{\Gamma_1 \in \mu, \Gamma_2 \in \nu} (\Gamma_1.newsId = \Gamma_2.newsId)$;
        if $c \geq \tau_c$ **then**
            add add edge between μ and ν;
            set $\omega_t(\mu, \nu)$ based on equation 4.1;

Output G as the Assertion Graph;

Chapter 5

Problem Formalization and Algorithms

5.1 Preliminaries

As we discussed in section 4.1, each node is associated with *an assertion*, which can be divided into multiple *keywords*. We assume *assertion graph* consists of n nodes $V = \{\nu_1, \nu_2, ..., \nu_n\}$, and let $K = \{\kappa_1, \kappa_2, ..., \kappa_m\}$ be a universe of m keywords. Each node ν_i is associated with a set of keywords $A_i \subseteq K$. If keyword $\kappa_j \in A_i$, we say that node ν_i *contains* keyword κ_j; otherwise node ν_i does not contain keyword κ_j; We say that a subset of nodes$V' \subseteq V$ contain keyword κ_j if there exists at least one node in V' that contains keyword κ_j.

A query Q is a subset of keywords specified by users to find connections between them in the *assertion graph*. We assume that $Q \subseteq K$ as we won't be able to cover any keyword which is not in K. Followed [21], we denote the *cover* of a set of nodes V' with respect to query set Q as $C(V', Q)$, and $C(V', Q) = Q \cap (\cup_{i \in V'} A_i)$. This specifies the intersection of query set Q and the union of keywords associated with node subset V'.

We denote any subgraph as $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$. d is the diameter (the longest shortest path between any two nodes in the graph) of G'. $W' = \sum_{e \in E'} w_e$, representing the sum of weights of all edges E' in subgraph G'. w_{max} is the maximum edge weight in G', and $|E'|$ is the number of edges. To differentiate different kinds of nodes in the subgraph, we define *query node* and *intermediate node* as follows:

Definition 5. *(Query node and Intermediate node) Query node is the node whose set of keywords contains at least one keyword in query Q. Intermediate node is the node whose set of keywords does not contain any keyword in query Q.*

5.2 Problem formalization

5.2.1 Problem statement

Problem 1. *(Query-driven Event Search)Given a news corpus, a set of keywords Q, an objective function f, we seek to build an undirected, weighted graph $G = (V, E)$ first and find a connected subgraph $G' = (V', E')$ such that:*

1. *$C(V', Q) = Q$*
2. *all intermediate nodes have to be on a path connecting query nodes*
3. *$f(G')$ is maximized among all candidates for G'*

The objective function is defined as follows:

$$f(G') = \alpha \frac{1}{d} + (1 - \alpha) \frac{W'}{|E'| w_{max}} \tag{5.1}$$

where α is the parameter trading off two parts' importance, d is the diameter, W' is the total edge weight, $|E'|$ is the number of edges, and w_{max} is the maximum single edge weight of generated graph G'.

Theorem 1. *Problem 1 is NP-complete.*

Proof. We prove this by a reduction from the *Team Formation* (TF)problem. An instance of the TF problem contains a set of n individuals $X = 1, ..., n$, a graph(X, E), and task T, find $X' \subseteq X$, so that $C(X', T) = T$, and the diameter of X' is minimized. It is easy to see that TF is a special case of Problem 1 by setting $\alpha = 1$, in which case, f is maximized exactly when the diameter is minimized. Then there is a solution for TF problem only if there is a solution for our problem, and vice versa. □

5.2.2 Discussion about objective function

The objective function f makes a combination of *structure connectivity* and *semantic relatedness* of a graph. To relatively explain these two parts: 1) Diameter $\frac{1}{d}$ limits the distance between pairs of nodes in generated graph G'. The smaller d is, the more connected G' is. As d is normally in the range of [1, 10], $\frac{1}{d}$ is naturally in the range of (0,1]. $\frac{1}{d}$ can also be regarded as $\frac{d_{min}}{d}$ where d_{min} is the minimum diameter of a generated graph (except for single isolated nodes). 2) In the latter part, $\frac{W'}{|E'|}$ represents the average

edge weight of G', $\frac{W'}{|E'|w_{max}}$ thus is the ratio of average edge weight and max edge weight of G'. As introduced before, $w_{\mu\nu}$ between nodes μ and ν measures the degree of their semantic relatedness, which is a function of number of co-occurences and their time differences as in function 4.1. It may be unfair to compare the total edge weight of two different subgraphs as the frequency of their nodes' text being referred in the dataset influences edge weight. However, measuring the *ratio* of average edge weight and maximum edge weight is independent on other features, thus makes it the good way to constrain the semantic relatedness of a graph. Also $\frac{W'}{|E'|w_{max}}$ is in the range of (0,1], which avoids the possibility that any part dominating the result of their linear combination.

5.3 Heuristic algorithms

In this section, we propose three heuristic algorithms for *query-driven event search* problem:*EdgeGreedy* (`Edge`), *EdgeGreedy_star*(`EG*`)and *RarestFirst_star*(`RF*`). Another fourth algorithm *RarestFirst*(`Rare`) is baseline from [21]. `Edge` differs from `Rare` in the way in which the next node is added to the graph. The change between `Edge` and `EG*` lies in two process called *edge_star* and *node_star*. Details of all algorithms are explained in this section.

5.3.1 Rare

Algorithm 3 (`Rare`) is a variation of the algorithm presented in [21], we regard it as a baseline of our problem. First, for each keyword query $q \in Q$, we compute $S(q)$, which is the set of nodes that contain q, defined as *support set*. Then, we rank all support sets by size and choose one node from the minimum size support set to start building the solution subgraph. As V' in output graph $G'(V', E')$ must contain at least one node in each set $S(q)$, we assume smaller size set has more priority. $p(\nu, S(q))$ refers to the shortest path between node ν and every node in set $S(q)$, that is:

$$p(\nu, S(q)) = min_{\mu \in S(q)} p(\nu, \mu) \tag{5.2}$$

and we call the node picked in set $S(q)$ as ν_{can} . After fixing the node ν in $S(1)$(query q_1), we then pick nodes with $p(\nu, S(q))$ in all other set $S(q)(q \neq q_1)$. When adding nodes to S_{node}, we also add all intermediate nodes that are along the path $p(\nu, \nu_{can})$. After scanning all remaining support sets, all added nodes and paths form a connected subgraph G'_{can}; line 13 and line 14

in algorithm 3 calculate G'_{can}'s diameter and edge weight sum. After calculating all possible solutions, algorithm 3 chooses the candidate subgraph with maximum f value. Recall that we pre-computed all pairs shortest path and use hash tables to store specific attributes(hops, edge weight sum etc.). The running time of algorithm 3 is $O(c|S(1)| \times |Q| + |S(1)| \times |S(node)|^2)$ where c is the average size(around 213) of support set, $|S(1)|$ is the size of rarest support set, averagely numbering 11. $|Q|$ is the size of query set, ranging between [4,10], $|S(node)|$ is the average number of nodes in generated graph, fewer than 12. $|S(1)| \times |S(node)|^2$ is the cost for calculating diameter (line 14) for each candidate subgraph. Algorithm 4 calculates a graph's diameter by exhaustively expanding each node outwards and recording the furtherest step.

Also, we pre-compute the shortest path(minimum hops) between nodes, and the total weight of this path.There may exist multiple shortest paths(same hops) between two nodes but with different total weight, we always choose the one with maximum total weight.

5.3.2 Edge

Algorithm 5 (`Edge`) ranks $S(q)(q \in Q)$ based on increasing size, starts with $S(1)$(support set with minimum size), and keeps adding edges from other sets following increasing size sequence. When picking edges to add to solution set, instead of fixing one node in algorithm 3, `Edge` scans all nodes in the solution set and choose the edge with the endpoint node μ that maximizes the marginal gain of f. The example below explains how algorithm 5 chooses the next node and how it benefits the result.

Example 1. *(Edge example) Assume edges have unit edge weight. Existed subgraph S contains nodes a, b, c and edge (a, b) and (b, c). Candidate nodes are d, e. Figure 5.1(a) shows how node d connects to S, and figure 5.1(b) shows how node e connects to S. d and e are both only one hop away from S. Yet, $f(S \cup e \cup (b, e))$ - $f(S)$ = 0; $f(S \cup d \cup (a, d))$ - $f(S)$ = -0.25(α = 0.5). Even though solution 1 and 2 have the same influence on total edge weight(both are three), the ratio of average weight and maximum weight (both are one), solution 1 increases diameter by 1, and solution 2 still stays one. Thus, Edge will add e to existed subgraph S.*

As each diameter calculating costs $|S(node)^2|$, the total complexity for algorithm 5 is $O(c|S(1)| \times |S(node)|^2)$ where c is the average size of support set, $|S(1)|$ is the minimum size of all support sets, and $|S(node)|$ is the average number of nodes in generated solution.

Algorithm 3: The **Rare** algorithm

input : Graph G(V, E); Query set Q; α
output: A subgraph G(V', E')

find support set for each query in Q;
for *each query* $q \in Q$ **do**
  $S(q) = \{\nu_i | q \in \kappa_i\}$
rank all sets based on the size, and rename them to $S(1)$ *to* $S(|Q|)$;
initiate node solution set S_{node}*, path solution set* S_{path} ;
$S_{node} \leftarrow \phi$;
$S_{path} \leftarrow \phi$;
for *each* $\nu \in S(1)$ **do**
  $S_{node} \leftarrow S_{node} \cup \nu$;
  for $1 < j \leq |Q|$ **do**
    $\nu_{can} \leftarrow arg\ p(\nu, S(j))$;
    $S_{node} \leftarrow \nu_{can} \cup \{\nu_{path(\nu,\nu_{can})}\} \cup S_{node}$;
    $S_{path} \leftarrow path(\nu, \nu_{can}) \cup S_{path}$;
  $d \leftarrow diameter(S_{node}, S_{path})$;
  $W \leftarrow \sum_e e \in S_{path}$;
  $f = \alpha \frac{1}{d} + (1 - \alpha)W$;
  $S_{node} \leftarrow \phi$; $S_{path} \leftarrow \phi$;
$\nu' \leftarrow argmax f$;
$G' = \nu' \cup S'_{node} \cup S'_{path}$

5.3.3 Edge_star

Definition 6. *(Edge_star) Edge_star is a process of adding edges incident on the existing nodes which will not decrease the value of objective function* f*.*

Edge_star is the first step of post-processing generated graph. Algorithm 5 can generate a subgraph $G' = (V', E')$ covering all query keywords and maximizes f among all candidates. However, each addition in algorithm 5 is *one* query node and *one* path connecting two query nodes(along with all intermediate nodes). Thus not all edges between pairs of nodes in V' are covered by E'. To make G' more compact, we propose *Edge_star*(Algorithm 6) to add back missing edges under the condition that this edge addition will not decrease f value. As we don't introduce new nodes to G', G''s diameter

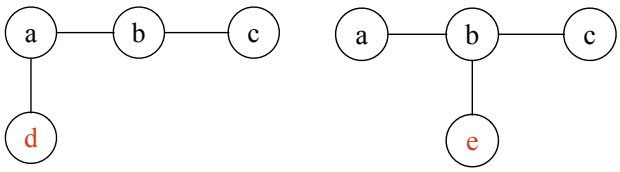
Algorithm 4: The **diameter calculating** algorithm

input : Subgraph $G_1(V_1, E_1)$
output: Diameter of G_1

```
1  /* calculate diameter of G1; record the longest hop for each node */;
2  map ← φ ; d ← −1 ;
3  for each node ν ∈ V1 do
4      ν_step = −1
5  for each node ν ∈ V1 do
6      map_ν = 0; ν_step = 0;
7      queue ← φ;  queue ← queue ∪ ν;
8      while !queue.IsEmpty() do
9          μ ← queue.poll();
10         if d < μ_step then
11             d ← μ_step
12         if map_ν < μ_step then
13             map_ν ← μ_step
14         if N_ν.size() > 0 then
15             for ω ∈ N_ν and ω_step = −1 do
16                 ω_step = μ_step + 1;
17                 queue ← queue ∪ ω;
18 return d as the diamter;
```

will not increase. The only possible way to decrease f is that added edge lowers down the ratio of average edge weight and maximum edge weight in G', and does not decrease the diameter to compensate.

To add back missing edges, we first get each node's(ν) original neighbor set S_1(line 2) and current neighbor set S_2(line 3), and then any node (μ)which is in their difference set ($\{S_1 - S_2\}$) and also in V' can be ν's candidate neighbor. If this addition does not decrease f, algorithm 6 will add it to G'. Let c be the average candidate size for each node in V'(which is less than 1), and the complexity of f value check is $|V'|^2$ (all for diameter calculating), then the total complexity is $c|V'|^2$. G''s average $|V'|$ size is 6(see details in section 6).



(a) Candidate solution 1 (b) Candidate solution 2

Figure 5.1: An example for algorithm 5

5.3.4 Node_star

Definition 7. *(Node_star) Node_star is a process of adding node μ and edge (μ, ν) that connects μ to existing node ν if the addition will not decrease the value of objective function f.*

Node_star aims to add more information to existed subgraph $G_1 = (V_1, E_1)$ without ruining the structure connectivity and semantic relatedness, in other words, decreasing the value of f. As introduced in section 4.3, there are two types of nodes: query and intermediate nodes. We only add intermediate nodes when they are on the path of connecting two query nodes. Therefore, if node *top* is intermediate node, Algorithm 7 doesn't add it first, instead it puts *top*'s neighbors in the queue(see line 26, 27) that created in the beginning and see if it can connect two query nodes afterwards. On the other hand, when it is a query node, Algorithm 7 checks the path connecting *top* and G_1 first as *top* may not be the direct neighbor of V_1(see line 13-17), and then calculate if f is decreased. It is possible that query nodes are quite a few steps away and that it will be expensive to explore too far. Therefore Algorithm 7 breaks when as many as d steps have been explored.

5.3.5 RF* and EG*

We combine `Edge_star` and `Node_star` as *post-process* of generated graphs. Normally `Edge_star` and `Node_star` contribute to each other and iteratively grow the subgraph under the condition of not decreasing objective function value. We stop the process when no new edge nor node can be added. `RF*` is algorithm `Rare` plus the iterative *post-process*, and `EG*` is `Edge` plus the *post-process*.

Algorithm 5: The **Edge** algorithm for Problem 1

input : Graph G(V, E); Query set Q; $\alpha(> 0)$
output: A subgraph H(V', E')

/* *find support set for each query in* Q */;
for *each query* $q \in Q$ **do**
  $S(q) = \{\nu_i | q \in \kappa_i\}$
// *rank all sets based on the size, and rename them to* $S(1)$ *to* $S(|Q|)$;
$S_{node} \leftarrow \phi$;
$S_{path} \leftarrow \phi$;
for *each* $\nu \in S(1)$ **do**
  $S_{node} \leftarrow S_{node} \cup \nu$;
  for $1 < j \leq |Q|$ **do**
    for *each* $\mu \in S(node)$ **do**
      $\delta(\mu) = f(S_{node} \cup S_{path} \cup \mu \cup e(\mu, S_{node})) - f(S_{node} \cup S_{path})$
    $\nu_{can} \leftarrow argmin_{\nu' \in S(j)} \delta(\mu)$;
    $S_{node} \leftarrow S_{node} \cup \{\nu_{path(\nu,\nu_{can})}\} \cup \nu_{can}$;
    $S_{path} \leftarrow path(\nu, \nu_{can}) \cup S_{path}$;
  $d \leftarrow diameter(S_{node}, S_{path})$;
  $W \leftarrow \sum_e e \in S_{path}$;
  $f = \alpha d + (1 - \alpha)W$;
  $S_{node} \leftarrow \phi$; $S_{path} \leftarrow \phi$;
$\nu' \leftarrow argmax f$;
$G' = \nu' \cup S'_{node} \cup S'_{path}$
Output subgraph G';

Algorithm 6: The **Edge_star** algorithm

input : Subgraph G'(V', E'); neighbor information
output: Subgraph $G_1(V_1, E_1)$

```
1  for each node ν ∈ V' do
2      S_1 ← N_ν;
3      S_2 ← N'_ν;
4      for each node μ ∈ {S_1 − S_2} do
5          if μ ∈ V' then
6              if adding μ will not decrease f then
7                  E' ← E' ∪ e(ν, μ);
8              else
9                  continue
10 V_1 ← V'; E_1 ← E';
```

Algorithm 7: The **Node_star** algorithm

input : Subgraph $G_1(V_1, E_1)$
output: Subgraph $G'(V', E')$

```
1  // add all neighbors to a queue;
2  d ← diameter(G_1); step ← 0;
3  q ← an empty queue ;
4  for each node ν ∈ V_1 do
5      S_1 ← N_ν ; S_2 ← N'_ν;
6      for each node μ ∈ {S_1 − S_2} do
7          q.offer(μ)
8  while !q.IsEmpty() do
9      top ← q.poll();
10     prev ← top.prev;
11     L ← φ; P ← φ;
12     if top is query node then
13         while ! prev ∈ V_1 do
14             L ← L ∪ top;
15             P ← P ∪ (top, prev);
16             top ← prev;
17             prev ← prev.prev;
18         δ ← f(G_1 ∪ L ∪ P) - f(G_1);
19         if δ ≥ 0 then
20             G_1 ← G_1 ∪ L ∪ P;
21     else if top is intermediate node then
22         step ← step + 1;
23         if step == d then
24             break;
25         S_top ← N_top;
26         for each node μ ∈ S_top do
27             q.offer(μ);
28 V_1 ← V'; E_1 ← E';
```

Chapter 6

Experimental Results

6.1 Datasets

There are news datasets available online, such as Reuters-21578 [12], and New York Times Annotated Corpus[13]. However, these datasets are mostly from only one source, and are processed(tagged) for specific purposes such as text categorization and summarization. Also, we aim to find interesting events with respect to user-input queries, so it is better that all news in the dataset are from breaking news category instead of some regular news report. Because of these reasons and also the intention of getting more current results, we subscribed to RSS feeds to collect three datasets from different categories.

We collect Worldwide news from sixteen news websites(see table 6.1) from June 1st, 2013 until May 1st, 2014. Each feed provides different number of news every day, approximately ten to thirty breaking news. We crawled news' headlines, introduction, main text and corresponding timestamps. The whole dataset is in English. As we only analyze *text* in our project, we get rid of all top news contains only slideshows, gallery, pictures or video. Distribution of news articles from different web sources is indicated in the following table 6.1. As shown in table 6.1, three of news sources are in UK, one from Canada, one from Australia, and the rest are from US. As it turned out most breaking news we crawled falls into the political category, we then crawled another dataset from Verge to technology category (see table 6.2).

The Onion is unveiled, by *The Boston Globe*, to be a not a legitimate news source but a website full of satire and fake news [14]. As there are only 1162 out of 210,206 news articles, we think it won't have much influence on the assertion graph we created and following event search on it.

We created an *assertion graph* following all steps introduced in chap-

[12] http://www.daviddlewis.com/resources/testcollections/reuters21578/

[13] https://catalog.ldc.upenn.edu/LDC2008T19

[14] http://www.boston.com/culturedesk/2013/03/06/the-onion-fake-globe-uncovers/t7S8gjs3Wsmzr2L8gC6ZOO/story.html

News Sources	Nation	Number	News Sources	Nation	Number
BBC	UK	32430	CBC	Canada	9273
CBS	US	14733	CNN	US	17729
Denver Post	US	3416	Fox News	US	5182
Guardian	UK	19130	Huffington Post	US	18912
NBC	US	10968	News.com.au	Australia	30431
Reuters	UK	11295	TheOnion	US	1162
Time	US	11214	USA	US	18869
WorldNews	US	30022	Washington Post	US	3259

Table 6.1: Worldwide dataset news distribution

Type	News Sources	Nation	Number
Tech	Verge	US	12187

Table 6.2: Verge dataset

ter 3 and 4 for each dataset. The statistics for each dataset's generated triples and corresponding assertion graph are presented in table 6.3. Items in "Connected" column specifies how many dense connected subgraphs this assertion graph contains.

Type	News	Assertions	Nodes	Edges	Connected
Worldwide	210,206	3,847,640	120,507	301,696	142
Tech	12,187	241,217	15,143	23,062	24

Table 6.3: Statistics for worldwide and verge dataset

6.2 Performance evaluation

This section evaluates four algorithms `Rare`, `RF*`, `Edge`, and `EG*` on the *structural density* and *semantic relatedness* of generated subgraphs. We report sugraph's multiple features such as number of nodes, number of edges and so on. Yet, we evaluate performance mostly based on diameter, weightratio and objective function value. All algorithms are implemented in Java and were run on a Linux server with Intel Xeon CPU X5570 (2.93GHz) and 128GB RAM.

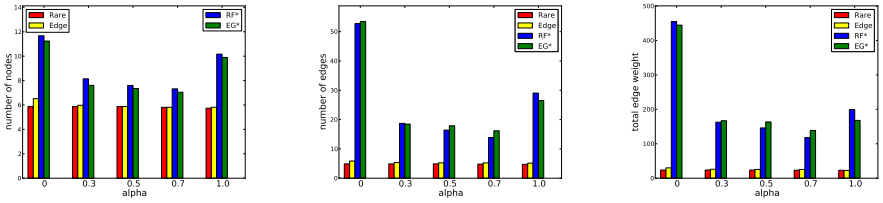
6.2.1 Query set generation

Each query set consists of multiple keywords between which users are interested in exploring connections. To acquire more meaningful events with respect to query sets, we constrain keywords in query set satisfying the following conditions: 1) $Q = \{q_1, q_2, q_3...\}$; K is the universal set of keywords of total assertion graph G. $Q \subseteq K$. 2) Keywords in Q are related with each other, meaning that for each pair of keywords (q_i, q_j), there is at least one path between (ν, μ) where $\nu \in S_i$ and $\mu \in S_j$. S_i and S_j are respectively support sets for query q_i and q_j (see details in section 5.1).

To generate legal query sets, we collect one keyword file for each dense connected subgraph in assertion graph G, and rank those keywords based on frequency. We then pick top three as fixed keywords and randomly pick a few(depending on how many keywords needed) from the remaining to form a query set.

6.2.2 Parameter changing

Figures 6.1(a) to 6.4(a) shows how generated graph G' changes with α increasing from [0,1]. We input 142 query sets(each one with six key words) for Worldwide dataset and twenty-four query sets for Verge dataset, and calculated the average value for every feature. Horizontally comparing all features, figure 6.1(a), 6.1(b) and 6.1(c) have the similar trend when $\alpha = 0$ or $\alpha = 1$ they have greater values than those when α in between. The reason behind this is because only weightratio or diameter is constraining the linear function when $\alpha = 0$ or 1, thus more nodes and edges are added. And, the total edge weight and number of edges of G' are dependent on the number of nodes.

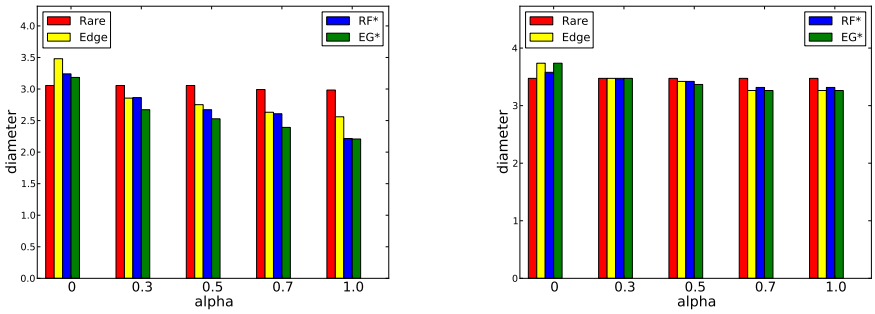


(a) Number of nodes (b) Number of edges (c) Total edge weight

Figure 6.1: Statistics of G' when α ranges between [0,1]

Diameter d, weightratio r and f value are three metrics we use to measure

the algorithms' performance recall that we defined $f = \alpha \frac{1}{d} + (1 - \alpha)\ r$. Figure 6.2(a) and figure 6.2(b) plot the diameter as a function of α. With α increasing(d is getting more power over f), every algorithm except for `Rare` outputs denser subgraph. As we can see, in Worldwide dataset(figure 6.2(a)), `EG*` yields the smallest diameter for each α value (excluding $\alpha = 0$), leading baseline `Rare` with about 12% to 26% decrease. As expected, `RF*` outperforms `Rare` (about 6% to 25% decrease), and `EG*` performs better than `Edge` (about 6% to 14% decrease) as post process `node_star` and `edge_star` help adjust generated graph. `Edge` outperforms `Rare` by about 6% to 15 % as the former takes f value change into consideration when adding next node to existed subgraph. However, for Verge Dataset(figure 6.2(b)), even though `EG*` still performs best, the gap between approaches is smaller than that of Worldwide dataset, even the largest gap is 6%. We will explain the reason in the next paragraph combining weightratio and f.

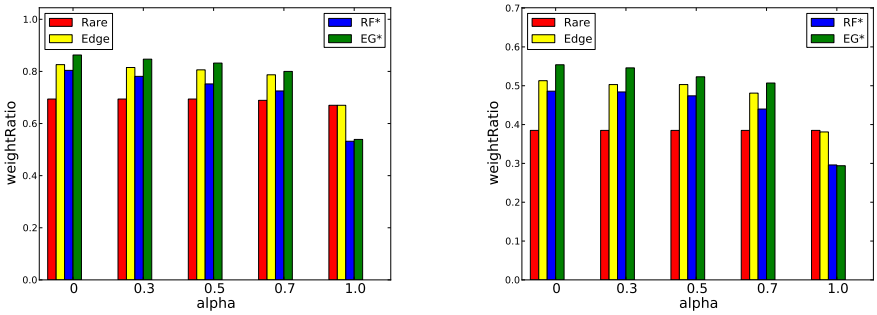


(a) Worldwide (b) Verge

Figure 6.2: Diameter of G' when α ranges between [0,1]

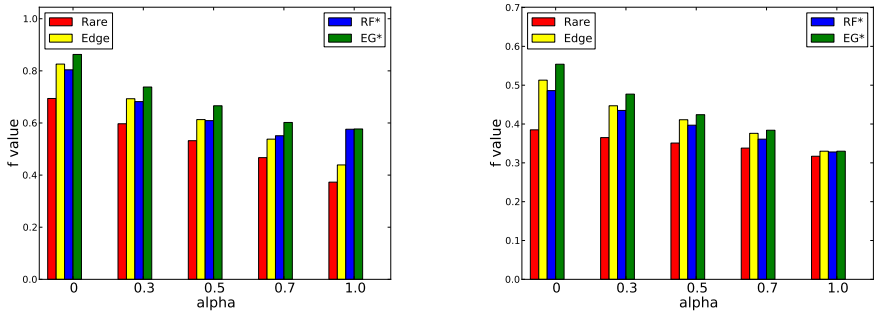
Figure 6.3(a) and figure 6.3(b) shows how weightratio r changes with increasing α. `Rare` still has little change as it builds the subgraph by only measuring shortest path between fixed node and candidate nodes. With no surprise, `EG*` still outperforms other approaches by leading runner-up `Edge` by around 3%, `RF*` by 10%, and `Rare` 20% in Worldwide Dataset. Combined with diameter's influence, `Edge` also leads in the objective function value, separately leading `Edge`, `RF*` and `Rare` 8%, 9 % and 25% in figure 6.4(a). In comparison, Verge Dataset has bigger gap between algorithms in weightratio (see figure 6.3(b)), which are 4%, 10%, and 36% between first three approaches and `EG*`, and yet smaller gap in objective function value(3%, 6%, 20%). The reason behind this is that Verge Dataset is sparser than World-

wide, so when running `node_star` and `edge_star`, Verge Dataset tends to expand to neighboring nodes so that the weightratio can compensate the loss of diameter as there are not available edges to add to existed subgraph.



(a) Worldwide (b) Verge

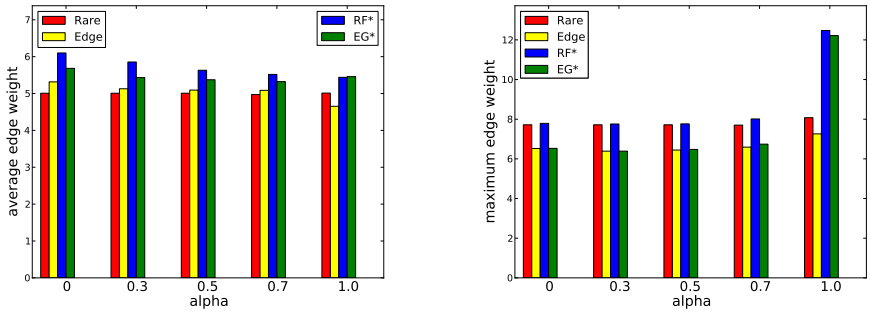
Figure 6.3: Weightratio of G' when α ranges between [0,1]



(a) Worldwide (b) Verge

Figure 6.4: Objective function value of G' when α ranges between [0,1]

Figure 6.5(a) and figure 6.5(b) plots the average edge weight and maximum edge weight trend with increasing α. We don't use these features to measure algorithms' performance, yet we can observe from the trend that α doesn't impact these features that much(except for maximum edge weight when $\alpha = 1$). What causes this may be no matter which part is dominating f, no noisy edge can be added as either d or r can constrain it.



(a) Worldwide (b) Verge

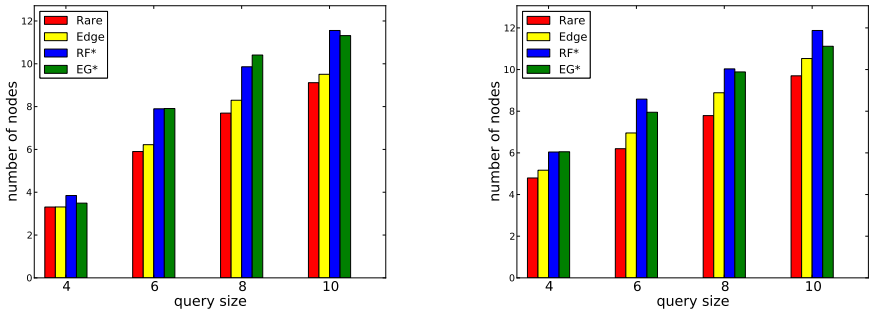
Figure 6.5: Average and maximum edge weight of G' when α ranges between [0,1]

6.2.3 Query set size changing

To evaluate how the size of query set influences results, we conduct experiments for all algorithms by varying query set size from four to ten. The number of nodes, number of edges and total edge weight increase as expected. Yet, the increase rate decreases as query size increases, such as the number of nodes (figure 6.6(a)) when query size almost doubles when query size was four, while the rate drops to 25%, and 17% when query size changing from six to eight and eight to ten. What's interesting that at most eleven nodes are among the four approaches, which makes it still readable for users. The trend still applies to the number of edges and total edge weight(we omit them as they have similar trends as figure 6.6(a)).

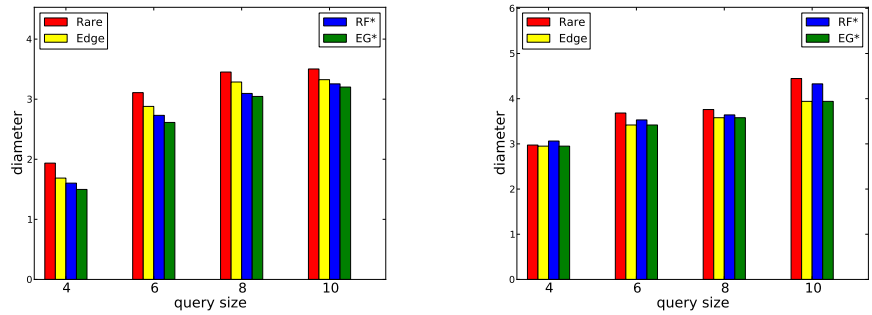
Figure 6.7(a), 6.8(a) and 6.9(a) present feature diameter, weighRatio and f value of G', which are three metrics we use to measure algorithms' performance. `EG*` outperforms others in each plot, having the smallest diameter, largest weightratio(except for query size = 4), and the most f value in both datasets. However, the gap between diameter in Worldwide is larger than that of Verge dataset, and the gap between weightratio is the other way around. Same reasoning to that of 6.2.2 applies here: when more keywords are input, Verge tends to add more nodes instead of adding more edges between existed nodes. What's also worth mentioning is that `Edge` performs better than `RF*` in Verge Dataset(smaller diameter, greater weightratio, greater f value) because node_start and edge_star do not have as much influence as those in Worldwide dataset.

Horizontally comparing, diameter increases with query size expanding.



(a) Worldwide (b) Verge

Figure 6.6: Number of nodes in G' when query size ranges between [4, 10]



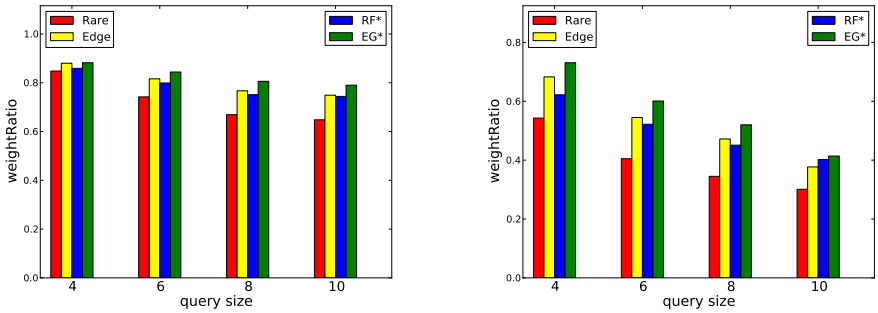
(a) Worldwide (b) Verge

Figure 6.7: Diameter of G' when query size ranges between [4, 10]

Yet, similar to trend in figure 6.6(a), the increase rate becomes smaller and almost flattens out when query number changes from eight to ten. Weight-ratio is quite steady during the change. Thus f values experiences a drop from four to six, and stays steady afterwards. These plots illustrate that our algorithm is robust enough to deal with large query set sizes, still being able to output structurally dense subgraph and semantically related subgraph even though quite a few keywords are input.

6.2.4 Running time

Table 6.4 and 6.5 report the running time and steps of `node_star` and `edge_star` in `EG*` and `RF*` separately for Worldwide and Verge Dataset.



(a) Worldwide (b) Verge

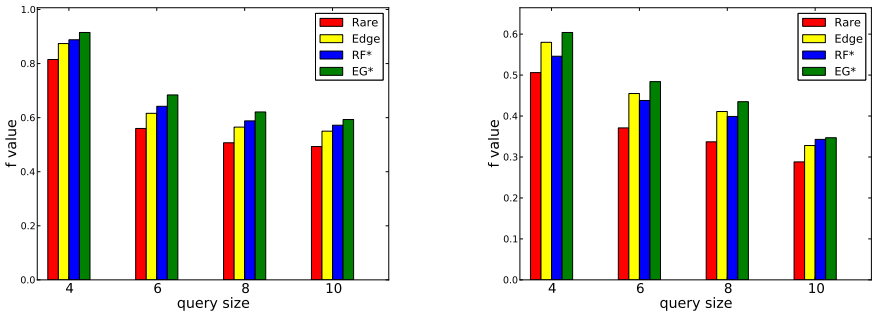
Figure 6.8: Weightratio of G' when query size ranges between [4, 10]

Still, the running time for each approach is the average time of separately running 142 and twenty-three query sets for each dataset, and α is set to 0.5. With no surprise, `Rare` runs fastest among all approaches as it picks the next node only depending on shortest path while `Edge` needs to calculate marginal f gain for all candidates. Items in e*(E) and n*(E) respectively represent the count of steps in which `Edge_star` and `Node_star` actually add edges and nodes to existed subgraph in algorithm `EG*`, and e*(R) and n*(R) are for `RF*`. We can observe from both tables that `EG*` has much more edge and node addition than `RF*`, and when query size is four, six and eight, Verge has more node addition and less edge addition than Worldwide, which verifies the explanation in section 6.2.3 that Verge tends to add more nodes to compensate for less edge addition.

size	Edge	EG*	e*(E)	n*(E)	Rare	RF*	e*(R)	n*(R)
4	102	110	1.90	1.67	11	13	0.58	0.22
6	103	109	2.57	2.18	7	12	0.99	0.41
8	110	130	2.86	2.29	3	15	1.11	0.43
10	152	166	3.06	2.43	4	16	1.19	0.47

Table 6.4: Running time(ms.) for worldwide dataset

In summary, our extensive experiments on news datasets show that our approaches are effective and efficient, generating much preferred results than the baseline. In particular, `EG*` outperforms all other algorithms.



(a) Worldwide (b) Verge

Figure 6.9: Objective function value of G' when query size ranges between [4, 10]

size	Edge	EG*	e*(E)	n*(E)	Rare	RF*	e*(R)	n*(R)
4	47	50	1.87	2	10	14	0.42	0.26
6	49	51	2.47	2.37	4	8	0.76	0.75
8	45	47	2.47	2.40	3	6	0.93	0.87
10	139	145	2.07	2	7	28	1.23	0.76

Table 6.5: Running time(ms.) for verge dataset

6.3 Case study

These experiments aim to show that our problem definition and corresponding algorithms can produce meaningful and reasonable results. As it is difficult to evaluate the quality of results in a way that does not involve graph's internal features but semantics, we conducted a case study in which we provide results of algorithm `EG*` for a few breaking-news-related query sets. To be more convincing, we pick three breaking news from Worldwide news, and one from Technology dataset. Chosen breaking news are 1) Snowden leaking classified documents[15]; 2) 2014 Ukrainian revolution[16] and 3) Malaysia Airlines Flight 370 disappeared [17]. In this case study, we seek to find if generated subgraph conveys related *actions* and is readable for users. We pick query keywords we are interested in exploring, and the total query sets

[15]http://en.wikipedia.org/wiki/Edward_Snowden
[16]http://en.wikipedia.org/wiki/2014_Ukrainian_revolution
[17]http://en.wikipedia.org/wiki/Malaysia_Airlines_Flight_370

are as follows:

Q1 = { Snowden, NSA, Russia, Chinese, United States, asylum}
Q2 = { Crimea, Russia, Putin, Kiev, moscow, Obama, yanukovych}
Q3 = { mh370, Malaysia, search, route, Chinese, Australia, Indian}

We search $Q1$, $Q2$ and $Q3$ in Worldwide dataset, and $Q4$ in Verge dataset. The generated subgraphs are shown separately in figure 6.10, 6.11 and 6.12. Resulting subgraphs have small diameter, high weightratio and f value as presented in table 6.6. The structure density and semantic relatedness of generated graph are very much dependent on the original assertion graph and also input queries. Thus, results may demonstrate different patterns. For instance, nodes in figure 6.10 have multiple paths between each other, while in figure 6.11 nodes are connected by a key node in the middle and there is only one path between pair of nodes.

Case	# of nodes	# of edges	diameter	weightratio	f value
Q1	12	16	4	0.89	0.57
Q2	12	11	2	0.99	0.728
Q3	6	10	2	0.98	0.727

Table 6.6: Statistics of generated subgraphs

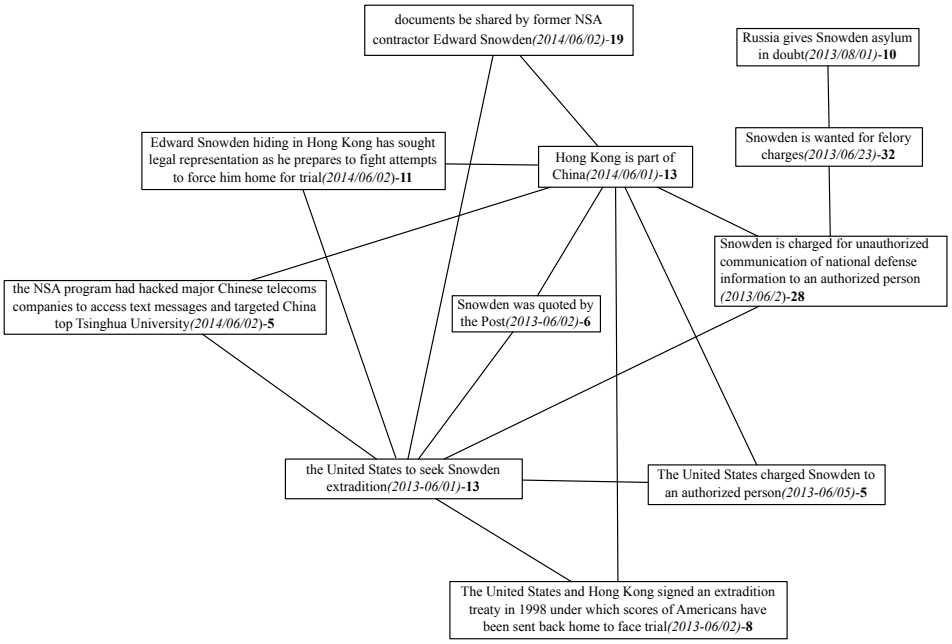


Figure 6.10: Generated subgraph for "snowden" scenario

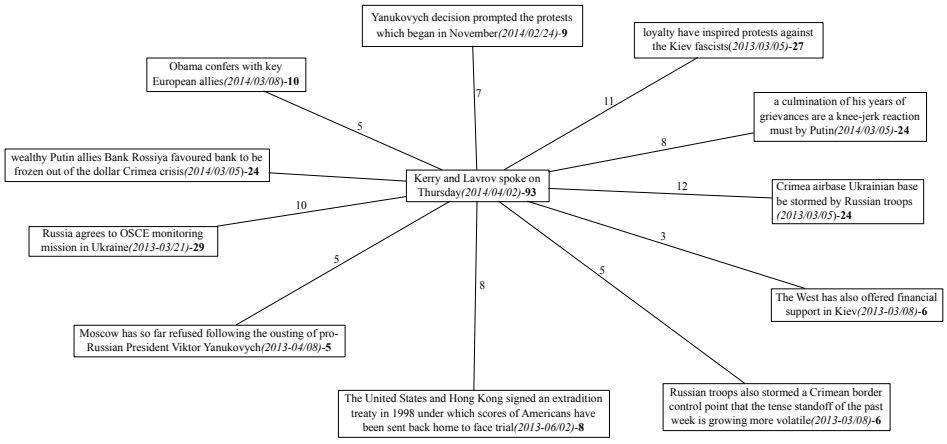


Figure 6.11: Generated subgraph for "ukraine" scenario

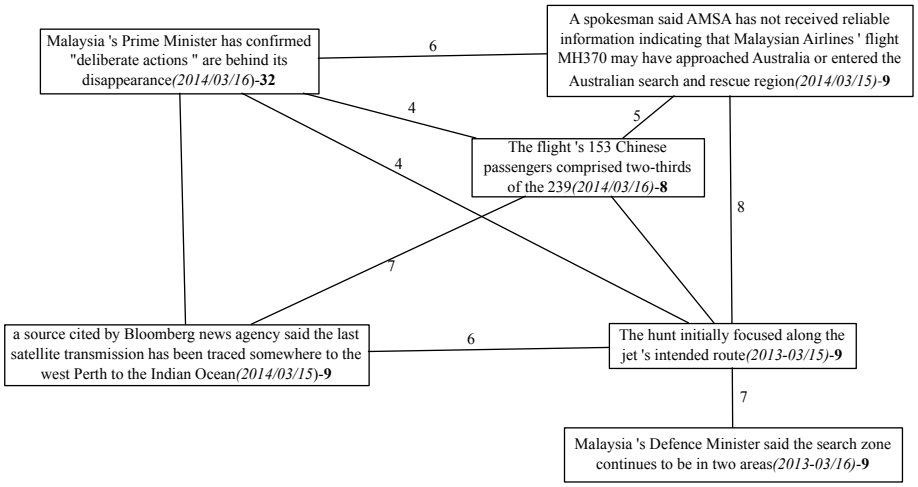


Figure 6.12: Generated subgraph for “mh370” scenario

Chapter 7

Conclusion and Future work

In this paper, we studied the problem of building a query-driven event search engine which determines a densely-linked, semantically related and query-covering subgraph from a news information network built from raw news corpus. We first introduced the methodology to build a news information network, an undirected and weighted graph, from raw news articles, and formally defined the *Query-driven event search* problem and developed a new objective function with which rank candidate subgraphs. We proved this problem is NP-complete and proposed a few heuristics algorithms to solve it. In experiment with real-life news dataset, we evaluate our algorithms' performance by measuring output graph's related features such as diameter, weight, number of edges etc. We also did a few case studies by inputing a few event-specific keywords, and examined if generated subgraphs make sense. The experiment shows that our problem definition has practical meaning by demonstrating users the best option of how those queries are related with each other and form an event with multiple actions. Performance study also indicates that our algorithms can generate answers which combines the structure density and semantic relatedness.

In future work, we want to extend our framework to apply to other types of graphs to explore more applications. For instance, if applied in DBLP dataset, it can be used for beginner graduate student to narrow down k related papers covering areas (captured by keywords) he or she is interested in. However, processing such queries may call for adaption to our current techniques. We need to carefully think about the data model, particularly the meaning of edges and edge weights. We also want to upgrade our framework to adjust on-line update, which means adding most current news article to existing news information network incrementally and enables users to get the most updated event with respect to queries.

Bibliography

[1] Charu C. Aggarwal and Haixun Wang. *Managing and Mining Graph Data*. Springer Publishing Company, Incorporated, 1st edition, 2010.

[2] Reid Andersen and Kumar Chellapilla. Finding dense subgraphs with size bounds. In *WAW*, pages 25–37, 2009.

[3] Albert Angel, Nick Koudas, Nikos Sarkas, and Divesh Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. *CoRR*, abs/1203.0060, 2012.

[4] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. Greedily finding a dense subgraph. *J. Algorithms*, 34(2):203–221, February 2000.

[5] P. Berkhin. A survey of clustering data mining techniques. *Grouping Multidimensional Data*, pages 25–71, 2006.

[6] Alexander Budanitsky and Graeme Hirst. Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures. *Workshop on WordNet and Other Lexical Resources, Second meeting of the North American Chapter of the Association for Computational Linguistics, Pittsburgh, USA*, 2001.

[7] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization*, APPROX '00, pages 84–95, London, UK, UK, 2000. Springer-Verlag.

[8] James Cheng, Yiping Ke, Wilfred Ng, and Jeffrey Xu Yu. Context-aware object connection discovery in large graphs. In Yannis E. Ioannidis, Dik Lun Lee, and Raymond T. Ng, editors, *ICDE*, pages 856–867. IEEE, 2009.

[9] Dipanjan Das and Andr F. T. Martins. A survey on automatic text summarization, 2007.

[10] Yon Dourisboure, Filippo Geraci, and Marco Pellegrini. Extraction and classification of dense communities in the web. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 461–470, New York, NY, USA, 2007. ACM.

[11] Günes Erkan and Dragomir R. Radev. Lexrank: Graph-based lexical centrality as salience in text summarization. *J. Artif. Int. Res.*, 22(1):457–479, December 2004.

[12] Martin Ester, Hans peter Kriegel, Jrg S, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[13] Christos Faloutsos, Kevin S. McCurley, and Andrew Tomkins. Fast discovery of connection subgraphs. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 118–127, New York, NY, USA, 2004. ACM.

[14] David Gibson, Ravi Kumar, and Andrew Tomkins. Discovering large dense subgraphs in massive graphs. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 721–732. VLDB Endowment, 2005.

[15] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. Blinks: Ranked keyword searches on graphs. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 305–316, New York, NY, USA, 2007. ACM.

[16] Graeme Hirst and David St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. In Christiane Fellbaum, editor, *WordNet: An Electronic Lexical Database*, pages 305–332. MIT Press, 1998.

[17] Arvind Hulgeri and Charuta Nakhe. Keyword searching and browsing in databases using banks. In *Proceedings of the 18th International Conference on Data Engineering*, ICDE '02, pages 431–, Washington, DC, USA, 2002. IEEE Computer Society.

[18] J.J. Jiang and D.W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proc. of the Int'l. Conf. on Research in Computational Linguistics*, pages 19–33, 1997.

[19] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In Klemens Bhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-ke Larson, and Beng Chin Ooi, editors, *VLDB*, pages 505–516. ACM, 2005.

[20] Gjergji Kasneci, Shady Elbassuoni, and Gerhard Weikum. Ming: Mining informative entity relationship subgraphs. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 1653–1656, New York, NY, USA, 2009. ACM.

[21] Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 467–476, New York, NY, USA, 2009. ACM.

[22] C. Leacock and M. Chodorow. Combining local context and wordnet similarity for word sense identification. In Christiane Fellfaum, editor, *MIT Press*, pages 265–283, Cambridge, Massachusetts, 1998.

[23] Heeyoung Lee, Angel Chang, Yves Peirsman, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Deterministic coreference resolution based on entity-centric, precision-ranked rules. *Comput. Linguist.*, 39(4):885–916, December 2013.

[24] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky. Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, CONLL Shared Task '11, pages 28–34, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[25] Pei Lee, Laks V. S. Lakshmanan, and Evangelos E. Milios. Keysee: supporting keyword search on evolving events in social streams. In *KDD*, pages 1478–1481, 2013.

[26] Dekang Lin. An information-theoretic definition of similarity. In *In Proceedings of the 15th International Conference on Machine Learning*, pages 296–304. Morgan Kaufmann, 1998.

[27] Gang Luo, Chunqiang Tang, and Ying li Tian. Answering relationship queries on the web. In Carey L. Williamson, Mary Ellen Zurko, Peter F.

Patel-Schneider, and Prashant J. Shenoy, editors, *WWW*, pages 561–570. ACM, 2007.

[28] Inderjeet Mani. Multi-document summarization by graph search and matching. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-97*, pages 622–628. AAAI, 1997.

[29] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[30] Mausam, Michael Schmitz, Robert Bart, Stephen Soderland, and Oren Etzioni. Open language learning for information extraction. In *Proceedings of Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CONLL)*, 2012.

[31] Yuval Merhav, Filipe Mesquita, Denilson Barbosa, Wai Gen Yee, and Ophir Frieder. Extracting information networks from the blogosphere. *ACM Trans. Web*, 6(3):11:1–11:33, October 2012.

[32] Filipe Mesquita, Jordan Schmidek, and Denilson Barbosa. Effectiveness and efficiency of open relation extraction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 447–457. Association for Computational Linguistics, October 2013.

[33] Ruslan Mitkov and Wolverhampton Wv Sb. Anaphora resolution: The state of the art. Technical report, 1999.

[34] Evangelos E. Milios Pei Lee, Laks V.S. Lakshmanan. Incremental cluster evolution tracking from highly dynamic network data. *ICDE*, abs/1309.7313, 2014.

[35] Kira Radinsky and Eric Horvitz. Mining the web to predict future events. In *WSDM*, pages 255–264, 2013.

[36] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, Cambridge, 2012.

[37] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *In Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 448–453, 1995.

[38] Mauro Sozio and Aristides Gionis. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '10, pages 939–948, New York, NY, USA, 2010. ACM.

[39] Hanghang Tong and Christos Faloutsos. Center-piece subgraphs: Problem definition and fast solutions. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 404–413, New York, NY, USA, 2006. ACM.

[40] Ying Xu, Mi-Young Kim, Kevin Quinn, Randy Goebel, and Denilson Barbosa. Open information extraction with tree kernels. In *HLT-NAACL*, pages 868–877, 2013.

[41] Rui Yan, Liang Kong, Congrui Huang, Xiaojun Wan, Xiaoming Li, and Yan Zhang. Timeline generation through evolutionary trans-temporal summarization. In *EMNLP*, pages 433–443. ACL, 2011.