

TOWARD AUTOMATED  
FACT MONITORING AND CHECKING

by

NAEEMUL HASSAN

Presented to the Faculty of the Graduate School of  
The University of Texas at Arlington in Partial Fulfillment  
of the Requirements  
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

AUGUST 2016

Copyright © by NAEEMUL HASSAN 2016

All Rights Reserved

To Ammu, Abbu, Niapu,  
wife Titly  
and uncle Hero.

## ACKNOWLEDGEMENTS

I would like to thank my supervising professor Dr. Chengkai Li for constantly motivating and encouraging me, and also for his invaluable advice during the course of my doctoral study. I wish to thank Dr. Sharma Chakravarthy, Dr. Gautam Das, Dr. Leonidas Fegaras and Dr. Ingmar Weber for their interest in my research and for taking the time to serve on my dissertation committee.

I would also like to extend my appreciation to the Computer Science and Engineering department for providing financial support for my doctoral studies. I wish to thank Pamela McBride and Sherri Gotcher for their support and encouragement. I am especially grateful to Dr. Jun Yang, Dr. Nan Zhang, Professor Bill Adair and Dr. Cong Yu for their interest in my research and for the helpful discussions and invaluable comments.

I am grateful to all the teachers who taught me during the years I spent in school, first in Ideal School and College, then in Notre Dame College, then in Bangladesh University of Engineering and Technology (BUET) and finally in this university at the Unites States. I would like to thank Sayeed Hyder and Nafees Ahmed Kuntal for encouraging and inspiring me to pursue graduate studies.

I would like to thank my current and previous lab-mates – Xiaonan Li, Ning Yan, Nandish Jayaram, Afroza Sultana, Gensheng Zhang, Abolfazl Asudeh, Sona Hasani, Fatma Arslan and many others of the Innovative Database and Information

Systems Research (IDIR) Lab for collaborating on the research projects making my stay at ERB 514 an enjoyable one.

I would like to express my deep gratitude to my parents who have encouraged and inspired me and sponsored my undergraduate and graduate studies. I am extremely fortunate to be so blessed. I am also extremely grateful to my sister Rubama, wife Titly, relatives and especially uncle Hero for their sacrifice, encouragement and patience. I also thank several of my friends who have helped me throughout my career.

Finally, I thank Allah, my creator, for letting me through all the difficulties. I have experienced Your guidance day by day. You are the one who let me finish my degree. I will keep on trusting You for my future. Thank you, Lord.

JULY 18, 2016

## ABSTRACT

### TOWARD AUTOMATED FACT MONITORING AND CHECKING

NAEEMUL HASSAN, Ph.D.

The University of Texas at Arlington, 2016

Supervising Professor: Dr. Chengkai Li

Public figures such as politicians make claims about “facts” all the time. Oftentimes there are false, exaggerated and misleading claims on important topics, due to careless mistakes and even deliberate manipulation of information. With technology and modern day media helping spread information to mass audiences through all types of channels, there is a pressing need for checking the veracity of factual claims important to the public. Journalists and citizens spend a good amount of time doing that. More and more dedicated platforms and institutes are being created for fact-checking. This nascent genre of investigative reporting has become a basic feature of political coverage, especially during elections, and plays an important role in improving political discourse and increasing democratic accountability. Part of the goal of *computational journalism* is use computing to automate fact-checking. There are many computational and journalistic challenges toward a fully automated fact-checking system. This dissertation presents these challenges and focuses on the research areas where breakthroughs are needed. Toward automated fact-checking, we developed tools to find check-worthy factual claims from natural language sentences.

Specifically, we prepared a U.S. presidential debate dataset and built classification models to distinguish check-worthy factual claims from non-factual claims and unimportant factual claims. We also identified the most effective features based on their impact on the classification models’ accuracy. We built a platform, ClaimBuster, which uses the classification model and presents check-worthy factual claims spoken during all the 2016 U.S. presidential election primary debates.

Like for automated fact-checking, advanced computation techniques are also necessary for newsworthy fact discovery, especially from live events. Reporters always try hard to bring out attention-seizing factual statements backed by data, which may lead to news stories and investigation. Facts can be stated on data from domains outside of sports and social media, including stock data, weather data, and criminal records. These facts are not only interesting to reporters but also useful to financial analysts, scientists, and citizens. Database and data mining researchers have started to push the frontiers of automated significant fact discovery and monitoring. This dissertation addresses the problem of significant facts monitoring during live events such as a basketball game, hourly weather updates and so on. Technically, we consider an ever-growing table of objects with *dimension* and *measure* attributes. We define situational fact, a “contextual” skyline tuple that stands out against historical tuples in a context when a set of measure attributes are compared. A context is specified by a conjunctive constraint involving dimension attributes. New tuples are constantly added to the table, reflecting events happening in the real world in a live fashion. The goal is to discover constraint-measure pairs that qualify a new tuple as a contextual skyline tuple and discover them quickly before the event becomes yesterday’s news. A brute-force approach requires exhaustive comparison with every tuple, under every constraint, and in every measure subspace. We design algorithms in response to these

challenges using three corresponding ideas—tuple reduction, constraint pruning, and sharing computation across measure subspaces. Furthermore, we present an end-to-end system, including fact ranking, fact-to-statement translation and keyword-based fact search.

In addition to addressing the fact-checking and fact-monitoring problem and thereby pushing the boundary of *computational journalism* forward, this dissertation also focuses on multi-dimensional Pareto-optimal analysis; specifically, given a set of multi-dimensional points, finding the set of points which are not worse than any other points on all dimensions. This dissertation finds applications of Pareto-optimality and its variants in group recommendation, crowdsourcing, and other domains. Traditional Pareto frontier (skyline) computation is inadequate to answer queries which need to analyze not only individual points but also groups of points. To fill this gap, this dissertation proposes a novel concept *Skyline Groups* that represents groups which are not dominated by any other groups. It also demonstrates applications of *Skyline Group* through a web-based system in question answering, expert team formation and paper reviewer selection.



## TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . .	iv
ABSTRACT . . . . .	vi
LIST OF ILLUSTRATIONS . . . . .	xii
LIST OF TABLES . . . . .	xiv
Chapter	Page
1. Introduction . . . . .	1
2. Finding Check-worthy Factual Statements . . . . .	4
2.1 Introduction . . . . .	4
2.2 Limitations Of Current Practices Of Fact-Checking . . . . .	5
2.3 The “HOLY GRAIL” . . . . .	7
2.3.1 Computational Challenges . . . . .	8
2.3.2 Journalistic Challenges . . . . .	10
2.4 Related Work . . . . .	11
2.5 ClaimBuster . . . . .	12
2.5.1 Classification and Ranking . . . . .	12
2.5.2 Data Collection . . . . .	13
2.5.3 Dataset . . . . .	13
2.5.4 Ground-truth Collection . . . . .	15
2.5.5 Feature Extraction . . . . .	19
2.5.6 Evaluation . . . . .	22
2.5.7 ClaimBuster Platform . . . . .	26
2.6 CASE STUDY: 2016 U.S. Presidential Election Primary Debates . . . . .	29

2.6.1	Data Collection . . . . .	29
2.6.2	Finding Check-worthy Factual Claims . . . . .	29
2.6.3	Topic Detection . . . . .	30
2.6.4	Verdict Collection . . . . .	33
2.6.5	Analysis . . . . .	34
2.7	Future Plan . . . . .	38
3.	Automatic Fact Monitoring . . . . .	41
3.1	Introduction . . . . .	41
3.2	Related Work . . . . .	47
3.3	Problem Statement . . . . .	49
3.4	Solution Overview . . . . .	53
3.5	Prototype System: <b>FactWatcher</b> . . . . .	58
3.6	Usage Scenario of <b>FactWatcher</b> . . . . .	62
4.	Expert Team Finding . . . . .	64
4.1	Introduction . . . . .	64
4.2	Related Work . . . . .	70
4.3	Skyline Groups Problem . . . . .	72
4.4	Finding Skyline Groups . . . . .	75
4.4.1	Output Compression for MIN and MAX . . . . .	77
4.4.2	Input Pruning . . . . .	79
4.4.3	Search Space Pruning: Anti-Monotonicity . . . . .	80
4.5	Algorithms . . . . .	86
4.5.1	Dynamic Programming Algorithm Based on Order-Specific Property . . . . .	86
4.5.2	Iterative Algorithm Based on Weak Candidate-Generation Property . . . . .	90

4.5.3	From Distinct Vectors to Equivalent Groups . . . . .	92
4.6	Experiments . . . . .	97
4.6.1	Study of Different Aggregate Functions . . . . .	98
4.6.2	Experiments on NBA Dataset . . . . .	101
4.6.3	Experiments on Stock Dataset . . . . .	103
4.6.4	Experiments on Synthetic Datasets . . . . .	105
4.7	Prototype System: CrewScout . . . . .	109
4.8	Usage Scenario of CrewScout . . . . .	115
5.	Conclusion . . . . .	117
	REFERENCES . . . . .	118

## LIST OF ILLUSTRATIONS

Figure	Page
2.1 Distribution of Sentences . . . . .	14
2.2 Data Collection Interface . . . . .	15
2.3 Frequency Distribution of Participants' Quality . . . . .	17
2.4 Feature Importance . . . . .	21
2.5 ClaimBuster Platform . . . . .	25
2.6 Find Factual Claims in any Text . . . . .	26
2.7 A Debate Page . . . . .	27
2.8 Distributions of ClaimBuster scores over all the sentences for both parties	30
2.9 Distributions of ClaimBuster scores over all the sentences for the major candidates . . . . .	31
2.10 Distribution of verdicts for each party . . . . .	33
2.11 Distribution of verdicts for each major candidate . . . . .	34
2.12 Distribution of topics over all the sentences for each party . . . . .	35
2.13 Distribution of topics over all the sentences from the major candidates	36
2.14 Distribution of topics over sentences scored high ( $\geq 0.5$ ) by ClaimBuster	37
2.15 Distribution of topics over sentences scored low ( $\leq 0.3$ ) by ClaimBuster	38
2.16 Comparison of topic distributions of CNN, PolitiFact fact-checked sen- tences and sentences scored high ( $\geq 0.5$ ) by ClaimBuster . . . . .	39
3.1 Lattice $\mathcal{C}^{t_5}$ . . . . .	57
3.2 FactWatcher User Interface . . . . .	59
4.1 Running example in 2-d space . . . . .	74

4.2	(a) Calculate $Sky_k^n$ from $Sky_{k-1}^{n-1}$ and $Sky_k^{n-1}$ ; (b) Dynamic programming algorithm for calculating $Sky_k^n$ . . . . .	89
4.3	(a)-(b): Execution time (seconds, log scale) and (c)-(d): number of candidate groups (log scale), mixture of SUM/MAX/MIN . . . . .	100
4.4	(a)-(b): Execution time (seconds, logarithmic scale) and (c)-(d): number of candidate groups (logarithmic scale), SUM . . . . .	104
4.5	(a)-(b): Execution time (seconds, logarithmic scale) and (c)-(d): number of candidate groups (logarithmic scale), MIN . . . . .	105
4.6	(a)-(b): Execution time (seconds, logarithmic scale) and (c)-(d): number of candidate groups (logarithmic scale), MAX . . . . .	106
4.7	Number of pairwise group comparisons by different methods for MIN (a)-(b) and MAX (c)-(d) . . . . .	107
4.8	Effect of input pruning on OSM, $k = 3$ . . . . .	107
4.9	Finding all skyline groups for MAX, $n = 100$ , OSM . . . . .	108
4.10	Execution time (seconds, logarithmic scale) on stock dataset, $k = 3$ . . . . .	108
4.11	Execution time (seconds) of OSM/WCM on correlated synthetic dataset with 5 attributes, $k = 4$ . . . . .	108
4.12	Execution time (seconds, logarithmic scale) of OSM on different synthetic datasets, $k = 3$ , $n = 10$ million . . . . .	109
4.13	Execution time (seconds, logarithmic scale) of WCM on different synthetic datasets, $k = 3$ , $n = 10$ million . . . . .	110
4.14	CrewScout Interface . . . . .	111
4.15	Display Panel Showing Skyline Teams . . . . .	112
4.16	Selecting and Comparing Teams . . . . .	113

## LIST OF TABLES

Table	Page
2.1 Performance . . . . .	18
2.2 Ranking Accuracy: Past Presidential Debates . . . . .	19
2.3 Ranking Accuracy: 2015 Republican Debate . . . . .	20
2.4 Comparison with [17] . . . . .	23
2.5 Comparison with [18] . . . . .	24
2.6 Performance of a Rule-based Classifier . . . . .	24
2.7 Score differences between sentences fact-checked and those not chosen for checking . . . . .	32
3.1 A Mini-world of Basketball Gamelogs . . . . .	43
3.2 Comparing Related Work on Three Modeling Aspects . . . . .	49
3.3 Running Example . . . . .	50
3.4 A Data Table for the Running Example . . . . .	59
4.1 Experts . . . . .	65
4.2 All possible 2-expert teams . . . . .	66
4.3 Notations . . . . .	73
4.4 Running example . . . . .	74
4.5 Examples of aggregate-based comparison . . . . .	74
4.6 Counter-example for proving Theorem 3 . . . . .	85
4.7 Number of all groups (G), skyline groups (S), distinct skyline group vectors (V), under various $n$ , $k$ , and functions. Correlated synthetic dataset. M: million, B: billion . . . . .	99

4.8	Sample skyline groups from 512 players, 5 players per group . . . . .	101
4.9	Number of tuples dominated by $< k$ tuples in NBA . . . . .	103
4.10	Number of tuples dominated by less than $k$ tuples in stock dataset . . . . .	109

## CHAPTER 1

### Introduction

Computational journalism emerged recently as a young interdisciplinary field [1] that brings together experts in journalism, social sciences and computer science, and advances journalism by innovations in computational techniques. Database and data mining researchers have also started to push the frontiers of this field [2], [3], [4]. One of the important tasks in journalism field is fact-checking. The growing movement of political fact-checking plays an important role in increasing democratic accountability and improving political discourse [5, 6]. Politicians and media figures make claims about “facts” all the time, but the new army of fact-checkers can often expose claims that are false, exaggerated or half-truths. The number of active fact-checking websites has grown from 44 a year ago to 64 this year, according the Duke Reporters’s Lab <sup>1</sup>. In this dissertation, we study the process of fact-checking and investigate opportunities for automating it. We discuss the technical challenges we will face in automating fact-checking and propose potential solutions. Specifically, we study the problem of finding important checkworthy factual claims from political discourses. We model this problem as a multi-class classification task and we follow a supervised learning approach to tackle it. We constructed a labeled dataset of spoken sentences by presidential candidates during 1960–2012 U.S. presidential debates. Each sentence is given one of three possible labels—it is not a factual claim; it is an unimportant factual claim; it is an important factual claim. We trained and tested several multi-class classification models using the labeled dataset. Experiment results

---

<sup>1</sup><http://reporterslab.org/snapshot-of-fact-checking-around-the-world-july-2015/>



demonstrated promising accuracy of the models. We also built a fact-finding platform named ClaimBuster. It allows to find important factual claims from any document. The platform has live-covered some of the latest primary debates of 2016 U.S. presidential elections and archived all of them. It also has a social component which keeps monitoring Twitter for checkworthy factual tweets.

Another task where journalist and reporters spend a good amount of time is bringing out attention-seizing factual statements backed by data, which may lead to news stories and investigation. While such statements take many different forms, we consider some common forms. In this dissertation, we study how to find situational facts pertinent to the new tuples in an ever-growing database, where the tuples capture real-world events. Specifically, we study the novel problem of finding situational facts and formalize it as discovering constraint-measure pairs that qualify a tuple as a contextual skyline tuple. We devise efficient algorithms based on three main ideas tuple reduction, constraint pruning and sharing computation across measure subspaces. We use a simple prominence measure for ranking situational facts and discovering prominent situational facts. We conduct extensive experiments on two real datasets (NBA dataset and weather dataset) to investigate their prominent situational facts and to study the efficiency of various proposed algorithms and their tradeoffs.

Another interesting problem we study in this dissertation is finding one of the best teams of experts to complete a task. In day-to-day journalism, oftentimes it becomes necessary to form a team of experts to investigate/report an event. We approach this team formation problem with a novel technique called *Skyline Groups*. Specifically, we formulate and investigate the novel problem of finding the *skyline  $k$ -tuple groups* from an  $n$ -tuple dataset—i.e., groups of  $k$  tuples which are not dominated by any other group of equal size, based on aggregate-based group dominance

relationship. The major technical challenge is to identify effective anti-monotonic properties for pruning the search space of skyline groups. To this end, we first show that the anti-monotonic property in the well-known *Apriori* algorithm does not hold for skyline group pruning. Then, we identify two anti-monotonic properties with varying degrees of applicability: *order-specific property* which applies to SUM, MIN, and MAX as well as *weak candidate-generation property* which applies to MIN and MAX only. Experimental results on both real and synthetic datasets verify that the proposed algorithms achieve orders of magnitude performance gain over the baseline method.

## CHAPTER 2

### Finding Check-worthy Factual Statements

Our purpose in this chapter is twofold: To argue for the advancement of research on automated fact-checking and to report our progress toward that end with a tool called ClaimBuster. We describe the current state-of-the-art of fact checking research and describe the approach we have taken with ClaimBuster. We report the preliminary results of a field test comparing ClaimBuster’s ability to identify check-worthy factual claims made during the 2016 U.S. presidential election primary debates with those of professional journalists and fact-checking organizations. Significant overlaps between the machine scoring and human judgments were observed. Using the ClaimBuster model, we developed a platform which was used to live-cover some of the latest 2016 Republican and Democrat primary debates. We describe various components of the platform and explain how each of them works.

#### 2.1 Introduction

Technology, social media and new forms of journalism have made it easier than ever to disseminate falsehoods and half-truths faster than the fact-checkers can expose them. The challenge is that the human fact-checkers frequently have difficulty keeping up with the rapid spread of misinformation. Computation may hold the key to far more effective and efficient fact-checking, as Cohen et al. [7, 8] and Diakopolous<sup>1</sup> have pointed out. Over and over again, computing has reshaped journalism. Tasks that required huge amounts of manual labor such as analyzing data and finding

---

<sup>1</sup><http://towknight.org/research/thinking/scaling-fact-checking/>

patterns and relationships are now accomplished with ease. There is little doubt that computers can substantially aid fact-checking too.

The eternal quest, the “Holy Grail”, is a completely automatic fact-checking platform that can detect a claim as it appears in real time, and instantly provide the voter with a rating about its accuracy. It makes its calls by consulting databases of already checked claims, and by conducting novel analysis of relevant data from reputable sources.

In this chapter, we advocate the pursuit of the “Holy Grail” and discuss the technical challenges we will face in automating fact-checking and potential solutions. The “Holy Grail” may remain far beyond our reach for many, many years to come.

But in pursuing this ambitious goal, we can help fact-checking and improve the political discourse. One such advancement is our own progress on ClaimBuster, a tool that helps journalists find political claims to fact-check. We have used it on the presidential debates of U.S. Election 2016. During a debate, for every sentence spoken by the candidates and extracted into transcripts, ClaimBuster immediately determines if the sentence has a factual claim and whether its truthfulness is important to the public.

## 2.2 Limitations Of Current Practices Of Fact-Checking

Fact-checking is difficult and time-consuming for journalists, which creates a significant gap between the moment a politician makes a statement and when the fact-check is ultimately published.

The growth of fact-checking has been hampered by the nature of the work. It is time-consuming to find claims to check. Journalists have to spend hours going through transcripts of speeches, debates and interviews to identify claims they will research.

Also, fact-checking requires advanced research techniques. While ordinary journalism can rely on simple “on-the-one-hand, on-the-other-hand” quotations, a fact-check requires more thorough research so the journalist can determine the accuracy of a claim.

Fact-checking also requires advanced writing skills that go beyond “just the facts” to persuade the reader whether the statement was true, false or somewhere in between. Fact-checking is a new form that has been called “reported conclusion” journalism.

Those factors mean that fact-checking often takes longer to produce than traditional journalism, which puts a strain on staffing and reduces the number of claims that can be checked. It also creates a time gap between the moment the statement was made and when the fact-check is ultimately published. That can take as little as 15 to 30 minutes for the most simple fact-check to a full day for a more typical one. A complicated fact-check can take two or more days. (By contrast, Leskovec, Backstrom and Kleinberg [9] found a meme typically moves from the news media to blogs in just 2.5 hours.)

For voters, that means a long gap between the politician’s claim and a determination whether it was true. The voters don’t get the information when they really need it. They must wait and look up on a fact-checking site to find out if the claim was accurate. This is one of several factors that emboldens politicians to keep repeating claims even when they are false.

Another limitation is the outdated nature of the fact-checkers’ publishing platforms. Many fact-checking sites still use older content management systems built for newspapers and blogs that are not designed in a modern style for structured journalism. This limits how well they can be used in computational projects.

## 2.3 The “HOLY GRAIL”

We should not be surprised if we can get very close but never reach the “Holy Grail”. A fully automated fact-checker calls for fundamental breakthroughs in multiple fronts and, eventually, it represents a form of Artificial Intelligence (AI). As remote and intangible as AI may have appeared initially, though, in merely 60 years scientists have made leaps and bounds that profoundly changed our world forever. The quest for the “Holy Grail” of fact-checking will likewise drive us to constantly improve this important journalistic activity.

The Turing test [10] was proposed by Alan Turing as a way of gauging a machine’s ability to exhibit artificial intelligence. Although heavily criticized, the concept has served well in helping advance the field. Similarly, we need explicit and tangible measures for assessing the ultimate success of a fact-checking machine. The “Holy Grail” is a computer-based fact-checking system bearing the following characteristics:

**Fully automated:** It checks facts without human intervention. It takes as input the video/audio signals and texts of a political discourse and returns factual claims and a truthness rating for each claim (e.g., the Truth-O-Meter by PolitiFact).

**Instant:** It immediately reaches conclusions and returns results after claims are made, without noticeable delays.

**Accurate:** It is equally or more accurate than any human fact-checker.

**Accountable:** It self-documents its data sources and analysis, and makes the process of each fact-check transparent. This process can then be independently verified, critiqued, improved, and even extended to other situations.

Such a system mandates many complex steps—extracting natural language sentences from textual/audio sources; separating factual claims from opinions, beliefs, hyperboles, questions, and so on; detecting topics of factual claims and discerning

which are the “check-worthy” claims; assessing the veracity of such claims, which itself requires collecting information and data, analyzing claims, matching claims with evidence, and presenting conclusions and explanations. Each step is full of challenges. We now discuss in more detail these challenges and potential solutions.

### 2.3.1 Computational Challenges

On the computational side, there are mainly two fundamental challenges. One is to understand what one says. Computer scientists have made leaps and bounds in speech recognition and Natural Language Processing (NLP). But these technologies are far from perfect. The other challenge lies in our capability of collecting sufficient evidence for checking facts. We are in the big-data era. A huge amount of useful data is accessible to us and more is being made available at every second. Semantic web, knowledge base, database and data mining technologies help us link together such data, reason about the data, efficiently process the data and discover patterns. But, what is being recorded is still tiny compared to the vast amount of information the universe holds. Below we list some of the more important computational hurdles to solve.

#### **Finding claims to check**

—Going from raw audio/video signals to natural language. Extracting contextual information such as speaker, time, and occasion.

—Defining “checkable” and “check-worthy” of claims. Is the claim factual (falsifiable) or is it opinion? Should or can we check opinions? How “interesting” is the claim? How do we balance “what the public should know” and “what the public wants to consume”? Can these judgements be made computationally?

—Extracting claims from natural language. What to do when a claim spans multiple sentences? What are the relevant features useful for determining whether a claim is

“checkable” or “check-worthy”?

### **Getting data to check claims**

—We should consider at least two types of data sources: 1) claims already checked by various organizations; 2) unstructured, semi-structured and structured data sources that provide raw data useful for checking claims, e.g., voting records, government budget, historical stock data, crime records, weather records, sports statistics, and Wikipedia.

—Evaluating quality and completeness of sources.

—Matching claims with data sources. This requires structure/metadata in the database of already checked claims, as well as data sources.

—Synthesizing and corroborating multiple sources.

—Cleansing data. Given a goal (e.g., to verify a particular claim), help journalists decide which data sources—or even which data items in a database—are worthy investigating as high priority.

### **Checking claims**

—How to remove (sometimes intentional) vagueness, how to spot cherry-picking of data (beyond correctness), how to evaluate and how to come up with convincing counterarguments using data [11, 12, 13].

—The methods in [11, 12, 13] rely on being able to cast a claim as a mathematical function that can be evaluated over structured data. Who translate a claim into this function? Can the translation process be automated?

—Fact verification may need human participation (e.g., social media as social sensors) or even crowdsourcing (e.g., checking whether a bridge really just collapsed). Can a computer system help coordinate and plan human participation on an unprecedented level? How to remove bias and do quality control of human inputs? Should such a system be even considered fully automated?



## Monitoring and anticipating claims

—Given evolving data, we can monitor when a claim turns false/true [14, 13]. Can we anticipate what claims may be made soon? That way, we can plan ahead and be proactive.

—Challenges in scalable monitoring and parallel detection of a massive number of claim types/templates.

### 2.3.2 Journalistic Challenges

A major barrier to automation is the lack of structured journalism in fact-checking. Although there's been tremendous growth in the past few years – 20 new sites around the world just in the last year, according to the Duke Reporters' Lab – the vast majority of the world's fact-checkers are still relying on old-style blog platforms to publish their articles. That limits the articles to a traditional headline and text rather than a newer structured journalism approach that would include fields such as statement, speaker and location that would allow for real-time matching. There are no standards for data fields or formatting. The articles are just published as plain text.

There also is no single repository where fact-checks from various news organizations are catalogued. They are kept in the individual archives of many different publications, another factor that makes real-time matching difficult. Another journalistic barrier is the inconsistency of transparency. Some fact-checkers distill their work to very short summaries, while others publish lengthy articles with many quotations and citations.<sup>2</sup> The lack of structure, the absence of a repository and the inconsistency in publishing provides a lack of uniformity for search engines, which

---

<sup>2</sup><http://reporterslab.org/study-explores-new-questions-about-quality-of-global-fact-checking/>

do not distinguish fact-checks from other types of editorial content in their search results.

Another challenge is the length of time it takes to publish more difficult fact-checks and to check multiple claims from the same event. PolitiFact, for example, boasted that it published 20 separate checks from the Aug. 6 Republican presidential debate. But it took six days for it to complete all of those checks.<sup>3</sup>

## 2.4 Related Work

To the best of our knowledge, no prior study has focused on computational methods for detecting factual claims and discerning their importance. The most relevant line of work is subjectivity analysis of text (e.g., [12, 1, 10]) which classifies sentences into objective and subjective ones. However, not all objective sentences are check-worthy important factual claims. Part of the goals of computational journalism [3, 4] is use computing to automate fact checking [11, 9]. Wu et al. [11] studied how to model the quality of facts and find their supporting arguments and counterarguments. Vlachos and Riedel [9] analyzed the tasks in fact-checking and presented a dataset of factual claims collected from PolitiFact.com and Channel4.com. Another area of related research is checking information credibility in micro-blog platforms. For instance, [13] finds trending rumors containing disputed factual claims. [2, 6] focus on assigning credibility scores to tweets. The scoring models are highly dependent on Twitter specific features such as the credibility of twitter users. A tweet with high credibility does not necessarily contain a check-worthy factual claims.

---

<sup>3</sup><http://www.politifact.com/truth-o-meter/article/2015/aug/12/20-fact-checks-republican-debate/>

## 2.5 ClaimBuster

ClaimBuster is a tool that helps journalists find claims to fact-check. Figure 2.6 is the screenshot of the current version of ClaimBuster. For every sentence spoken by the participants of a presidential debate, ClaimBuster determines whether the sentence has a factual claim and whether its truthfulness is important to the public. As shown in Figure 2.6, to the left of each sentence there is a score ranging from 0 (least likely an important factual claim) to 1 (most likely). The calculation is based on machine learning models built from thousands of sentences from past debates labeled by humans. The ranking scores help journalists prioritize their efforts in assessing the veracity of claims. ClaimBuster will free journalists from the time-consuming task of finding check-worthy claims, leaving them with more time for reporting and writing. Ultimately, ClaimBuster can be expanded to other discourses (such as interviews and speeches) and also adapted for use with social media. Note that ClaimBuster does not yet determine if a factual claim is true or false. This will be a direction of the future work further toward the “Holy Grail”.

### 2.5.1 Classification and Ranking

We model ClaimBuster as a classifier and ranker and we take a supervised learning approach to construct it. We categorize sentences in presidential debates into three categories:

**Non-Factual Sentence (NFS):** Subjective sentences (opinions, beliefs, declarations) and many questions fall under this category. These sentences do not contain any factual claim. Below are some examples.

- *But I think it's time to talk about the future.*
- *You remember the last time you said that?*

**Unimportant Factual Sentence (UFS):** These are factual claims but not check-

worthy. The general public will not be interested in knowing whether these sentences are true or false. Fact-checkers do not find these sentences as important for checking. Some examples are as follows.

- *Next Tuesday is Election Day.*
- *Two days ago we ate lunch at a restaurant.*

**Check-worthy Factual Sentence (CFS):** They contain factual claims and the general public will be interested in knowing whether the claims are true. Journalists look for these type of claims for fact-checking. Some examples are:

- *He voted against the first Gulf War.*
- *Over a million and a quarter Americans are HIV-positive.*

Given a sentence, the objective of ClaimBuster is to derive a score that reflects the degree by which the sentence belongs to CFS. Many widely-used classification methods support ranking naturally. For instance, consider a Support Vector Machine (SVM). We treat CFSs as positive examples and both NFSs and UFSs as negative examples. SVM finds a decision boundary between the two types of training examples. Following Platt’s scaling technique [15], for a given sentence  $x$  to be classified, we calculate the posterior probability  $P(class = CFS|x)$  using the SVM’s decision function. The probability scores of all sentences are used to rank them.

## 2.5.2 Data Collection

This section describes the U.S. presidential debate dataset and our data collection procedure.

## 2.5.3 Dataset

The custom of organizing debates between U.S. presidential candidates before a general election started in 1960. There have been a total of 14 presidential elections

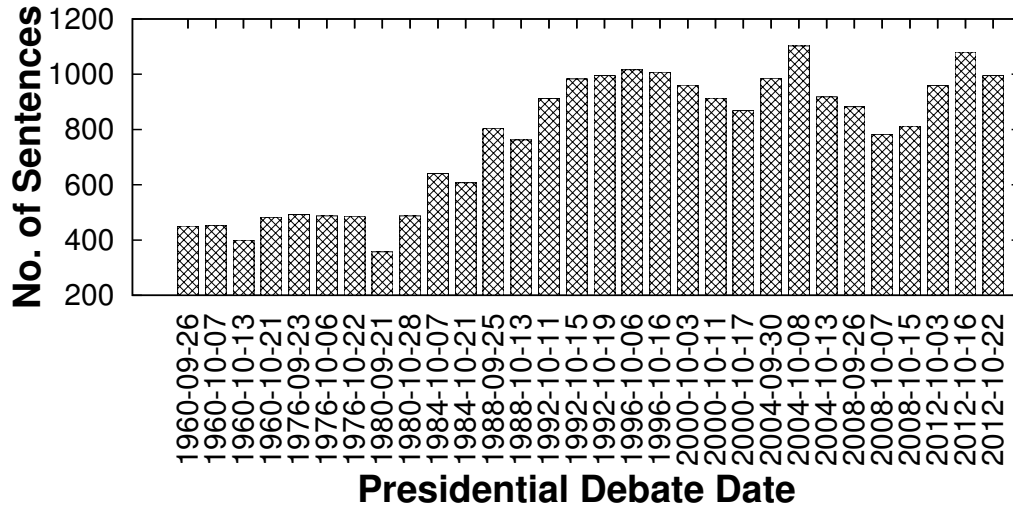


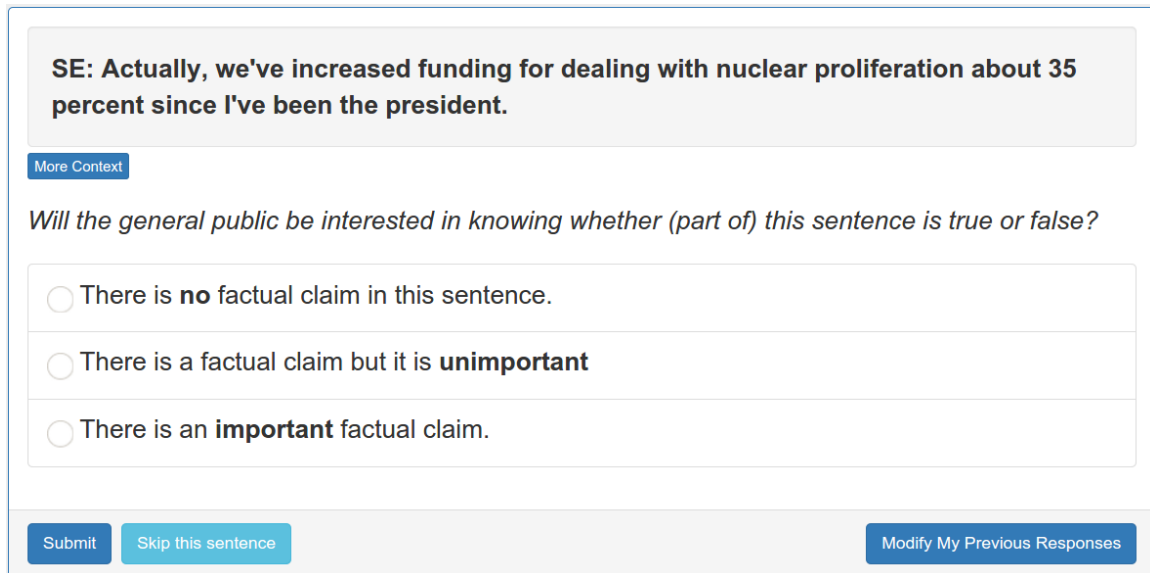
Figure 2.1. Distribution of Sentences

from 1960 to 2012. Except 1964, 1968 and 1972 there have been debates before all the 11 remaining election years. Number of debates before an election varies from year to year; for example, there were two and three debates before 1988 and 2012 elections, respectively. In total, there are 30 debates in these 11 election years. We have collected the transcripts of these debates and used in experiments.

There are 28029 sentences in the transcripts. Using parsing rules and human annotation, we determined the speaker identity of each sentence. 23075 sentences are spoken by presidential candidates and 4815 are by the debate moderators. For our experiments, we concentrated on the 20788 sentences spoken by the candidates which have at least 5 words. Figure 2.1 shows the distribution of sentences among these 30 debates.

#### 2.5.4 Ground-truth Collection

We developed a rich and controlled data collection platform to collect ground-truth of the sentences <sup>4</sup>. We have invited university students, professors, journalists and reporters who are aware of U.S. politics to contribute in the data collection process. The process has been running for 15 months in multiple phases and it is still going on. Figure 2.2 shows the interface of the data collection platform. A participant is presented one sentence at a time from the pool of all sentences. S/he can assign one of three [NFS, UFS, CFS] possible labels for the sentence. If the participant is not confident to assign a label for the sentence, it is possible to skip and move forward. To maintain high quality and avoid spammers or low quality participants, we undertook some control mechanisms.



**SE: Actually, we've increased funding for dealing with nuclear proliferation about 35 percent since I've been the president.**

[More Context](#)

*Will the general public be interested in knowing whether (part of) this sentence is true or false?*

There is **no** factual claim in this sentence.

There is a factual claim but it is **unimportant**

There is an **important** factual claim.

[Submit](#) [Skip this sentence](#) [Modify My Previous Responses](#)

Figure 2.2. Data Collection Interface

---

<sup>4</sup>[http://idir-server2.uta.edu/classifyfact\\_survey/](http://idir-server2.uta.edu/classifyfact_survey/)

#### 2.5.4.1 Context

With just the text of the sentence under consideration, it is sometimes hard for a participant to assign a label. To resolve this issue, we provided a button to show the context of the sentence. When the button is clicked, the system shows four preceding sentences of the sentence in the question.

#### 2.5.4.2 Screening Sentences

To detect spammers and low-quality participants, we used 1032 (731 NFS, 63 UFS, 238 CFS) screening sentences, picked from all the sentences. Three experts agreed upon the labels of these sentences. On average, one out of every ten sentences given to a participant (without letting the participant know) was randomly chosen to be a screening sentence selected from the pool. A random number  $X$ , where  $1 \leq X \leq 3$ , decides the type of the screening question. If  $X = 1$ , the type is NFS, if  $X = 2$ , the type is UFS, if  $X = 3$ , the type is CFS. Once the type is selected, the screening question is randomly picked from the pool of screening questions of that particular type. The participants were ranked by the degree of agreement on screening sentences between them and the three experts.

#### 2.5.4.3 Quality Measure

We defined quality measures to order the participants according to their job quality. These measures help us to identify low-quality participants. We observed that not all mislabeling has equal significance. For example, labeling an NFS sentence as a CFS is a more critical mistake than labeling a UFS as a CFS. We defined weights for different types of mistakes and incorporated them into the quality measure. Formally,

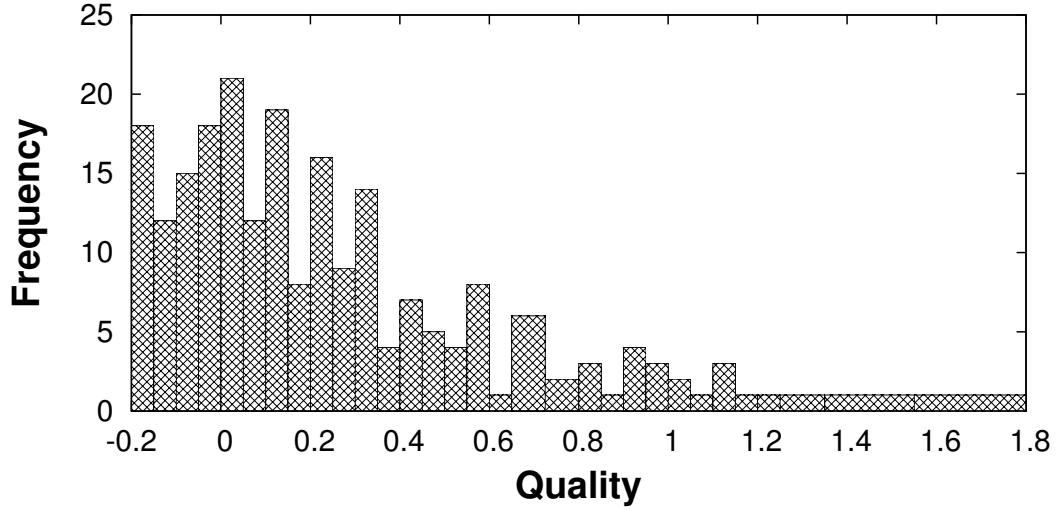


Figure 2.3. Frequency Distribution of Participants' Quality

if  $X_Y$  is the number of times a participant  $p$  labels an  $X$  sentence as  $Y$ , then the quality of  $p$ ,  $Q(p)$ , is-

$$Q(p) = (-0.2 * (NFS\_NFS + UFS\_UFS + CFS\_CFS) + 0.7 * (NFS\_UFS + UFS\_CFS + CFS\_UFS + UFS\_NFS) + 2.5 * (NFS\_CFS + CFS\_NFS)) / \text{total number of screening sentences labeled by } p$$

The weights are defined empirically. The maximum possible quality is  $-0.2$  and the minimum possible quality is  $2.5$ . Figure 2.3 shows the frequency distribution of  $Q(p)$  for all participants. If  $Q(p) \leq 0$  for a participant  $p$ , we define  $p$  as a top-quality participant. A total of 329 participants contributed in the data collection process. Among them, 52 are top-quality participants.

#### 2.5.4.4 Reward Program

We devised a reward program to encourage participants for performing a high-quality job. For a participant  $p$ , his/her reward is dependent on  $Q(p)$ , number of



	Precision	Recall	F-measure
NFS	0.90	0.96	0.93
UFS	0.65	0.26	0.37
CFS	0.79	0.74	0.77

Table 2.1. Performance

sentences labeled, average length of sentences labeled and number of sentences skipped by  $p$ . In general, it is possible to earn up to 10 cents for each sentence as the reward. In addition to this reward, we also offered about \$1000 as first, second, third and honorary prizes.

#### 2.5.4.5 Leaderboard & Tips

To provide participants a sense of competition, our data-collection website features a leaderboard of participants. It allows a participant to see his/her rank among other participants. This is a mechanism to encourage serious participants to perform better and discourage spammers from further participation. Along with the leaderboard, the website provided helpful tips and messages from time to time to keep the participants motivated.

We collected responses from two participants for each sentence. For training and evaluating our classification models, we only used a sentence if its label was agreed upon by two top-quality participants. Thereby we got 8015 sentences (5860 NFSs, 482 UFSs, 1673 CFSs).

k	P@k	AvgP	nDCG
10	1.000	1.000	1.000
25	1.000	1.000	1.000
50	0.980	0.995	0.987
100	0.943	0.979	0.956
200	0.916	0.955	0.931
300	0.848	0.937	0.874
400	0.764	0.915	0.801
500	0.679	0.897	0.827

Table 2.2. Ranking Accuracy: Past Presidential Debates

### 2.5.5 Feature Extraction

We extracted multiple categories of features from the sentences. We use the following sentence to explain the features.

*When President Bush came into office, we had a budget surplus and the national debt was a little over five trillion.*

**Sentiment:** We used AlchemyAPI to calculate a sentiment score for each sentence. The score ranges from -1 (most negative sentiment) to 1 (most positive sentiment). The above sentence has a sentiment score -0.846376.

**Length:** This is the word count of a sentence. Natural language toolkit NLTK was used for tokenizing a sentence into words. The example sentence has length 21.

**Word:** We used words in sentences to build tf-idf features. After discarding rare words that appear in less than three sentences, we got 6130 words. We did not apply

k	P@k	AvgP	nDCG
10	0.400	0.642	0.441
20	0.450	0.520	0.456
30	0.367	0.500	0.401
40	0.325	0.477	0.368
50	0.300	0.456	0.346
60	0.300	0.431	0.356
70	0.300	0.411	0.390
80	0.275	0.406	0.401
90	0.267	0.394	0.422
100	0.270	0.381	0.452

Table 2.3. Ranking Accuracy: 2015 Republican Debate

stemming or stopword removal.

**Part-of-Speech (POS) Tag:** We applied NLTK POS tagger on all sentences. There are 43 POS tags in the corpus. We constructed a feature for each tag. For a sentence, the count of words belonging to a POS tag is the value of the corresponding feature. In the example sentence, there are 3 words (came, had, was) with POS tag VBD (Verb,Past Tense) and 2 words (five, trillion) with POS tag CD (Cardinal Number).

**Entity Type:** We used AlchemyAPI to extract entities from sentences. There are 2727 entities in the labeled sentences. They belong to 26 types. The above sentence has an entity “Bush” of type “Person”. We constructed a feature for each entity type. For a sentence, its number of entities of a particular type is the value of the corresponding feature.

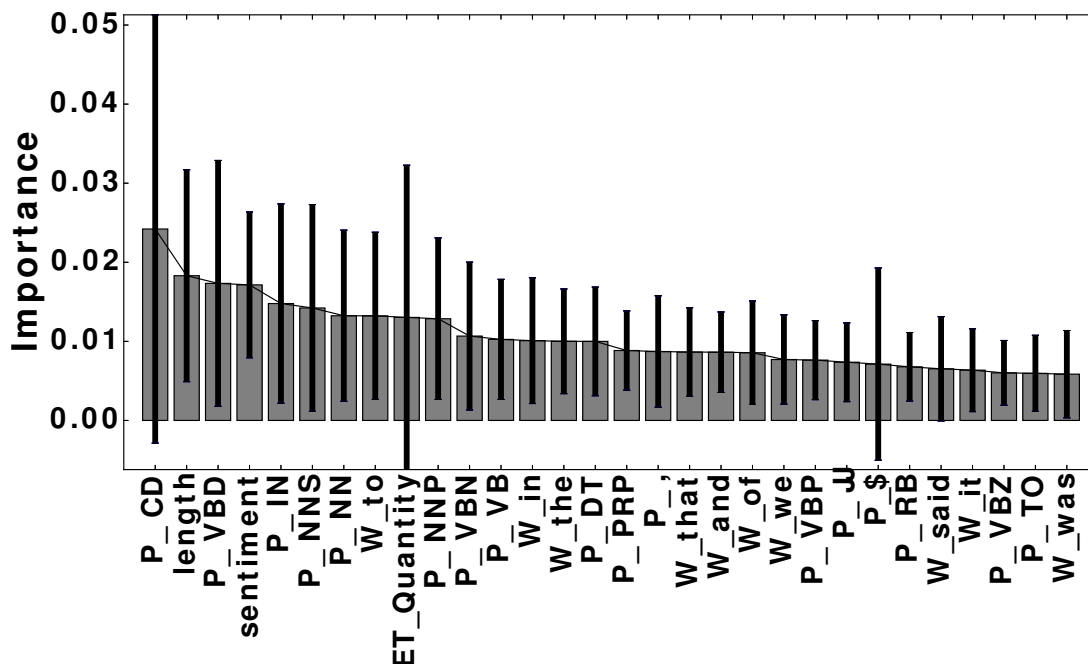


Figure 2.4. Feature Importance

**Feature Selection:** There are 6201 features in total. To avoid over-fitting and attain a simpler model, we performed feature selection. We trained a random forest classifier for which we used GINI index to measure the importance of features in constructing each decision tree. The overall importance of a feature is its average importance over all the trees. Figure 2.4 shows the importance of the 30 best features in the forest. The black solid lines indicate the standard deviations of importance values. Category types are prefixes to feature names. We observed that unsurprisingly POS tag CD (Cardinal Number) is the best feature—check-worthy factual claims are more likely to contain numeric values (45% of CFSs in our dataset) and non-factual sentences are less likely to contain numeric values (6% of NFSs in our dataset).

### 2.5.6 Evaluation

We performed 3-class (NFS/UFS/CFS) classification using several supervised learning methods, including Multinomial Naive Bayes Classifier (NBC), Support Vector Machine (SVM) and Random Forest Classifier (RFC). These methods were evaluated by 4-fold cross-validation. SVM had the best accuracy in general. We experimented with various combinations of the extracted features. Table 2.1 shows the performance of SVM using words and POS tag features. On the CFS class, ClaimBuster achieved 79% precision (i.e., it is accurate 79% of the time when it declares a CFS sentence) and 74% recall (i.e., 74% of true CFSs are classified as CFSs). The classification models had better accuracy on NFS and CFS than UFS. This is not surprising, since UFS is between the other two classes and thus the most ambiguous. More detailed results and analyses based on data collected by an earlier date can be found in [16].

We used SVM to rank all 8015 sentences (cf. Section 2.5.2) by the method in Section 2.5.1. We measured the accuracy of the top-k sentences by several commonly-used measures, including Precision-at-k ( $P@k$ ), AvgP (Average Precision), nDCG (Normalized Discounted Cumulative Gain). Table 2.2 shows these measure values for various k values. In general, ClaimBuster achieved excellent performance in ranking. For instance, for top 100 sentences, its precision is 0.96. This indicates ClaimBuster has a strong agreement with high-quality human coders on the check-worthiness of sentences.

#### 2.5.6.1 Comparison with Subjectivity Classifiers

We also compared the performance of ClaimBuster with state-of-the-art *subjectivity* classifiers [17, 18]. Our hypothesis was that an *subjectivity* classifier can be used to separate NFS from UFS and CFS. However, experiment results showed that

	NFS	UFS	CFS
subjective	157	5	44
objective	574	58	194

Table 2.4. Comparison with [17]

the *subjectivity* classifiers failed to filter NFS. We used the *OpinionFinder*<sup>5</sup> package for classification. This tool provides two *subjectivity* classifiers [17, 18].

The first classifier [17] tags each sentence as either subjective or objective based on a model trained on the *MPQA Corpus*<sup>6</sup>. Based on a 10 fold-cross validation on 11168 sentences extracted from the MPQA Opinion Corpus, this classifier has an accuracy of 76%, subjective precision of 79%, subjective recall of 76%, and subjective F-measure of 77.5%.

The second classifier [18] is a rule-based classifier. It optimizes precision at the expense of recall. That is, it classifies a sentence as subjective or objective only if it can do so with confidence. Otherwise, it labels the sentence as “unknown”. This rule-based classifier is reported to have about 91.7% subjective precision and 30.9% subjective recall. Objective precision is 83.0% and objective recall is 32.8%.

Table 2.4 shows the comparison between [17] and ClaimBuster. We used the 1032 screening sentences for this experiment. 574 NFS sentences were labeled as objective sentence and 44 CFS sentences were labeled as subjective sentence. This invalidates our hypothesis that a *subjectivity* classifier may be used to filter NFS sentences

<sup>5</sup><http://mpqa.cs.pitt.edu/opinionfinder/>

<sup>6</sup><http://mpqa.cs.pitt.edu/corpora/>

	NFS	UFS	CFS
subjective	21	0	4
unknown	175	5	45
objective	535	58	189

Table 2.5. Comparison with [18]

	NFS	UFS	CFS
NFS prediction	5803	0	93
UFS prediction	470	0	16
CFS prediction	1508	0	341

Table 2.6. Performance of a Rule-based Classifier

from UFS and CFS. Table 2.5 also shows similar comparison between ClaimBuster and [18].

### 2.5.6.2 Comparison with Rule-based Classifiers

We used a simple rule-based classifier as a baseline and compared it with an SVM-based model. Table 2.6 shows performance of the baseline. The following regular expression were used as the rules.

1.  $\cdot * (\textit{percent}|\textit{times}|\%) . * \textit{of} . * \$$
2.  $(?! . * \textit{think} | . * \textit{believe} ) . * (\textit{more}|\textit{less}|\textit{lower}|\textit{higher}|\textit{bigger}|\textit{smaller}|\textit{increasing}|\textit{decreasing} ) . * \textit{than} . * \$$



1

Find factual claims in your own text

	<p>2016 Democratic Party Presidential Debate Live. April 14, 2016, 9 p.m. EST <span style="float: right;">3</span></p> <p>Participants: Hillary Clinton, Bernie Sanders Moderators: Wolf Blitzer, Dana Bash, Errol Louis</p>
	<p>2016 Democratic Party Presidential Debate. April 14, 2016, 9 p.m. EST</p> <p>Participants: Bernard Sanders, Hillary Clinton Moderators: Wolf Blitzer, Dana Bash, Errol Louis</p>
	<p>2016 Republican Party Presidential Debate. March 10, 2016, 8:30 p.m. EST</p> <p>Participants: John Kasich, Marco Rubio, Ted Cruz, Donald Trump Moderators: Jake Tapper, Stephen Dinan, Dana Bash, Hugh Hewitt</p>
	<p>2016 Democratic Party Presidential Debate. March 9, 2016, 9 p.m. EST</p> <p>Participants: Hillary Clinton, Bernard Sanders Moderators: Maria Elena Salinas, Jorge Ramos, Karen Tumulty</p>
	<p>2016 Democratic Party Presidential Debate. March 6, 2016, 8 p.m. EST</p> <p>Participants: Bernard Sanders, Hillary Clinton Moderators: Anderson Cooper, Don Lemon</p>
	<p>2016 Republican Party Presidential Debate. March 3, 2016, 9 p.m. EST</p> <p>Participants: Donald Trump, Marco Rubio, Ted Cruz, John Kasich Moderators: Bret Baier, Chris Wallace, Megyn Kelly</p>

Tweets by @ClaimBusterTM 2

ClaimBuster Retweeted  
  
**Rebecca Ballhaus** @rebeccaballhaus  
 Trump's joint RNC fund raised \$25 million in Q2. It transferred just \$2.2 million to his campaign: [wsj.com/articles/trump...](http://wsj.com/articles/trump...)

**Trump Camp Saw \$2.2 Million Fro...**  
 Donald Trump's joint fund with the R...  
[wsj.com](http://wsj.com)

ClaimBuster Retweeted  
  
**Chad Pergram** @ChadPergram  
 Fed deficit for this fiscal yr hits \$600b. Increase of \$162b over last yr. Annual deficits had improved in recent yrs. 1 yr was \$1.4t alone

Figure 2.5. ClaimBuster Platform

3. (?!. \* think|. \* believe). \* (increas|decreas). \* \$
4. · \* [0 - 9] + (to|or|and)[0 - 9] + . \* \$
5. · \* there. \* (is|was|are|has|had). \* (never|not). \* \$
6. · \* (support|oppose). \* (abortion|tax|civil|wage|gun|health|security|energy). \* \$

This baseline approach performed significantly worse than an SVM-based model. For example, 1508 CFS sentences were labeled as NFS by this baseline technique. The results validate the hypothesis that it is not simple to produce a reasonably accurate rule-based classifier and a bag-of-words model with POS tags outperforms a rule-based classifier.



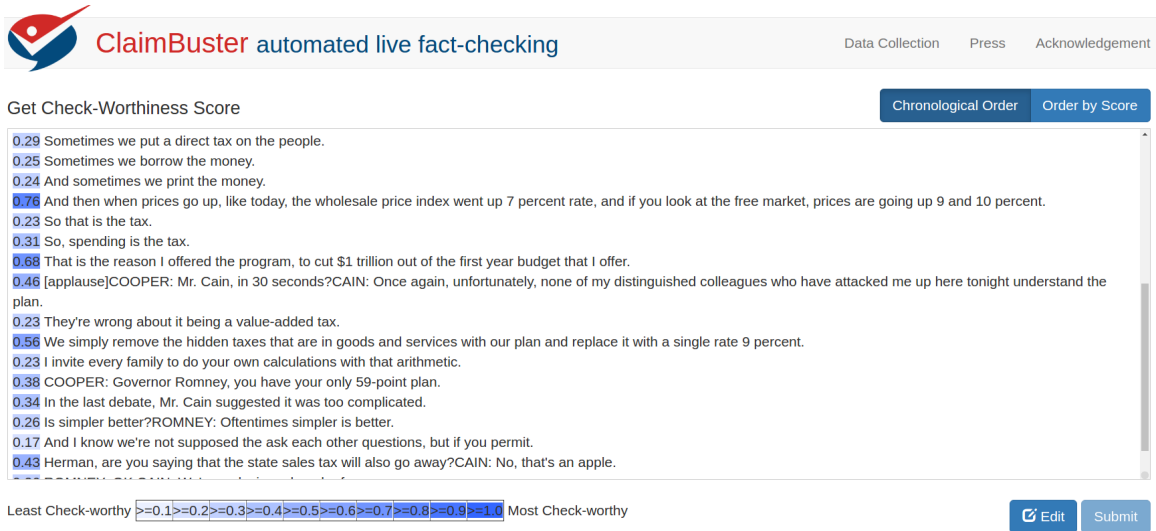


Figure 2.6. Find Factual Claims in any Text

### 2.5.7 ClaimBuster Platform

We have developed a platform <sup>7</sup> for fact-checking enthusiasts. It allows to find check-worthy factual claims from any political discourse. Major components of the platform are- text mining, social media analysis and collaborative fact-checking. The platform has covered all the Democratic and Republican primary debates of the 2016 U.S. presidential election. It has received substantial media attention from multiple news outlets, including Austin American Statesman, Poynter, PolitiFact and New Scientist, to name a few <sup>8</sup>. Figure 2.5 shows the landing page of the ClaimBuster platform. Below, each part of the platform is explained-

1. Find factual claims in any text: Clicking this button takes the user to a new page as shown in Figure 2.6. A user can write/paste any text in the textbox, press the *Submit* button and the system will show check-worthiness score for each sentence of the text. The *Edit* button allows the user to edit the text.

<sup>7</sup><http://idir.uta.edu/claimbuster>

<sup>8</sup><http://idir.uta.edu/claimbuster/press>

2016 Republican Party Presidential Debate. March 10, 2016, 8:30 p.m. EST

Venue: University of Miami, Miami, Florida. Broadcasted by: CNN/Salem Radio/Washington Times.

Moderators: Jake Tapper, Stephen Dinan, Dana Bash, Hugh Hewitt

Participants: John Kasich, Marco Rubio, Ted Cruz, Donald Trump

Transcript Source: <https://www.washingtonpost.com/news/the-fix/wp/2016/03/10/the-cnn-miami-republican-debate-transcript-annotated/>



Chronological Order Order by Score

Least Check-worthy >=0.1 >=0.2 >=0.3 >=0.4 >=0.5 >=0.6 >=0.7 >=0.8 >=0.9 >=1.0 Most Check-worthy

take -- every time this maniac from North Korea does anything, we immediately send our ships. We get virtually nothing. We have 25,000 soldiers on the line, on the border between North and South Korea. We have so many places. Saudi Arabia was making a billion dollars a day, and we were getting virtually nothing to protect them. We are going to be in a different world. We're going to negotiate real deals now, and we're going to bring the wealth back to our country. We owe \$19 trillion! We're going to bring wealth back to our country.

Dana Bash: Senator Rubio, will that be enough to save Social Security?

Marco Rubio: No. And I -- and I think you've outlined why. The numbers don't add up. You know, when I ran for the Senate in 2010, I came out and said we're going to have to make changes to Social Security, and everyone said that's the end of your campaign. In Florida, you can't talk about that, but people know that it's the truth here in Florida. Fraud is not enough. Certainly, let's wipe out the fraud, but as you said, it won't add up. You already gave those numbers. The second point is on foreign aid. I hear that all the time as well. I'm against any sort of wasting of money on foreign aid, but it's less than 1 percent of our federal budget. The numbers don't add up. The

Note: You can select the text and annotate it. The underlined sentences are annotated.

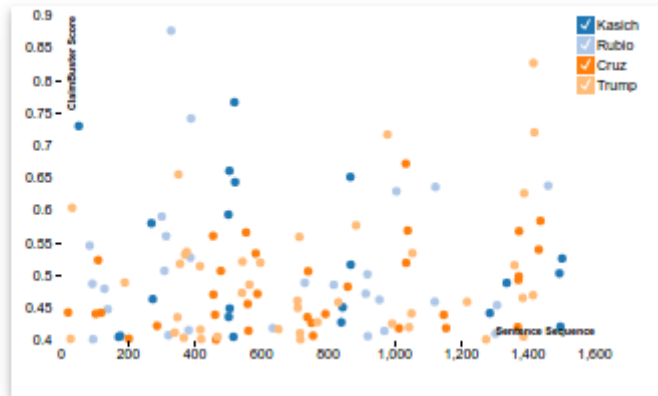


Figure 2.7. A Debate Page

*Order by Score* allows the user to see the highly check-worthy sentences in the text.

2. Twitter factual claim monitor: This part shows the automatically detected check-worthy factual claims related to U.S. politics in Twitter. A Twitter account <sup>9</sup> is used to retweet the detected tweets. The underlying system first collects tweets from a list of U.S. political Twitter accounts, news organization and applies multiple filters to perform preprocessing. Then it applies ClaimBuster model over these tweets for finding the check-worthiness score of each tweet, and then applies several filters to eliminate redundant and irrelevant tweets. Only highly scored tweets are retweeted using the account.
3. Event list: This part shows a list of debates covered by the platform. When a debate is broadcasted in a television channel, this platform covers the debate live and the top-entry of the list features the live debate. Closed captions of the debate are fed to the platform through a device called TextGrabber. Past debates are listed below the live debates in inverse chronological order. Figure 2.7 is a screenshot of the platform when a past debate is selected. The background colors of the sentences indicate how check-worthy they are. Darker colors correspond to higher check-worthiness scores. By default, all sentences having scores higher than or equal to 0.5 are highlighted. A slider allows the user to modify this threshold. An *Order by Score* button allows the user to order all the sentences by their check-worthiness scores. This way of ranking helps fact-checkers prioritize their efforts in assessing the veracity of claims. Thus, ClaimBuster will free journalists from the time-consuming task of finding check-worthy claims, leaving them with more time for reporting and writing.

---

<sup>9</sup><http://twitter.com/ClaimBusterTM>

## 2.6 CASE STUDY: 2016 U.S. Presidential Election Primary Debates

Using the 21 primary debates, we compared ClaimBuster against the human fact-checkers at several popular fact-checking organizations. We are interested in testing the hypothesis that the claims picked by ClaimBuster are also more likely to be fact-checked by professionals. If the hypothesis is true, we can expect ClaimBuster to be effective in assisting professionals choose what to fact-check and thus helping improve their work efficiency.

### 2.6.1 Data Collection

There have been 12 Republican<sup>10</sup> and 9 Democratic primary debates in the 2016 U.S. presidential election. The debates featured as many as 11 Republican Party candidates and 5 Democratic Party candidates at the beginning, respectively. These debates took place between August, 2015 and April, 2016. We collected the transcripts of all these debates from several news media websites, including Washington Post, CNN, Times, and so on. There are a total of 30737 sentences in the 21 transcripts. We pre-processed these transcripts and identified the speaker of each sentence. Furthermore, we identified the role of the speaker. Sentences spoken by debate moderators were excluded from the study.

### 2.6.2 Finding Check-worthy Factual Claims

We use ClaimBuster to calculate the check-worthiness scores of the sentences and thereby identify highly check-worthy factual claims. Figure 2.8 shows the distributions of ClaimBuster scores on all the sentences for both political parties. The distributions for the two parties are similar. One distinction is that the distribution for the Republican Party has a higher peak and a slightly thinner right tail than

---

<sup>10</sup>We only considered the “prime time” debates which included the more popular candidates.

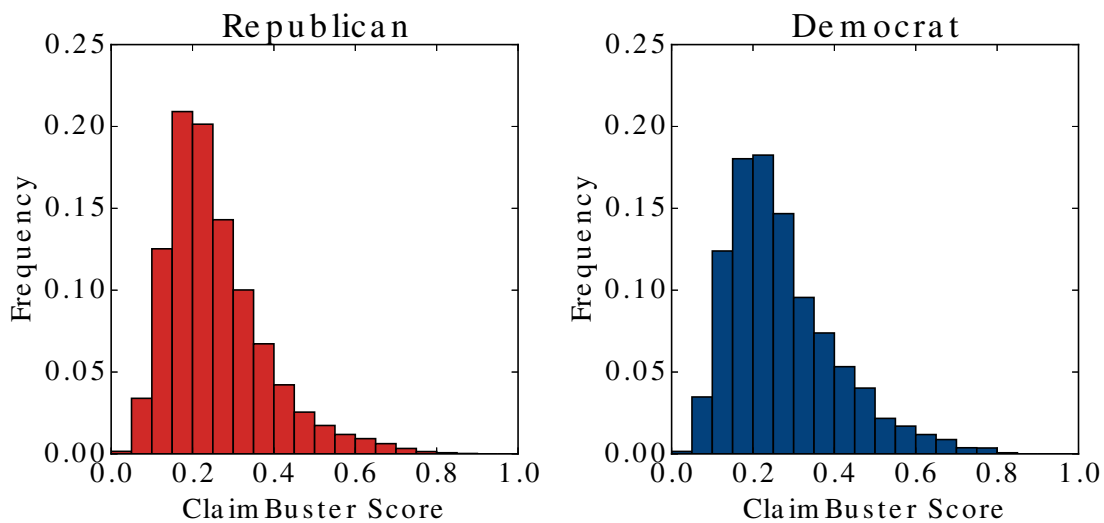


Figure 2.8. Distributions of ClaimBuster scores over all the sentences for both parties

the distribution for the Democratic party. There are 776 check-worthy factual claims spoken by the Republicans with ClaimBuster scores over 0.5. This is 5.06% of all the sentences spoken by the Republican candidates. From Democrat candidates, there are 484 (6.73%) sentences with ClaimBuster score higher than 0.5.

Figure 2.9 shows the check-worthiness score distributions for the major candidates (nomination winners and runner-ups) from both parties. Among these four candidates, *Donald Trump* appears to have presented less number of highly check-worthy factual claims (ClaimBuster score  $\geq 0.5$ ) than the other three candidates. He has used more non-factual sentences (ClaimBuster score  $\leq 0.3$ ) compared to the other candidates.

### 2.6.3 Topic Detection

From each of the 21 debates, the 20 highest-scoring sentences were selected and manually placed in topic categories, a modified version of the most important prob-

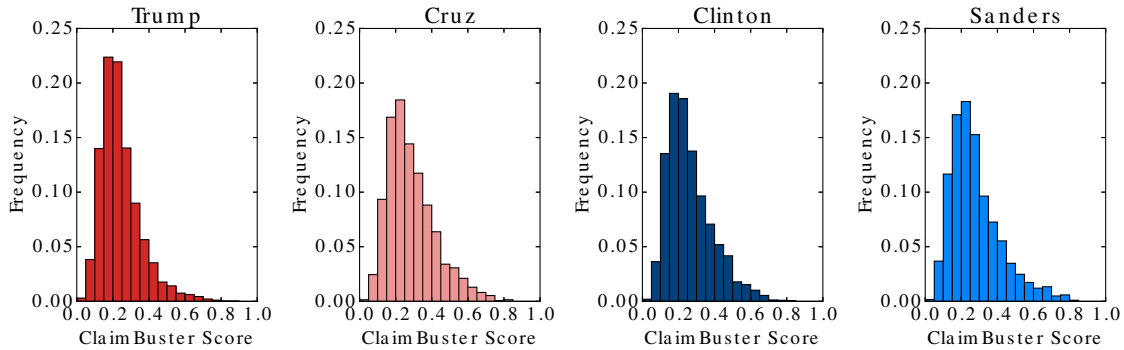


Figure 2.9. Distributions of ClaimBuster scores over all the sentences for the major candidates

lems (MIP) used by Gallup and other researchers for decades [19, 20, 21]. The major topics in the primary debates were: economy, crime, international affairs, immigration, health care, social issues, education, campaign finance, environment, Supreme Court, privacy and energy. Some of these topics were further broken down into subtopics. The 420 sample sentences were used to cultivate a list of keywords most often found for each of these topics. For example, the keywords for subtopic “abortion” were abortion, pregnancy and planned parenthood. Some topics had a small number of keywords, others had more than 20.

A topic-detection program is created to detect each debate sentence’s topic. Provided a sentence, the program computes a score for each topic in our list based on presence of each topic’s keywords in the sentence. The score is the total number of occurrences of such keywords. The sentence is assigned to the topic attaining the highest score among all the topics. However, if the highest score is lower than a threshold (two occurrences of topic keywords), the program does not assign any of the topics to the sentence. If there is a tie between two or more topics, the program uses

Platforms	avg(YES)	avg(NO)	t-value	p-value
CNN	0.433	0.258	21.137	1.815E-098
PolitiFact	0.438	0.258	16.362	6.303E-060

Table 2.7. Score differences between sentences fact-checked and those not chosen for checking

the topic of the preceding sentence if it matches one of the tied topics. Otherwise, it randomly picks one of the tied topics.

In order to evaluate the above approach to detect topics, we created ground-truth data for one Republican debate and one Democratic debate. We only used sentences with at least 0.5 ClaimBuster score. In our ground-truth data for the Democratic debate, there are 52 sentences and 39 of them are labeled with a topic. The program detected topics for 27 of the 39 sentences and only one sentence was assigned with a incorrect topic. For the Republican debate ground-truth data, there are 62 sentences and 44 sentences are labeled with a topic. The program found topics for 30 out of the 44 sentences and 5 of these sentences were mis-classified.

We applied the topic detection program on all remaining sentences of these debates. The topics of the sentences allow us to gain better insight into the data. The results of our study which leverages the detected topics are reported in Section 2.6.5. The high accuracy of the topic-detection program on the ground-truth data gives us confidence on the results.

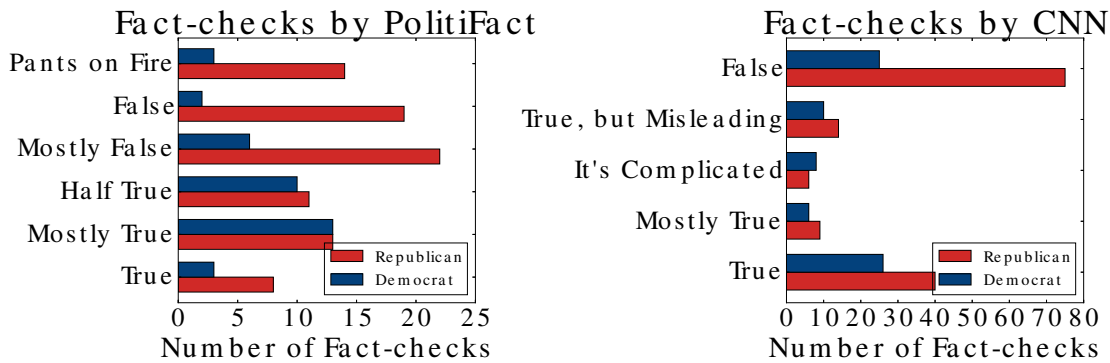


Figure 2.10. Distribution of verdicts for each party

#### 2.6.4 Verdict Collection

We used CNN and PolitiFact as the means for comparing ClaimBuster’s results. These two organizations were selected because each identifies claims they judge to be worth checking and then rates each claim on a truthfulness scale. The verdicts for CNN are true, mostly true, true but misleading, false or it’s complicated. PolitiFact uses true, mostly true, half true, mostly false, false and “pants on fire” (egregiously false). Other organizations focus specifically on false or misleading claims only (Factcheck.org) or write about debate statements they found interesting or suspicious (Washington Post) which makes a comparison to ClaimBuster problematic.

For each of the 21 debates CNN and PolitiFact prepared a summary of the factual claims they chose to check and rendered a verdict on them. We collected all of these verdicts, 224 from CNN and 118 from PolitiFact.

Table 2.7 shows scores given by ClaimBuster to the claims fact-checked by CNN and PolitiFact. The ClaimBuster average for sentences fact-checked by CNN is 0.433 compared to 0.258 for those sentences not selected by CNN, a statistically significant difference. Likewise, the ClaimBuster average for sentences checked by PolitiFact



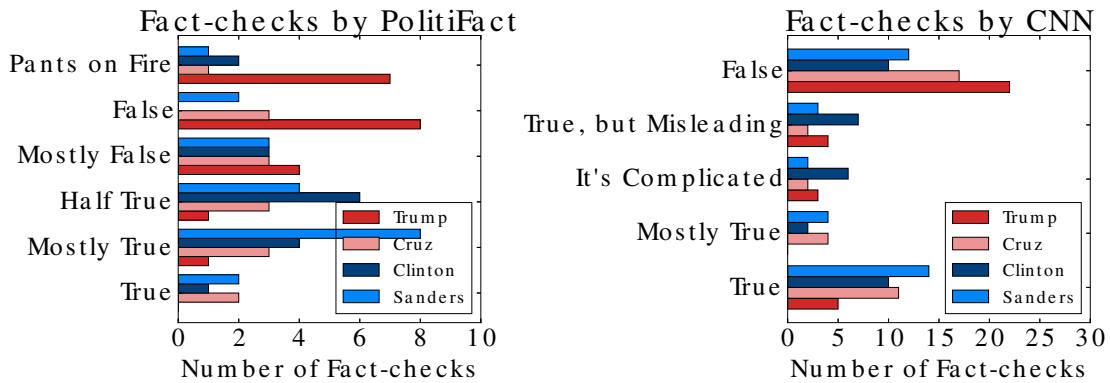


Figure 2.11. Distribution of verdicts for each major candidate

is 0.438 compared to 0.258 for those not selected, also a significant difference. The results of these comparisons demonstrate the utility of ClaimBuster in identifying sentences likely to contain important factual claims.

Figure 2.10 shows, for each party, the number of fact-checks of different veracity by CNN and PolitiFact. Figure 2.11 shows number of fact-checks for each major candidates. One observation is, *Donald Trump* has presented more *Pants on Fire*, *False* and *Mostly False* factual claims than other candidates according to PolitiFact. Similar observation is also evident according to CNN.

### 2.6.5 Analysis

With the ClaimBuster score, topic and veracity of the sentences at hand, we study the relation among these and try to find answers to questions such as which candidate presented more factual claims pertaining to a certain topic compared to others and so on.

Figure 2.12 shows the distribution of topics among sentences for each party. Republican candidates are more vocal about *Economy*, *International Affairs*, and

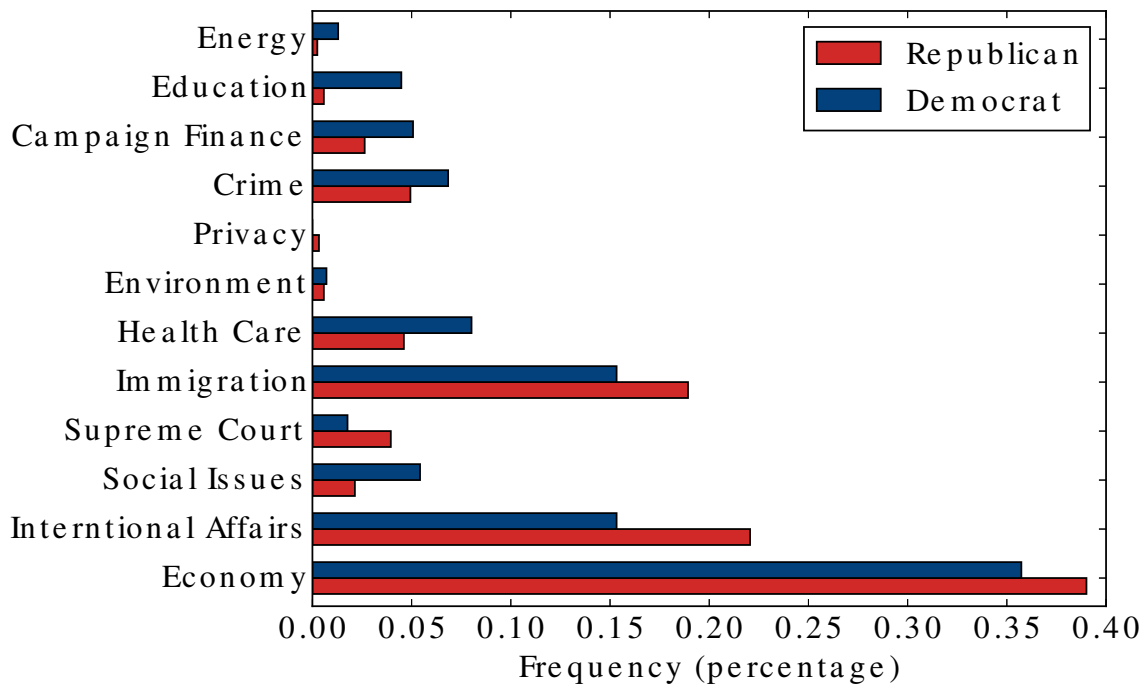


Figure 2.12. Distribution of topics over all the sentences for each party

*Immigration* compared to the Democrats. On the other hand, Democrats are more vocal on *Energy*, *Education*, *Social Issues* and *Health Care*. We roll down to the candidate level and try to understand the most vocal candidates on each of the topics. Figure 2.13 shows the topic distribution for each major candidate. *Bernie Sanders* was the most vocal on *Social Issues* among the candidates. *Ted Cruz* spoke significantly more on *International Affairs* compared to other candidates.

We analyzed the check-worthiness of the sentences of each topic. Figure 2.14 shows the topic distribution of sentences having ClaimBuster score  $\geq 0.5$ . This figure explains how often the candidates used factual claims while speaking about different topics. For example, both *Donald Trump* and *Bernie Sanders* presented significantly more check-worthy factual claims relating to the *Economy* compared to their debate competitors.

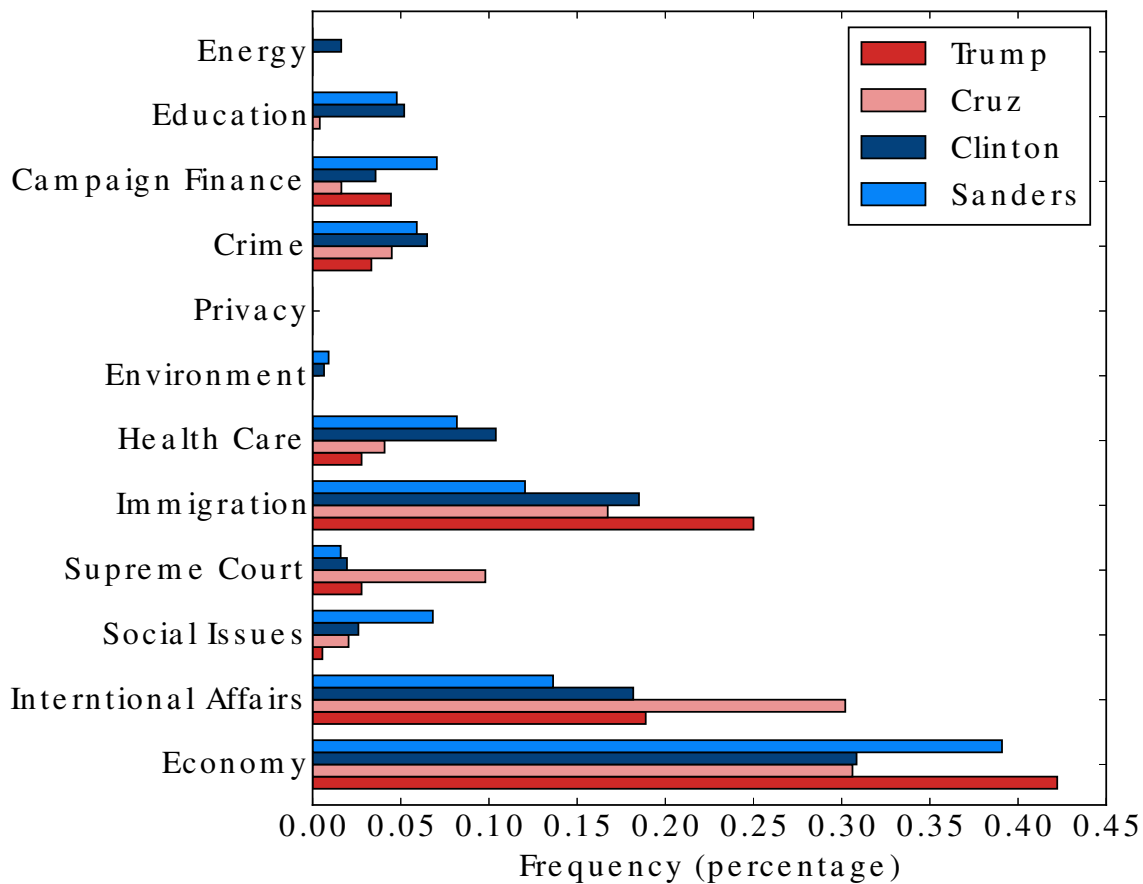


Figure 2.13. Distribution of topics over all the sentences from the major candidates

Figure 2.15 shows the topic distribution of sentences having ClaimBuster score  $\leq 0.3$ . This figure explains how much the candidates spoke about different topics without presenting factual claims. One interesting observation derived from Figures 2.14 and 2.15 is that Republican candidates spoke about *Health Care* but used fewer factual claims regarding this topic. On the other hand, Democratic candidate *Hillary Clinton* presented factual statements related to *Environment* rather than presenting non-factual, subjective statements.

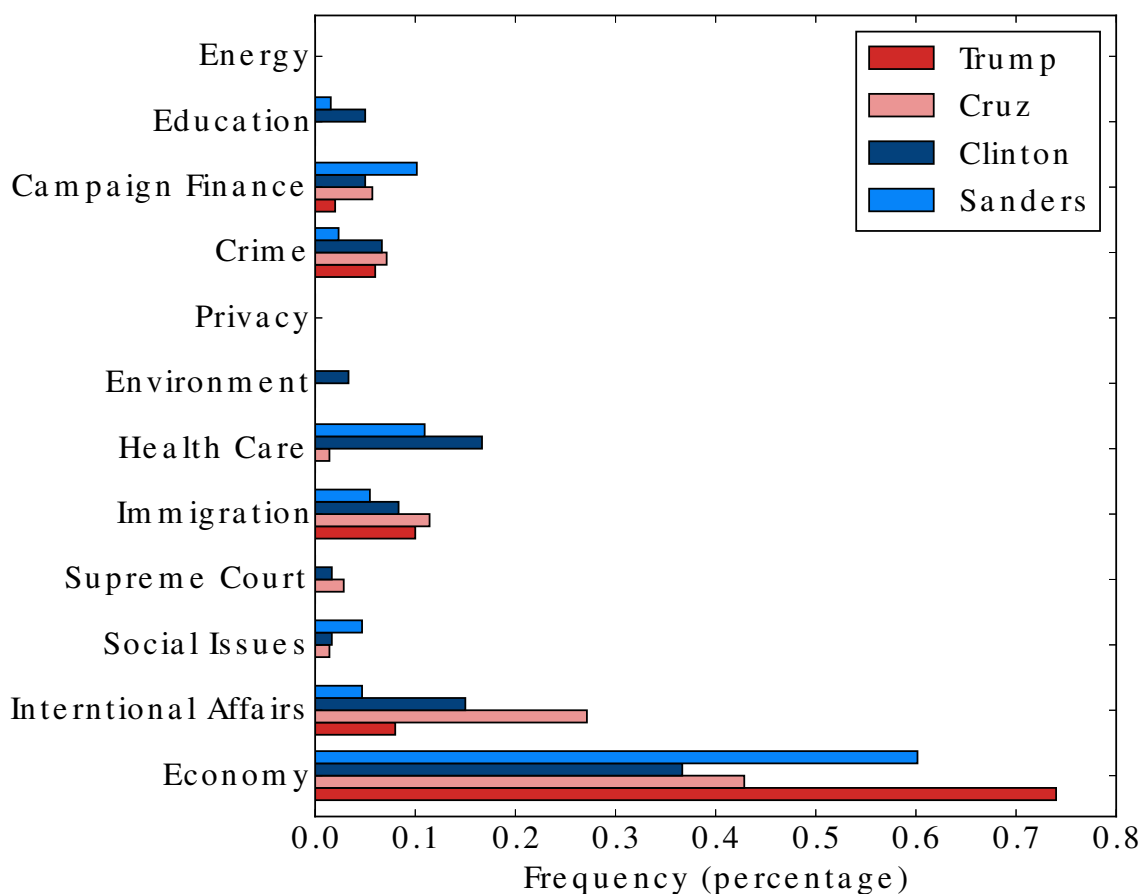


Figure 2.14. Distribution of topics over sentences scored high ( $\geq 0.5$ ) by ClaimBuster

Figure 2.16 shows the topic distributions of CNN, PolitiFact sentences as well as of highly check-worthy factual sentences (ClaimBuster score  $\geq 0.5$ ). This figure signifies that there are strong similarities between ClaimBuster and the fact-checking organizations. ClaimBuster tends to give high scores to the topics which CNN and PolitiFact tend to choose for fact checking. For example, all three have about 50 percent of the fact checks (or high ClaimBuster scores) associated with *Economy*, about 14 percent for *International Affairs*, about 10 percent for *Immigration* and 4 percent for *Crime*. One topic where ClaimBuster showed a difference with the human

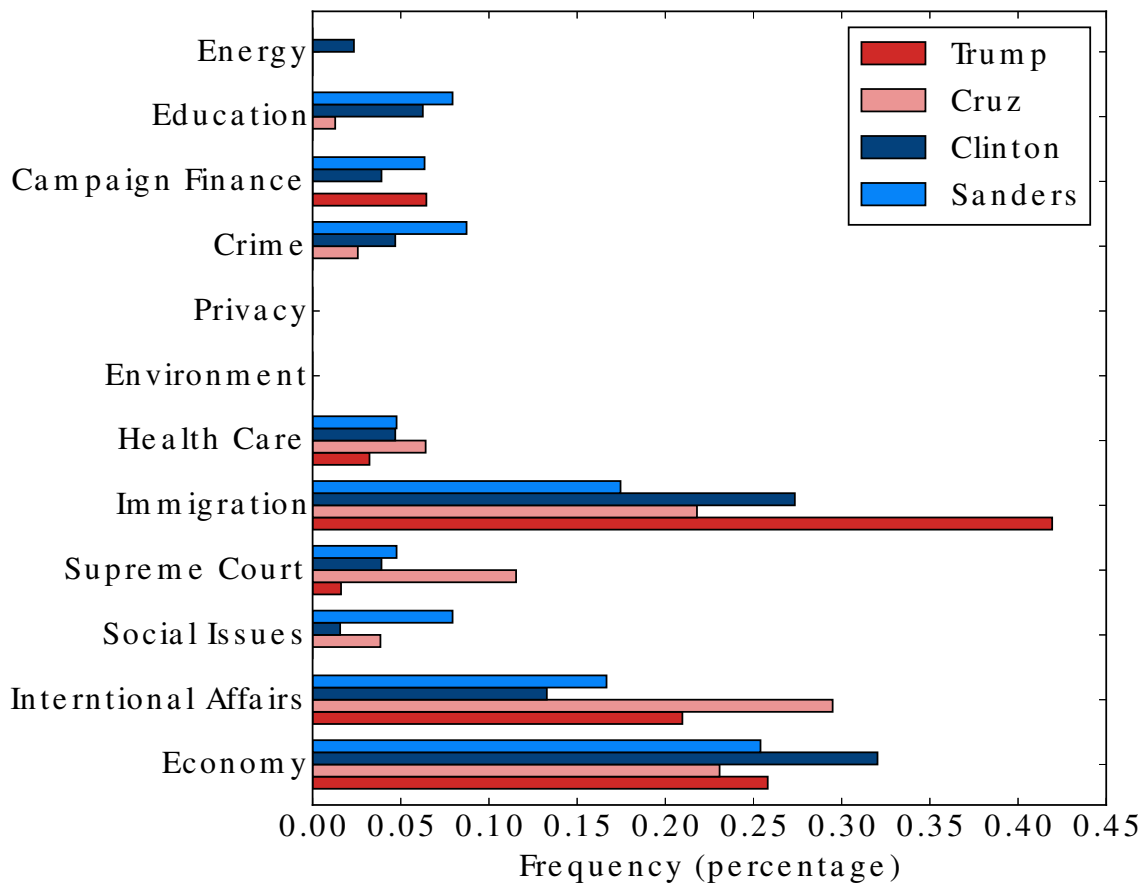


Figure 2.15. Distribution of topics over sentences scored low ( $\leq 0.3$ ) by ClaimBuster

fact-checkers was *Social Issues*. That topic represented about 9 percent of the CNN and PolitiFact fact-checks but only about 2 percent of the highly scored ClaimBuster sentences.

## 2.7 Future Plan

We look forward to making progress on several fronts in building ClaimBuster in the future. We are applying ClaimBuster on Australian Parliament Hansard <sup>11</sup>. This will facilitate fact-checking statements made by the members of parliament.

<sup>11</sup>[http://www.apf.gov.au/Parliamentary\\_Business/Hansard](http://www.apf.gov.au/Parliamentary_Business/Hansard)

Building a repository of fact-checks done by professionals is also in our agenda. This will enable automatic matching of claims during a live event with known fact-checks in the repository and instantly informing the audience about the claims' veracity.

We are also studying claims found in the media about various domains such as politics, sports and so on. Particularly, we are interested in investigating how numbers, actions and comparisons are used in factual claims. Furthermore, we will

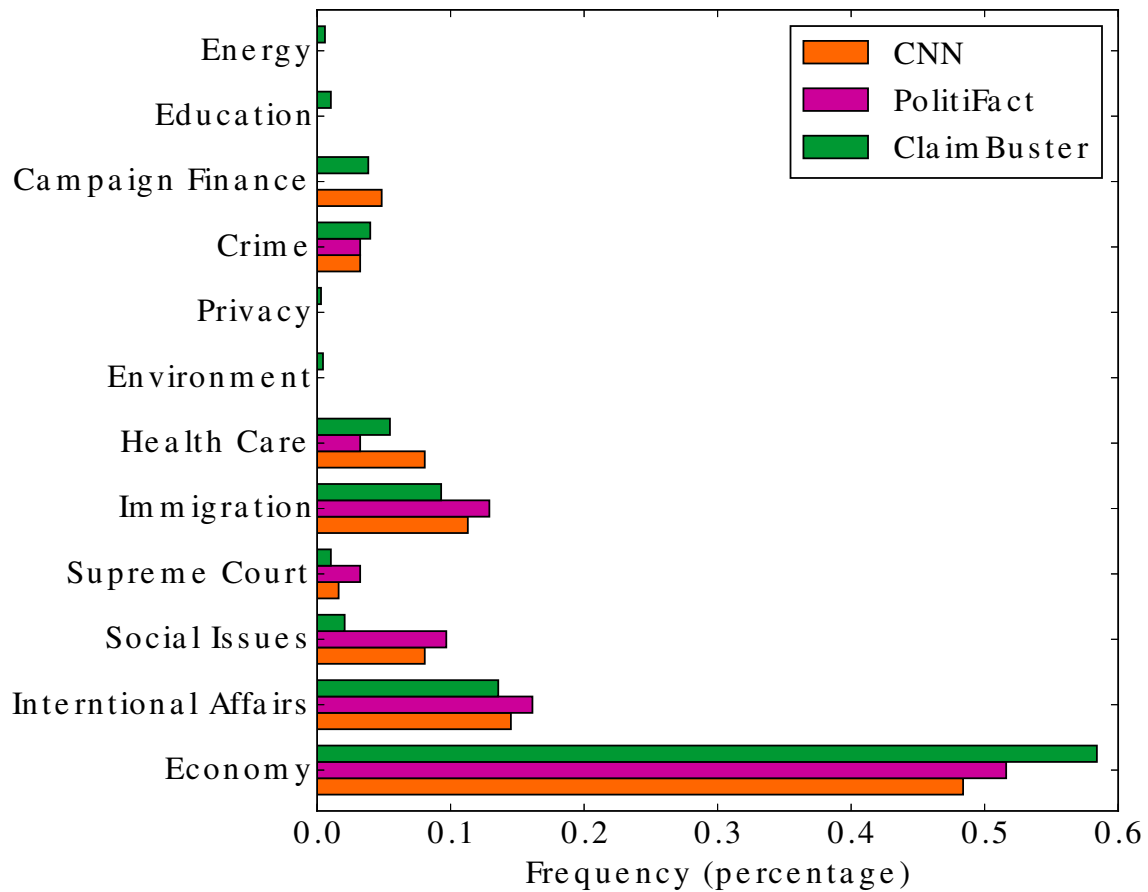


Figure 2.16. Comparison of topic distributions of CNN, PolitiFact fact-checked sentences and sentences scored high ( $\geq 0.5$ ) by ClaimBuster

formulate templates of factual claims. Such templates will enable programs to automatically categorize claims, which can be valuable for improving fact-checking accuracy.

All these efforts will bring us closer towards the “Holy Grail” of automated fact-checking - a fully automated, live, end-to-end fact-checking system [22].

## CHAPTER 3

### Automatic Fact Monitoring

Towards computational journalism, we present **FactWatcher**, a system that helps journalists identify data-backed, attention-seizing facts which serve as leads to news stories. **FactWatcher** discovers three types of facts, including situational facts, one-of-the-few facts, and prominent streaks, through a unified suite of data model, algorithm framework, and fact ranking measure. Given an append-only database, upon the arrival of a new tuple, **FactWatcher** monitors if the tuple triggers any new facts. Its algorithms efficiently search for facts without exhaustively testing all possible ones. Furthermore, **FactWatcher** provides multiple features in striving for an end-to-end system, including fact ranking, fact-to-statement translation and keyword-based fact search.

#### 3.1 Introduction

*Computational journalism* emerged recently as a young interdisciplinary field [1] that brings together experts in journalism, social sciences and computer science, and advances journalism by innovations in computational techniques. Database and data mining researchers have also started to push the frontiers of this field [2, 3, 4]. One of the goals in computational journalism is *newsworthy fact discovery*. Reporters always try hard to bring out attention-seizing factual statements backed by data, which may lead to news stories and investigation. While such statements take many different forms, we consider a common form exemplified by the following excerpts from real-world news media:



- “Paul George had 21 points, 11 rebounds and 5 assists to become the first Pacers player with a 20/10/5 (points/rebounds/assists) game against the Bulls since Detlef Schrempf in December 1992.” (<http://espn.go.com/espn/elias?date=20130205>)
- “The social world’s most viral photo ever generated 3.5 million likes, 170,000 comments and 460,000 shares by Wednesday afternoon.” ([http://www.cnbc.com/id/49728455/President\\_Obama\\_Sets\\_New\\_Social\\_Media\\_Record](http://www.cnbc.com/id/49728455/President_Obama_Sets_New_Social_Media_Record))

What is common in the above two statements is a prominent fact with regard to a context and several measures. In the first statement, the context includes the performance of Pacers players in games against the Bulls since December 1992 and the measures are points, rebounds, assists. By these measures, no performance in the context is better than the mentioned performance of Paul George. For the second statement, the measures are likes, comments, shares and the context includes all photos posted to Facebook. The story is that no photo in the context attracted more attention than the mentioned photo of President Barack Obama, by the three measures. In general, facts can be put in many contexts, such as photos posted in 2012, photos posted by political campaigns, and so on.

Similar facts can be stated on data from domains outside of sports and social media, including stock data, weather data, and criminal records. For example: 1) “Stock A becomes the first stock in history with price over \$300 and market cap over \$400 billion.” 2) “Today’s measures of wind speed and humidity are  $x$  and  $y$ , respectively. City B has never encountered such high wind speed and humidity in March.” 3) “There were 35 DUI arrests and 20 collisions in city C yesterday, the first time in 2013.” Some of these facts are not only interesting to reporters but also useful to financial analysts, scientists, and citizens.

In technical terms, a fact considered here is a *contextual skyline* object that stands out against other objects in a context with regard to a set of measures. Con-

tuple id	player	day	month	season	team	opp_team	points	assists	rebounds
$t_1$	Bogues	11	Feb.	1991-92	Hornets	Hawks	4	12	5
$t_2$	Seikaly	13	Feb.	1991-92	Heat	Hawks	24	5	15
$t_3$	Sherman	7	Dec.	1993-94	Celtics	Nets	13	13	5
$t_4$	Wesley	4	Feb.	1994-95	Celtics	Nets	2	5	2
$t_5$	Wesley	5	Feb.	1994-95	Celtics	Timberwolves	3	5	3
$t_6$	Strickland	3	Jan.	1995-96	Blazers	Celtics	27	18	8
$t_7$	Wesley	25	Feb.	1995-96	Celtics	Nets	12	13	5

\* Attribute `opp_team` is the short form of opposition team.

Table 3.1. A Mini-world of Basketball Gamelogs

sider a table  $R$  whose schema includes a set of measure attributes  $\mathcal{M}$  and a set of dimension attributes  $\mathcal{D}$ . A context is a subset of  $R$ , resulting from a conjunctive constraint defined on a subset of the dimension attributes  $D \subseteq \mathcal{D}$ . A measure subspace is defined by a subset of the measure attributes  $M \subseteq \mathcal{M}$ . A tuple  $t$  is a contextual skyline tuple if no other tuple in the context dominates  $t$ . A tuple  $t'$  dominates  $t$  if  $t'$  is better than or equal to  $t$  on every attribute in  $M$  and better than  $t$  on at least one of the attributes. Such is the standard notion of dominance relation adopted in *skyline analysis* [23].

We study how to find *situational facts* pertinent to new tuples in an ever-growing database, where the tuples capture real-world events. We propose algorithms that, whenever a new tuple  $t$  enters an append-only table  $R$ , discover constraint-measure pairs that qualify  $t$  as a contextual skyline tuple. Each such pair constitutes a situational fact pertinent to  $t$ 's arrival.

**Example 1.** Consider the mini-world of basketball gamelogs  $R$  in Table 3.1, where  $\mathcal{D}=\{\text{player, month, season, team, opp\_team}\}$  and  $\mathcal{M}=\{\text{points, assists, rebounds}\}$ . The existing tuples are  $t_1$  to  $t_6$  and the new tuple is  $t_7$ . If the context is the whole table (i.e., no constraint) and the measure subspace  $M=\mathcal{M}$ ,  $t_7$  is not a skyline tuple since it is dominated by  $t_3$  and  $t_6$ . However, with regard to context  $\sigma_{\text{month=Feb.}}(R)$  (corresponding to constraint  $\text{month=Feb.}$ ) and the same measure subspace  $M$ ,  $t_7$  is in the skyline along with  $t_2$ . In yet another context  $\sigma_{\text{team=Celtics} \wedge \text{opp\_team=Nets}}(R)$  under measure subspace  $M=\{\text{assists, rebounds}\}$ ,  $t_7$  is in the skyline along with  $t_3$ . Tuple  $t_7$  is also a contextual skyline tuple for other constraint-measure pairs, which we do not further enumerate.  $\square$

Discovering situational facts is challenging as timely discovery of such facts is expected. In finding news leads centered around situational facts, the value of a news piece diminishes rapidly after the event takes place. Consider NBA games again. Sports media need to identify and discuss sensational records quickly as they emerge; any delay makes fans less interested in the records and risks losing them to rival media. Timely identification of situational facts is also critical in areas beyond journalism. To make informed investment decisions, investors want to know facts related to stock trading as soon as possible. Facts discovered from weather data can assist scientists in identifying extreme weather conditions and help government and the public in coping with the weather.

Simple situational facts on a single measure and a complete table, e.g., the all-time NBA scoring record, can be conveniently detected by database triggers. However, general and complex facts involving multiple dimension and measure attributes are much harder to discover. Exhaustively using triggers leads to an exponential explosion of constraint-measure pairs to check for each new tuple. In reality, news media relies on instincts and experiences of domain experts on this endeavor. The experts,

impressed by an event such as the outstanding performance of a player in a game, hypothesize a fact and manually craft a database query to check it. This is how Elias Sports Bureau tackles the task and provides sports records (such as the aforementioned one by Paul George) to many sports media [24]. With ever-growing data and limited human resources, such manual checking is time-consuming and error-prone. Its low efficiency not only leads to delayed and missing facts, but also ties up precious human expertise that could be otherwise devoted to more important journalistic activities.

The technical focus of this chapter is thus on efficient automatic approach to discovering situational facts, i.e., finding constraint-measure pairs that qualify a new tuple  $t$  as a contextual skyline tuple. A straightforward brute-force approach would compare  $t$  with every historical tuple to determine if  $t$  is dominated, repeatedly for every conjunctive constraint satisfied by  $t$  under every possible measure subspace. The obvious low-efficiency of this approach has three culprits—exhaustive comparison with *every tuple*, under *every constraint*, and over *every measure subspace*. We thus design algorithms to counter these issues by three corresponding ideas, as follows:

**1) Tuple reduction** Instead of comparing  $t$  with every previous tuple, it is sufficient to only compare with current skyline tuples. This is based on the simple property that, if any tuple dominates  $t$ , then there must exist a skyline tuple that also dominates  $t$ . For example, in Table 3.1, under constraint `month=Feb.` and the full measure space  $\mathcal{M}$ , the corresponding context contains  $t_1$ ,  $t_2$ ,  $t_4$  and  $t_5$ , and the contextual skyline has two tuples— $t_1$  and  $t_2$ . When the new tuple  $t_7$  comes, with regard to the same constraint-measure pair, it suffices to compare  $t_7$  with  $t_1$  and  $t_2$ , not the remaining tuples.

**2) Constraint pruning** If  $t$  is dominated by  $t'$  in a particular measure subspace  $M$ , then  $t$  does not belong to the contextual skyline of constraint-measure pair

$(C, M)$  for any  $C$  satisfied by both  $t$  and  $t'$ . For example, since  $t_7$  is dominated by  $t_3$  in the full measure space  $\mathcal{M}$ , it is not in the contextual skylines for  $(\text{team}=\text{Celtics} \wedge \text{opp\_team}=\text{Nets}, \mathcal{M})$ ,  $(\text{team}=\text{Celtics}, \mathcal{M})$ ,  $(\text{opp\_team}=\text{Nets}, \mathcal{M})$  and  $(\text{no constraint}, \mathcal{M})$ . Furthermore, since  $t_7$  is dominated by  $t_6$  in  $\mathcal{M}$ , it does not belong to the contextual skylines for  $(\text{season}=1995-96, \mathcal{M})$  and  $(\text{no constraint}, \mathcal{M})$ . Based on this, we examine the constraints satisfied by  $t$  in a certain order, such that comparisons of  $t$  with skyline tuples associated with already examined constraints are used to prune remaining constraints from consideration.

**3) Sharing computation across measure subspaces** Since repeatedly visiting the constraints satisfied by  $t$  for every measure subspace is wasteful, we pursue sharing computation across different subspaces. The challenge in such sharing lies in the anti-monotonicity of dominance relation—a skyline tuple in space  $M$  may or may not be in the skyline of a superspace or subspace  $M'$  [25]. Nonetheless, we can first consider the full space  $\mathcal{M}$  and prune various constraints from consideration for smaller subspaces. For instance, after comparing  $t_7$  with  $t_2$  in  $\mathcal{M}$ , the algorithms realize that  $t_7$  has smaller values on `points` and `rebounds`. It is dominated by  $t_2$  in three subspaces— $\{\text{points}, \text{rebounds}\}$ ,  $\{\text{points}\}$  and  $\{\text{rebounds}\}$ . When considering these subspaces, we can skip two contexts—corresponding to constraint `month=Feb.` and empty constraint, respectively—as  $t_2$  and  $t_7$  are in both contexts.

It is crucial to report truly *prominent* situational facts. A newly arrived tuple  $t$  may be in the contextual skylines for many constraint-measure pairs. Reporting all of them will overwhelm users and make important facts harder to spot. We measure the *prominence* of a constraint-measure pair by the cardinality ratio of all tuples to skyline tuples in the corresponding context. The intuition is that, if  $t$  is one of the very few skyline tuples in a context containing many tuples under a measure subspace, then the corresponding constraint-measure pair brings out a prominent fact. We thus

rank all situational facts pertinent to  $t$  in descending order of prominence. Reporters and experts can choose to investigate top- $k$  facts or the facts with prominence values above a threshold.

## 3.2 Related Work

Pioneers in data journalism have considerable success in using computer programs to write stories about sports games and stock earnings (e.g., StatSheet <http://statsheet.com/> and Narrative Science <http://www.narrativescience.com/>). The stories follow writing patterns to narrate box scores and play-by-play data and a company’s earnings data. They focus on capturing what happened in the game or what the earnings numbers indicate. They do not find situational facts pertinent to a game or an earnings report in the context of historical data.

Skyline query is extensively investigated in recent years, since Börzsönyi et al. [23] brought the concept to the database field. In [23] and the studies afterwards, it is assumed both the context of tuples in comparison and the measure space are given as query conditions. A high-level perspective on what distinguishes our work is—while prior studies *find answers* (i.e., skyline points) for a given query (i.e., a context, a measure space, or their combination), we study the reverse problem of *finding queries* (i.e., constraint-measure pairs that qualify a tuple as a contextual skyline tuple, among all possible pairs) for a particular answer (i.e., a new tuple).

From a technical perspective, Table 3.2 summarizes the differences among the more relevant previous studies and this chapter, along three aspects—whether they consider all possible contexts defined on dimension attributes, all measure subspaces, and incremental computation on dynamic data. With regard to context, Zhang et al. [26] integrate the evaluation of a constraint with finding skyline tuples in the corresponding context in a given measure space. With regard to measure, Pei et

al. [25] compute on static data the *skycube*—skyline points in all measure subspaces. Xia et al. [27] studied how to update a compressed skycube (CSC) when data change. The CSC stores a tuple  $t$  in its *minimum subspaces*—the measure subspaces in which  $t$  is a skyline tuple and of which the subspaces do not contain  $t$  in the skyline. They proposed an algorithm to update CSC when new tuples come and also an algorithm to use CSC to find all skyline tuples for a given measure subspace.

We can adapt [27] to find situational facts. While [28] provides experimental comparisons with the adaptation, here we analyze its shortcomings. Since [27] does not consider different contexts, the adaptation entails maintaining a separate CSC for every possible context. Upon the arrival of a new tuple  $t$ , for every context, the adaptation will update the corresponding CSC. Since a CSC only stores  $t$  in its minimum subspaces, the adaptation needs to run their query algorithm to find the skyline tuples for all measure subspaces, in order to determine if  $t$  is one of the skyline tuples. This is clearly an overkill, caused by that CSC is designed for finding all skyline tuples. Furthermore, while our algorithms can share computation across measure subspaces, there does not appear to be an effective strategy to share the computation of CSC algorithms across different contexts.

*Promotion analysis by ranking* [29] finds the contexts in which an object is ranked high. It ranks objects by a single score attribute, while we define object dominance relation on multiple measure attributes. It considers one-shot computation on static data, while we focus on incremental discovery on dynamic data. Due to these distinctions, the algorithmic approaches in the two works are also fundamentally different.

Wu et al. [4] studied the *one-of-the- $\tau$  object* problem, which entails finding the largest  $k$  value and the corresponding  *$k$ -skyband objects* (objects dominated by less than  $k$  other objects) such that there are no more than  $\tau$   $k$ -skyband objects.

	all possible contexts	measure subspaces	incremental
[26]	no	no	no
[25]	no	yes	no
[27]	no	yes	yes
[29]	yes	no	no
[4]	no	yes	no
[30]	no	no	yes
this work	yes	yes	yes

Table 3.2. Comparing Related Work on Three Modeling Aspects

They consider all measure subspaces but not different contexts formed by constraints. Similar to [29], it focuses on static data.

Alvanaki et al. [30] worked on detecting interesting events through monitoring changes in ranking, by using materialized view maintenance techniques. The work focuses on top- $k$  queries on single ranking attribute rather than skyline queries defined on multiple measure attributes. Their ranking contexts have at most three constraints. The work is similar to [31] which studied how to predict significant events based on historical data and correspondingly perform lazy maintenance of ranking views on a database.

### 3.3 Problem Statement

This section provides a formal description of our data model and problem statement. Consider a relational schema  $R(\mathcal{D}; \mathcal{M})$ , where the *dimension space* is a set of *dimension attributes*  $\mathcal{D} = \{d_1, \dots, d_n\}$  on which *constraints* are specified, and the *mea-*



$id$	$d_1$	$d_2$	$d_3$	$m_1$	$m_2$
$t_1$	$a_1$	$b_2$	$c_2$	10	15
$t_2$	$a_1$	$b_1$	$c_1$	15	10
$t_3$	$a_2$	$b_1$	$c_2$	17	17
$t_4$	$a_2$	$b_1$	$c_1$	20	20
$t_5$	$a_1$	$b_1$	$c_1$	11	15

Table 3.3. Running Example

sure space is a set of *measure attributes*  $\mathcal{M}=\{m_1, \dots, m_s\}$  on which dominance relation for skyline operation is defined. Any set of dimension attributes  $D \subseteq \mathcal{D}$  defines a *dimension subspace* and any set of measure attributes  $M \subseteq \mathcal{M}$  defines a *measure subspace*. In Table 3.3,  $R(\mathcal{D}; \mathcal{M}) = \{t_1, t_2, t_3, t_4, t_5\}$ ,  $\mathcal{D} = \{d_1, d_2, d_3\}$ ,  $\mathcal{M}=\{m_1, m_2\}$ .

We will use this table as a running example.

**Definition 1** (Constraint). A constraint  $C$  on dimension space  $\mathcal{D}$  is a conjunctive expression of the form  $d_1=v_1 \wedge d_2=v_2 \wedge \dots \wedge d_n=v_n$  (also written as  $\langle v_1, v_2, \dots, v_n \rangle$  for simplicity), where  $v_i \in \text{dom}(d_i) \cup \{*\}$  and  $\text{dom}(d_i)$  is the value domain of dimension attribute  $d_i$ . We use  $C.d_i$  to denote the value  $v_i$  assigned to  $d_i$  in  $C$ . If  $C.d_i=*$ , we say  $d_i$  is unbound, i.e., no condition is specified on  $d_i$ . We denote the number of bound attributes in  $C$  as  $\text{bound}(C)$ .

The set of all possible constraints over dimension space  $\mathcal{D}$  is denoted  $\mathcal{C}_{\mathcal{D}}$ . Clearly,  $|\mathcal{C}_{\mathcal{D}}| = \prod_i (|\text{dom}(d_i)| + 1)$ .

Given a constraint  $C \in \mathcal{C}_{\mathcal{D}}$ ,  $\sigma_C(R)$  is the relational algebra expression that chooses all tuples in  $R$  that satisfy  $C$ . □

---

**Algorithm 1:** Find  $\mathcal{C}^t$ 

---

**Input:**  $t \in R$ **Output:**  $\mathcal{C}^t$ : constraints satisfied by  $t$ 

```
1  $\mathcal{C}^t \leftarrow \emptyset$ ;  
2  $Q \leftarrow \emptyset$ ;  $Q.enqueue(\top)$ ;  
3 while not  $Q.empty()$  do  
4    $C \leftarrow Q.dequeue()$ ;  
5    $\mathcal{C}^t \leftarrow \mathcal{C}^t \cup \{C\}$ ;  
6    $i \leftarrow n$ ;  
7   while  $i > 0$  and  $C.d_i = *$  do  
8      $C' \leftarrow C$ ;  
9      $C'.d_i \leftarrow t.d_i$ ;  
10     $Q.enqueue(C')$ ;  
11     $i \leftarrow i - 1$ ;  
12 return  $\mathcal{C}^t$ ;
```

---

**Example 2.** For Table 3.3, an example constraint is  $C = \langle a_1, *, c_1 \rangle$  in which  $d_2$  is unbound.  $\sigma_C(R) = \{t_2, t_5\}$ .  $\square$

**Definition 2** (Skyline). Given a measure subspace  $M$  and two tuples  $t, t' \in R$ ,  $t$  dominates  $t'$  with respect to  $M$ , denoted by  $t \succ_M t'$  or  $t' \prec_M t$ , if  $t$  is equal to or better than  $t'$  on all attributes in  $M$  and  $t$  is better than  $t'$  on at least one attribute in  $M$ . A tuple  $t$  is a skyline tuple in subspace  $M$  if it is not dominated by any other tuple in  $R$ . The set of all skyline tuples in  $R$  with respect to  $M$  is denoted by  $\lambda_M(R)$ , i.e.,  $\lambda_M(R) = \{t \in R \mid \nexists t' \in R \text{ s.t. } t' \succ_M t\}$ .  $\square$

We use the general term “better than” in Def. 2, which can mean either “larger than” or “smaller than” for numeric attributes and either “ordered before” or “ordered after” for ordinal attributes, depending on applications. Further, the preferred ordering of values on different attributes are allowed to be different. For example, in a basketball game, 10 points is better than 5 points, while 3 fouls is worse than 1 foul. Without loss of generality, we assume measure attributes are numeric and a larger value is better than a smaller value.

---

**Algorithm 2:** BruteForce

---

**Input:**  $R(\mathcal{M}, \mathcal{D})$ : existing tuples;  $t$ : the new tuple

**Output:**  $S^t$ : the contextual skylines for  $t$

```
1  $S^t \leftarrow \emptyset$ ;  
2 foreach  $M \subseteq \mathcal{M}$  do  
3   foreach  $C \in \mathcal{C}^t$  do  
4      $pruned \leftarrow \mathbf{false}$ ;  
5     foreach  $t' \in R$  do  
6       if  $t \prec_M t'$  and  $t' \in \sigma_C(R)$  then  
7          $pruned \leftarrow \mathbf{true}$ ;  
8         break;  
9     if not  $pruned$  then  $S^t \leftarrow S^t \cup \{(C, M)\}$ ;  
10  $R \leftarrow R \cup \{t\}$ ;  
11 return  $S^t$ ;
```

---

**Definition 3** (Contextual Skyline). *Given a relation  $R(\mathcal{D}; \mathcal{M})$ , the contextual skyline under constraint  $C \in \mathcal{C}_{\mathcal{D}}$  over measure subspace  $M \subseteq \mathcal{M}$ , denoted  $\lambda_M(\sigma_C(R))$ , is the skyline of  $\sigma_C(R)$  in  $M$ .  $\square$*

**Example 3.** *For Table 3.3, if  $M = \mathcal{M}$ ,  $\lambda_M(R) = \{t_4\}$ . In fact,  $t_4$  dominates all other tuples in space  $M$ . If the constraint is  $C = \langle a_1, b_1, c_1 \rangle$ ,  $\sigma_C(R) = \{t_2, t_5\}$ ,  $\lambda_M(\sigma_C(R)) = \{t_2, t_5\}$  for  $M = \mathcal{M}$ , and  $\lambda_M(\sigma_C(R)) = \{t_2\}$  for  $M = \{m_1\}$ .  $\square$*

**Problem Statement** Given an append-only table  $R(\mathcal{D}; \mathcal{M})$  and the last tuple  $t$  that was appended onto  $R$ , the *situational fact discovery problem* is to find each constraint-measure pair  $(C, M)$  such that  $t$  is in the contextual skyline. The result, denoted  $S^t$ , is  $\{(C, M) | C \in \mathcal{C}_{\mathcal{D}}, M \subseteq \mathcal{M}, t \in \lambda_M(\sigma_C(R))\}$ . For simplicity of notation, we call  $S^t$  “the contextual skylines for  $t$ ”, even though rigorously speaking it is the set of  $(C, M)$  pairs whose corresponding contextual skylines include  $t$ .

---

**Algorithm 3:** BaselineSeq

---

**Input:**  $R(\mathcal{M}, \mathcal{D})$ : existing tuples;  $t$ : the new tuple

**Output:**  $S^t$ : the contextual skylines for  $t$

```
1  $S^t \leftarrow \emptyset$ ;  
2 foreach  $M \subseteq \mathcal{M}$  do  
3    $S \leftarrow \mathcal{C}^t$ ;  
4   foreach  $t' \in R$  do  
5      $\lfloor$  if  $t \prec_M t'$  then  $S \leftarrow S - \mathcal{C}^{t,t'}$ ;  
6     foreach  $C \in S$  do  
7        $\lfloor S^t \leftarrow S^t \cup \{(C, M)\}$ ;  
8  $R \leftarrow R \cup \{t\}$ ;  
9 return  $S^t$ ;
```

---

### 3.4 Solution Overview

Discovering situational facts for a new tuple  $t$  entails finding constraint-measure pairs that qualify  $t$  as a contextual skyline tuple. We identify three sources of inefficiency in a straightforward brute-force method, and we propose corresponding ideas to tackle them. To facilitate the discussion, we define the concept of *tuple-satisfied constraints*, which are all constraints pertinent to  $t$ , corresponding to the contexts containing  $t$ .

**Definition 4** (Tuple-Satisfied Constraint). *Given a tuple  $t \in R(\mathcal{D}; \mathcal{M})$  and a constraint  $C \in \mathcal{C}_{\mathcal{D}}$ , if  $\forall d_i \in \mathcal{D}$ ,  $C.d_i = *$  or  $C.d_i = t.d_i$ , we say  $t$  satisfies  $C$ . We denote the set of all such satisfied constraints by  $\mathcal{C}_{\mathcal{D}}^t$  or simply  $\mathcal{C}^t$  when  $\mathcal{D}$  is clear in context. It follows that given any  $C \in \mathcal{C}^t$ ,  $t \in \sigma_C(R)$ .  $\square$*

For  $C \in \mathcal{C}^t$ ,  $C.d_i$  can attain two possible values  $\{*, t.d_i\}$ . Hence,  $\mathcal{C}^t$  has  $2^n$  constraints in total for  $|\mathcal{D}|=n$ . Alg.1 is a simple routine used in all algorithms for finding all constraints of  $\mathcal{C}^t$ . It generates the constraints from the most general constraint  $\top = \langle *, *, \dots, * \rangle$  to the most specific constraint  $\langle t.d_1, t.d_2, \dots, t.d_n \rangle$ .  $\top$  has no bound attributes, i.e.,  $\text{bound}(\top)=0$ . Alg.1 makes sure a constraint is not generated twice, for

efficiency, by not continuing the while-loop in Line 7 once a specific attribute value is found in  $C$ .

A brute-force approach to the contextual skyline discovery problem would compare a new tuple  $t$  with every tuple in  $R$  to determine if  $t$  is dominated, repeatedly for every constraint satisfied by  $t$  in every possible measure subspace. It is shown in Alg.2. The obvious inefficiency of this approach has three culprits—the exhaustive comparison with *every tuple*, for *every constraint* and in *every measure subspace*. We devise three corresponding ideas to counter these causes, as follows:

**(1) Tuple reduction** For a constraint-measure pair  $(C, M)$ ,  $t$  is in the contextual skyline  $\lambda_M(\sigma_C(R))$  if  $t$  belongs to  $\sigma_C(R)$  and is not dominated by any tuple in  $\sigma_C(R)$ . Instead of comparing  $t$  with every tuple, it suffices to only compare with current skyline tuples. This simple optimization is based on the following proposition which ways, if any tuple dominates  $t$ , there must exist a skyline tuple that also dominates  $t$ .

**Proposition 1.** *Given a new tuple  $t$  inserted into  $R$ , a constraint  $C \in \mathcal{C}^t$  and a measure subspace  $M$ ,  $t \in \lambda_M(\sigma_C(R))$  if and only if  $\nexists t' \in \lambda_M(\sigma_C(R))$  such that  $t' \succ_M t$ .  $\square$*

To exploit this idea, our algorithms conceptually maintain the contextual skyline tuples for each context (i.e., measure subspace and constraint), and compare  $t$  only with these tuples for constraints that  $t$  satisfies.

**(2) Constraint pruning** For constraints satisfied by  $t$ , we need to determine whether  $t$  enters the contextual skyline. To prune constraints from consideration, we note the following property: if  $t$  is dominated by a skyline tuple  $t'$  under measure subspace  $M$ ,  $t$  is not in the contextual skyline of constraint-measure pair  $(C, M)$  for any  $C$  satisfied by both  $t$  and  $t'$ .

To enable constraint pruning, we organize all constraints in  $\mathcal{C}^t$  into a lattice by their subsumption relation. The constraints satisfied by both  $t$  and  $t'$ , denoted  $\mathcal{C}^{t,t'}$ , also form a lattice, which is the intersection of lattices  $\mathcal{C}^t$  and  $\mathcal{C}^{t'}$ . Below we formalize the concepts of lattice and lattice intersection.

**Definition 5** (Constraint Subsumption). *Given  $C_1, C_2 \in \mathcal{C}_{\mathcal{D}}$ ,  $C_1$  is subsumed by or equal to  $C_2$  (denoted  $C_1 \sqsubseteq C_2$  or  $C_2 \supseteq C_1$ ) iff*

1.  $\forall d_i \in \mathcal{D}, C_2.d_i = C_1.d_i$  or  $C_2.d_i = *$ .

*$C_1$  is subsumed by  $C_2$  (denoted  $C_1 \triangleleft C_2$  or  $C_2 \triangleright C_1$ ) iff  $C_1 \sqsubseteq C_2$  but  $C_1 \neq C_2$ . In other words, the following condition is also satisfied in addition to the above one—*

2.  $\exists d_i \in \mathcal{D}$  such that  $C_2.d_i = *$  and  $C_1.d_i \neq *$ , i.e,  $d_i$  is bound to a value belonging to  $\text{dom}(d_i)$  in  $C_1$  but is unbound in  $C_2$ .

*By definition,  $\sigma_{C_1}(R) \subseteq \sigma_{C_2}(R)$  if  $C_1 \sqsubseteq C_2$ . □*

**Example 4.** *Consider  $C_1 = \langle a, b, c \rangle$  and  $C_2 = \langle a, *, c \rangle$ . Here  $C_1.d_1 = C_2.d_1$ ,  $C_1.d_3 = C_2.d_3$ ,  $C_1.d_2 = b$  and  $C_2.d_2 = *$ . By Definition 5,  $C_1$  is subsumed by  $C_2$ , i.e.  $C_1 \triangleleft C_2$ . □*

**Definition 6** (Partial Order on Constraints). *The subsumption relation  $\sqsubseteq$  on  $\mathcal{C}_{\mathcal{D}}$  forms a partial order. The partially ordered set (poset)  $(\mathcal{C}_{\mathcal{D}}, \sqsubseteq)$  has a top element  $\top = \langle *, *, \dots, * \rangle$  that subsumes every other constraint in  $\mathcal{C}_{\mathcal{D}}$ .  $\top$  is the most general constraint, since it has no bound attributes. Note that  $(\mathcal{C}_{\mathcal{D}}, \sqsubseteq)$  is not a lattice and does not have a single bottom element. Instead, it has multiple minimal elements. Every minimal element  $C$  satisfies the condition that  $\forall d_i, C.d_i \neq *$ .*

*If  $C_1 \triangleleft C_2$ , we say  $C_1$  is a descendant of  $C_2$  ( $C_2$  is an ancestor of  $C_1$ ). If  $C_1 \triangleleft C_2$  and  $\text{bound}(C_1) - \text{bound}(C_2) = 1$ , then  $C_1$  is a child of  $C_2$  ( $C_2$  is a parent of  $C_1$ ). Given  $C \in \mathcal{C}_{\mathcal{D}}$ , we denote  $C$ 's ancestors, descendants, parents and children by  $\mathcal{A}_C$ ,  $\mathcal{D}_C$ ,  $\mathcal{P}_C$  and  $\mathcal{CH}_C$ , respectively. □*

**Definition 7** (Lattice of Tuple-Satisfied Constraints). *Given  $t \in R(\mathcal{D}; \mathcal{M})$ ,  $\mathcal{C}^t \subseteq \mathcal{C}_{\mathcal{D}}$  by definition. In fact,  $(\mathcal{C}^t, \sqsubseteq)$  is a lattice. Its top element is  $\top$ . Its bottom element  $\langle t.d_1, t.d_2, \dots, t.d_n \rangle$ , denoted  $\perp(\mathcal{C}^t)$ , is a minimal element in  $\mathcal{C}_{\mathcal{D}}$ .*

*Given  $C \in \mathcal{C}^t$ , we denote  $C$ 's ancestors, descendants, parents and children within  $\mathcal{C}^t$  by  $\mathcal{A}_C^t$ ,  $\mathcal{D}_C^t$ ,  $\mathcal{P}_C^t$  and  $\mathcal{CH}_C^t$ , respectively.  $|\mathcal{CH}_C^t| = n - \text{bound}(C)$  where  $n = |\mathcal{D}|$ , i.e., each child of  $C$  is a constraint by adding conjunct  $d_i = t.d_i$  into  $C$  for unbound attribute  $d_i$ . It is clear that  $|\mathcal{P}_C^t| = \text{bound}(C)$ . By definition,  $\mathcal{A}_C^t = \mathcal{A}_C$  and  $\mathcal{P}_C^t = \mathcal{P}_C$ , while  $\mathcal{D}_C^t \subseteq \mathcal{D}_C$  and  $\mathcal{CH}_C^t \subseteq \mathcal{CH}_C$ .  $\square$*

**Example 5.** *Fig.3.1 presents lattice  $\mathcal{C}^{t_5}$  for  $t_5$  in Table 3.3. For simplicity, we omit values on unbound dimension attributes (e.g.,  $\langle *, *, c_1 \rangle$  is represented as  $c_1$ ). Consider  $C = \langle a_1, *, c_1 \rangle$ .  $\mathcal{A}_C^{t_5} = \{\top, \langle a_1, *, * \rangle, \langle *, *, c_1 \rangle\}$ ,  $\mathcal{P}_C^{t_5} = \{\langle a_1, *, * \rangle, \langle *, *, c_1 \rangle\}$ ,  $\mathcal{CH}_C^{t_5} = \{\langle a_1, b_1, c_1 \rangle\}$  and  $\mathcal{D}_C^{t_5} = \{\langle a_1, b_1, c_1 \rangle\}$ .  $\square$*

**Definition 8** (Lattice Intersection). *Given  $t, t' \in R(\mathcal{D}; \mathcal{M})$ ,  $\mathcal{C}^{t,t'} = \mathcal{C}^t \cap \mathcal{C}^{t'}$  is the intersection of lattices  $\mathcal{C}^t$  and  $\mathcal{C}^{t'}$ .  $\mathcal{C}^{t,t'}$  is non-empty and is also a lattice. By Definition 7, the lattices for all tuples share the same top element  $\top$ . Hence  $\top$  is also the top element of  $\mathcal{C}^{t,t'}$ . Its bottom  $\perp(\mathcal{C}^{t,t'}) = \langle v_1, v_2, \dots, v_n \rangle$  where  $v_i = t.d_i$  if  $t.d_i = t'.d_i$  and  $v_i = *$  otherwise.  $\perp(\mathcal{C}^{t,t'})$  equals  $\top$  when  $t$  and  $t'$  do not have common attribute value.  $\square$*

**Example 6.** *Fig.3.4 shows  $\mathcal{C}^{t_4}$  and  $\mathcal{C}^{t_5}$  for  $t_4$  and  $t_5$  in Table 3.3. The constraints connected by solid lines represent the lattice intersection  $\mathcal{C}^{t_4, t_5}$ . Its bottom is  $\perp(\mathcal{C}^{t_4, t_5}) = \langle *, b_1, c_1 \rangle$ . In addition to  $\mathcal{C}^{t_4, t_5}$ ,  $\mathcal{C}^{t_4}$  and  $\mathcal{C}^{t_5}$  further include the constraints connected by dashed and dotted lines, respectively.  $\square$*

The algorithms we are going to propose consider the constraints in certain lattice order, compare  $t$  with skyline tuples associated with visited constraints, and use  $t$ 's dominating tuples to prune unvisited constraints from consideration—thereby

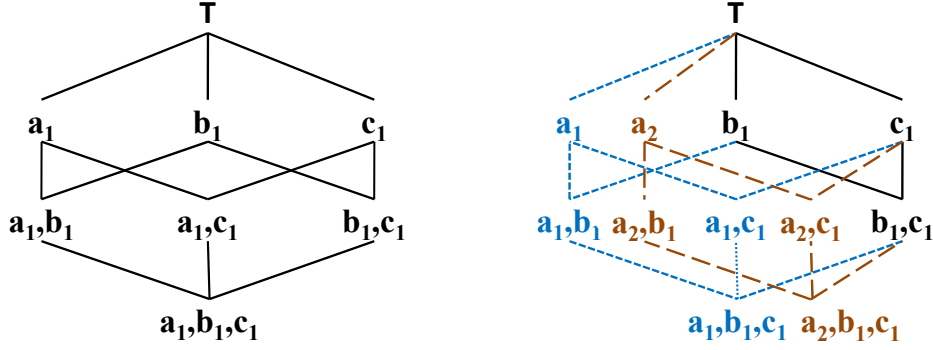


Figure 3.1. Lattice  $\mathcal{C}^{t_5}$

reducing cost. This idea of lattice-based pruning of constraints is justified by Propositions 2 and 3 below.

**Proposition 2.** *Given a tuple  $t$ , if  $t \notin \lambda_M(\sigma_C(R))$ , then  $t \notin \lambda_M(\sigma_{C'}(R))$ , for all  $C' \in \mathcal{A}_C$ .  $\square$*

If  $t \prec_M t'$ , then  $t \notin \lambda_M(\sigma_{\perp(C^{t,t'})}(R))$ . Hence, according to Proposition 2, we have the following Proposition 3.

**Proposition 3.** *Given two tuples  $t$  and  $t'$ , if  $t \prec_M t'$ , then  $t \notin \lambda_M(\sigma_C(R))$ , for all  $C \in \mathcal{C}^{t,t'}$ .  $\square$*

**(3) Sharing computation across measure subspaces** Given  $t$ , we need to consider not only all constraints satisfied by  $t$ , but also all possible measure subspaces. Sharing computation across measure subspaces is challenging because of anti-monotonicity of dominance relation—a skyline tuple under space  $M$  may or may not be a skyline tuple in another space  $M'$ , regardless of whether  $M'$  is a superspace or subspace of  $M$  [25]. We thus propose algorithms that first traverse the lattice in the full measure space, during which a frontier of constraints is formed for each measure subspace. Top-down (respectively, bottom-up) lattice traversal in a subspace com-



mences from (respectively, stops at) the corresponding frontier instead of the root, which in effect prunes some top constraints.

**Two Baseline Algorithms** We introduce two baseline algorithms `BaselineSeq` (Alg.3) and `Baselineldx`. They are not as naive as the brute-force Alg.2. Instead, they exploit Proposition 3 straightforwardly. Upon  $t$ 's arrival, for each subspace  $M$ , they identify existing tuples  $t'$  dominating  $t$ . `BaselineSeq` sequentially compares  $t$  with every existing tuple.  $S$  is initialized to be  $\mathcal{C}^t$  (Line 3). Whenever `BaselineSeq` encounters a  $t'$  that dominates  $t$ , it removes constraints in  $\mathcal{C}^{t,t'}$  from  $S$  (Line 5). By Proposition 3,  $t$  is not in the contextual skylines for those constraints. After  $t$  is compared with all tuples, the constraints having  $t$  in their skylines remain in  $S$ . The same is independently repeated for every  $M$ . The pseudo code of `Baselineldx` is similar to Alg.3 and thus omitted. Instead of comparing  $t$  with all tuples, `Baselineldx` directly finds tuples dominating  $t$  by a one-sided range query  $\bigwedge_{m_i \in M} (m_i \geq t.m_i)$  using a  $k$ -d tree [32] on full measure space  $\mathcal{M}$ .

### 3.5 Prototype System: FactWatcher

Figure 3.2 shows `FactWatcher`'s customized GUI for NBA (National Basketball Association) data, where new tuples—players' statistics in individual games—come into the database when a game is over. The GUI's structure is dataset-agnostic as long as the data table is modeled by  $R(\mathcal{D}; \mathcal{M})$  as given in Section 3.3. For instance, for data analytics of a social network service, suppose the dimension attributes are  $\mathcal{D}=\{user\ age, user\ city, post\ type, timestamp\}$  and the measure attributes are  $\mathcal{M}=\{number\ of\ likes, number\ of\ comments, number\ of\ shares\}$ . `FactWatcher` finds facts such as “no one in Dallas has posted a photo that gets as many likes, comments and shares.” The GUI consists of four areas, as follows.

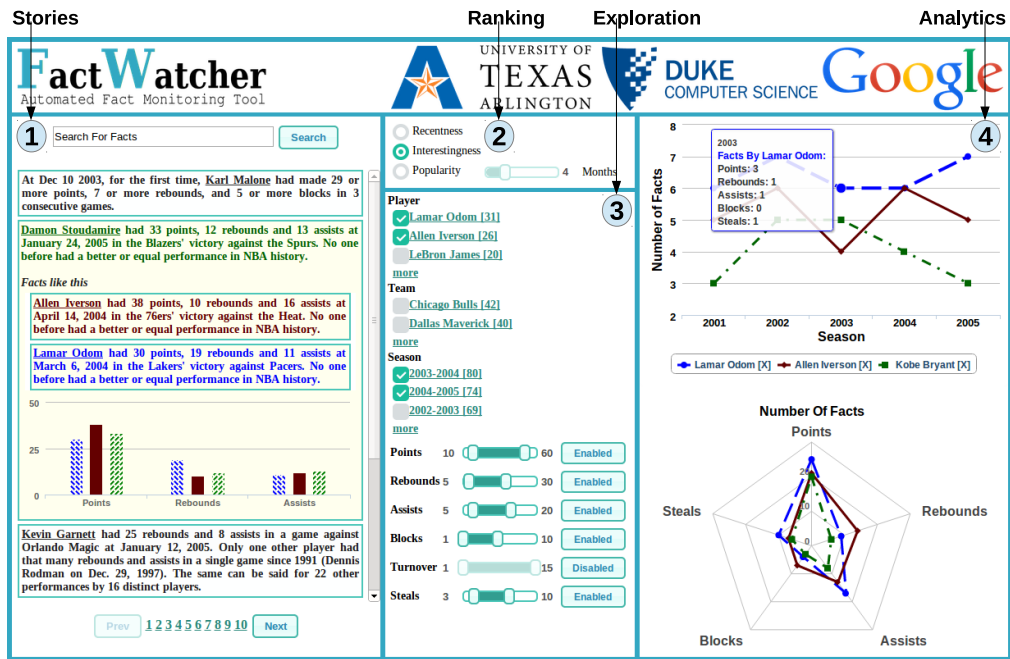


Figure 3.2. FactWatcher User Interface

$id$	$player$	$team$	$opposition\ team$	$pts$	$ast$	$reb$
$t_1$	Lamar Odom	Clippers	Nets	12	9	13
$t_2$	Lamar Odom	Clippers	Lakers	8	11	6
$t_3$	Lamar Odom	Clippers	Lakers	9	9	7
$t_4$	Eddie House	Heat	Nets	9	7	8
$t_5$	Lamar Odom	Heat	Nets	10	11	12
$t_6$	Eddie House	Clippers	Nets	10	11	10
$t_7$	Eddie House	Heat	Wizards	8	6	9
$t_8$	Lamar Odom	Clippers	Lakers	10	11	11

Table 3.4. A Data Table for the Running Example

**1. Stories** This area shows a ranked list of textual news leads (stories) translated from facts that have ever been discovered and are still valid. It also allows users search for translated stories by keywords. The translation is guided by a set of templates and rules. For example, if  $t_8$  in Table 3.4 triggers a situational fact with regard to constraint-measure pair  $(\langle *, Clippers, Lakers \rangle, \{pts, ast, reb\})$ , the story is “Lamar Odom had 10 points, 11 assists and 11 rebounds to become the first Clippers player with a 10/11/11 (points/ assists/ rebounds) game against the Lakers.”

If a story is clicked, FactWatcher shows below it more stories for the same constraint-measure pair  $(C, M)$  or grouping-measure pair  $(G, M)$ , as illustrated in Figure 3.2. It also presents bar charts to compare the stories by their values on  $M$ .

**2. Ranking** FactWatcher allows users to choose from several ways of ranking facts and their corresponding stories.

*Recentness* This default option simply orders facts by their triggering tuples’ timestamps.

*Popularity* This option ranks facts by the frequencies of facts appearing in search results within the last  $x$  months, where  $x$  can be controlled using a slider.

*Interestingness* This option ranks facts by the elapsed time since their last comparable facts were discovered. Suppose tuple  $t$  triggers a new situational fact  $f_1$  with regard to constraint-measure pair  $(C, M)$  and a new prominent streak  $f_2$  with regard to grouping-measure pair  $(G, M)$ . The interestingness of  $f_1$  ( $f_2$ ) is the elapsed time since the last fact was discovered in  $(C, M)$  ( $(G, M)$ ). The longer the elapsed time is, the more interesting a fact is, since a long elapsed time indicates the fact does not come by easily.

**3. Exploration** This area presents a faceted interface for exploring the stories. Each facet corresponds to a dimension or measure attribute. Under the facet for a dimension attribute, the attribute values are associated with and ordered by

numbers, which indicate how many facts involve the values. For instance, Figure 3.2 shows that there are 31 facts for such  $(C, M)$  that the constraint  $C$  has a conjunct  $player=Lamar\ Odom$ . **FactWatcher** places a checkbox beside each attribute value. A user can select/unselect the checkboxes across multiple facets. The selected values within one facet correspond to a disjunctive condition, and the disjuncts from different facets form a conjunctive condition. They together correspond to multiple constraints. Each fact (story) displayed in the “stories” area must satisfy one such constraint.

Beside the facet for a measure attribute, **FactWatcher** presents a sidebar and a button. A user can click the button to enable or disable a measure attribute. Accordingly the “stories” area displays such stories whose corresponding measure subspaces  $M$  only involve one or more enabled measure attributes. The user can also use the sidebar to set the minimum and maximum desired values on an enabled attribute  $m_i$ . Accordingly the displayed facts (stories) must have values on  $m_i$  within the range.

**4. Analytics** This area visualizes statistics on facts related to objects selected by a user. The “stories” area highlights the objects (values on object identification attributes) in stories, e.g., *Allen Iverson*, *Lamar Odom*, etc. in Figure 3.2. When a user clicks on an object, it is added to the object list in the middle of the “analytics” area. The user can remove an object by clicking the crossmark beside it. The top part of the “analytics” area is a line chart, which shows one line per selected object that represents the number of facts (among the displayed facts in the “stories” area) triggered by the object over each time period. When the user hovers the mouse on a data point, a pop-up box shows, for each measure attribute, the number of facts whose measure subspaces contain the measure attribute. The bottom part of this

area is a radar chart, which shows one polygon per selected object that represents how many facts triggered by the object are related to each measure attribute.

### 3.6 Usage Scenario of FactWatcher

We will demonstrate **FactWatcher** on several datasets, including an NBA dataset and a weather dataset. The NBA dataset has 317,371 tuples of NBA box scores from 1991-2004, on 8 dimension and 7 measure attributes. The weather dataset has 7.8 million daily weather forecast records for 5,365 locations of UK from Dec. 2011 to Nov. 2012. It has 7 dimension attributes and 7 measure attributes. When we explain the demonstration steps below, we refer to the GUI in Figure 3.2 and its corresponding NBA scenario.

**Stories** When a user visits **FactWatcher**, **FactWatcher** shows a list of stories in area “stories” of Figure 3.2. The user enters a keyword query in the search box. The list of stories will be updated. The faceted interface in area “exploration” and the line chart and radar chart in area “analytics” will change accordingly. The user then clicks a particular story. Similar stories will be shown below it, with bar charts to compare the stories.

**Ranking** By default, the stories are ordered by recentness. The user explores other ranking schemes by choosing the radio button for *interestingness* or *popularity* in area “ranking”. When *popularity* is chosen, the user further uses the sidebar beside it to control the period for assessing popularity of stories.

**Exploration** The user uses the faceted interface in area “exploration” to explore stories. The user checks *Lamar Odom*, *Allen Iverson* and some others under *player* and *2003-2004*, *2004-2005* under *season*. The area “stories” will show stories related to any of the selected players when they played for or against any team (as she did not select anything under *team*) during *2003-2004* or *2004-2005* season. The

user further uses the slidebars for measure attributes to adjust the ranges of values on these attributes. The area “stories” will only show those stories whose measure attribute values do not fall out of the ranges. If the user wants to exclude a measure attribute from the filtering criteria, she can click the button beside its sidebar to disable it, e.g., *Turnover* in Figure 3.2.

**Analytics** When the user reads the stories, she can click on any underlined objects, i.e., players. After a while, the user has clicked on multiple objects, which are shown in the box in the middle of area “analytics”. The top line chart and bottom radar chart in that area visualize the statistics on facts related to these objects, as described in Section 3.5. If the user is not interested in an object anymore, she can remove the object from comparison by clicking the “X” beside the object in the middle box.

## CHAPTER 4

### Expert Team Finding

#### 4.1 Introduction

The traditional *skyline tuple* problem has been extensively investigated in recent years [23, 33, 34, 35, 36, 37, 38]. Consider a database table of  $n$  tuples and  $m$  numeric attributes. The domain of each attribute has an application-specific preference order, with “better” values being preferred over “worse” values. A tuple  $t_1$  *dominates*  $t_2$  if and only if every attribute value of  $t_1$  is either better than or equal to the corresponding value of  $t_2$  according to the preference order and  $t_1$  has better value on at least one attribute. The set of skyline tuples are those tuples that are not dominated by any other tuples in the table.

In this chapter, we formulate and investigate the novel problem of computing *skyline groups*. In contrast to the skyline tuple problem which has been extensively investigated, the skyline groups problem surprisingly has not been studied in prior work. In this problem, we refer to any subset of  $k$  tuples in the table as a *k-tuple group*. Our objective is to find, for a given  $k$ , all  $k$ -tuple skyline groups, i.e.,  $k$ -tuple groups that are not *dominated* by any other  $k$ -tuple groups.

The notion of dominance between groups is analogous to the dominance relation between tuples in skyline analysis. The dominance relation between two groups of  $k$  tuples is defined by comparing their aggregates. To be more specific, we calculate for each group a single aggregate tuple, whose attribute values are aggregated over the corresponding attribute values of the tuples in the group. The groups are then compared by their aggregate tuples using traditional tuple dominance. While many ag-

	database	indexing
$t_1$	3	0
$t_2$	0	3
$t_3$	2	1
$t_4$	2	2
$t_5$	0	2

Table 4.1. Experts

aggregate functions can be considered in calculating aggregate tuples, we focus on three distinct functions that are commonly used in database applications—SUM (i.e., AVG, since groups are of equal size), MIN and MAX. Intuitively, SUM captures the collective strength of a group, while MIN/MAX compares groups by their weakest/strongest member on each attribute. Note that throughout the paper, we assume the larger the SUM/MIN/MAX values are, the better a group is. As a simple example, consider two 3-tuple groups— $G = \{\langle 0, 3 \rangle, \langle 2, 1 \rangle, \langle 2, 2 \rangle\}$  and  $G' = \{\langle 2, 1 \rangle, \langle 2, 2 \rangle, \langle 0, 2 \rangle\}$ . Their aggregate tuples under the function SUM are  $\text{SUM}(G) = \langle 4, 6 \rangle$  and  $\text{SUM}(G') = \langle 4, 5 \rangle$ . Hence  $G$  dominates  $G'$ .

The need for finding expert groups prevails in several application areas, including question answering, crowdsourcing, panel selection, project team formation, and so on. This is illustrated by the following motivating examples.

*CrowdSourcing* Consider forming a team of Wikipedia editors to write a new Wikipedia article related to “database” and “indexing”. Table 4.1 shows all relevant editors  $t_1, \dots, t_5$  and their expertise on the two topics. We want to assign the task to a team of 2 editors. Table 4.2 shows the aggregate vectors under AVG, MIN and MAX, for



team	<i>AVG</i>	<i>MIN</i>	<i>MAX</i>
$G_{1,2}$	$\langle 1.5, 1.5 \rangle$	$\langle 0, 0 \rangle$	$\langle \mathbf{3}, \mathbf{3} \rangle$
$G_{1,3}$	$\langle 2.5, 0.5 \rangle$	$\langle 2, 0 \rangle$	$\langle 3, 1 \rangle$
$G_{1,4}$	$\langle \mathbf{2.5}, \mathbf{1.0} \rangle$	$\langle 2, 0 \rangle$	$\langle 3, 2 \rangle$
$G_{1,5}$	$\langle 1.5, 1.0 \rangle$	$\langle 0, 0 \rangle$	$\langle 3, 2 \rangle$
$G_{2,3}$	$\langle 1.0, 2.0 \rangle$	$\langle 0, 1 \rangle$	$\langle 2, 3 \rangle$
$G_{2,4}$	$\langle \mathbf{1.0}, \mathbf{2.5} \rangle$	$\langle \mathbf{0}, \mathbf{2} \rangle$	$\langle 2, 3 \rangle$
$G_{2,5}$	$\langle 0, 2.5 \rangle$	$\langle \mathbf{0}, \mathbf{2} \rangle$	$\langle 0, 3 \rangle$
$G_{3,4}$	$\langle \mathbf{2.0}, \mathbf{1.5} \rangle$	$\langle \mathbf{2}, \mathbf{1} \rangle$	$\langle 2, 2 \rangle$
$G_{3,5}$	$\langle 1.0, 1.5 \rangle$	$\langle 0, 1 \rangle$	$\langle 2, 2 \rangle$
$G_{4,5}$	$\langle 1.0, 2.0 \rangle$	$\langle \mathbf{0}, \mathbf{2} \rangle$	$\langle 2, 2 \rangle$

Table 4.2. All possible 2-expert teams

all possible 2-expert teams where  $G_{i,j}$  stands for a team of experts  $t_i$  and  $t_j$ . A simple scheme such as picking top editors on individual topics does not work. For example,  $G_{1,2}$  consists of the top editor on each topic and has an aggregated vector  $\langle 1.5, 1.5 \rangle$  with regard to AVG.  $G_{3,4}$ , with vector  $\langle 2.0, 1.5 \rangle$ , dominates  $G_{1,2}$  (denoted  $G_{3,4} \succ G_{1,2}$ ) under AVG. Hence,  $G_{3,4}$  is a better team in terms of collective expertise. In fact,  $G_{3,4}$  is a 2-expert skyline team, since no other team dominates it under AVG. Table 4.2 highlights all 2-expert skyline teams for every aggregate function.

*Question Answering* Consider a question-answering platform such as [Quora.com](https://www.quora.com). A question is displayed to users who might answer it. The question asker can also explicitly solicit answers from certain users, oftentimes by offering rewards. To receive quality answers, it is necessary to intelligently post the question to users with proper expertise. More often than not, a question requires expertise on several aspects that

cannot be fulfilled by any single user, needing attention from a diverse team of experts who collectively excel. For instance, consider question “Is C or Python better for high-performance computing?” To get a comprehensive answer, we need experts in “high performance computing”, “C”, and so on.

*Other Motivating Applications* The need for finding expert teams arises in several other applications. **1)** Consider the task of choosing a panel of experts to evaluate a research paper or a grant proposal. An expert can be modeled as a tuple in the multi-dimensional space defined by the paper’s topics, to reflect the expert’s strength on these topics. The collective expertise of a panel is modeled as the aggregate vector of the corresponding tuples. **2)** Forming collaborative teams for a software development project can be viewed as finding programmers who are collectively strong in the multi-dimensional space of desired skills for the project. **3)** In a variety of applications we look for “teams” in more general sense, such as bundles of products, reviews, stocks, and so on. For instance, to summarize a product’s many customer reviews, choosing a set of diverse reviews is forming a “team” of reviews, where the reviews are modeled by attributes such as “sentiment”, “length”, “quality”, etc. Another example is online fantasy sports where gamers compete by forming and managing team rosters of real-world athletes, aiming at outperforming other gamers’ teams. The teams are compared by aggregated statistics (e.g., “points”, “rebounds”, “assists” in basketball games) of the athletes in real games.

The capability of recommending groups is valuable in the above-mentioned applications. An attractive property of skyline groups is that a skyline group cannot be dominated by any other group. In contrast, given a non-skyline group, there always exists a better group in the skyline. Hence the skyline groups present those groups that are worth recommending. They become the input to further process that ultimately recommends one group.

Recommending a few groups becomes non-trivial when there are many skyline groups. In addition to eyeballing skyline groups by browsing and visualization interface, such post-processing can also be automatic. One approach is to filter and rank skyline groups according to user preference. For instance, if groups are ranked by a *monotonic* scoring function on attributes  $A_1, \dots, A_m$ , regardless of the specific scoring function, the skyline always contains a group attaining the best score. Another automatic approach is to return a small number of representative skyline groups, by criteria proposed for skyline tuples [39, 40, 41, 42], since each group corresponds to an aggregate tuple. We do not further investigate such post-processing here. In Section 4.2, we provide a more detailed discussion of previous work on choosing from a large number skyline tuples.

To find  $k$ -tuple skyline groups in a table of  $n$  tuples, there can be  $\binom{n}{k}$  different candidate groups. *How do we compute the skyline groups of  $k$  tuples each from all possible groups?* Interestingly, the skyline groups problem is significantly different from the traditional skyline tuple problem, to the extent that algorithms for the later are quite inapplicable.

A simple solution is to first list all  $\binom{n}{k}$  groups, compute the aggregate tuple for each group, and then use any traditional skyline tuple algorithm to identify the skyline groups. The main problem with such an approach is the significant computational and storage overhead in creating this huge intermediate input for the traditional skyline tuple algorithm (i.e.,  $O(\binom{n}{k})$  aggregate tuples). The skyline groups problem also has another idiosyncrasy that is not shared by the skyline tuple problem. For certain aggregate functions, specifically MAX and MIN, even the output size—i.e., the number of skyline groups—while significantly smaller than  $\binom{n}{k}$ , may be nevertheless too large to explicitly compute. To address these two problems, we develop novel techniques, namely *output compression*, *input pruning*, and *search space pruning*.

For MAX and MIN aggregates, we observe that numerous groups may share the same aggregate tuple. Our approach to compressing the output is to list the distinct aggregate tuples, each representing possibly many skyline groups, and also provide enough additional information so that actual skyline groups can be reconstructed if required. Interestingly, there is a difference between MIN and MAX in this regard: while the compression for MIN is relatively efficient, the compression for MAX requires solving the NP-Hard Set Cover Problem (which fortunately is not a real issue in practice, as we shall show in the paper). Note that both output compression and the aforementioned post-processing are for tackling large output. Output compression is incorporated into the process of finding skyline groups and is performed before post-processing.

Our approach to input pruning is to filter input tuples and significantly reduce the input size to the search of skyline groups. The idea is that if a tuple  $t$  is dominated by  $k$  or more tuples, then we can safely exclude  $t$  from the input without influencing the distinct aggregate tuples found at the end. We also find that, for MAX, we can safely exclude any non-skyline tuple without influencing the results.

Our final ideas (perhaps, technically the most sophisticated of the paper) are on search space pruning. Instead of enumerating every  $k$ -tuple combination, we exclude from consideration many combinations. To enable such candidate pruning, we identify two properties inspired by the anti-monotonic property in the well-known *Apriori* algorithm for frequent itemset mining [43]. It is important to emphasize here that the anti-monotonic property in *Apriori* *does not hold* for skyline groups defined by SUM, MIN or MAX. More specifically, a subset of a skyline group may not necessarily be a skyline group itself. We identify two anti-monotonic properties with different applicability—while the *Order-Specific Anti-Monotonic Property* (OSM) applies to SUM, MIN and MAX, the *Weak Candidate-Generation Property* (WCM) applies to

MIN and MAX but not SUM. We develop a dynamic programming algorithm and an iterative algorithm to compute skyline groups, based on OSM and WCM, respectively. Our algorithms iteratively generate larger candidate groups from smaller ones and prune candidate groups by these properties.

We briefly summarize our contributions as follows.

- We motivate and formulate the novel problem of computing skyline groups, and discuss the inapplicability of traditional skyline tuple algorithms in solving this problem.
- We develop novel algorithmic techniques for output compression, input pruning, and search space pruning. In particular, for search space pruning, we identify interesting anti-monotonic properties to filter out candidate groups.
- We run comprehensive experiments on real and synthetic datasets to evaluate the proposed algorithms.

## 4.2 Related Work

Skyline query has been intensively studied over the last decade. Kung et al. [44] first proposed in-memory algorithms to tackle the skyline problem. Börzsönyi et al. [23] was the original work that studied how to process skyline queries in database systems. Since then, this line of research includes proposals of improved algorithms [33, 34], progressive skyline computation [35, 36, 37], query optimization [38], and the investigation of many variants of skyline queries [25, 27, 45, 46, 47, 48, 49, 50, 51, 52, 53].

With regard to the concept of skyline groups, the most related previous works are [54] and [55]. In [54] the groups are defined by GROUP BY in SQL, while the groups in our work are formed by combinations of  $k$  tuples in a tuple set. Zhang et

al. [55] studied set preferences where the preference relationships between  $k$ -subsets of tuples are based on features of  $k$ -subsets. The features are more general than numeric aggregate functions considered in our work. The preferences given on each individual feature form a partial order over the  $k$ -subsets instead of a total order by numeric values. Their general framework can model many different queries, including our skyline groups problem. The optimization techniques for that framework, namely the *superpreference* and *M-relation* ideas, when instantiated for our specific problem, are essentially equivalent to input pruning in our solution as well as merging identical tuples. Hence such an instantiation is a baseline solution to our problem. However, the important search space pruning properties (OSM and WCM) and output compression in Section 4.4 are specific to our problem and were not studied before. These ideas bring substantial performance improvement, as the comparison with the baseline in Section 4.6 shall demonstrate.

With regard to the problem of forming expert teams to solve tasks, the most related prior works are [56] and [57]. In [57] teams are ranked by a scoring function, while in our case groups are compared by skyline-based dominance relation. Hence the techniques proposed in [57] are not applicable to our setting. In [56], instead of measuring how well teams match tasks, the focus was on measuring if the members in a team can effectively collaborate with each other, based on information from social networks.

A large number of skyline points may exist in a given dataset, due to various reasons such as high dimensionality. Such large size of skyline hinders the usefulness of skyline to end users. Researchers have noticed this issue and various approaches are proposed to alleviate the problem. One direction is to perform skyline analysis in subspaces instead of the original full space [25, 58]. Another direction is to choose a small number of representative skyline points. The semantics and methods proposed

in various works on this line can be directly adopted for post-processing when there are many skyline groups, since each group corresponds to an aggregate tuple. Specifically, Chan et al. [39] propose to return only frequent points and they measure the frequency of a point by how often it is in the skyline of different subspaces. Lin et al. [40] select  $k$  most representative points such that the total number of data points dominated by the  $k$  points is maximized. Tao et al. [41] define representative skyline points differently, aiming at minimizing the maximal distance between non-representative skyline points and their closest representatives. Chan et al. [42] define  $k$ -dominant skyline as the points that are not dominated by any other points in any  $k$ -attribute subspace.

### 4.3 Skyline Groups Problem

In this section we formulate the skyline groups problem. Table 4.3 lists the major notations used in the paper. Consider a database table  $D$  of  $n$  tuples  $t_1, \dots, t_n$  and  $m$  attributes  $A_1, \dots, A_m$ . We refer to any subset of  $k$  tuples in the table, i.e.,  $G : \{t_{i_1}, \dots, t_{i_k}\} \subseteq D$ , as a  $k$ -tuple group. Our objective is to find the skyline of  $k$ -tuple groups. Whether a  $k$ -tuple group belongs to the skyline or not is determined by the dominance relation between this group and other  $k$ -tuple groups. The dominance test, when taking two groups  $G_1$  and  $G_2$  as input, produces one of three possible outputs— $G_1$  dominates  $G_2$ ,  $G_2$  dominates  $G_1$ , or neither dominates the other. A  $k$ -tuple group is a *skyline  $k$ -tuple group*, or *skyline group* in short, if and only if it is not dominated by any other  $k$ -tuple group in  $D$ . Note that a tuple  $t_i$  may be in multiple skyline groups.

Groups are compared by their aggregates. Each group is associated with an *aggregate vector*, i.e., an  $m$ -dimensional vector with the  $i$ -th element being an aggregate value of  $A_i$  over all  $k$  tuples in the group. While this definition allows general

$t$	a tuple with $m$ attributes $\{A_1, \dots, A_m\}$
$D$	a database table of $n$ tuples $\{t_1, \dots, t_n\}$
$G$	$\{t_{i_1}, \dots, t_{i_k}\} \subseteq D$ , a $k$ -tuple group
$G \succ G'$	$G$ dominates $G'$
$v$	an aggregate vector
$\mathcal{F}(G)$	aggregate vector of group $G$ under function $\mathcal{F}$
$T_n$	the first $n$ tuples
$Sky_k^n$	all $k$ -tuple skyline groups in $T_n$
$Sky_k$	all $k$ -tuple skyline groups in $D$
OSM	order-specific property
WCM	weak candidate-generation property

Table 4.3. Notations

aggregate functions, we focus on three commonly used functions—SUM (i.e, AVG, since groups are of equal size), MIN, and MAX. Functions such as MEDIAN and VARIANCE do not satisfy the properties in Section 4.4 and thus do not lend themselves to efficient algorithms proposed in the paper. The aggregate vectors for two groups are compared according to the traditional tuple dominance relation which is defined according to certain application-specific preferences. Such preferences are captured as a combination of total orders for all attributes, where each total order is defined over (all possible values of) an attribute, with “larger” values preferred over “smaller” values. Hence, an aggregate vector  $v_1$  dominates  $v_2$  if and only if every attribute value of  $v_1$  is larger than or equal to the corresponding value of  $v_2$  according to the preference order and  $v_1$  is larger than  $v_2$  on at least one attribute.



	$A_1$	$A_2$
$t_1$	3	0
$t_2$	0	3
$t_3$	2	1
$t_4$	2	2
$t_5$	0	2

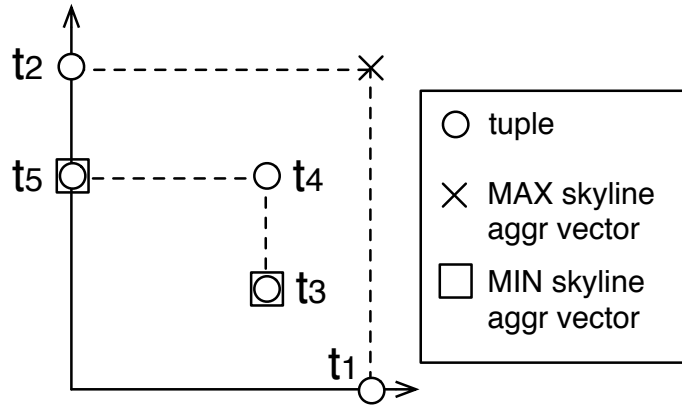


Table 4.4. Running example

Figure 4.1. Running example in 2-d space

	Tuples	SUM	MAX	MIN
$G$	$t_2\langle 0, 3 \rangle t_3\langle 2, 1 \rangle t_4\langle 2, 2 \rangle$	$\langle 4, 6 \rangle$	$\langle 2, 3 \rangle$	$\langle 0, 1 \rangle$
$G'$	$t_3\langle 2, 1 \rangle t_4\langle 2, 2 \rangle t_5\langle 0, 2 \rangle$	$\langle 4, 5 \rangle$	$\langle 2, 2 \rangle$	$\langle 0, 1 \rangle$
Dominance Relation		$G \succ G'$	$G \succ G'$	$G = G'$

Table 4.5. Examples of aggregate-based comparison

Table 4.4 depicts a 5-tuple, 2-attribute table which we shall use as a running example. Figure 4.1 depicts the tuples on a 2-dimensional plane defined by the two attributes. We consider the natural order of real numbers as the preference order. For instance,  $t_2$  dominates  $t_5$  while neither  $t_2$  nor  $t_3$  dominates each other. Table 4.5 shows a sample case of comparing two 3-tuple groups for each aggregate function. Figure 4.1 also shows the symbols corresponding to MIN and MAX aggregate vectors of skyline 2-tuple groups in the running example. For instance, the skyline 2-tuple group under MAX function is  $\{t_1, t_2\}$ , with aggregate vector  $\langle 3, 3 \rangle$ . The aggregate

vectors of skyline 2-tuple groups under MIN are  $\langle 2, 1 \rangle$  (for group  $\{t_3, t_4\}$ ) and  $\langle 0, 2 \rangle$  (for groups  $\{t_2, t_4\}$ ,  $\{t_2, t_5\}$ ,  $\{t_4, t_5\}$ ).

Our methods allow a mixture of different aggregate functions on different attributes. For example, if we use SUM on the first attribute and MAX on the second attribute, then for the two groups in Table 4.5, the aggregated vectors for  $G$  and  $G'$  are  $\langle 4, 3 \rangle$  and  $\langle 4, 2 \rangle$ , respectively. Our order-specific property (Section 4.4.3.1) can handle arbitrary mixture of SUM, MIN, and MAX, while the weak candidate-generation property (Section 4.4.3.2) handles any mixture of MIN and MAX. Section 4.6 presents the experimental results on such mixed functions.

#### 4.4 Finding Skyline Groups

In this section, we develop our main ideas for finding skyline groups. We start by considering a brute-force approach which first enumerates each possible combination of  $k$  tuples in the input table, computes the aggregate vector for each combination, and then invokes a traditional skyline-tuple-search algorithm to find all skyline groups. This approach has two main problems. One is its significant computational overhead, as the input size to the final step, i.e., skyline tuple search, is  $\binom{n}{k}$ , which can be extremely large.

The other problem is on the seemingly natural strategy of listing all skyline groups as the output. For certain aggregate functions (e.g., MAX and MIN), even the output size, i.e., the number of skyline groups produced, may be nevertheless too large to explicitly compute and store. Consider an extreme example under MAX. If a tuple  $t$  dominates all other tuples, then every  $k$ -tuple combination that contains  $t$  is a MAX skyline group—leading to a total of  $O(n^{k-1})$  skyline groups. Such a large output size not only leads to significant overhead in computing skyline groups, but also makes post-processing (e.g., ranking and browsing of skyline groups) costly.

Another idea is to consider skyline tuples only. While seemingly intuitive, this idea will not work correctly in general. In particular, we have the following two observations:

1) A group solely consisting of skyline tuples may *not* be a skyline group. Consider  $G=\{t_1, t_2\}$  in the running example. Note that both  $t_1$  and  $t_2$  are skyline tuples. Nonetheless, under SUM,  $G$  is dominated by  $G'=\{t_3, t_4\}$ , as  $\text{SUM}(G)=\langle 3, 3 \rangle$  while  $\text{SUM}(G')=\langle 4, 3 \rangle$ . As such,  $G$  is not on the skyline.

2) A group containing non-skyline tuples could be a skyline group. Again consider the running example, this time with  $G = \{t_4, t_5\}$  and MIN function. Note that  $t_5$  is not on the skyline as it is dominated by  $t_2$  and  $t_4$ . Nonetheless,  $G$  (with  $\text{MIN}(G) = \langle 0, 2 \rangle$ ) is actually on the skyline, because the only other groups which can reach  $A_2 \geq 2$  in the aggregate vector are  $\{t_2, t_4\}$  and  $\{t_2, t_5\}$ , both of which yield an aggregate vector of  $\langle 0, 2 \rangle$ , the same as  $\text{MIN}(G)$ . Thus,  $G$  is on the skyline despite containing a non-skyline tuple.

To address these challenges, we develop several techniques, namely *output compression*, *input pruning*, and *search space pruning*. We start with developing an *output compression* technique that significantly reduces the output size when the number of skyline groups is large, thereby enabling more efficient downstream processes that consume the skyline groups. Then, we consider how to efficiently find skyline groups. In particular, we shall describe two main ideas. One is *input pruning*—filtering the input tuples to significantly reduce the input size to the search of skyline groups. The other is *search space pruning*—instead of enumerating each and every  $k$ -tuple combination, we develop techniques to quickly exclude from consideration a large number of combinations. Note that the two types of pruning techniques are transparent to each other and therefore can be readily integrated.

#### 4.4.1 Output Compression for MIN and MAX

**Main Idea:** A key observation driving our design of output compression is that while the number of skyline groups may be large, many of them share the same aggregate vector. Thus, our main idea for compressing skyline groups is to store not all skyline groups, but only the (much fewer) distinct skyline aggregate vectors (in short *skyline vector*) as well as one skyline group for each skyline vector.

Among the three aggregate functions we consider in the paper, SUM rarely, if ever, requires output compression. The intuitive reason is that, for any attribute, the SUM aggregate of a skyline group is sensitive to all tuples in the group, while MIN (resp. MAX) aggregate is in general only sensitive to tuples with minimum (resp. maximum) values on certain attributes, making it much more likely for two groups to share the same MIN (resp. MAX) vector. In the rest of the paper, we shall focus on finding all skyline  $k$ -tuple groups for SUM, and finding all distinct skyline vectors and their accompanying (sample) skyline groups for MIN and MAX. We use the term “skyline search” to refer to the process in solving the problem.

**Reconstructing all Skyline Groups for a Skyline Vector:** While the distinct skyline vectors and their accompanying (sample) skyline groups may suffice in many cases, a user may be willing to spend time on investigating all groups equivalent to a particular skyline vector, and to choose a group after factoring in her knowledge and preference. Thus, we now discuss how one can reconstruct the skyline groups corresponding to a given skyline vector, if required.

Consider MIN first. For a given MIN skyline vector  $v$ , the process is as simple as finding  $\Omega(v)$ , the set of all input tuples which dominate or are equal to  $v$ . The reason is as follows. Given any  $k$ -tuple subset of  $\Omega(v)$ , its aggregate vector  $v'$  must be equal to  $v$ , because (1)  $v'$  cannot dominate  $v$  (otherwise  $v$  is not a skyline vector)

and (2)  $v'$  does not contain smaller value than  $v$  on any attribute, by definition of  $\Omega(v)$ . On the other hand, if a group contains a tuple outside of  $\Omega(v)$ , its aggregate vector must have smaller values than  $v$  on some attributes, and therefore cannot be in the skyline. The time complexity of a linear scan in finding  $\Omega(v)$  is  $O(n)$ . Given  $\Omega(v)$ , the only additional step is to enumerate all  $k$ -tuple subsets of  $\Omega(v)$ .

For MAX, interestingly, the problem is much harder. To understand why, consider each tuple as a set consisting of all attributes for which the tuple reaches the same value as a MAX skyline vector. The problem is now transformed to finding all combination of  $k$  tuples such that the union of their corresponding sets is the universal set of all attributes—i.e., finding all set covers of size  $k$ . For instance, in finding the 2-tuple skyline groups for a skyline vector  $v=\langle 4, 5, 6\rangle$ , consider two tuples  $t_1=\langle 3, 5, 2\rangle$  and  $t_2=\langle 4, 1, 6\rangle$ . The set for  $t_1$  is  $\{A_2\}$ , because  $t_1$  has the same value as  $v$  on attribute  $A_2$ . Similarly the set for  $t_2$  is  $\{A_1, A_3\}$ . Since the union of the two sets is  $\{A_1, A_2, A_3\}$ , the two tuples together is a set cover of size 2. The NP-hardness of this problem directly follows from the NP-completeness of SET-COVER, seemingly indicating that MAX skyline groups should not be compressed.

Fortunately, despite of the theoretical intractability, finding all skyline groups matching a MAX skyline vector  $v$  is usually efficient in practice. This is mainly because the number of tuples that “hit” the MAX attribute values in  $v$  is typically small. As such, even a brute-force enumeration can be efficient, as demonstrated by experimental results in Section 4.6. Nonetheless, when a large number of tuples “hit” the MAX attribute values, it is unclear how one can efficiently find and store all skyline groups - e.g., by using certain efficient indexing schemes. We leave the design of such indexing schemes as an open problem for future research.

Before algorithmic discussions, we make an important observation for the case of MAX when  $k \geq m$ , where  $k$  is the size of a skyline group and  $m$  is the number of

attributes. Since it takes at most  $m$  tuples to cover the MAX values of all attributes, there is only one distinct skyline vector—the vector that takes the MAX value on every attribute. In reconstructing skyline groups, for each skyline group, after finding tuples that cover the MAX values, the remaining tuples can be arbitrary.

#### 4.4.2 Input Pruning

We now consider the pruning of input to skyline groups searches, which is originally the set of all  $n$  tuples. An important observation is that if a tuple  $t$  is dominated by  $k$  or more tuples in the original table, then we can safely exclude  $t$  from the input without influencing the distinct skyline vectors found at the end. To understand why, suppose that a skyline group  $G$  contains a tuple  $t$  that is dominated by  $h$  ( $h \geq k$ ) tuples. There is always an input tuple  $t'$  that dominates  $t$  and is not in  $G$ . Since  $t'$  dominates  $t$ , there must be less than  $h$  tuples that dominate  $t'$ . Note that if  $t'$  is still dominated by  $k$  or more tuples, we can repeat this process until finding  $t' \notin G$  that is dominated by less than  $k$  tuples. Now consider the construction of another group  $G'$  by replacing  $t$  in  $G$  with  $t'$ . For SUM,  $G'$  always dominates  $G$ , contradicting our assumption that  $G$  is a skyline group. Thus, no skyline group under SUM can contain any tuple dominated by  $k$  or more tuples.

For MIN and MAX, it is possible that the aggregate vectors of the above  $G'$  and  $G$  are exactly the same. Even in this case, we can still safely exclude  $t$  from the input without influencing the distinct skyline vectors. If other tuples in  $G$  are dominated by  $k$  or more tuples, we can use the same process to remove them and finally reach a group that (1) features the same aggregate vector as  $G$ , and (2) has no tuple dominated by  $k$  or more other tuples. Thus, we can safely remove all tuples with at least  $k$  dominators for SUM, MIN and MAX.

Another observation for input pruning is that, for MAX only, we can safely exclude any non-skyline tuple  $t$  from the input without influencing the skyline vectors. The reason is as follows. Suppose a skyline group  $G$  contains a non-skyline tuple  $t$  that is dominated by a skyline tuple  $t'$ . If  $t' \notin G$ , then we can replace  $t$  in  $G$  with  $t'$  to achieve the same (skyline) aggregate vector (because  $G$  is a skyline group). If  $t' \in G$ , we can remove  $t$  from  $G$  without changing the aggregate vector of  $G$ . In either way,  $t$  can be safely excluded from the input. By repeatedly replacing or removing non-skyline tuples in the above way, we will obtain a group of size at most  $k$  that is formed solely by skyline tuples.<sup>1</sup> Padding the group with arbitrary additional tuples to reach size  $k$  will result in a group of the same aggregate vector as  $G$ .

#### 4.4.3 Search Space Pruning: Anti-Monotonicity

Our principal idea for search space pruning is to find and leverage two *anti-monotonic properties* for skyline search, somewhat in analogy to the Apriori algorithm for frequent itemset mining [43]. Nonetheless, it is important to note that the original anti-monotonic property in Apriori—every subset of a group “of interest” (e.g., a group of frequent items) must also be “of interest” itself—does not hold for skyline search over SUM, MIN or MAX. In fact, two examples in Section 4.3 can serve as proof by contradiction, for SUM and MIN. Specifically, for SUM, skyline 2-tuple group  $\{t_3, t_4\}$  contains a non-skyline tuple  $t_3$ , i.e., a non-skyline 1-tuple group. For MIN, skyline group  $\{t_4, t_5\}$  contains a non-skyline tuple  $t_5$ . For MAX, the inapplicability can be easily observed from the fact that the set of all tuples is always a skyline  $n$ -

---

<sup>1</sup>Note that if the resulting group has size smaller than  $k$ , then it (and thus  $G$ ) reaches the maximum values on all attributes. If there are fewer than  $k$  skyline tuples in the input, then we can immediately conclude that any skyline  $k$ -tuple group must reach the maximum values on all attributes.

tuple group, while many subsets of it are not on their corresponding skylines of equal group size. Thus, the key challenge is to find anti-monotonic properties that hold for skyline search. We stress that the main contribution here is not about *proving* these properties, but rather about *finding* the right ones that can effectively prune the search space.

#### 4.4.3.1 Order-Specific Anti-Monotonic Property

Our first idea is to make a revision to the classic property in the Apriori algorithm, by factoring in an order of all tuples. To understand how, consider aggregate function SUM and a skyline  $k$ -tuple group  $G_k$  which violates the Apriori property, i.e., a  $(k-1)$ -tuple subset of it,  $G_{k-1} \subset G_k$ , is not a skyline  $(k-1)$ -tuple group. We note for this case that all  $(k-1)$ -tuple groups which dominate  $G_{k-1}$  must contain tuple  $t_k = G_k \setminus G_{k-1}$ . To understand why, suppose that there exists a  $(k-1)$ -tuple group  $G'$  which dominates  $G_{k-1}$  but does not contain  $t_k$ . Then,  $G' \cup \{t_k\}$  would always dominate  $G_k = G_{k-1} \cup \{t_k\}$  under SUM, contradicting the skyline assumption for  $G_k$ . One can see from this example that while a subset of a skyline group may not be on the skyline for the entire input table, it is always a skyline group over a subset of the input table—in particular,  $D \setminus \{t_k\}$  in the above example. This observation can be extended to MIN and MAX, with a small tweak. That is, although  $G_{k-1}$  might be dominated by a  $(k-1)$ -tuple group  $G'$  not containing  $t_k$ , the aggregate vectors of  $G' \cup \{t_k\}$  and  $G_k$  must be equal. Therefore, considering  $G'$  and ignoring  $G_{k-1}$  will still lead us to the same skyline vector. If we require every subset  $G_{k-1}$  of a skyline group  $G_k$  to be a skyline group over table  $D \setminus \{t_k\}$ , where  $t_k = G_k \setminus G_{k-1}$ , we will not miss any skyline vector. This leads to the following property:

**Definition 9. Order-Specific Property** *An aggregate function  $\mathcal{F}$  satisfies the order-specific anti-monotonic property if  $\forall k$ , given a skyline  $k$ -tuple group  $G_k$  with*



aggregate vector  $v$  (i.e.,  $v = \mathcal{F}(G_k)$ ), for each tuple  $t$  in  $G_k$ , there must exist a set of  $(k - 1)$ -tuples  $G_{k-1} \subseteq D$  with  $t \notin G_{k-1}$ , such that (1)  $G_{k-1}$  is a skyline  $(k - 1)$ -tuple group over an input table  $D \setminus \{t\}$ , and (2)  $G_{k-1} \cup \{t\}$  is a skyline  $k$ -tuple group over the original input table  $D$  which satisfies  $\mathcal{F}(G_{k-1} \cup \{t\}) = v$ .

It may be puzzling from the definition where the order comes from. We note that it actually lies in the way search-space pruning can be done according to this property. Consider an arbitrary order of all tuples, say,  $\langle t_1, \dots, t_n \rangle$ . For any  $r < n$ , if we know that an  $h$ -tuple group  $G_h$  ( $h \leq r$ ) is *not* a skyline group over  $\{t_1, \dots, t_r\}$ , then we can safely prune from the search space all  $k$ -tuple groups whose intersection with  $\{t_1, \dots, t_r\}$  is  $G_h$ —a reduction of the search space size by  $O((n - r)^{k-h})$ . The reason is that, if the aggregate function satisfies the property in Definition 9, either (1) such groups are not skyline  $k$ -tuple groups or (2) the aggregate vectors of such groups are unchanged if we replace  $G_h$  by  $h$ -tuple groups that are subsets of  $\{t_1, \dots, t_r\}$  and dominate  $G_h$ . Such a pruning technique considers all tuples in a specific order—hence the name of order-specific anti-monotonic property.

**Theorem 1.** *SUM, MIN and MAX satisfy the order-specific anti-monotonic property.*

*Proof.* Suppose  $G_k$  is a  $k$ -tuple skyline group with aggregate vector  $v$  (i.e.,  $v = \mathcal{F}(G_k)$ ) and  $t \in G_k$ . Consider  $G_{k-1} = G_k \setminus \{t\}$ . (A) If  $G_{k-1}$  is a skyline  $(k - 1)$ -tuple group over  $D \setminus \{t\}$ , then  $G_{k-1}$  itself satisfies the two conditions in Definition 9; (B) Otherwise, by definition of skyline group, there must exist a skyline  $(k - 1)$ -tuple group over  $D \setminus \{t\}$ ,  $G'$ , such that  $G' \succ G_{k-1}$ . For SUM, only the above (A) is possible, i.e.,  $G_{k-1}$  must be a skyline  $(k - 1)$ -tuple group over  $D \setminus \{t\}$ . If (B) is possible, then  $G' \cup \{t\} \succ G_{k-1} \cup \{t\} = G_k$  by the concept of SUM, which contradicts with the assumption that  $G_k$  is a  $k$ -tuple skyline group. For MIN and MAX, for the above case (B), we prove that  $\mathcal{F}(G' \cup \{t\}) = \mathcal{F}(G_k)$ , i.e.,  $G'$  satisfies both condition (1) and

(2) in Definition 9. According to the semantics of MIN (MAX), if  $G' \succ G_{k-1}$ , then either  $G' \cup \{t\} \succ G_{k-1} \cup \{t\} = G_k$  or  $G' \cup \{t\} = G_{k-1} \cup \{t\} = G_k$ .  $G' \cup \{t\} \succ G_k$  would contradict with the assumption that  $G_k$  is a skyline group. Therefore  $G' \cup \{t\} = G_{k-1} \cup \{t\} = G_k$  and thus  $\mathcal{F}(G' \cup \{t\}) = \mathcal{F}(G_k)$ .  $\square$

We note a limitation of the order-specific property. To prune based on it, one has to compute for every  $h \in [k, n - k]$  the aggregate vectors of skyline 1, 2,  $\dots$ ,  $\min(k, h)$ -tuple groups over the first  $h$  tuples (by the order), because any of these groups may grow into a skyline  $k$ -tuple group when latter tuples (again, by the order) are brought into consideration. Given a large  $n$ , the order-specific pruning process may incur a significant overhead, as we shall show in Section 4.6.

#### 4.4.3.2 Weak Candidate-Generation Property

We now describe an order-free anti-monotonic property which “loosens” the classic Apriori property to one that holds for skyline search. The main idea is that, instead of requiring *every*  $(k - 1)$ -tuple subset of a skyline  $k$ -tuple group to be a skyline  $(k - 1)$ -tuple group, we consider the following property which only requires *at least one* subset to be on the skyline.

**Definition 10. (Weak Candidate-Generation Property)** *An aggregate function  $\mathcal{F}$  satisfies the weak candidate-generation property if,  $\forall k$  and for any aggregate vector  $v_k$  of a skyline  $k$ -tuple group, there must exist an aggregate vector  $v_{k-1}$  for a skyline  $(k-1)$ -tuple group, such that for any  $(k-1)$ -tuple group  $G_{k-1}$  which reaches  $v_{k-1}$  (i.e.,  $\mathcal{F}(G_{k-1}) = v_{k-1}$ ), there must exist an input tuple  $t \notin G_{k-1}$  which makes  $G_{k-1} \cup \{t\}$  a skyline  $k$ -tuple group that reaches  $v_k$  (i.e.,  $\mathcal{F}(G_{k-1} \cup \{t\}) = v_k$ ).*

An intuitive way to understand the definition is to consider the case where every skyline group has a distinct aggregate vector. In this case, the weak anti-monotonic property holds when every skyline  $k$ -tuple group has at least one  $(k-1)$ -tuple subset

being a skyline  $(k-1)$ -tuple group. The property is clearly “weaker” than the classic (Apriori) anti-monotonic property when being used for pruning, in the sense that it allows more candidate sets to be generated than directly (and mistakenly) applying the classic property.

In general, this property avoids the pitfall of order-specific property by removing the requirement of enumerating all tuples in order and generating skyline groups for each subset of tuples along the way. However, its limitation is that it only holds for MIN and MAX, but not for SUM.

**Theorem 2.** *MIN and MAX satisfy the weak candidate-generation property.*

*Proof.* We prove the theorem for MAX. The proof for MIN is similar. Suppose  $G_k$  is a skyline  $k$ -tuple group with  $\mathcal{F}(G_k)=v_k$ . Consider an arbitrary tuple  $t_1 \in G_k$  and the corresponding  $(k-1)$ -tuple subset of  $G_k$ ,  $G=G_k \setminus \{t_1\}$ .

If  $G$  is a skyline  $(k-1)$ -tuple group in  $D$ , then for any  $G'$  (including  $G$  itself) such that  $\mathcal{F}(G')=\mathcal{F}(G)$ , there are two possible cases to consider: (A)  $t_1 \notin G'$  and (B)  $t_1 \in G'$ . In Case (A),  $\mathcal{F}(G' \cup \{t_1\})=\mathcal{F}(G \cup \{t_1\})=\mathcal{F}(G_k)$ . In Case (B), note that since  $G'$  and  $G$  are of equal size, there must exist at least one tuple  $t_2 \in G$  and  $t_2 \notin G'$ . Consider  $G' \cup \{t_2\}$ . Since  $t_2 \in G$  and  $\mathcal{F}(G')=\mathcal{F}(G)$ , we have that  $\mathcal{F}(G' \cup \{t_2\})=\mathcal{F}(G \cup \{t_2\})=\mathcal{F}(G)$ . Furthermore, since  $t_1 \in G'$ , under MAX,  $\mathcal{F}(G' \cup \{t_2\})=\mathcal{F}(G \cup \{t_1\})=\mathcal{F}(G_k)$ . If  $G$  is not a skyline  $(k-1)$ -tuple group in  $D$ , consider a skyline  $(k-1)$ -group  $G'' \succ G$ . The same analysis above applies to  $G'$  instead of  $G$ .

In all cases, we always find a skyline  $(k-1)$ -tuple group and an extra tuple such that the aggregate vector of their union equals the original  $v_k$  under MAX. Therefore MAX satisfies the weak candidate-generation property in Definition 10.  $\square$

**Theorem 3.** *SUM does not satisfy the weak candidate-generation property.*

---

$t_1$	$\langle -131,-40,-4,-4,-98,-20,4,4,-69,-49,-9,-49,-9,54,-59,16,20,20,-107,-22,27,-22,27,61,-39,13,17,13,17,68,-12,-12,89,59,82,35,29,29,46,51,40,51,40,55,27,56,20,56,20,40,37,37,103,44,104,53,47,53,47,42,85,85,78,76,64,64,90,50,106 \rangle$
$t_2$	$\langle -40,-79,-38,-38,-80,-66,-52,-52,-85,-59,-67,-59,-67,-54,14,-47,-15,-15,-56,0,-41,0,-41,1,-76,-18,-52,-18,-52,-22,-63,-63,18,-52,3,-50,-32,-32,-60,-11,-47,-11,-47,-26,-67,-34,-51,-34,-51,-38,-59,-59,-22,-51,-18,-4,-32,-4,-32,-21,-17,-17,7,-27,-39,-39,-10,-39,-31 \rangle$
$t_3$	$\langle -49,50,-28,-28,51,33,10,10,64,15,35,15,35,20,-102,39,-44,-44,39,-79,14,-79,14,-65,81,-22,28,-22,28,-13,58,58,-51,44,-63,15,-24,-24,62,-52,8,-52,8,-31,57,-1,12,-1,12,-8,45,45,-7,19,6,-56,-8,-56,-8,-35,-9,-9,-68,-10,22,22,-30,5,25 \rangle$
$t_4$	$\langle 15,-23,-34,-34,-9,-42,-49,-49,-15,-16,-39,-16,-39,-52,-24,-58,-55,-55,13,-27,-47,-27,-47,-57,-28,-46,-54,-46,-54,-71,-29,-29,-48,-59,-67,-60,-57,-57,-41,-52,-55,-52,-55,-59,-53,-62,-54,-62,-54,-61,-50,-50,-68,-57,-75,-62,-63,-62,-63,-61,-63,-63,-63,-67,-64,-64,-72,-64,-70 \rangle$
$t_5$	$\langle 67,39,75,-94,68,22,52,-62,58,145,57,-97,-32,-42,22,11,39,-84,86,94,82,-106,-107,-58,50,111,47,-144,-53,-50,130,-87,-77,-29,-42,-8,13,-54,8,51,28,-129,-66,-41,7,39,20,-105,-33,-27,58,-75,-69,-22,-34,18,14,-95,-62,-32,51,-139,-61,-45,35,-89,-60,-27,-55 \rangle$
$t_6$	$\langle 67,39,-94,75,68,22,-62,52,58,-97,-32,145,57,-42,22,11,-84,39,86,-106,-107,94,82,-58,50,-144,-53,111,47,-50,-87,130,-77,-29,-42,-8,-54,13,8,-129,-66,51,28,-41,7,-105,-33,39,20,-27,-75,58,-69,-22,-34,-95,-62,18,14,-32,-139,51,-61,-45,-89,35,-60,-27,-55 \rangle$
$t_7$	$\langle 94,-82,44,44,-25,-47,-3,-3,-83,12,-50,12,-50,-11,147,-90,56,56,-66,119,-40,119,-40,84,-122,46,-46,46,-46,20,-86,-86,90,-75,72,-40,30,30,-124,69,-26,69,-26,38,-91,7,-22,7,-22,12,-71,-71,17,-34,-57,70,-5,70,-5,43,-13,-13,87,-5,-47,-47,38,-17,-54 \rangle$
$t_8$	$\langle -28,93,75,75,21,95,95,95,68,46,101,46,101,123,-23,115,79,79,1,19,107,19,107,87,80,55,110,55,110,114,84,84,51,136,52,112,91,91,97,69,112,69,112,101,109,96,104,96,104,104,111,111,111,119,104,73,107,73,107,93,100,100,77,119,114,114,101,115,130 \rangle$

---

Table 4.6. Counter-example for proving Theorem 3

We would like to note that while the only proof needed here is one counter-example, our study showed that finding such a counter-example is non-trivial. In

particular, the weak candidate-generation property indeed holds when  $k \leq 3$ , but fails when  $k \geq 4$ . For  $k=4$ , we constructed through MATLAB an 8-tuple, 69-attribute table as a counter-example, as shown in Table 4.6. With this counter-example,  $\{t_1, t_2, t_3, t_4\}$  is a skyline group for SUM, whereas none of  $\{t_1, t_2, t_3\}$ ,  $\{t_1, t_2, t_4\}$ ,  $\{t_1, t_3, t_4\}$ , or  $\{t_2, t_3, t_4\}$  is on the 3-tuple skyline.

## 4.5 Algorithms

### 4.5.1 Dynamic Programming Algorithm Based on Order-Specific Property

Consider an arbitrary<sup>2</sup> order of the  $n$  tuples in the input table, denoted by  $t_1, \dots, t_n$ . Let  $T_r$  be the set of the first  $r$  according to this order, i.e.,  $T_r = \{t_1, \dots, t_r\}$ . Let  $Sky_k^r$  be set of all skyline  $k$ -tuple groups with regard to  $T_r$ , i.e., each group in  $Sky_k^r$  is not dominated by any other  $k$ -tuple group consisting solely of tuples in  $T_r$ . One can see that our original problem can be considered as finding  $Sky_k^n$ . We now develop a dynamic programming algorithm which finds  $Sky_k^n$  by recursively solving the “smaller” problems of finding  $Sky_k^{n-1}$  and  $Sky_{k-1}^{n-1}$ , etc.

For ease of presentation, we assume aggregate function SUM in all the propositions, algorithms, and explanations in this section. At the end of the section, we shall explain why the idea is also applicable for MIN and MAX. The algorithm is based on the following idea—All skyline  $k$ -tuple groups in  $Sky_k^n$  can be partitioned into two disjoint sets  $S_1$  and  $S_2$  ( $Sky_k^n \equiv S_1 \cup S_2$  and  $S_1 \cap S_2 = \emptyset$ ) according to whether a group contains  $t_n$  or not. In particular,  $S_1 = \{G | G \in Sky_k^n, t_n \notin G\}$  and  $S_2 = \{G | G \in Sky_k^n, t_n \in G\}$ . One can see that  $S_1 \subseteq Sky_k^{n-1}$ . On the other hand,  $S_2$  is subsumed by a set of groups that can be expanded from  $Sky_{k-1}^{n-1}$ , the skyline  $(k-1)$ -tuple groups with regard to  $T_{n-1}$ . More specifically, given a skyline  $k$ -tuple

---

<sup>2</sup>We consider a random order in the experimental studies and leave the problem of finding an optimal order (in terms of efficiency) to future work.

group that contains  $t_n$ , if we remove  $t_n$  from it, then the resulting group belongs to  $Sky_{k-1}^{n-1}$ . These two properties are formally presented as follows. proof. Note that Proposition 5 can be directly derived from Theorem 1.

**Proposition 4.** *Given  $G \in Sky_k^n$ , if  $t_n \notin G$ , then  $G \in Sky_k^{n-1}$ .*

*Proof.* We prove this by contradiction. Assume  $G \notin Sky_k^{n-1}$ . Then, there must be a  $k$ -tuple group  $G' \in Sky_k^{n-1}$  such that  $G' \succ G$ . There are two possible cases. (A)  $G' \in Sky_k^n$ : It contradicts with  $G \in Sky_k^n$ . (B)  $G' \notin Sky_k^n$ : There must exist a  $k$ -tuple group  $G'' \in Sky_k^n$  such that  $G'' \succ G'$ . By transitivity of dominance relationship,  $G'' \succ G$ . This also contradicts with  $G \in Sky_k^n$ . Hence  $G \in Sky_k^{n-1}$ .  $\square$

**Proposition 5.** *Under aggregate function SUM, given  $G \in Sky_k^n$ , if  $t_n \in G$ , then  $G \setminus \{t_n\} \in Sky_{k-1}^{n-1}$ .*

We further explain the dynamic programming algorithm by referring to the outline in Algorithm 4. The idea is also illustrated in Figure 4.2. The function  $sky\_group(k, n)$  is for finding  $Sky_k^n$ . It first recursively computes  $Sky_{k-1}^{n-1}$  (Line 7). By adding  $t_n$  into each group in  $Sky_{k-1}^{n-1}$  (Line 8-10), the algorithm obtains a superset of the aforementioned  $S_2$ , according to Proposition 5. We denote this superset  $S_2^+$ . By recursively calling the  $sky\_group$  function (Line 12), it further computes  $Sky_k^{n-1}$ , which is a superset of the aforementioned  $S_1$ , according to Proposition 4. We also denote  $Sky_k^{n-1}$  by  $S_1^+$ .  $S_1^+$  and  $S_2^+$  thus contain all necessary candidate groups for  $Sky_k^n$ . Thus, the skyline over candidate groups ( $C_k^n = S_1^+ \cup S_2^+$ , Line 15) is guaranteed to be equal to  $Sky_k^n$ . Existing skyline query algorithms (e.g., [23, 33, 34]) can be applied over  $C_k^n$ . We use  $skyline()$  to refer to such algorithms (Line 16). The number of candidate groups considered ( $|S_1^+ \cup S_2^+|$ ) can potentially be much smaller than the number of all possible groups formed by all tuples, i.e.,  $\binom{n}{k}$ .

Note that  $Sky_k^n$  is needed in calculating both  $Sky_k^{n+1}$  and  $Sky_{k+1}^{n+1}$ . The algorithm recursively calls  $sky\_group(k, n)$  inside  $sky\_group(k, n+1)$ , to compute and

---

**Algorithm 4:**  $sky\_group(k, n)$ : Dynamic programming algorithm based on order-specific property

---

**Input:**  $n$ : input tuples  $T_n = \{t_1, \dots, t_n\}$ ;  $k$ : group size;  $k \leq n$

**Output:**  $Sky_k^n$ : skyline  $k$ -tuple groups among  $T_n$

```

1 if  $Sky_k^n$  is computed then
2   | return  $Sky_k^n$ ;
3 if  $k == 1$  then
4   |  $S_2^+ \leftarrow \{\{t_n\}\}$ ;
5 else
6   |  $S_2^+ \leftarrow \emptyset$ ;
7   |  $Sky_{k-1}^{n-1} \leftarrow sky\_group(k-1, n-1)$ ;
8   | foreach group  $G \in Sky_{k-1}^{n-1}$  do
9     |   |  $candidate\_group \leftarrow G \cup \{t_n\}$ ;
10    |   |  $S_2^+ \leftarrow S_2^+ \cup \{candidate\_group\}$ ;
11 if  $k < n$  then
12   |  $Sky_k^{n-1}$  (i.e.,  $S_1^+$ )  $\leftarrow sky\_group(k, n-1)$ ;
13 else
14   |  $S_1^+ \leftarrow \emptyset$ ;
15  $C_k^n \leftarrow S_1^+ \cup S_2^+$ ;
16  $Sky_k^n \leftarrow skyline(C_k^n)$ ;
17 return  $Sky_k^n$ ;

```

---

memoize  $Sky_k^n$ . Later it calls  $sky\_group(k, n)$  again inside  $sky\_group(k + 1, n+1)$ . This time  $Sky_k^n$  is not recomputed. Instead, the stored result is directly used (Line

1). Hence it is a dynamic programming algorithm. The sequence of calculating  $Sky_1^1$ , ...,  $Sky_k^n$  is shown by the dashed directed lines in Figure 4.2(b).

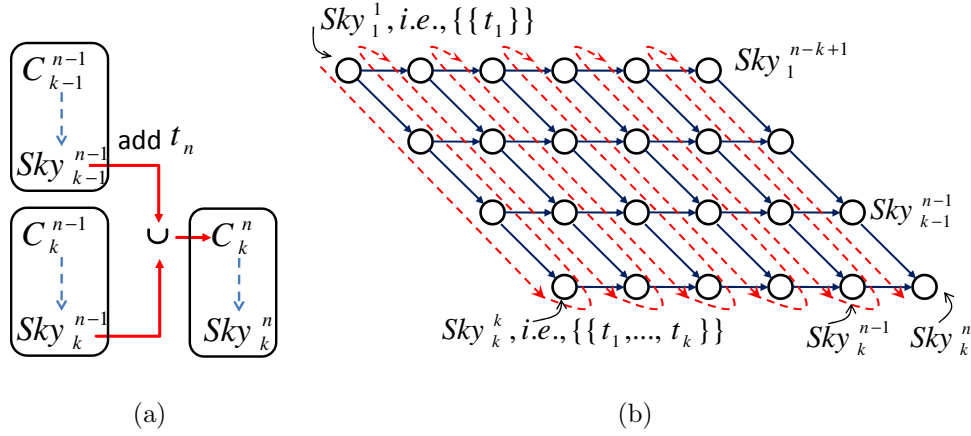


Figure 4.2. (a) Calculate  $Sky_k^n$  from  $Sky_{k-1}^{n-1}$  and  $Sky_k^{n-1}$ ; (b) Dynamic programming algorithm for calculating  $Sky_k^n$

Our discussion in this section so far assumed SUM. For MIN and MAX, Proposition 5 requires a small modification, as shown in the following Proposition 6.

**Proposition 6.** *Under aggregate function MIN and MAX, given  $G \in Sky_k^n$ , if  $t_n \in G$ , then there exists a group  $G' \in Sky_{k-1}^{n-1}$  such that  $F(G' \cup \{t_n\}) = F(G)$ .*

The implication of the applicability of Proposition 6 (instead of Proposition 5) for MIN and MAX is that, if we still apply Algorithm 4, the  $S_2^+$  produced by Line 8-10 is not guaranteed to be a superset of the aforementioned  $S_2$ . In other words, Line 16, which applies the skyline operation over candidate groups, cannot guarantee to produce  $Sky_k^n$ . However, the algorithm can still guarantee that the result of it contains all distinct aggregate vectors in  $Sky_k^n$ , based on Proposition 6. Note that our goal is to find all distinct skyline vectors and their accompanying (sample) skyline groups for MIN and MAX. Hence the algorithm suffices for our goal without change.



## 4.5.2 Iterative Algorithm Based on Weak

### Candidate-Generation Property

The weak candidate-generation property (Definition 10) can be summarized as follows. Consider the scenario when every skyline group has a distinct aggregate vector. Given a skyline group  $G$  and any  $i$ , at least one  $i$ -tuple sub-group of  $G$  must be a skyline  $i$ -tuple group. Based on this property, Algorithm 5 iteratively generates candidate  $i$ -tuple groups by adding new tuples into skyline  $(i - 1)$ -tuple groups (Line 6-12) and applies skyline algorithm over these candidates to find skyline  $i$ -tuple groups (Line 14). At every step of iteration, the algorithm only needs to generate  $i$ -tuple candidates by extending skyline  $(i - 1)$ -tuple groups instead of all  $(i - 1)$ -tuple groups. Hence it effectively prunes candidate groups by generation.

In reality, multiple skyline groups can have the same aggregate vector. The aforementioned statement is not true anymore. That is, given a skyline group  $G$  and any  $i$ , it is possible that none of its  $i$ -tuple sub-groups is a skyline  $i$ -tuple group. However, by Definition 10 and Theorem 2, a slightly different statement can be made for MIN and MAX—Given a skyline  $k$ -tuple group  $G_k$  and any  $i$ , there exists at least a skyline  $i$ -tuple group  $G_i$  that, when padded with other  $k - i$  tuples, will result in a skyline  $k$ -tuple group  $G'_k$  such that  $F(G'_k) = F(G_k)$ . Furthermore, given any skyline  $i$ -tuple group  $G'_i$  such that  $F(G'_i) = F(G_i)$ , we can pad  $G'_i$  with  $k - i$  other tuples to get a skyline  $k$ -tuple group that has the same aggregate vector as  $G_k$ . Therefore, although Algorithm 5 does not produce all skyline groups, it guarantees to find all distinct skyline vectors.

### **On Feasibility of Combining Order-Specific and Weak Candidate-Generation**

**Properties:** The order-specific and weak candidate-generation properties cannot be meaningfully combined. The candidate and skyline groups generated in Algorithm 5 are with respect to all  $n$  tuples, for different group size  $k$ . However, the candidate

---

**Algorithm 5:** *sky\_group*( $k, n$ ): Iterative algorithm based on weak candidate-generation property

---

**Input:**  $n$ : input tuples  $T_n = \{t_1, \dots, t_n\}$ ;  $k$ : group size;  $k \leq n$

**Output:**  $Sky_k$ : skyline  $k$ -tuple groups among  $T_n$

```

1  $C_1 \leftarrow T_n$ ;
2  $Sky_1 \leftarrow skyline(C_1)$ ;
3 for  $i \leftarrow 2$  to  $k$  do
4     //generate candidate  $i$ -tuple groups  $C_i$  from skyline  $i-1$ -tuple groups
    $Sky_{i-1}$ .
5      $C_i \leftarrow \emptyset$ ;
6     foreach  $G \in Sky_{i-1}$  do
7         foreach  $t \in T_n$  do
8             //generate candidate group
9             if  $t \notin G$  then
10                 $G' \leftarrow G \cup \{t\}$ ;
11                if  $G' \notin C_i$  then
12                     $C_i \leftarrow C_i \cup \{G'\}$ ;
13     //generate skyline  $i$ -tuple groups  $Sky_i$  based on candidates  $C_i$ 
14      $Sky_i \leftarrow skyline(C_i)$ ;
15 return  $Sky_k$ 

```

---

and skyline groups in Algorithm 4 are with respect to the first  $i$  ( $i=1..n$ ) tuples by a particular order. The combination is possible at the last step of Algorithm 4. We can take the intersection of  $C_k^n$  from Algorithm 4 and  $C_k$  from Algorithm 5 and then invoke  $skyline(C_k^n \cap C_k)$ . Even for this last step, the cost saving in  $skyline()$  due to

less candidates may not make up for the extra cost in producing both candidate sets  $C_k^n$  and  $C_k$ .

**Complexity Analysis:** The worst-case complexity of both Algorithms 4 and 5 is  $O(\binom{n}{k})$ , which is as poor as the complexity of the brute-force approach of enumerating all possible groups as candidates. We note that similarly the worst-case complexity of frequent itemset mining algorithms [43] is also exponential and equally poor as that of a brute-force approach. For both problems, it is the characteristics of real datasets that enables the algorithms to prune many candidates and thus to achieve better efficiency in reality. Specifically, one can see from Algorithms 4 and 5 that a critical factor determining the average-case complexity of these algorithms is the number of unique  $i$ -tuple skyline vectors in a  $j$ -tuple subset of the database (where  $i \in [1, k]$  and  $j \in [1, n]$ ) - which in turn depends on the underlying data distribution. For example, the number of unique skyline vectors tends to be small when values of different attributes are positively correlated: In the extreme-case scenario where all attributes share the same value, the number of unique skyline vector is always 1 for all  $i$  and  $j$ . On the other hand, there tends to be a large number of unique skyline vectors when the attributes are independently distributed. We shall evaluate the efficiency of Algorithms 4 and 5 over real-world datasets in the experiments section.

#### 4.5.3 From Distinct Vectors to Equivalent Groups

For MIN and MAX, even the output size - i.e., the number of skyline groups produced - may be too large to explicitly compute and store. As discussed in Section 4.4.1, for output compression, we only need to retain one representative skyline group for each distinct aggregated vector. To be more specific, it is sufficient for  $Sky_k^n$  in Algorithm 4 and  $Sky_k$  in Algorithm 5 to contain one representative group for each distinct aggregated vector of  $k$ -tuple groups. It can be easily achieved by a simple

modification of the skyline algorithm at Line 16 of Algorithm 4 and Line 14 of Algorithm 5. Whenever a candidate group is compared with current groups in the skyline, we prune it if it is equivalent to some existing group. This will further reduce the size of candidate groups and the number of group comparisons in succeeding iterations.

For input pruning, in the case of SUM and MIN, we remove all tuples dominated by at least  $k$  others. In the case of MAX, we remove all tuples not on the skyline. We showed in Section 4.4.2 that such input pruning techniques are safe - i.e., we will still obtain all distinct vectors and their representatives.

As discussed in Section 4.4.1, although in many cases distinct vectors and their representative groups suffice, a user may request all skyline groups equivalent to a particular aggregated vector, for applying further criteria in choosing a group. To return such equivalent groups, various postprocessing steps are required, due to output compression and input pruning. Below we discuss such postprocessing for individual functions.

Note that the same Algorithm 4 and 5 work if we do not apply output compression and input pruning. However, even if our application is to ultimately find all skyline groups, it is still beneficial to apply these two techniques and use postprocessing steps to find all skyline groups. Output compression and input pruning together not only reduce the output size, but also save computational cost by allowing the algorithms to deal with smaller input and intermediate results. In Section 4.6 we present experimental results to compare the execution time of our methods with and without  $k$ -dominator tuple pruning. The results verify the benefit of applying this pruning technique regardless of the ultimate output—representative groups for all distinct aggregated vectors or all skyline groups.

**SUM:** No postprocessing is necessary for SUM. First, a  $k$ -dominator tuple cannot appear in any skyline  $k$ -tuple group, as discussed in Section 4.4.2. Thus, input pruning

will not trigger postprocessing for SUM. Second, if the ultimate goal is to fetch all skyline groups, output compression should not be applied, because there is no effective way of reconstructing skyline groups from distinct aggregated vectors. In Line 16 of Algorithm 4, all skyline  $i$ -tuple groups should be retained, without applying the aforementioned simple modification that removes equivalent groups. Note that SUM only satisfies the order-specific property. Thus, only Algorithm 4 applies.

---

**Algorithm 6:** Finding skyline groups with identical aggregated vectors

(MIN function)

---

**Input:** input tuples  $R$ ;  $k$ : group size;  $k < |R|$

**Output:**  $Sky$ : skyline  $k$ -tuple groups for  $R$

```

1  $Sky \leftarrow \emptyset$ ;
2  $T \leftarrow$  remove  $k$ -dominator tuples from  $R$ ;
3  $n \leftarrow |T|$ ; /* number the tuples in  $T$  as  $t_1, \dots, t_n$  */
4  $Sky_k \leftarrow sky\_group(k, n)$ ; /* Algorithm 4 or Algorithm 5 */
5 foreach skyline  $k$ -tuple group  $G \in Sky_k$  do
6    $R_G \leftarrow$  the set of tuples in  $R$  that dominate or are equivalent to the
   aggregated vector of  $G$ ;
7   foreach  $k$ -combination  $G'$  of tuples in  $R_G$  do
8      $Sky \leftarrow Sky \cup \{G'\}$ ;
9 return  $Sky$ ;
```

---

**MIN:** Two factors contribute to the need for postprocessing. First, the pruned  $k$ -dominator tuples may appear in skyline groups. Second, the aforementioned equivalent group removal performed at Line 16 of Algorithm 4 and Line 14 of Algorithm 5

---

**Algorithm 7:** Finding skyline groups with identical aggregated vectors

---

(MAX function)

---

**Input:** input tuples  $R$ ;  $k$ : group size;  $k < |R|$   
**Output:**  $Sky$ : skyline  $k$ -tuple groups among  $R$

```
1  $Sky \leftarrow \emptyset$ ;  
2  $T \leftarrow$  remove  $k$ -dominator tuples from  $R$ ;  
3  $n \leftarrow |T|$ ; /* number the tuples in  $T$  as  $t_1, \dots, t_n$  */  
4  $Sky_k \leftarrow sky\_group(k, n)$ ; /* Algorithm 4 or Algorithm 5 */  
5 foreach skyline  $k$ -tuple group  $G \in Sky_k$  do  
6    $v \leftarrow$  the aggregated vector of  $G$   
7    $candidate\_group \leftarrow \emptyset$ ;  
8    $i \leftarrow 1$ ;  
9    $p[1] \leftarrow null$ ;  
10  while  $i > 0$  do  
11    /* Note that it is fine to select a tuple multiple times because a tuple can get the same value  
12    as  $v$  on multiple dimensions. */  
13     $candidate\_group \leftarrow candidate\_group \setminus \{p[i]\}$ ;  
14     $p[i] \leftarrow$  get the next tuple in  $R$  that has  $v$ 's value on the  $i$ th dimension;  
15    if  $p[i] == null$  then  $i \leftarrow i-1$ ; continue;  
16     $candidate\_group \leftarrow candidate\_group \cup \{p[i]\}$ ;  
17    if  $|candidate\_group| > k$  then continue;  
18    if  $i == d$  then  
19      /*  $d$  is the number of dimensions. */  $k' \leftarrow k - |candidate\_group|$ ;  
20      if  $k' == 0$  then  
21         $Sky \leftarrow Sky \cup \{candidate\_group\}$ ;  
22      else  
23         $R' \leftarrow R \setminus candidate\_group$ ;  
24        foreach  $k'$ -tuple combination  $G'$  among the tuples in  $R'$  do  
25           $Sky \leftarrow Sky \cup \{candidate\_group \cup G'\}$ ;  
26      else  
27         $i \leftarrow i + 1$ ;  
28         $p[i] \leftarrow null$ ;  
29  return  $Sky$ ;
```

---

will only keep one representative for each distinct aggregated vector. Note that both algorithms are applicable to MIN since MIN satisfies both order-specific and weak

candidate-generation properties. At the end of both algorithms, we obtain  $Sky_k$ , which contains representatives of all distinct aggregated vectors, but not necessarily all skyline  $k$ -tuple groups. To generate all skyline groups from  $Sky_k$  for MIN, we follow Algorithm 6. For each representative group, we find all the tuples that dominate or are equal to its aggregated vector. Any  $k$ -combination of these tuples is a skyline  $k$ -tuple group. This is based on the results from Section 4.4.1.

**MAX:** Algorithms 4 and 5 are both applicable to MAX. Similar to MIN, MAX needs postprocessing due to both input pruning and output compression. We thus devise Algorithm 7 to produce all skyline groups from representative groups.

For each representative group  $G$  that is found by Algorithms 4 and 5, Algorithm 7 uses a backtracking process to find all skyline groups that are equivalent to  $G$ . Denote the aggregated vector for  $G$  as  $v$ . On each dimension, we maintain a list of tuples from  $R$  (all input tuples to be considered) that attain  $v$ 's value on that dimension. We use the backtracking algorithm to enumerate all possible groups of the tuples from these lists, such that the groups have the same aggregated vector  $v$  and have less than or equal to  $k$  tuples. If a group has less than  $k$  tuples, it means there can be some “free” tuples. Any combination of other tuples will complement this group to form a skyline  $k$ -tuple group (Line 25-27).

A special case for MAX function is when there is only one distinct aggregated vector, i.e., all skyline  $k$ -tuple groups reach the highest possible value on every dimension. In Algorithms 4 and 5, whenever an  $i$ -tuple candidate group ( $i \leq k$ ) is generated, we test if this group attains the highest possible value on every attribute. If so, we have already found the aggregated vector for all skyline groups. Using that vector, we either find one representative group or all skyline groups, by a backtracking process that is essentially the same as Algorithm 7. We omit the details.

## 4.6 Experiments

The algorithms were implemented in C++. We executed all experiments on a Dell PowerEdge 2900 III server running Linux kernel 2.6.27-7, with dual quad-core Xeon 2.0GHz processors, 2x6MB cache, 8GB RAM, and three 250GB SATA HDs in RAID5.

**Datasets:** We collected 512 tuples of NBA players who played in the 2009 regular season [59]. The tuple of each player has 5 statistics (i.e., 5 attributes) that measure the player’s performance. The statistics are points per game (PPG), rebounds per game (RPG), assists per game (APG), steals per game (SPG), and blocks per game (BPG). Players and groups of players are compared by these statistics and their aggregates.

Another dataset is a collection of 35000 tuples that represent stocks for all the publicly traded firms as of December 31<sup>st</sup>, 2009 in several international markets [60]. Each tuple has 4 attributes, which are market capital (MC), stock price (SP), interest coverage ratio (ICR) and net income (NI). All the values were converted to US dollars.

To study the scalability of our methods, we also experimented with synthetic datasets produced by the data generator in [23]. The datasets have 1 to 10 million tuples, on 5 attributes. The data generator allows us to produce datasets where the attributes are correlated, independent, and anti-correlated. In independent datasets, the attribute values of a tuple were generated by a uniform distribution. In correlated datasets, attribute values were generated using normal distributions. Anti-correlated datasets were generated by a more complex procedure, which involves adding and subtracting values from otherwise uniformly distributed attribute values.

**Aggregate Functions and Methods Compared:** We investigated the performance of the two algorithms discussed in Section 4.5, namely the algorithms based on order-specific property (OSM) and weak candidate-generation property (WCM).



We also compared these methods with the baseline method (BASELINE), which is a direct adaptation of the general framework in [55] for our skyline groups problem. (The detailed discussion of [55] is in Section 4.2.) We executed these methods for the aggregate functions discussed in previous sections—SUM, MIN, and MAX.

**Parameters:** We ran our experiments under combinations of two parameter values, which are number of tuples, i.e., dataset size ( $n$ ) and number of tuples per group, i.e., group size ( $k$ ).

**Values Measured:** For each applicable combination of aggregate function, method, and parameter values, we measured the execution time needed to find all distinct aggregate vectors and their representative groups, as well as the time to find all skyline groups. Besides execution time, we also measured the total number of candidate groups generated and number of pairwise group (aggregated vector) comparisons in the process. Due to the iterative nature of OSM and WCM, they call the basic skyline function multiple times. Hence, the total number of generated candidate groups is the cumulative sizes of inputs to all skyline function invocations. Furthermore, OSM produces candidate groups by merging two disjoint sets of smaller groups. Here input size was calculated as the summation of the sizes of disjoint sets.

#### 4.6.1 Study of Different Aggregate Functions

**Size of Output under Different Functions:** Table 4.7 shows, for different  $n$ ,  $k$ , and aggregate functions, the number of all possible groups (G), the number of all skyline groups (S), and the number of distinct aggregate vectors (V) for the skyline groups. The table is for correlated synthetic datasets. The observations made on the NBA dataset were similar. It can be seen that G quickly becomes very large, which indicates that any exhaustive method will suffer due to the large space of possible answers. We want to point out that the number of skyline vectors (V) can be large

$n$		$k = 2$			$k = 4$			$k = 6$		
		G	S	V	G	S	V	G	S	V
1 M	SUM		247	247		1654	1654		6146	6146
	MIN	$4 \times 10^{11}$	187	141	$4 \times 10^{22}$	1914	436	$1 \times 10^{33}$	12816	870
	MAX		368	220		147	73		2.9 M	1
4 M	SUM		219	219		1610	1610		7482	7482
	MIN	$8 \times 10^{12}$	179	131	$1 \times 10^{25}$	2182	461	$6 \times 10^{36}$	17784	1148
	MAX		396	274		164	78		11 M	1
7 M	SUM		221	221		1374	1374		5825	5825
	MIN	$2 \times 10^{13}$	188	134	$1 \times 10^{26}$	2193	455	$2 \times 10^{38}$	16347	1002
	MAX		552	323		354	90		55 M	1
10 M	SUM		210	210		1300	1300		4487	4487
	MIN	$4 \times 10^{13}$	183	133	$4 \times 10^{26}$	2130	450	$1 \times 10^{39}$	15442	913
	MAX		402	224		968	63		0.8 B	1

Table 4.7. Number of all groups (G), skyline groups (S), distinct skyline group vectors (V), under various  $n$ ,  $k$ , and functions. Correlated synthetic dataset. M: million, B: billion

(e.g., under  $k=6$ ). As discussed in Section 4.1, these distinct vectors become the input to further post-processing such as filtering, ranking and browsing. When a particular skyline vector is chosen by a user, the corresponding equivalent skyline groups are generated upon request.

Among the three functions, in general SUM has the largest number of skyline vectors and MAX results in the smallest output size (V). This is due to the intrinsic

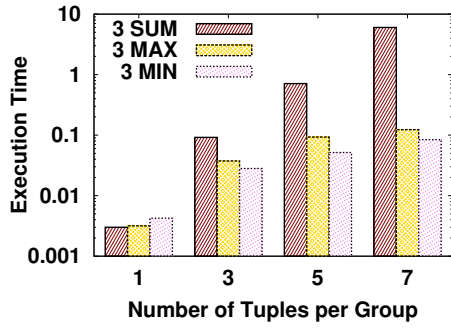
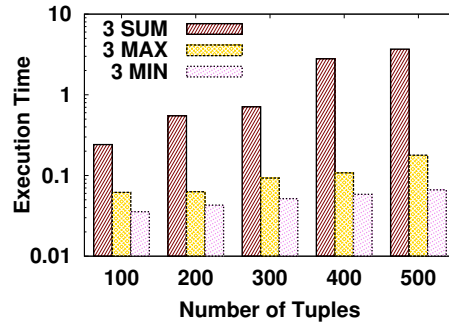
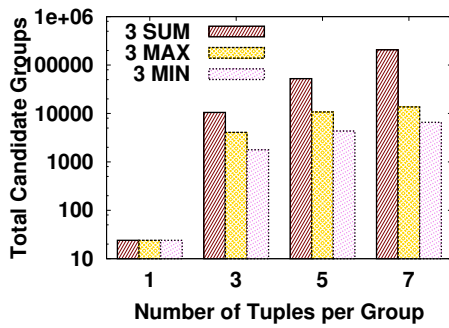
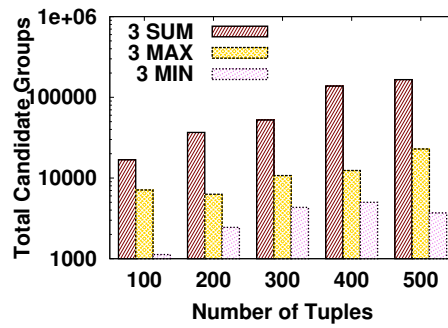
(a)  $n = 300$ (b)  $k = 3$ (c)  $n = 300$ (d)  $k = 3$ 

Figure 4.3. (a)-(b): Execution time (seconds, log scale) and (c)-(d): number of candidate groups (log scale), mixture of SUM/MAX/MIN

characteristics of these functions. In computing the aggregate vector for a group, SUM reflects the strength of all group members on each dimension. Hence it is more difficult for a group to dominate or equal to another group on every dimension. In contrast, MIN (MAX) chooses the lowest (highest) value among group members on each dimension. Hence skyline groups are formed by relatively small number of extremal tuples.

On the other hand, if we compare the sizes of all skyline groups including the equivalent ones, it is rare under SUM to have multiple skyline groups sharing the same aggregate vector. MAX results in much more equivalent groups. Moreover,

						PPG	RBG	APG	SPG	BPG
G1	Carmelo Anthony	Kobe Bryant	Kevin Durant	LeBron James	Dwyane Wade	283.2	63.4	52.2	15.2	7.6
G2	Andrew Bogut	Marcus Camby	Monta Ellis	Dwight Howard	Josh Smith	166.2	96.4	32.2	13.4	19.4
G3	Trevor Ariza	Monta Ellis	Dwyane Wade	Dwight Howard	Josh Smith	202	72.6	43.2	16.6	14
G4	Carlos Boozer	Baron Davis	LeBron James	Rajon Rondo	Chris Paul	193.8	61.2	80.6	17.6	4.8
G5	Andrew Bogut	LeBron James	Chris Paul	Dwight Howard	Jason Kidd	185.8	81	64	14	13.8

PPG:Point Per Game, RBG: ReBound per Game, APG: Assist Per Game, SPG:

Steal Per Game, BPG: Block Per Game

Table 4.8. Sample skyline groups from 512 players, 5 players per group

under MAX, when group size  $k$  is larger than or equal to the number of attributes (5 for the datasets), all skyline groups have the same aggregate vector that attains the highest value on every attribute.

**Dealing with a Mixture of Aggregate Functions:** Our methods allow a mixture of different aggregate functions applied on different attributes. OSM can handle arbitrary mixture of SUM, MIN, and MAX, while WCM can handle any mixture of MIN and MAX. Figure 4.3 shows the execution time of OSM over the 5-attribute NBA dataset, for 3 different mixtures of functions. For example, 3SUM means SUM function on the first 3 of the 5 attributes, and MIN and MAX on the remaining 2 attributes. From Figure 4.3 we can see that SUM function is typically more expensive. This is because output compression has less effect on SUM, under which it is more difficult for a group to dominate other groups.

#### 4.6.2 Experiments on NBA Dataset

**Sample Resultant Skyline Groups:** Table 4.8 shows several sample skyline 5-tuple groups under aggregate function SUM, from the NBA dataset. We see the sample groups are formed by elite players with different strengths. For instance, G1

is excellent in scoring (PPG), G2 excels in defense (RBG and BPG), and G3 is a very balanced group that is strong on many aspects although not the best on any dimension.

**Comparison of Various Methods:** Figure 4.4-4.6 show the execution time and number of generated candidate groups, by BASELINE/OSM/WCM under all applicable functions, over the NBA dataset. Figure 4.7 further shows the number of pairwise group (aggregate vector) comparisons performed by these algorithms under MIN and MAX. In sub-figure (a) and (c) of these figures, we fix the size of dataset ( $n$ ) to 300 tuples and vary group size ( $k$ ). In sub-figure (b) and (d) of these figures, we fix the group size ( $k=5$  for SUM/MIN and  $k=3$  for MAX) and vary dataset size. We observed that OSM/WCM performed substantially (often orders of magnitude in execution time) better than BASELINE. Without the properties, BASELINE produced much more candidate groups than OSM/WCM and thus incurred much more pairwise group (aggregate vector) comparisons inside skyline function invocations.

**Effect of Input Pruning:** Input pruning was applied in all the experiments for Figure 4.4-4.6. It had a good impact on the performance of all algorithms, since it significantly reduced the size of input. Table 4.9 shows that, in all considered cases on NBA dataset, less than 100 tuples remained after  $k$ -dominator tuple pruning was applied. Figure 4.8 shows that substantial saving on execution time was achieved for all functions.

**Search Space Pruning Power of OSM and WCM:** Figure 4.5, 4.6 and 4.7 compare OSM and WCM, in terms of execution time, number of candidate groups produced, and number of pairwise group (aggregate vector) comparisons incurred. We observed that, in terms of execution time, OSM performed better than WCM on the NBA dataset under both MIN and MAX. Although WCM demonstrated better pruning power in most cases as it resulted in less candidate groups (Fig-

$n$	$k = 1$	$k = 3$	$k = 5$	$k = 7$
100	19	31	37	44
200	22	37	47	57
300	24	50	61	67
400	29	62	78	86
500	30	62	83	94

Table 4.9. Number of tuples dominated by  $< k$  tuples in NBA

ures 4.5(c), 4.5(d), 4.6(c), and 4.6(d)), WCM required more pairwise group comparisons than OSM (Figure 4.7). Hence it lost in comparison with OSM.

**Effect of Output Compression:** Figure 4.9 shows the cost (in execution time) of post-processing for obtaining all skyline groups from distinct skyline vectors, on the NBA dataset, for  $n = 100$ , MAX function, and OSM algorithm. We can see that in this configuration finding all skyline groups only doubled the execution time. This verifies that, even though the problem of finding all skyline groups from distinct vectors is an NP-hard problem, in practice it is usually efficient due to the small number of tuples that can “hit” MAX attribute values, as explained in Section 4.4.1. As  $n$  increases, naturally the cost of post-processing will also increase. However, in reality we may only need to produce the equivalent groups for a skyline vector chosen by the user, instead of for all skyline vectors.

#### 4.6.3 Experiments on Stock Dataset

We also experimented on the Stock dataset. As the behavior of our algorithms on this dataset is mostly similar to that on the NBA dataset, we do not present extensive results. Figure 4.10 shows the performance of OSM and WCM for group

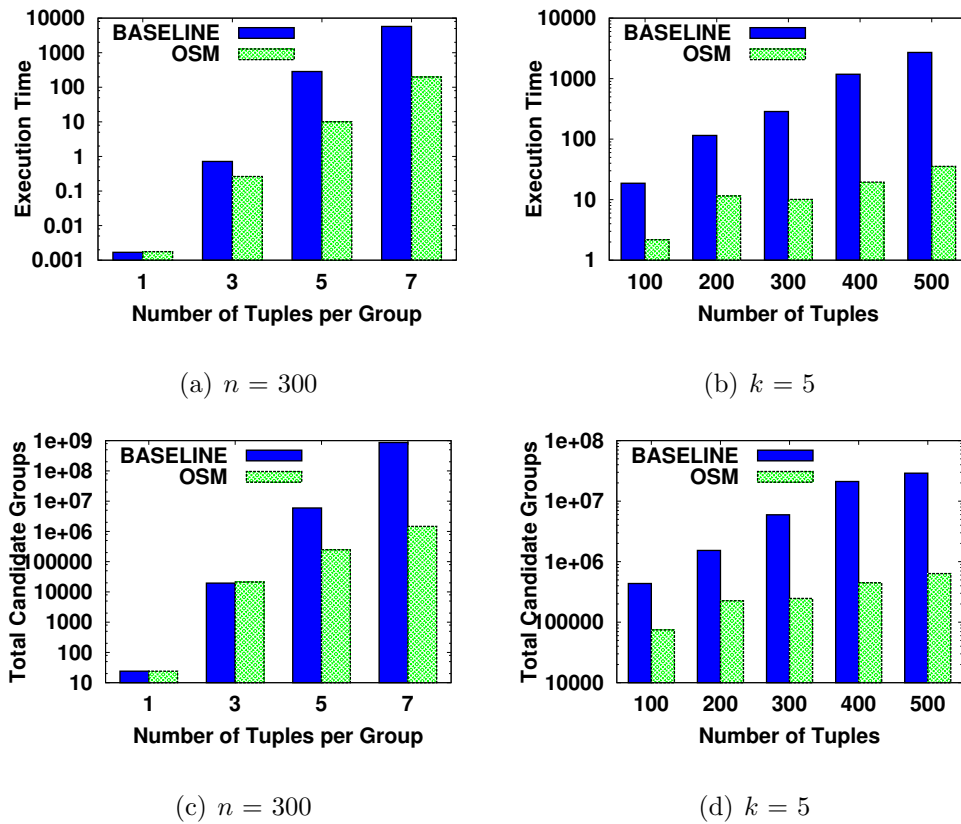


Figure 4.4. (a)-(b): Execution time (seconds, logarithmic scale) and (c)-(d): number of candidate groups (logarithmic scale), SUM

size  $k = 3$  under various input sizes. It is observed that, although the stock dataset is much bigger than the NBA dataset, the execution time is still considerably small. This is due to the effective input pruning. Table 4.10 shows that only less than 300 tuples remained after  $k$ -dominator tuple pruning was applied. We also see that, in this dataset, WCM took less execution time than OSM for MIN function. This is partly due to the overhead of OSM in performing candidate generation and skyline comparison for multiple (group size, table size) combinations, as mentioned in Section 4.4.3.1.

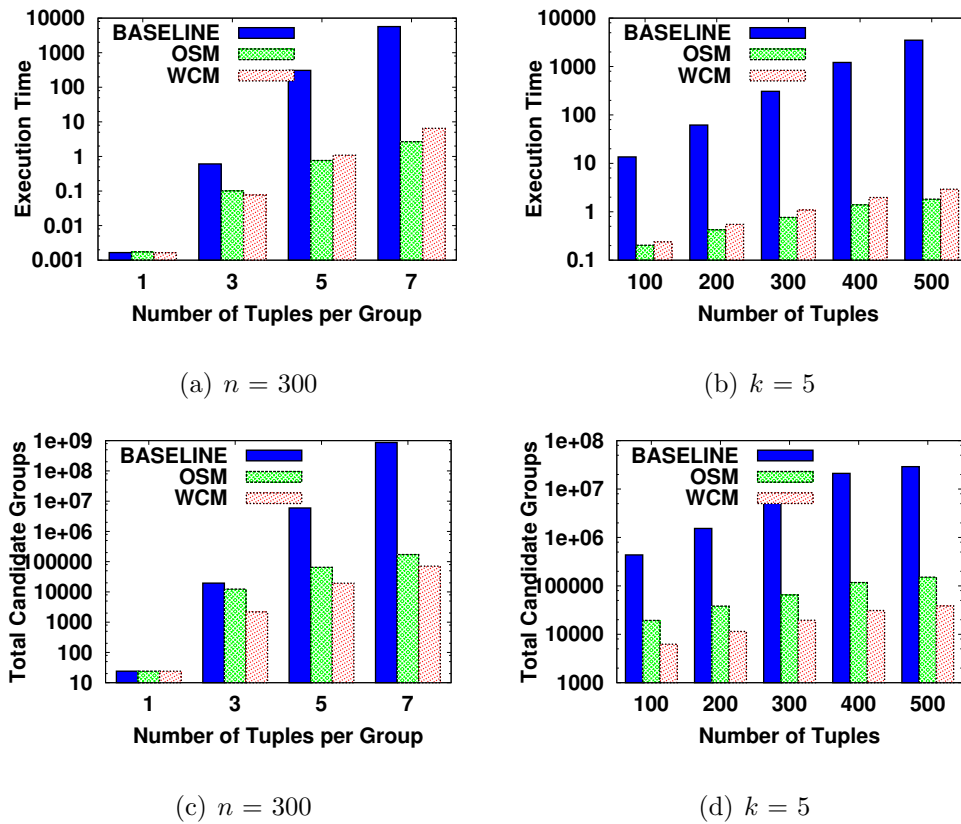


Figure 4.5. (a)-(b): Execution time (seconds, logarithmic scale) and (c)-(d): number of candidate groups (logarithmic scale), MIN

#### 4.6.4 Experiments on Synthetic Datasets

To show the scalability of our methods, we experimented on the synthetic datasets with 1 to 10 million tuples. In Figure 4.11, we see that OSM/WCM can finish within a minute on these large datasets, for  $k=4$  and all 3 functions.

The same methods will not be as efficient on independent or even anti-correlated data. Figures 4.12 and 4.13 show the performance of OSM/WCM on three different datasets of equal cardinality, under different number of attributes. We see that the execution time on anti-correlated and independent data increases quickly and soon the



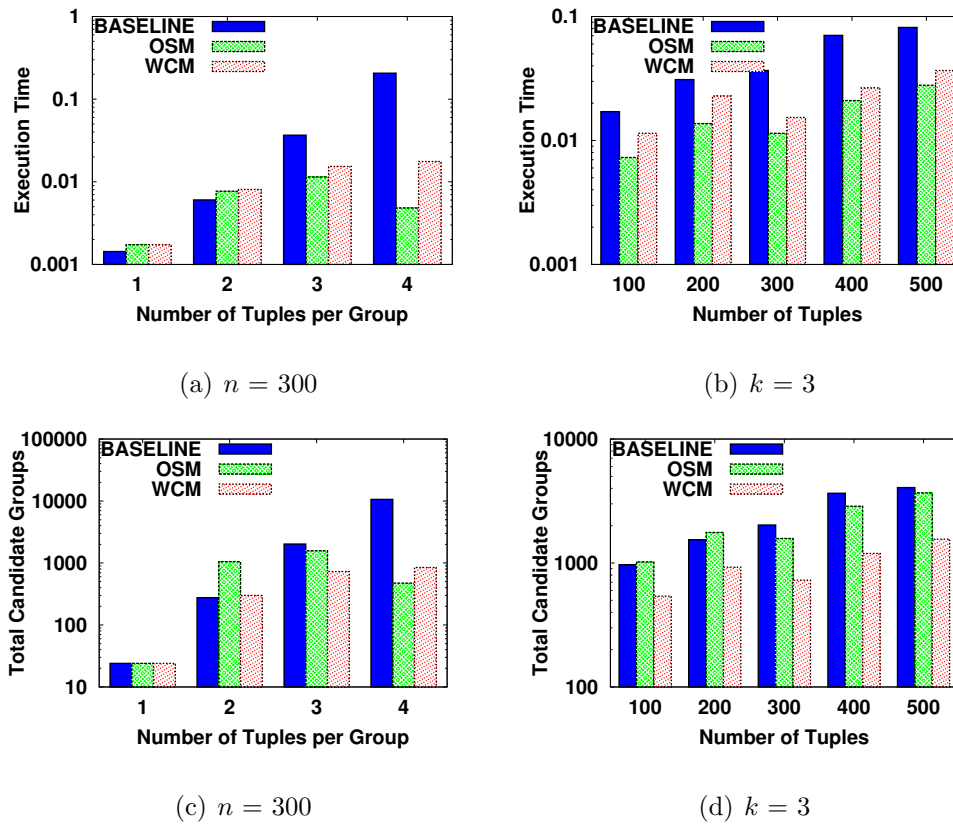
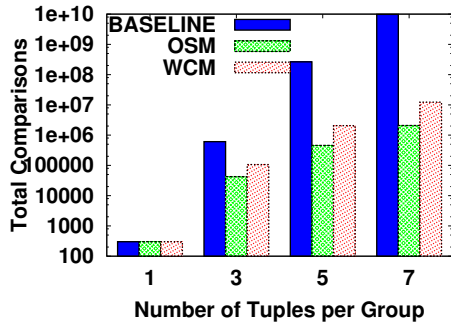
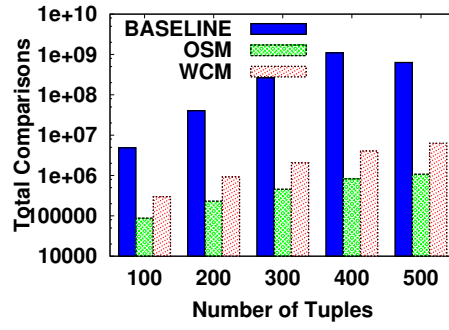


Figure 4.6. (a)-(b): Execution time (seconds, logarithmic scale) and (c)-(d): number of candidate groups (logarithmic scale), MAX

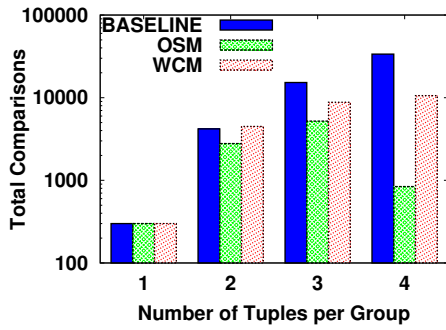
algorithm cannot finish within reasonable amount of time. (Thus the corresponding bars are not plotted.) This is not surprising. In anti-correlated dataset, values of a tuple on different attributes are negatively correlated. Hence it is more difficult to find a tuple dominating other tuples. This means input pruning in such a dataset cannot reduce the input size effectively, and OSM/WCM cannot prune many candidates either. Attributes in real datasets may neither be fully correlated nor fully anti-correlated. The attributes often form groups, such as *rebounds* and *blocks*, *assists* and *steals* in basketball games. The attributes within the same group are correlated, while



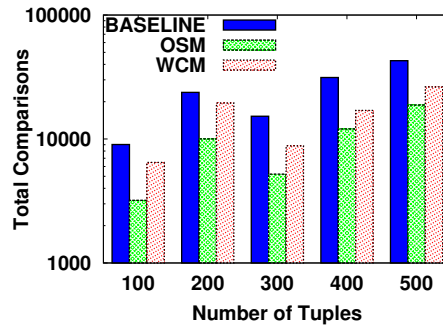
(a)  $n = 300$



(b)  $k = 5$

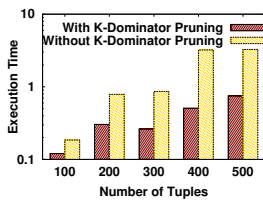


(c)  $n = 300$

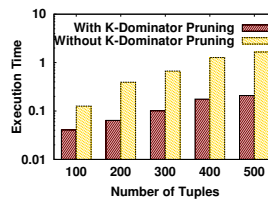


(d)  $k = 3$

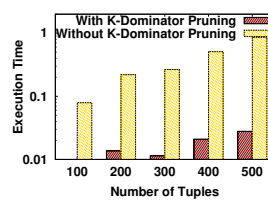
Figure 4.7. Number of pairwise group comparisons by different methods for MIN (a)-(b) and MAX (c)-(d)



(a) SUM



(b) MIN



(c) MAX

Figure 4.8. Effect of input pruning on OSM,  $k = 3$

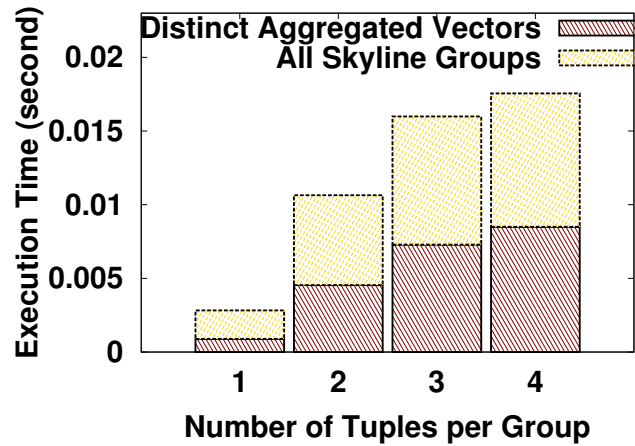


Figure 4.9. Finding all skyline groups for MAX,  $n=100$ , OSM

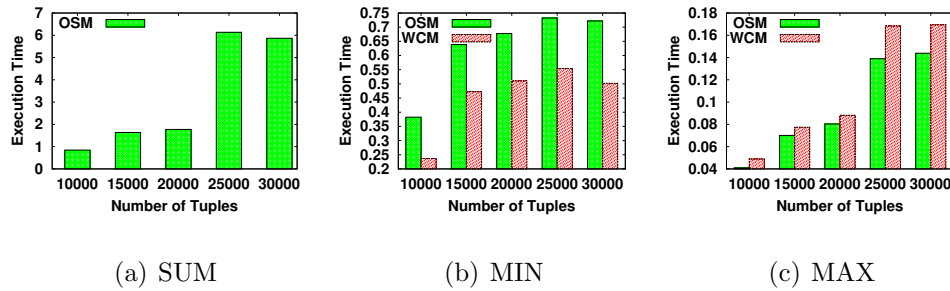


Figure 4.10. Execution time (seconds, logarithmic scale) on stock dataset,  $k=3$

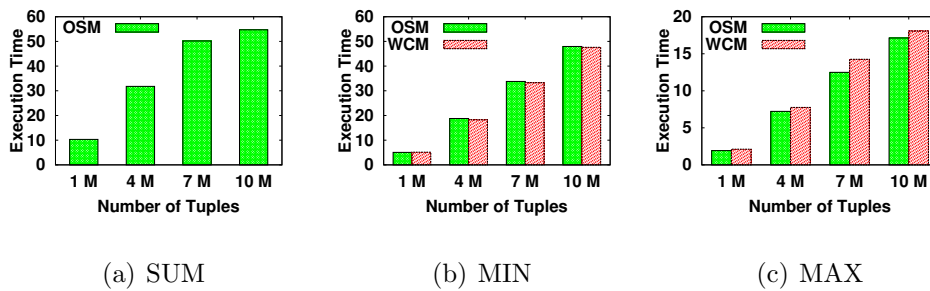


Figure 4.11. Execution time (seconds) of OSM/WCM on correlated synthetic dataset with 5 attributes,  $k=4$

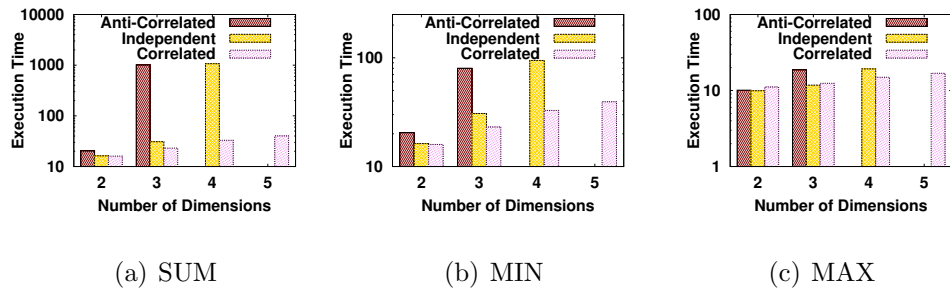


Figure 4.12. Execution time (seconds, logarithmic scale) of OSM on different synthetic datasets,  $k = 3$ ,  $n = 10$  million

$n$	$k = 3$	$k = 5$	$k = 7$
10000	99	139	178
15000	126	172	232
20000	130	180	239
25000	148	205	263
30000	143	217	281

Table 4.10. Number of tuples dominated by less than  $k$  tuples in stock dataset

the ones across different groups tend to be independent or anti-correlated. A direction for future study is to investigate the performance of our methods on synthetic data with such more realistic correlation patterns.

#### 4.7 Prototype System: CrewScout

We introduce CrewScout (<http://idir.uta.edu/crewsout>), a system for finding expert teams in accomplishing tasks. The underpinning concept of the system is *skyline teams* (called *skyline groups* in [61, 62]). The new contributions made in this

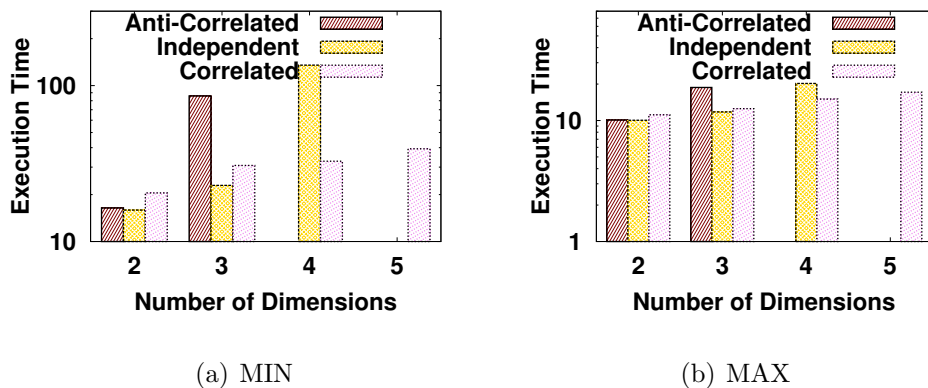


Figure 4.13. Execution time (seconds, logarithmic scale) of WCM on different synthetic datasets,  $k = 3$ ,  $n = 10$  million

dissertation include an end-to-end system with an interactive user interface that assists users in choosing teams among potentially many skyline teams and an extension of application and demonstration scenarios into more general areas ([61, 62] mostly focused on the application of forming teams in fantasy sports games.)

An attractive characteristic of a skyline team is that no other team of equal size can dominate it. In contrast, given a non-skyline team, there is always a better skyline team. This property distinguishes CrewScout from other team recommendation techniques. The skyline teams consist of the teams that are worth recommending. They become the input to further manual or automated post-processing that eventually finds one team. Admittedly, determining the “best” team is a complex task that may involve more factors than what skyline teams can capture—e.g., which experts are available for a task, whether they have good relationship to work together, and so on. The post-processing is thus crucial. Examples of such post-processing include eye-balling the skyline teams, filtering and ranking them by user preferences, and browsing and visualization of the skyline teams. Particularly, CrewScout provides

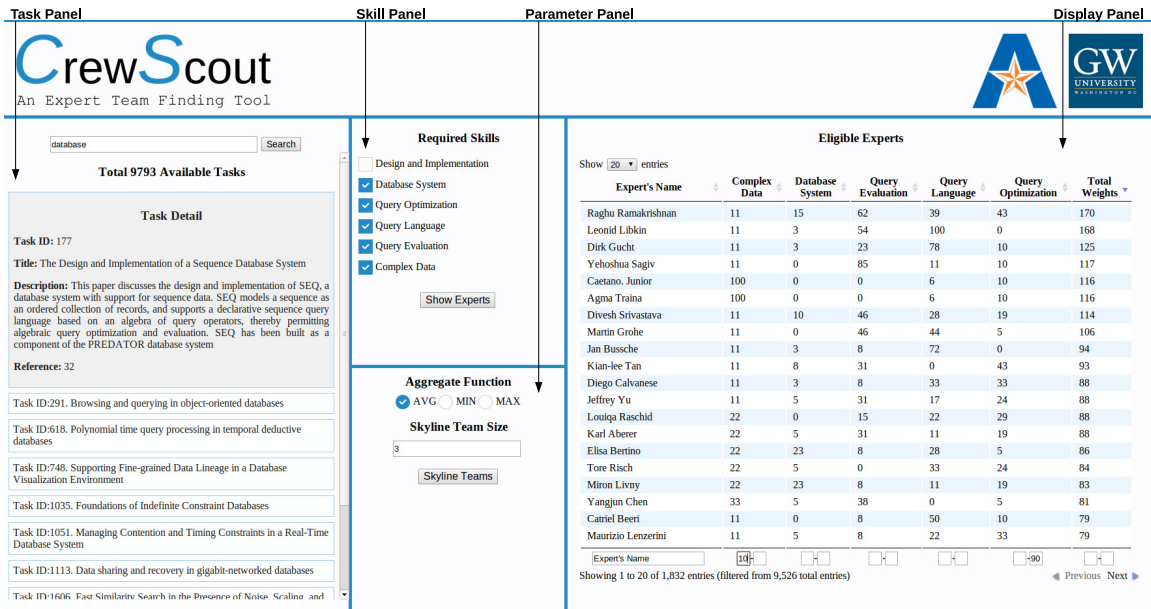


Figure 4.14. CrewScout Interface

an interactive tool to assist a human user in exploring and choosing skyline teams. Following portion explains CrewScout’s user interface. In a nutshell, a user starts their interaction with CrewScout by searching available tasks and selecting a task. CrewScout then proposes a set of experts who are eligible for forming teams. Then, the system provides a set of skyline teams and also guides the user to pick a team.

Figure 4.14 shows the GUI of CrewScout, which is comprised of a *task panel*, a *skill panel*, a *parameter panel*, and a *display panel*. The task panel presents a list of available tasks. When a user clicks a task, CrewScout provides more details about it. CrewScout also provides a keyword search box at the top of this panel for searching available tasks. The skill panel presents the skills required for completing the selected task. It shows a checkbox for each skill. By default, all the checkboxes are checked. The user can check/uncheck some of them according to their preference. When the

**Clustering Panel**

**Required Skills**

Design and Implementation

Database System

Query Optimization

Query Language

Query Evaluation

Complex Data

---

**Aggregate Function**

AVG  MIN  MAX

**Skyline Team Size**

---

Number of Clusters:

Clustering Algorithm:

Similarity Measure:

**Skyline Teams**

Show  entries

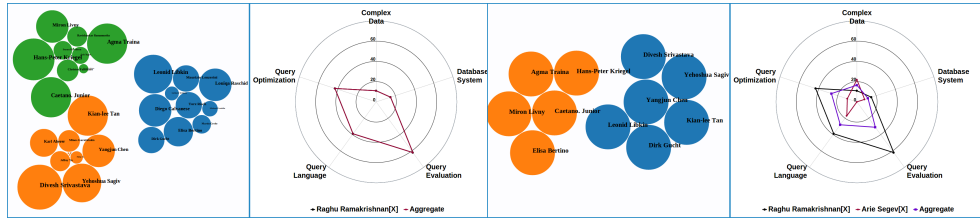
Team Members' Names	Complex Data	Database System	Query Evaluati
Dirk Gucht   Raghu Ramakrishnan   Leonid Libkin	11.00	7.00	46.33
Yehoshua Sagiv   Raghu Ramakrishnan   Leonid Libkin	11.00	6.00	67.00
Agma Traina   Raghu Ramakrishnan   Leonid Libkin	40.67	6.00	38.67
Caetano, Junior   Raghu Ramakrishnan   Leonid Libkin	40.67	6.00	38.67
Divesh Srivastava   Raghu Ramakrishnan   Leonid Libkin	11.00	9.33	54.00
Martin Grohe   Raghu Ramakrishnan   Leonid Libkin	11.00	6.00	54.00
Kian-lee Tan   Raghu Ramakrishnan   Leonid Libkin	11.00	8.67	49.00
Diego Calvanese   Raghu Ramakrishnan   Leonid Libkin	11.00	7.00	41.33
Jeffrey Yu   Raghu Ramakrishnan   Leonid Libkin	11.00	7.67	49.00
Raghu Ramakrishnan   Karl Aberer   Leonid Libkin	14.67	7.67	49.00
Raghu Ramakrishnan   Louiqa Raschid   Leonid Libkin	14.67	6.00	43.67
Raghu Ramakrishnan   Elisa Bertino   Leonid Libkin	14.67	13.67	41.33
Tore Risch   Raghu Ramakrishnan   Leonid Libkin	14.67	7.67	38.67
Miron Livny   Raghu Ramakrishnan   Leonid Libkin	14.67	13.67	41.33
Yangjun Chen   Raghu Ramakrishnan   Leonid Libkin	18.33	7.67	51.33
Maurizio Lenzerini   Raghu Ramakrishnan   Leonid Libkin	11.00	7.67	41.33
Hans-Peter Kriegel   Raghu Ramakrishnan   Leonid Libkin	14.67	20.33	38.67
Dirk Gucht   Yehoshua Sagiv   Raghu Ramakrishnan	11.00	6.00	56.67
Dirk Gucht   Agma Traina   Raghu Ramakrishnan	40.67	6.00	28.33
Dirk Gucht   Caetano, Junior   Raghu Ramakrishnan	40.67	6.00	28.33

Showing 1 to 20 of 504 entries

Figure 4.15. Display Panel Showing Skyline Teams

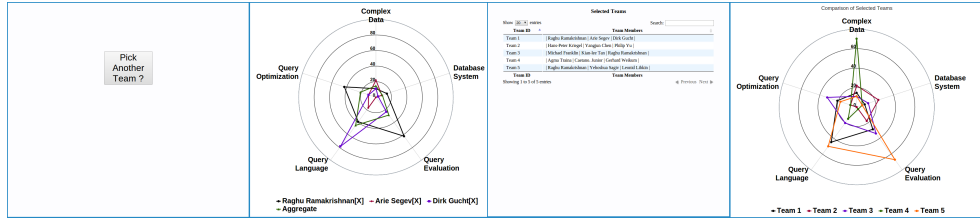
user clicks the “Show Experts” button, the display panel presents a paginated list of all experts who have expertise in at least one checked skill. If the user further checks/unchecks some skills, the expert list is automatically refreshed to reflect the change. Experts are ordered by summations of their expertise in all selected skills in the current implementation. In the expert list, a filter is provided for each skill. The user can filter the experts by setting the minimum and maximum expertise for one or more skills. The user can also filter the experts by their names through partial string matching.

The parameter panel allows the user to set parameters for skyline team computation. It includes a textbox for specifying the skyline team size and radio but-



(a) 1 expert selected

(b) 2 experts selected



(c) 3 experts selected

(d) Selected skyline teams

Figure 4.16. Selecting and Comparing Teams

tons for choosing an aggregate function (AVG, MIN, or MAX). Once the user clicks the “Skyline Teams” button, CrewScout calculates all skyline teams (considering all experts satisfying the aforementioned filters) and shows them in the display panel (Figure 4.15). Similar to the filters on experts, CrewScout also provides filters for the skyline team list, including filters on team members’ names and minimum/maximum aggregated expertise on individual skills. The teams satisfying the conditions are called the *qualifying skyline teams*. When the display panel exhibits the skyline teams, a *clustering panel* is added below the parameter panel. It provides a “Pick a Team” button and three drop-down lists that allow the user to choose a clustering algorithm (e.g., *K-means*), a similarity/distance function (e.g., Euclidean distance) for the clustering algorithm, and the number of clusters. When the user clicks the button, CrewScout will display below the current panels a visualization interface (Figure 4.16) that clusters the experts in the qualifying skyline teams and assists the user in exploring and choosing teams.



The visualization interface has two panels. The left panel visualizes the clusters. Each expert that belongs to at least one qualifying skyline team is represented as a circle. Circles in the same cluster are annotated with the same color. Their positions are automatically determined by the *multi-foci force layout* (<https://github.com/mbostock/d3/wiki/Force-Layout>). The size of a circle is proportional to the number of qualifying skyline teams containing the corresponding expert. At the beginning, only the labels of big circles (containing information of corresponding experts) are visible. When the user hovers the mouse over a circle, the expert’s profile (including the name and the number of skyline teams containing the expert) is displayed in a small pop-up window. The user can gradually zoom in to see the labels of smaller circles. The user can iteratively select  $k$  experts. Whenever the user selects an expert, CrewScout removes those circles whose corresponding experts do not belong to any skyline teams with all selected experts so far. The remaining circles are re-clustered and resized, based on only qualifying skyline teams containing the selected experts. The right panel presents a polar-chart. Each polygon in the polar-chart represents a selected expert’s expertise on the chosen skills. The aggregated expertise of the selected experts is also represented by a polygon. The selected experts are listed under the polar-chart. The user can remove any expert by clicking the cross sign beside it, and the clusters of circles are refreshed accordingly. Once  $k$  experts are selected, a skyline team is chosen. A “Pick Another Team?” button appears in the left panel. If the user clicks it, two more panels are added to the lower portion of the visualization interface—the left one lists all selected teams and the right one presents another polar-chart that compares them.

## 4.8 Usage Scenario of CrewScout

An online demonstration of CrewScout is hosted at <http://idir.uta.edu/crewsout>. Its front-end UI is developed in PHP+JavaScript. The system demonstrates three application scenarios, including paper reviewer selection, question answering, and team formation, on a 900K-publication dataset collected through Microsoft Academic Search API, a [stackoverflow.com](http://stackoverflow.com) dataset and an NBA dataset from [databasebasketball.com](http://databasebasketball.com), respectively. It also supports user-uploaded datasets. Below we describe the demonstration steps for the reviewer selection scenario, with an imaginary user Amy.

1. Amy searches for, say “database”, and matching publications are displayed in the task panel. A default publication is highlighted.
2. Amy clicks a publication to show or hide its abstract, depending on its status. When a publication is selected, the skill and display panels are refreshed with the corresponding required expertise and qualifying reviewers, respectively. Amy checks/unchecks one or more skills, the qualifying reviewers are automatically refreshed. Amy filters the reviewers by setting minimum and/or maximum thresholds on one or more skills. (Figure 4.14)
3. Amy specifies an aggregate function and a skyline team size in the parameter panel. Once Amy clicks the “Skyline Teams” button, the display panel shows the skyline teams (Figure 4.15). Amy can filter them by reviewer name and thresholds on aggregated skills.
4. After choosing clustering parameters, Amy clicks the “Pick a Team” button and the visualization interface presents the reviewer clusters (Figure 4.16).
5. Amy moves the mouse over the circles to see the reviewers’ profiles and she also zooms in and out. When Amy selects a reviewer, the corresponding expertise polygon is inserted into the polar-chart. Amy repeats this step multiple times until a  $k$ -reviewer team is formed.

6. Amy clicks the “Pick Another Team?” button to select another team. In this way, Amy chooses multiple teams and compares them in a polar-chart.

## CHAPTER 5

### Conclusion

In this dissertation, I studied some of the most challenging and novel problems of *Computational Journalism*. I motivated the problems with meaningful applications and use cases. I designed algorithms and techniques to solve these problems and presented extensive experimental results. I also developed prototype of the systems and demonstrated real-life usage scenarios of the systems. This is a small step towards *Computational Journalism*. Going forward, I plan to explore several directions, including further automation of the fact-checking process, automating comprehensive fact finding from live events and story telling. I hope that this study will help researchers to have a better understanding of *Computational Journalism* field and will also serve as the base for many automatic fact monitoring and checking solutions to come.

## REFERENCES

- [1] S. Cohen, J. T. Hamilton, and F. Turner, “Computational journalism,” *Commun. ACM*, vol. 54, no. 10, pp. 66–71, 2011.
- [2] S. Cohen, C. Li, J. Yang, and C. Yu, “Computational journalism: A call to arms to database researchers,” in *CIDR*, 2011, pp. 148–151.
- [3] X. Jiang, C. Li, P. Luo, M. Wang, and Y. Yu, “Prominent streak discovery in sequence data,” in *KDD*, 2011, pp. 1280–1288.
- [4] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu, “On “one of the few” objects,” in *KDD*, 2012, pp. 1487–1495.
- [5] B. Nyhan and J. Reifler, “The effect of fact-checking on elites: A field experiment on us state legislators,” *American Journal of Political Science*, vol. 59, no. 3, pp. 628–640, 2015.
- [6] L. Graves, “Deciding what’s true: Fact-checking journalism and the new ecology of news,” Ph.D. dissertation, COLUMBIA UNIVERSITY, 2013.
- [7] S. Cohen, J. T. Hamilton, and F. Turner, “Computational journalism,” *CACM*, vol. 54, no. 10, pp. 66–71, 2011.
- [8] S. Cohen, C. Li, J. Yang, and C. Yu, “Computational journalism: A call to arms to database researchers.” in *CIDR*, 2011.
- [9] J. Leskovec, L. Backstrom, and J. Kleinberg, “Meme-tracking and the dynamics of the news cycle,” in *KDD*, 2009.
- [10] A. M. Turing, “Computing machinery and intelligence,” *Mind*, vol. 59, no. 236, pp. 433–460, 1950.

- [11] Y. Wu, B. Walenz, P. Li, A. Shim, E. Sonmez, P. K. Agarwal, C. Li, J. Yang, and C. Yu, “iCheck: computationally combating ”lies, d-ned lies, and statistics”,” in *SIGMOD*, 2014.
- [12] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu, “Toward computational fact-checking,” in *PVLDB*, 2014.
- [13] B. Walenz *et al.*, “Finding, monitoring, and checking claims computationally based on structured data,” in *Computation+Journalism Symposium*, 2014.
- [14] N. Hassan, A. Sultana, Y. Wu, G. Zhang, C. Li, J. Yang, and C. Yu, “Data in, fact out: Automated monitoring of facts by FactWatcher,” *PVLDB*, vol. 7, no. 13, pp. 1557–1560, 2014.
- [15] J. Platt *et al.*, “Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods,” *Advances in large margin classifiers*, vol. 10, no. 3, 1999.
- [16] N. Hassan, C. Li, and M. Tremayne, “Detecting check-worthy factual claims in presidential debates,” in *CIKM*, 2015.
- [17] E. Riloff, J. Wiebe, and W. Phillips, “Exploiting subjectivity classification to improve information extraction,” 2005.
- [18] E. Riloff and J. Wiebe, “Learning extraction patterns for subjective expressions,” in *Proceedings of the 2003 conference on Empirical methods in natural language processing*. Association for Computational Linguistics, 2003, pp. 105–112.
- [19] M. E. McCombs and D. L. Shaw, “The agenda-setting function of mass media,” *Public opinion quarterly*, vol. 36, no. 2, pp. 176–187, 1972.
- [20] T. W. Smith, “America’s most important problem—a trend analysis, 1946–1976,” *Public Opinion Quarterly*, vol. 44, no. 2, pp. 164–180, 1980.

- [21] J.-H. Zhu, “Issue competition and attention distraction: A zero-sum theory of agenda-setting,” *Journalism & Mass Communication Quarterly*, vol. 69, no. 4, pp. 825–836, 1992.
- [22] N. Hassan, B. Adair, J. T. Hamilton, C. Li, M. Tremayne, J. Yang, and C. Yu, “The quest to automate fact-checking,” *Proceedings of the 2015 Computation+Journalism Symposium*, 2015.
- [23] S. Börzsönyi, D. Kossmann, and K. Stocker, “The skyline operator,” in *Proceedings of the 17th International Conference on Data Engineering*, 2001, pp. 421–430.
- [24] “<http://www.newsday.com/sports/columnists/neil-best/hirdt-enjoying-long-run-as-stats-guru-1.3174737>,” accessed: Jul. 2013.
- [25] J. Pei, Y. Yuan, X. Lin, W. Jin, M. Ester, Q. Liu, W. Wang, Y. Tao, J. X. Yu, and Q. Zhang, “Towards multidimensional subspace skyline analysis,” *ACM Trans. Database Syst.*, vol. 31, no. 4, pp. 1335–1381, 2006.
- [26] M. Zhang and R. Alhajj, “Skyline queries with constraints: Integrating skyline and traditional query operators,” *DKE*, vol. 69, no. 1, pp. 153 – 168, 2010.
- [27] T. Xia and D. Zhang, “Refreshing the sky: the compressed skycube with efficient support for frequent updates,” in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, 2006, pp. 491–502.
- [28] A. Sultana, N. Hassan, C. Li, J. Yang, and C. Yu, “Incremental discovery of prominent situational facts,” in *ICDE*, 2014, pp. 112–123.
- [29] T. Wu, D. Xin, Q. Mei, and J. Han, “Promotion analysis in multi-dimensional space,” *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 109–120, 2009. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1687627.1687641>
- [30] F. Alvanaki, E. Ilieva, S. Michel, and A. Stupar, “Interesting event detection through hall of fame rankings,” in *DBSocial*, 2013, pp. 7–12.

- [31] A. Bharadwaj, “Automatic Discovery of Significant Events From Databases,” Master’s thesis, University of Texas at Arlington, December 2011.
- [32] J. L. Bentley, “Multidimensional binary search trees in database applications,” *IEEE Trans. Softw. Eng.*, vol. 5, no. 4, pp. 333–340, July 1979.
- [33] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, “Skyline with presorting,” in *Proceedings of the International Conference on Data Engineering*, 2003, pp. 717–719.
- [34] P. Godfrey, R. Shipley, and J. Gryz, “Maximal vector computation in large data sets,” in *Proceedings of the 31st international conference on Very large data bases*, 2005, pp. 229–240.
- [35] K.-L. Tan, P.-K. Eng, and B. C. Ooi, “Efficient progressive skyline computation,” in *VLDB ’01: Proceedings of the 27th International Conference on Very Large Data Bases*, 2001, pp. 301–310.
- [36] D. Kossmann, F. Ramsak, and S. Rost, “Shooting stars in the sky: an online algorithm for skyline queries,” in *VLDB ’02: Proceedings of the 28th international conference on Very Large Data Bases*, 2002, pp. 275–286.
- [37] D. Papadias, Y. Tao, G. Fu, and B. Seeger, “Progressive skyline computation in database systems,” *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, pp. 41–82, 2005.
- [38] S. Chaudhuri, N. Dalvi, and R. Kaushik, “Robust cardinality and cost estimation for skyline operator,” in *Proceedings of the 22nd International Conference on Data Engineering*, April 2006, pp. 64–73.
- [39] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. Tung, and Z. Zhang, “On high dimensional skylines,” in *Advances in Database Technology - EDBT 2006*, Y. Ioannidis, M. Scholl, J. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, and C. Boehm, Eds., 2006, vol. 3896, pp. 478–495.



- [40] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, “Selecting stars: The k most representative skyline operator,” in *IEEE 23rd International Conference on Data Engineering*, April 2007, pp. 86–95.
- [41] Y. Tao, L. Ding, X. Lin, and J. Pei, “Distance-based representative skyline,” in *Proceedings of the 2009 IEEE International Conference on Data Engineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 892–903. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1546683.1547325>
- [42] C. Chan, H. Jagadish, K. Tan, A. Tung, and Z. Zhang, “Finding k-dominant skylines in high dimensional space,” in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, 2006, pp. 503–514.
- [43] R. Agrawal and R. Srikant, “Fast algorithms for mining association rules in large databases,” in *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, pp. 487–499.
- [44] H.T.Kung, F.Luccio, and F.P.Preparata, “On finding the maxima of a set of vectors,” *Journal of the ACM*, vol. 22, no. 4, pp. 469 – 476, 1975.
- [45] X. Lin, Y. Yuan, W. Wang, and H. Lu, “Stabbing the sky: efficient skyline computation over sliding windows,” in *Proceedings of the 21st International Conference on Data Engineering*, April 2005, pp. 502–513.
- [46] J. Pei, B. Jiang, X. Lin, and Y. Yuan, “Probabilistic skylines on uncertain data,” in *Proceedings of the 33rd international conference on Very large data bases*, 2007, pp. 15–26.
- [47] M. Morse, J. M. Patel, and H. V. Jagadish, “Efficient skyline computation over low-cardinality domains,” in *Proceedings of the 33rd international conference on Very large data bases*, 2007, pp. 267–278.

- [48] E. Dellis and B. Seeger, “Efficient computation of reverse skyline queries,” in *Proceedings of the 33rd international conference on Very large data bases*, 2007, pp. 291–302.
- [49] X. Lian and L. Chen, “Monochromatic and bichromatic reverse skyline search over uncertain databases,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, New York, NY, USA, 2008, pp. 213–226.
- [50] M. Sharifzadeh and C. Shahabi, “The spatial skyline queries,” in *Proceedings of the 32nd international conference on Very large data bases*. VLDB Endowment, 2006, pp. 751–762.
- [51] B.-C. Chen, K. LeFevre, and R. Ramakrishnan, “Privacy skyline: privacy with multidimensional adversarial knowledge,” in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 770–781.
- [52] P. Wu, C. Zhang, Y. Feng, B. Zhao, D. Agrawal, and A. El Abbadi, “Parallelizing skyline queries for scalable distribution,” in *Advances in Database Technology - EDBT 2006*. Springer Berlin / Heidelberg, 2006, pp. 112–130.
- [53] W.-T. Balke, U. Gntzer, and J. Zheng, “Efficient distributed skylining for web information systems,” in *Advances in Database Technology - EDBT 2004*, 2004, vol. 2992, pp. 573–574.
- [54] S. Antony, P. Wu, D. Agrawal, and A. El Abbadi, “Moolap: Towards multi-objective olap,” in *IEEE 24th International Conference on Data Engineering*, April 2008, pp. 1394–1396.
- [55] X. Zhang and J. Chomicki, “Preference queries over sets,” in *IEEE 27th International Conference on Data Engineering*, April 2011, pp. 1019–1030.

- [56] T. Lappas, K. Liu, and E. Terzi, “Finding a team of experts in social networks,” in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 467–476.
- [57] A. Anagnostopoulos, L. Becchetti, C. Castillo, A. Gionis, and S. Leonardi, “Power in unity: forming teams in large-scale community systems,” in *Proceedings of the 19th ACM international conference on Information and knowledge management*, 2010, pp. 599–608.
- [58] Y. Tao, X. Xiao, and J. Pei, “Subsky: Efficient computation of skylines in subspaces,” in *Proceedings of the 22nd International Conference on Data Engineering*, April 2006, p. 65.
- [59] “Stock dataset,” [http://pages.stern.nyu.edu/~adamodar/New\\_Home\\_Page/data.html](http://pages.stern.nyu.edu/~adamodar/New_Home_Page/data.html).
- [60] “Nba dataset,” <http://www.databasebasketball.com/>.
- [61] C. Li, N. Zhang, N. Hassan, S. Rajasekaran, and G. Das, “On skyline groups,” in *CIKM*, 2012, pp. 2119–2123.
- [62] N. Zhang, C. Li, N. Hassan, S. Rajasekaran, and G. Das, “On skyline groups,” *TKDE*, vol. 26, no. 4, pp. 942–956, April 2014.