Bulletin of the Technical Committee on

Data Engineering

September 2021 Vol. 45 No. 3

IEEE Computer Society

Letters

Special Issue on Data Engineering Challenges in Combating Misinformation

Conference and Journal Notices

The TC on Data Engineering

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems. The TCDE web page is http://tab.computer.org/tcde/index.html.

The Data Engineering Bulletin

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

The Data Engineering Bulletin web site is at http://tab.computer.org/tcde/bull_about.html.

Letter from the Editor-in-Chief

The September issue of the Data Engineering Bulletin, curated by associate editors Prof. Chengkai Li of University of Texas at Arlington and Prof. Jun Yang of Duke University, features a collection of papers on the topic of combating misinformation.

Combating misinformation has become our society's great urgency and top challenge. In the last couple of years, through the two elections in the U.S., the world-wide Covid-19 pandemic, as well as numerous incidents such as the January 6 United States Capitol attack, we have witnessed the sad havoc wreaked by misinformation.

In the most recent issue of The Atlantic, Stanford researcher Renée DiResta wrote about misinformation and amplified propaganda, "Understanding the incentives of influencers, recognizing the very common rhetorical techniques that precipitate outrage, developing an awareness of how online crowds now participate in crystallizing public opinion—that is an education that Americans need."

While definitely a societal, political, and educational issue, misinformation is a challenge that the tech industry and the academia must come together to address. The algorithms we develop, taking each individual act of clicking or rehashing as reinforcing signals, have the blind tendency to dramatically amplify an original message no matter how ludicrous it might be, enabling it to sway public opinions significantly on issues ranging from the pandemic to democracy.

The papers in this issue are a great starting point. They cover a broad spectrum of topics such as reliability scoring of information sources, fact-checking of individual claims, misinformation detection in a large corpus, and the promises and challenges of investigative journalism. One area we need to look into more seriously is the credibility of not only the information source, but also that of each participant in the information propagation process, that is, to how much extent a participant can be considered as a credible sponsor of the information in question.

Haixun Wang
Instacart

Letter from the Special Issue Editors

The problem of misinformation is not new, but there is now a growing consensus that it is posing a severe threat to the well-being of societies. We do not need to look far for compelling examples—during the ongoing COVID-19 pandemic, misinformation on the coronavirus and vaccines has greatly exacerbated the public health crises across the globe. As many have observed, the harm of misinformation today has been magnified by technologies that provide easy production, access, and dissemination of information with little regard to its accuracy and credibility. The computing community must take its share of responsibilities by developing effective defenses against misinformation. This special issue is devoted to exploring how the data engineering community can contribute to this fight. While a comprehensive solution will undoubtedly require concerted effort across disciplines, we believe our community has much to offer in this fight, with our expertise on working with data at scale, with our tradition of building abstractions that enable ease of use, and with our openness to embrace collaboration and ideas from other fields.

This special issue includes six articles that provide a sample of ongoing work on combating misinformation from across the world. We start with an overview article from the University of South Florida. It examines the strengths and limitations of various existing approaches to tackling three challenges that are vital for ensuring the integrity and credibility of online information. The first challenge is about how to gauge information reliability at source level, especially at the coarse level of web domains. The second one is about fact-checking at statement level, using NLP techniques and external data such as knowledge databases. The last challenge is about other signals that can be used to estimate information credibility without examining the veracity of information content itself.

The second paper is a European collaboration on a system called *ConnectionLens* for supporting *investigative journalism*, a vital part of any modern society that is playing an increasingly important role in investigating misinformation. *ConnectionLens* helps journalists by integrating a wide range of heterogeneous, schema-less data sources into a single graph for query and exploration; it employs scalable data processing techniques to reduce the cost of constructing and searching the graph. The paper demonstrates how such a system helps with a specific use case of detecting conflict of interests—a key factor of credibility—in biomedical research; however, it benefits a broad range of tasks including fact-checking by providing the source data needed for any data-driven investigation.

Assuming we have data, the next three papers address the challenge of vetting factual claims using reference data stored in relational tables. The third paper, from Eurocom, compares four recent fact-checking systems that use tables to verify statistical claims. The comparison was carried out both analytically, focusing on how these systems differ in terms of various input and output characteristics, and empirically using several datasets of claims. These systems are based on different methodologies, ranging from natural language inference, question answering, machine learning classifiers, to text-to-SQL generation. The results can provide useful insights that help future system designers produce more advanced fact-checking systems.

The fourth paper, from Renmin University of China, has a similar goal of study but examines in more detail a narrower set of fact-checking systems—it focuses on comparing several models that are based on natural language inference. It constructs a unified framework that can be instantiated into different existing models as well as new ones. They placed a particular emphasis on allowing the framework to encode table structures in the produced language models. Their experiment results demonstrated accuracy improvement due to the inclusion of such features.

The fifth paper, a collaboration between UIC, UMich, and Google, goes beyond merely verifying the *correctness* of a claim, and detects whether it could still mislead by "cherry-picking." Specifically, the paper tackles two popular claim types of trendlines and rankings; even from the same underlying data, people can claim different trends and rankings by cherry-picking their vantage points. The paper shows how to use *perturbation* analysis to capture the robustness of claims when their vantage points are perturbed, and how to perform such analysis efficiently even over large perturbation spaces.

Finally, the sixth paper, from Cornell, outlines an end-to-end system called *WebChecker* for detecting misinformation in a very large collection of documents, e.g., from Web or social media platforms. *WebChecker* employs a wide array of fact-checking methods with different cost-accuracy trade-offs. Running an expensive deep neural net on every text snippet to detect misinformation may be more accurate, but will be prohibitively expensive at Web-scale. To look for the combination of methods that correctly detects the most amount of misinformation while staying within a cost budget, *WebChecker* adaptively switches processing methods to sample their quality and uses reinforcement learning to principally evolve its strategies.

Overall, we hope this collection of six articles together offers a sample of the ongoing work as well as open data engineering challenges in combating misinformation. Working on this special issue has been a privilege for us, as the topic has been dear to our hearts for many years. You may recall our "call to arms" to the database community in CIDR 2011—a decade later, the call is still on, and perhaps more pressing now than ever. We would like to thank the authors of this issue for their contributions, and would welcome more from the data engineering community to join this fight against misinformation.

Chengkai Li and Jun Yang
University of Texas at Arlington and Duke University, USA

Preserving the Integrity and Credibility of the Online Information Ecosystem

Matthew Sumpter and Giovanni Luca Ciampaglia*

University of South Florida

Abstract

The Internet seems awash with information that is either inaccurate or shared with malicious intents, or both. While this is not the first time that society has had to deal with this problem, certain features of our modern, fast-paced, data-driven information ecosystem seem to exacerbate it. Thus it is extremely important to equip journalists and fact-checkers, and of course the public at large, with tools to help them deal with the proliferation of false and misleading information and to promote the quality of information circulating online. In this paper we survey the state of the work in this area, focusing in particular on the challenges stemming from dealing with the peculiar nature of social media data, and discuss recent proposals to devise scalable and accurate signals of information quality.

1 Introduction

In recent years the Internet seems to have become the source and vehicle of many societal ills. The explosion of hoaxes, conspiracy theories, and state-sponsored disinformation, has given prominence and extreme urgency to the problem. But false and misleading information has existed also in the past. The style and themes of so-called "fake news"[1] websites, that came to prominence during the 2016 U.S. Presidential Elections, carry a striking resemblance to those used by "yellow journalism" outlets from the early 20$^{\text{th}}$ century [2]. Thus it is important to understand what elements of the problem are really novel and which are not.

There are a variety of classifications of information, including but not limited to: *rumor*, *gossip*, *propaganda*, *conspiracy theories*, *hoaxes*, and *satire*. The distinction between many of these terms relies on a matter of intention. For example, *satire* may be intentionally false for the purpose of amusement. However, a *rumor* can simply be a misinterpretation of the available facts with no intention to mislead. Allowing for the additional flexibility of rhetoric, such as sarcasm and persuasion, it becomes clear how difficult it can be to discern whether information is being presented with the intention to inform, persuade, or entertain.

From a quantitative point of view, social media are obviously different from the media of the past like the telegraph or the printed press: for one, they allow for a much faster and broader dissemination of information.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Corresponding author. Email: glc3@usf.edu

[1]Fake news is defined as "impostor news", i.e. outlets featuring many of the trappings associated to the news publishing business without the necessary quality control mechanisms employed by professional publications [1].

But given the many striking similarities between then and now, one would be tempted to dismiss the whole phenomenon of fake news and online misinformation as the same of yellow journalism or gossip: a problem arising from fundamental human cognitive limits, and whose solution does not require any novel thinking or tools. In this paper, we argue that this is not the case. There are several novel factors at play at the nexus of the social, cognitive, and algorithmic levels, which require us to think in creative ways to address this fundamentally new societal problem. Many of these solutions could come from the field of data engineering, as we outline next.

2 The prevalence and sources of online misinformation

The introduction of the World Wide Web in the early 90s has dramatically lowered the barriers of entry to mass communication, allowing for anyone with an Internet connection to publish information affordably and quickly, regardless of its merit. Although this led to an enormous volume of information [3], it was still not easy to garner large audiences until the development of two key innovations: search engines [4], and social networking services [5], which have made it possible to retrieve information easily and have allowed for communities of homogeneous opinion to aggregate and share information.

A virtuous outcome of these innovations is the emergence of networks of like-minded individuals capable of self-organizing in novel and surprising ways [6], and to nurture strong ties that transcend the limits imposed by geographical distance or culture [7, 8], but one downside is the risk of polarization or cyberbalkanization [9–11]. Online communities are ripe for exploitation by bad actors seeking to manipulate large groups of people. Bolstered by content curation algorithms designed to increase engagement (filter bubbles [12]), these communities may fall prey to manipulation due to a mix of psychological and algorithmic biases that entrench accepted information while resisting opposing viewpoints (echo chambers [13]).

There are, unfortunately, numerous contemporary examples of widespread misinformation campaigns, including foreign influence in the 2016 election and the rise of conspiracy theories like 'Qanon', and the suspicion of widespread, yet unfounded, election fraud. However, gauging the actual prevalence of false and inaccurate information on social media is still the subject of much research [1, 14, 15], especially in the context of political communication [16]. Even though strong exposure to fake news is limited to the segment of most active news consumers, or 'superspreaders' [17], individual claims echoing the false or misleading content shared by these audiences can spread rapidly through social media [18, 19], amplified by bots [20] or other malicious actors [21], who often target elites, like celebrities, pundits, or politicians.

There are two inherent limitations to many of these studies. The first is that they typically rely on active engagement (e.g. likes and comments) as a proxy for exposure, but miss data about impressions, thus neglecting exposure due to "passive" forms of consumption such as scrolling and reading. The second stems from the impossibility of verifying each individual piece of content shared online. To get around this limitation, scientists have resorted to coarse-grained content annotation schemes to reduce sparsity and increase coverage. Source reliability ratings are a prime example of this approach [14, 17, 20, 22], as they typically identify sources by their web domain name. Lists of domain ratings compiled by expert journalists or fact-checkers have allowed to measure the prevalence of online misinformation, as well as the impact of social bots in spreading content from low-reliability sources. There are two advantages to this approach.

First, due to the highly skewed character of online popularity, one advantage of annotating sources at the web domain level is that it is possible to attain a high coverage by focusing only on the top most popular domains. Second, since web domains are the main form of identity on the web, source-level ratings are an effective way for social media platforms and search engines to implement content exclusion lists. Blocking or deprioritizing domains with low reliability introduces strong reputational costs for any agent who wishes to spread misinformation.

However source-level reliability ratings also come with limitations and challenges, which are especially relevant for data engineers. Maintaining such lists is typically time consuming, and authoring tools that facilitate

such annotations are of great help. On top of that, source-level ratings may miss important heterogeneity *within* individual sources. This is especially true of larger outlets, that may employ different journalistic standards between the newsroom and their editorial desk. Finally, when lists are used to enforce moderation standards, they cause misinformation agents to update their domain name frequently in an attempt to circumvent detection, requiring more frequent updates on the part of the reliability annotators.

3 Verifying content and fact-checking claims

The prevalence of misinformation has led to the rise of the fact-checking industry. Fact-checking has been recognized as a solution to misinformation, as those who risk being fact-checked are less likely to share misinformation in the first place [23]. Unfortunately, claims online are generated and spread at a rate far too quickly for human journalists to keep up with. This is because fact-checking is a non-trivial task requiring the claim to be researched and then the fact-check written, published, and distributed. The window of time between initial claim and published fact-check can be significant (an estimate places it at in the 10–20h range [24]), allowing the claim to have reached too wide of an audience for the fact-check to be effective.

In addition to the time requirement of fact-checking, there is the confounding human component. Although fact-checkers are typically trained journalists, they are still susceptible to both mistakes and bias. These issues have revealed an opportunity for computer scientists to get involved. Computational approaches to fact-checking have the potential to address both the time-dependent and human-dependent issues of manual fact-checking.

ClaimBuster [25], developed in 2014, is the first end-to-end fact-checking system. Using a variety of machine learning and NLP techniques, media sources are monitored for statements that match an existing repository of professional fact-checks so that they may be served expressly to an audience. On unseen claims, it attempts to frame the claim as a question for the purpose of querying knowledge bases and question answering engines, such as Wolfram Alpha and Google search results.

Another approach engages with the content in a more structured way. It does so by means of knowledge representation techniques such as knowledge graphs [26]. A knowledge graph aims to store and provide knowledge of the real world, where nodes are entities and the edges are the relations that exist between these entities. Knowledge graphs have a number of advantages over relational and non-relational models for the massive, open-world domain that "real-world" knowledge implies. The basic unit of knowledge contained in a knowledge graph is typically the semantic triple, composed of two entities (eg. a subject and an object) and a predicate relation that exists between them. An example of a semantic triple is `<Tallahassee, capitalOf, Florida>`. Compiling many series of semantic triples results in a knowledge graph which can be traversed to gain insights regarding the semantic relationships between entities.

There exist several general-purpose knowledge graphs, including Yago [27], DBpedia [28], and Wikidata [29]. It is notable that, although these contain lots of overlapping and related information, they lack consolidation and techniques must be developed to map entities and relations between these various knowledge graphs. This is a similar task to that of ontology alignment [30].

Fact-checking has been performed using existing knowledge graphs by receiving a claim as a semantic triple and checking its validity based on the sets of existing triples in a knowledge base that connect the subject and object of the claim triple [31–34]. Although this approach has proven promising, it is limited by its input because reducing a complex claim into a semantic triple is a nontrivial task, akin to the NLP task of relation extraction [35]. Fact-checking a political claim requires identifying an ontology best suited to modeling this type of claims and then developing a tool that can reduce claims into triples using the selected ontology. To address this task, we have built a pipeline which focuses on extracting relations by modeling a network of claims as a graph network using sentence dependency trees [36]. By modeling input claims in a format more analogous to the verification method (knowledge graphs), we were able to successfully extract relations from real-world claims and use them as input into fact-checking algorithms.

4 Toward signals of quality

Fact-checking has become a critical component of the socio-technical infrastructure devoted to preserving the integrity of the online information ecosystems. Social media platforms, despite initial reluctance, have embraced this valuable resource. Facebook, for example, partners with accredited fact-checking organization to review potentially misleading content in circulation on its platform. If a piece of content is flagged by a fact-checker as false, Facebook automatically reduces its visibility across the platform, reducing the chances that users may be exposed to it [37].

Of course, given the scale of social media, fact-checking every piece of content in circulation on it is likely an unfeasible task. Therefore, social media platforms are seeking to identify signals of credibility, or news quality, that could help them promote trustworthy and reliable information. These signals should rely on information about the content that is readily accessible to the platform or at least easy to estimate, without having to inspect the content itself or having to bring in a human to review it.

There is a vast literature devoted to identifying the reputation of content and actors on the Web [38–44], but many of these approaches are either hard to scale or make restrictive assumptions about the type of metadata available. These assumptions often reflect the specific Web platforms for which they were originally developed, for example the requirement that content is organized following wiki principles and that there is a full history of all user actions available [38]. More recently work in the context of political news consumption has proposed to use crowdsourcing to evaluate the reliability of news sources [45]. Finally, other work proposes instead to promote content that produces engagement within a politically diverse audience [46].

5 Discussion and future challenges

When it comes to the modern information ecosystem, misinformation and disinformation (in all its related forms) poses strong threats to the integrity of public discourse. Here we have outlined two areas where data engineering could help conduct the fight against fake news. The first is about measuring the actual prevalence of misinformation, the amount of exposure it gets, and the main actors responsible for producing and disseminating it. Source-level reliability ratings are the current standard used in much of the literature and form the backbone of several content moderation schemes used by real platforms [47, 48]. Lists curated by fact-checkers and other third-party organizations are thus a valuable tool, but present several challenges and limitations. First of all, the content of these lists is often static while the Web and social media are dynamic environments. How to update these lists? One possibility could be to look into the tail of the popularity distribution for potential candidates of up-and-coming misinformation producers. Another approach could be to focus on users that are routinely exposed to known low-credibility sources (or that share content from them) and identify what other sources they are routinely exposed to (co-exposure) or they share content from (co-sharing). This could reveal novel unrated sources that could be passed to expert journalist for rating and further evaluation (see Fig. 1). More broadly, combining data about co-exposure (or co-sharing) of multiple users it could be possible to define networks of low-credibility sources with overlapping audiences, potentially revealing the broader ecosystem of online misinformation. Several interesting questions arise: can we look into niche communities to see what content they are sharing? Can we identify collusion rings of sources pushing similar content in an orchestrated fashion? To support this endeavor, there is an urgent need for novel technical standards and methods. Novel Web standards could help provide more meaningful annotation for Internet sources, while novel methods to link different domains operated by the same organizations could mitigate the aforementioned problems with churn and active avoidance by malicious actors.

The second major area of activity is the quest to automate fact-checking or at least help support human fact-checkers in their task of verifying claims. Considering the unstructured nature of online discourse, there is in this case too a need for novel standards of annotation of online content. The development of a standardized

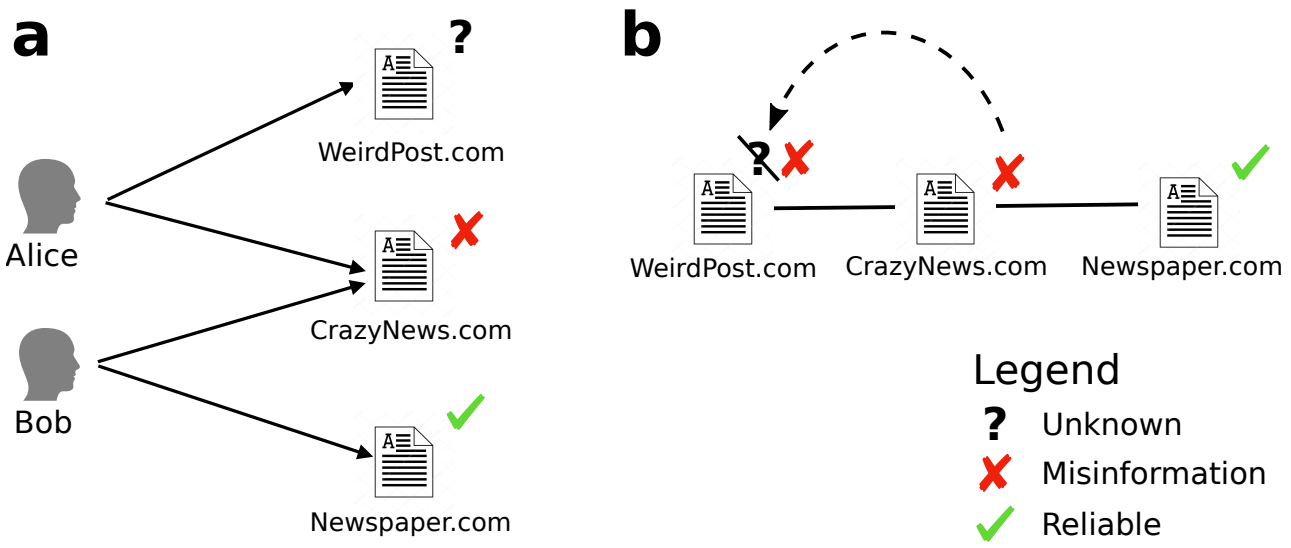


Figure 1: **Example of a co-exposure network**. **(a)** Alice and Bob follow three sources: a known misinformation source (CrazyNews.com), a known reliable source (Newspaper.com) and a source for which no rating is available (WeirdPost.com). **(b)** The co-exposure network is a graph whose nodes are sources and there is an edge (solid lines) between two sources if they share some users (i.e., they *co-expose*). Since WeirdPost.com as a co-exposure with CrazyNews.com due to Alice, it is possible to propagate the rating of 'misinformation source' to its neighbor with unknown rating (dashed arrow). Note that the propagation does not apply to neighbors with a known rating: Newspaper.com is already a known reliable source so its rating does not change.

schema and ontology for content annotation would provide a means by which multimedia content on the web could begin to properly be aggregated and cross-referenced in a dependable way. Journalists and publishers could be provided education and tools on how to annotate their own work such that computer scientists would not need to retroactively mine and process media on the web.

A good example is the ClaimReview schema [49], a content annotation format developed by researchers at Duke University. ClaimReview provides an important first step in the direction of content annotation as it relates to claims in news media. This schema allows professional fact checkers to annotate their fact-checks with distinct properties, such as the claim reviewed and the rating decision. This allows claims to be aggregated and queried by search engines. There has been some preliminary research regarding necessary schemas for annotating news media. Arslan et al. [50] have identified a set of 20 semantic frames (11 novel frames and 9 exisiting Berkeley FrameNet [51] frames) that can be used to model factual claims. Ciampaglia and Licato [52] identified the need for argumentation schemes to capture rhetorical methods present among claims found in news media.

References

[1] D. M. Lazer, M. A. Baum, Y. Benkler, A. J. Berinsky, K. M. Greenhill, F. Menczer, M. J. Metzger, B. Nyhan, G. Pennycook, D. Rothschild, *et al.*, "The science of fake news," *Science*, vol. 359, no. 6380, pp. 1094–1096, 2018.

[2] S. McQueen, "From yellow journalism to tabloids and clickbait: The origins of fake news in the united states," in *Information Literacy and Libraries in the Age of Fake News* (D. Agosto, ed.), pp. 12–35, ABC-CLIO, 2018.

[3] M. Hilbert and P. Lopez, "The world's technological capacity to store, communicate, and compute information," *Science*, vol. 332, pp. 60–65, Feb. 2011.

[4] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," Technical Report 1999-66, Stanford InfoLab, Nov. 1999. Previous number = SIDL-WP-1999-0120.

[5] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?," in *Proceedings of the 19th international conference on World wide web - WWW '10*, ACM Press, 2010.

[6] Z. Tufekci, *Twitter and tear gas : the power and fragility of networked protest*. New Haven: Yale University Press, 2017.

[7] D. Mocanu, A. Baronchelli, N. Perra, B. Gonçalves, Q. Zhang, and A. Vespignani, "The twitter of babel: Mapping world languages through microblogging platforms," *PLoS ONE*, vol. 8, p. e61981, Apr. 2013.

[8] P. S. Park, J. E. Blumenstock, and M. W. Macy, "The strength of long-range ties in population-scale social networks," *Science*, vol. 362, pp. 1410–1413, Dec. 2018.

[9] J. E. Katz, "Struggle in cyberspace: Fact and fiction on the world wide web," *The Annals of the American Academy of Political and Social Science*, vol. 560, pp. 194–199, Nov. 1998.

[10] M. Van Alstyne and E. Brynjolfsson, "Global village or cyber-balkans? modeling and measuring the integration of electronic communities," *Management Science*, vol. 51, no. 6, pp. 851–868, 2005.

[11] M. Conover, J. Ratkiewicz, M. Francisco, B. Gonçalves, F. Menczer, and A. Flammini, "Political polarization on Twitter," in *International AAAI Conference on Web and Social Media*, ICWSM '11, (Palo Alto, CA, USA), pp. 89–96, AAAI, 2011.

[12] E. Pariser, *The filter bubble : how the new personalized web is changing what we read and how we think*. New York: Penguin Press, 2011.

[13] C. R. Sunstein, *#Republic*. Princeton University Press, Apr. 2018.

[14] H. Allcott and M. Gentzkow, "Social media and fake news in the 2016 election," *Journal of economic perspectives*, vol. 31, no. 2, pp. 211–36, 2017.

[15] A. Bovet and H. A. Makse, "Influence of fake news in twitter during the 2016 us presidential election," *Nature Communications*, vol. 10, p. 7, Jan. 2019.

[16] A. M. Guess, B. Nyhan, and J. Reifler, "Exposure to untrustworthy websites in the 2016 us election," *Nature human behaviour*, vol. 4, no. 5, pp. 472–480, 2020.

[17] N. Grinberg, K. Joseph, L. Friedland, B. Swire-Thompson, and D. Lazer, "Fake news on twitter during the 2016 us presidential election," *Science*, vol. 363, no. 6425, pp. 374–378, 2019.

[18] S. Vosoughi, D. Roy, and S. Aral, "The spread of true and false news online," *Science*, vol. 359, no. 6380, pp. 1146–1151, 2018.

[19] Zhao, Zilong, Zhao, Jichang, Sano, Yukie, Levy, Orr, Takayasu, Hideki, Takayasu, Misako, Li, Daqing, Wu, Junjie, and Havlin, Shlomo, "Fake news propagates differently from real news even at early stages of spreading," *EPJ Data Sci.*, vol. 9, no. 1, p. 7, 2020.

[20] C. Shao, G. L. Ciampaglia, O. Varol, K.-C. Yang, A. Flammini, and F. Menczer, "The spread of low-credibility content by social bots," *Nature communications*, vol. 9, no. 1, pp. 1–9, 2018.

[21] J. Weedon, W. Nuland, and A. Stamos, "Information operations and facebook," tech. rep., Facebook, Inc., 2017.

[22] M. Zimdars, *Fake News: Understanding Media and Misinformation in the Digital Age*, ch. Viral "Fake News" Lists and the Limitations of Labeling and Fact-Checking, pp. 361–372. MIT Press Ltd, Feb. 2020.

[23] B. Nyhan and J. Reifler, "The effect of fact-checking on elites: A field experiment on u.s. state legislators," *American Journal of Political Science*, vol. 59, no. 3, pp. 628–640, 2015.

[24] C. Shao, G. L. Ciampaglia, A. Flammini, and F. Menczer, "Hoaxy: A platform for tracking online misinformation," in *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, (Republic and Canton of Geneva, Switzerland), pp. 745–750, International World Wide Web Conferences Steering Committee, 2016.

[25] N. Hassan, G. Zhang, F. Arslan, J. Caraballo, D. Jimenez, S. Gawsane, S. Hasan, M. Joseph, A. Kulkarni, A. K. Nayak, V. Sable, C. Li, and M. Tremayne, "Claim buster: The first ever end-to-end fact checking system," *Proceedings of the VLDB Endowment*, vol. 10, no. 12, pp. 1945–1948, 2017.

[26] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, J. E. L. Gayo, S. Kirrane, S. Neumaier, A. Polleres, R. Navigli, A.-C. N. Ngomo, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, and A. Zimmermann, "Knowledge graphs," tech. rep., CoRR, Mar. 2020.

[27] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: A core of semantic knowledge," in *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, (New York, NY, USA), pp. 697–706, Association for Computing Machinery, 2007.

[28] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The Semantic Web* (K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, eds.), (Berlin, Heidelberg), pp. 722–735, Springer Berlin Heidelberg, 2007.

[29] D. Vrandečić, "Wikidata: A new platform for collaborative data collection," in *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, (New York, NY, USA), pp. 1063–1064, Association for Computing Machinery, 2012.

[30] F. M. Suchanek, S. Abiteboul, and P. Senellart, "PARIS," *Proceedings of the VLDB Endowment*, vol. 5, pp. 157–168, nov 2011.

[31] N. Lao and W. W. Cohen, "Relational retrieval using a combination of path-constrained random walks," *Machine Learning*, vol. 81, no. 1, pp. 53–67, 2010.

[32] G. L. Ciampaglia, P. Shiralkar, L. M. Rocha, J. Bollen, F. Menczer, and A. Flammini, "Computational fact checking from knowledge networks," *PLOS ONE*, vol. 10, pp. 1–13, June 2015.

[33] P. Shiralkar, A. Flammini, F. Menczer, and G. L. Ciampaglia, "Finding streams in knowledge graphs to support fact checking," in *2017 IEEE International Conference on Data Mining (ICDM)*, (Piscataway, NJ), pp. 859–864, IEEE, Nov. 2017. Extended Version.

[34] B. Shi and T. Weninger, "Discriminative predicate path mining for fact checking in knowledge graphs," *Knowledge-Based Systems*, vol. 104, pp. 123–133, July 2016.

[35] A. Smirnova and P. Cudré-Mauroux, "Relation extraction using distant supervision," *ACM Computing Surveys*, vol. 51, pp. 1–35, jan 2019.

[36] M. Sumpter and G. L. Ciampaglia, "REMOD: Relation extraction for modeling online discourse," in *Proceedings of the 1st International Workshop on Knowledge Graphs for Online Discourse Analysis*, 2021.

[37] Full Fact, "Report on the Facebook Third-Party Fact-Checking programme," tech. rep., Full Fact, 2020.

[38] B. T. Adler and L. de Alfaro, "A content-driven reputation system for the Wikipedia," in *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, (New York, NY, USA), pp. 261–270, ACM, 2007.

[39] J.-H. Cho, K. Chan, and S. Adali, "A survey on trust modeling," *ACM Comput. Surv.*, vol. 48, pp. 28:1–28:40, Oct. 2015.

[40] J. A. Golbeck, *Computing and applying trust in web-based social networks*. PhD thesis, University of Maryland at College Park, 2005.

[41] A. Gupta, P. Kumaraguru, C. Castillo, and P. Meier, *TweetCred: Real-Time Credibility Assessment of Content on Twitter*, pp. 228–243. Cham: Springer International Publishing, 2014.

[42] H. Rashkin, E. Choi, J. Y. Jang, S. Volkova, and Y. Choi, "Truth of varying shades: Analyzing language in fake news and political fact-checking," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, (Copenhagen, Denmark), pp. 2931–2937, Association for Computational Linguistics, Sept. 2017.

[43] S. Jiang and C. Wilson, "Linguistic signals under misinformation and fact-checking: Evidence from user comments on social media," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, Nov. 2018.

[44] S. Jiang, S. Baumgartner, A. Ittycheriah, and C. Yu, *Factoring Fact-Checks: Structured Information Extraction from Fact-Checking Articles*, pp. 1592–1603. New York, NY, USA: Association for Computing Machinery, 2020.

[45] G. Pennycook and D. G. Rand, "Fighting misinformation on social media using crowdsourced judgments of news source quality," *Proceedings of the National Academy of Sciences*, vol. 116, no. 7, pp. 2521–2526, 2019.

[46] S. Bhadani, S. Yamaya, A. Flammini, F. Menczer, G. L. Ciampaglia, and B. Nyhan, "Political audience diversity and news reliability in algorithmic ranking," tech. rep., CoRR, July 2020.

[47] G. Morrow, B. Swire-Thompson, J. Polny, M. Kopec, and J. Wihbey, "The emerging science of content labeling: Contextualizing social media content moderation," *SSRN Electronic Journal*, 2020.

[48] J. Nassetta and K. Gross, "State media warning labels can counteract the effects of foreign disinformation," *Harvard Kennedy School Misinformation Review*, oct 2020.

[49] Schema.org, "ClaimReview." online at `https://schema.org/ClaimReview`, 2020.

[50] F. Arslan, J. Caraballo, D. Jimenez, and C. Li, "Modeling factual claims with semantic frames," in *Proceedings of the 12th Language Resources and Evaluation Conference*, (Marseille, France), pp. 2511–2520, European Language Resources Association, May 2020.

[51] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The berkeley framenet project," in *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*, pp. 86–90, 1998.

[52] G. L. Ciampaglia and J. Licato, "Fact-checking, false narratives, and argumentation schemes," in *Proceedings of the 1st International Workshop on Knowledge Graphs for Online Discourse Analysis*, 2021.

Empowering Investigative Journalism with Graph-based Heterogeneous Data Management

Angelos-Christos Anadiotis[⋆], Oana Balalau[◇], Théo Bouganim[◇], Francesco Chimienti[◇], Helena Galhardas[‡], Mhd Yamen Haddad[◇], Stéphane Horel[†], Ioana Manolescu[◇], Youssr Youssef[◇],
[⋆]École Polytechnique, EPFL &IPP, [◇]Inria & IPP, [‡]INESC-ID & IST, Univ. Lisboa, [†]Le Monde

Abstract

Investigative Journalism (IJ, in short) is staple of modern, democratic societies. IJ often necessitates working with large, dynamic sets of heterogeneous, schema-less data sources*, which can be* structured, semi-structured, or textual*, limiting the applicability of classical data integration approaches. In prior work, we have developed ConnectionLens, a system capable of integrating such sources into a single heterogeneous graph, leveraging Information Extraction (IE) techniques; users can then query the graph by means of keywords, and explore query results and their neighborhood using an interactive GUI. Our keyword search problem is complicated by the graph heterogeneity, and by the lack of a result score function that would enable pruning of the search space.*

In this work, we describe an actual IJ application *studying conflicts of interest in the biomedical domain, and we show how ConnectionLens supports it. Then, we present* novel techniques addressing the scalability challenges *raised by this application: one allows us to reduce the significant IE costs while building the graph, while the other is a novel, parallel, in-memory keyword search engine, which achieves orders of magnitude speed-up over our previous engine. Our experimental study on the real-world IJ application data confirms the benefits of our contributions.*

1 Introduction

Journalism and the press are a critical ingredient of any modern society. Like many other industries, such as trade, or entertainment, journalism has benefitted from the explosion of Web technologies, which enabled instant sharing of their content with the audience. However, unlike trade, where databases and data warehouses had taken over daily operations decades before the Web age, *many newsrooms discovered the Web and social media, long before building strong information systems where journalists could store their information and/or ingest data of interest for them.* As a matter of fact, journalists' desire to protect their confidential information may also have played a role in delaying the adoption of data management infrastructures in newsrooms.

At the same time, highly appreciated journalism work often requires *acquiring, curating, and exploiting large amounts of digital data*. Among the authors, S. Horel co-authored the "Monsanto Papers" series which obtained the European Press Prize Investigative Reporting Award in 2018 [1]; a similar project is the "Panama Papers" (later known as "Offshore Leaks") series of the International Consortium of Investigative Journalists [2]. In such

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

works, journalists are forced to work with *heterogeneous* data, potentially in *different data models* (*structured* such as relations, *semistructured* such as JSON or XML documents, or graphs, including but not limited to RDF, as well as *unstructured text*). We, the authors, are currently collaborating on such an **Investigative Journalism** (IJ, in short) application, focused on the study of **situations potentially leading to conflicts of interest**[1] (CoIs, in short) between biomedical experts and various organizations: corporations, industry associations, lobbying organizations or front groups. Information of interest in this setting comes from: scientific publications (in PDF) where authors declare e.g., "Dr. X. Y. has received consulting fees from ABC"; semi-structured metadata (typically XML, used for instance in PubMed), where authors may also specify such connections; a medical association, say, French Cardiology, may build its own disclosure database which may be relational, while a company may disclose its ties to specialists in a spreadsheet.

This paper builds upon our recent work [3], where we have identified a set of requirements (**R**) and constraints (**C**) that need to be addressed to efficiently support IJ applications. We recall them here for clarity and completeness:

R1. Integral source preservation and provenance: in journalistic work, it is crucial to be able to trace each information item back to the data source from which it came. This enables *adequately sourcing* information, an important tenet of quality journalism.

R2. Little to no effort required from users: journalists often lack time and resources to set up IT tools or data processing pipelines. Even when they are able to use a tool supporting one or two data models (e.g., most relational databases provide some support for JSON data), handling other data models remains challenging. Thus, the data analysis pipeline needs to be as automatic as possible.

C1. Little-known entities: interesting journalistic datasets feature some extremely well-known entities (e.g., world leaders in the pharmaceutical industry) next to others of much smaller notoriety (e.g., an expert consulted by EU institutions, or a little-known trade association). From a journalistic perspective, such lesser-known entities may play a crucial role in making interesting connections among data sources, e.g., the association may be created by the industry leader, and it may pay the expert honoraries.

C2. Controlled dataset ingestion: the level of confidence in the data required for journalistic use excludes massive ingestion from uncontrolled data sources, e.g., through large-scale Web crawls.

R3. Performance on "off-the-shelf" hardware: The efficiency of our data processing pipeline is important; also, the tool should run on general-purpose hardware, available to users like the ones we consider, without expertise or access to special hardware.

Further, IJ applications' data analysis needs entail:

R4. Finding connections across heterogeneous datasets is a core need. In particular, it is important for our approach to be tolerant of inevitable differences in the organization of data across sources. Heterogeneous data integration works, such as [4–6], and recent heterogeneous polystores, e.g., [7–9] assume that sources have well-understood schemas; other recent works, e.g., [10–12] focus on analyzing large sets of Open Data sources, all of which are tabular. IJ data sources do not fit these hypotheses: data can be semi-structured, structured, or simply text. Therefore, we opt for **integrating all data sources in a heterogeneous graph** (with no integrated schema) and for **keyword-based querying** (where users specify some terms); the system returns subtrees of the graph that connect nodes matching these terms.

C3. Lack of single, well-behaved answer score: After discussing several journalistic scenarios, we have not been able to identify a unique method (score) for deciding which are the best answers to a query. Instead: (i) it appears that "very large" answers (say, of more than 20 edges) are of limited interest; (ii) connections that "state the obvious", e.g., that a French scientist is connected to a French company through their nationality, are not of interest. Therefore, unlike prior keyword search algorithms, which fix a score function and exploit it to prune the search, our algorithm must be orthogonal and work it with any score function.

[1] According to the 2011 French transparency law, "A conflict of interest is any situation where a public interest may interfere with a public or private interest, in such a way that the public interest may be, or appear to be, unduly influenced."

Building upon our previous work, and years-long discussions of IJ scenarios, this paper makes the following contributions:

- We describe the CoI IJ application proposed by S. Horel (Section 2); we extract its technical requirements and we devise an end-to-end data analysis pipeline addressing these requirements (Section 3).
- We provide application-driven optimizations, inspired from the CoI scenario but reusable to other contexts, which speed up the graph construction process (Section 4).
- We introduce a parallel, in-memory version of the keyword search algorithm described in [3, 13], and we explain our design in both the physical database layout and the parallel query execution (Section 5).
- We evaluate the performance of our system on synthetic and real-world data. We demonstrate its scalability, and demonstrate performance improvements of several orders of magnitude over our prior work, thereby enabling the journalists to perform interactive exploration of their data (Section 6).

2 Use case: conflicts of interest in the biomedical domain

The topic. Biomedical experts such as health scientists and researchers in life sciences play an important role in society, advising governments and the public on health issues. They also routinely interact with industry (pharmaceutical, agrifood etc.), consulting, collaborating on research, or otherwise sharing work and interests. To trust advice coming from these experts, it is important to ensure the advice is not unduly influenced by vested interests. Yet, IJ work, e.g. [14–16], has shown that disclosure information is often scattered across multiple data sources, hindering access to this information. We now illustrate the data processing required to gather and collectively exploit such information.

Sample data. Figure 1 shows a tiny fragment of data that can be used to find connections between scientists and companies. *For now, consider only the nodes shown as a black dot or as a text label, and the solid, black edges connecting them; these model directly the data. The others are added by ConnectionLens as we discuss in Section 3.1.* (i) Hundreds of millions of bibliographic notices (in **XML**) are published on the PubMed web site; the site also links to research (in PDF). In recent years, PubMed has included an optional CoIStatement element where authors can declare (in free text) their possible links with industrial players; less than 20% of recent papers have this element, and some of those present are empty ("The authors declare no conflict of interest"). (ii) Within the **PDF** papers themselves, paragraphs titled, e.g., "Acknowledgments", "Disclosure statement" etc., may contain

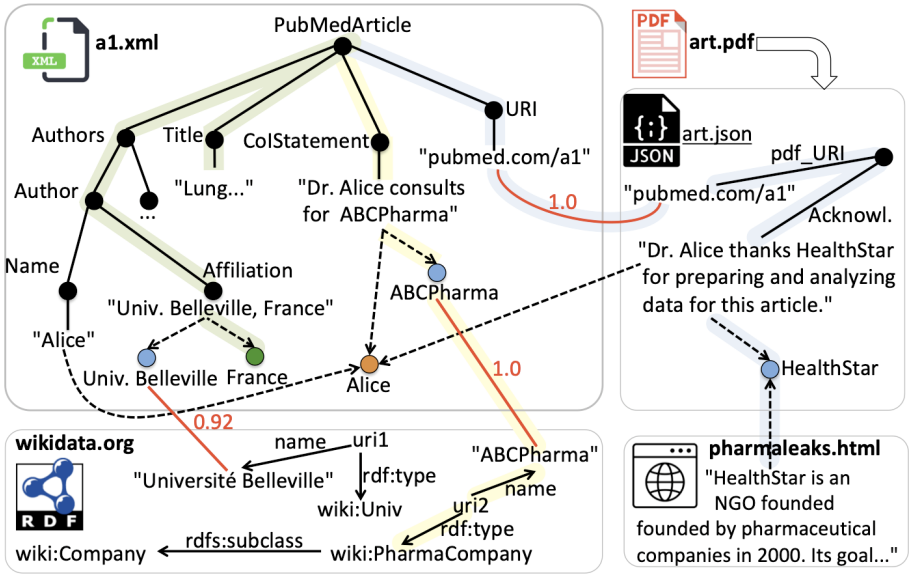


Figure 1: Graph data integration in ConnectionLens.

such information, even if the CoIStatement is absent or empty. This information is accessible if one converts the PDF in a format such as **JSON**. In Figure 1, Alice declares her consulting for ABCPharma in XML, yet the "Acknowledgments" paragraph in her PDF paper mentions HealthStar.[2] (iii) A (subset of a) knowledge base (in **RDF**) such as WikiData describes well-known entities, e.g., ABCPharma; however, less-known entities of interest in an IJ scenario are often missing from such KGs, e.g., HealthStar in our example. (iv) Specialized data sources, such as a trade catalog or a Wiki Web site built by other investigative journalists, may provide information on some such actors: in our example, the PharmaLeaks Web site shows that HealthStar is also funded by the industry. Such a site, established by a trusted source (or colleague), even if it has little or no structure, is a gold mine to be reused, since it saves days or weeks of tedious IJ work. *In this and many IJ scenarios, sources are highly heterogeneous, while time, skills, and resources to curate, clean, or structure the data are not available*.

Sample query. Our application requires *the connections of specialists in lung diseases, working in France, with pharmaceutical companies*. In Figure 1, the edges with *green* highlight and those with *yellow* highlight, together, form an answer connecting Alice to ABCPharma (spanning over the XML and RDF sources); similarly, the edges highlighted in *green* together with those in *blue*, spanning over XML, JSON and HTML, connect her to HealthStar.

The potential impact of a CoI database. A database of known relationships between experts and companies, built by integrating heterogeneous data sources, would be a valuable asset. In Europe, such a database could be used, e.g., to select, for a committee advising EU officials on industrial pollutants, experts with few or no such relationships. In the US, the Sunshine Act [17], just like the French 2011 transparency law, requires manufacturers of drugs and medical devices to declare such information. However, this does not extend to companies from other sectors.

3 Investigative journalism pipeline

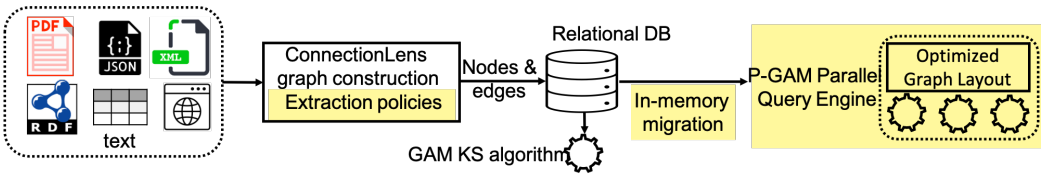


Figure 2: Investigative Journalism data analysis pipeline.

The pipeline we have built for IJ is outlined in Figure 2. First, we recall ConnectionLens graph construction (Section 3.1), which integrates heterogeneous data into a graph, stored and indexed in PostgreSQL. On this graph, the GAM keyword search algorithm (recalled in Section 3.2) answers queries such as our motivating example; these are both detailed in [3]. The modules with yellow background in Figure 2 are the novelties of this work, and will be introduced below: scenario-driven performance optimizations to the graph construction (Section 4), and an in-memory, parallel keyword search algorithm, called P-GAM (Section 5).

3.1 ConnectionLens graph construction

ConnectionLens integrates JSON, XML, RDF, HTML, relational or text data into a graph, as illustrated in Figure 1. Each source is mapped to the graph as close to its data model as possible, e.g., XML edges have no labels while internal nodes all have names, while in JSON conventions are different. Next, ConnectionLens **extracts named entities from all text nodes**, regardless the data source they come from, using trained language models. In the

[2]This example is inspired from prior work of S. Horel where she identified (manually inspecting thousands of documents) an expert supposedly with no industrial ties, yet who authored papers for which companies had supplied and prepared data.

figure, blue, green, and orange nodes denote Organization, Location, and Person entities, respectively. Each such entity node is connected to the text node it has been extracted from, by an *extraction edge* recording also the confidence of the extraction (dashed in the figure). **Entity nodes are shared across the graph**, e.g., Person:Alice has been found in three data sources, Org:BestPharma in two sources etc. ConnectionLens includes a *disambiguation* module which avoids mistakenly unifying entities with the same labels but different meanings. Finally, nodes with similar labels are *compared*, and if their similarity is above a threshold, a **sameAs** (red) edge connecting them is introduced, labeled with the similarity value.

A sameAs edge with similarity 1.0 is called an *equivalence edge*. Then, p equivalent nodes, e.g., the entity ABCPharma and the identical-label RDF literal, would lead to $p(p-1)/2$ equivalence edges. To keep the graph compact, one of the p nodes is declared the *representative* of all p nodes, and instead, we only store the $p-1$ equivalence edges adjacent to the representative. Details on the graph construction steps can be found in [3].

Formally, a ConnectionLens graph is denoted $G = (N, E)$, where nodes can be of different types (URIs, XML elements, JSON nodes, etc., including extracted entities) and edges encode data source structure, the connection between extracted entities and the text in which they were found, as well as node label similarity.

3.2 The GAM keyword search algorithm

We view our motivating query, on highly heterogeneous content with no a-priori known structure, as a **keyword search query over a graph**. Formally, a query $Q = \{w_1, w_2, \ldots, w_m\}$ is a set of m keywords, and an *answer tree* (AT, in short) is a set t of G edges which (i) together, form a tree, and (ii) for each w_i, contain at least one node whose label matches w_i. We are interested in *minimal* answer trees, i.e., answer trees that satisfy the following properties: (i) removing an edge from the tree will make it lack at least one keyword match, and (ii) if mulitple nodes match a query keyword, then all matching nodes are related through sameAs links with similarity 1.0. In the literature (see Section 7), a *score function* is used to compute the quality of an answer, and only the best k ATs are returned, for a small integer k. Our problem is harder since: (i) our ATs may span across different data sources, even of different data models; (ii) they may traverse an edge **in its original or in the opposite direction**, e.g., to go from JSON to XML through Alice; this doubles the search space, compared to a directed graph where a single direction is considered; and (iii) **no single score function serves all IJ needs** since, depending on the scenario, journalists may favor different (incompatible) properties of an AT, such as "being characteristic of the dataset" or, on the contrary, "being surprising". Thus, **we cannot rely on special properties of the score function** to help us prune unpromising parts of the search space, as done in prior work (see Section 7). Intuitively, tree size could be used to limit the search: very large answer trees (say, of more than 100 edges) generally do not represent meaningful connections. However, in heterogeneous, complex graphs, users find it hard to set a size limit for the exploration. Nor is a smaller solution always better than a larger one. For instance, an expert and a company may both have "nationality" edges leading to "French" (a solution of 2 edges), but that may be less interesting than finding that the expert has written an article specifying in its CoIStatement funding from the company (which could span 5 edges or more).

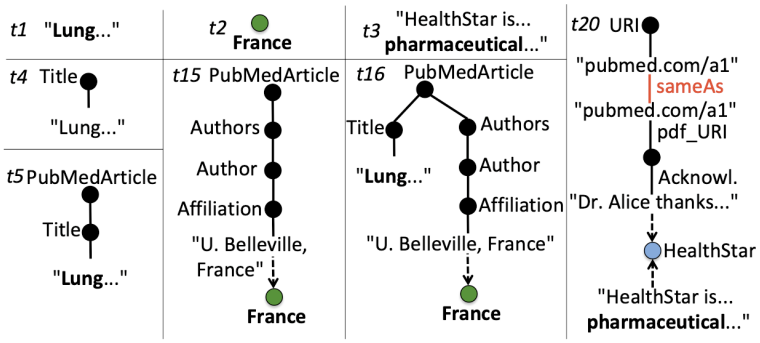


Figure 3: Trees built by GAM for our sample query.

Our **Grow-and-Aggressive-Merge (GAM)** algorithm [3, 13] enumerates trees exhaustively, until a number of answers are found, or a time-out. First, it builds 1-node trees from the nodes of G which match 1 or more keywords, e.g., t_1, t_2, t_3 in Figure 3, showing some partial trees built when answering our sample query. The

keyword match in each node label appears in bold. Then, GAM relies on two steps. **Grow** adds to the root of a tree one of its adjacent edges in the graph, leading to a new tree: thus t_4 is obtained by Grow on t_1, t_5 by Grow on t_4, and successive Grow steps lead from t_2 to t_{15}. Similarly, from t_3, successive Grow's go from the HTML to the JSON data source (the HealthStar entity occurs in both), and then to the XML one, building t_{20}. Second, as soon as a tree is built by Grow, it is **Merge**d with all the trees already found, rooted in the same node, matching different keywords and having disjoint edges wrt the given tree. For instance, assuming t_{15} is built after t_5, they are immediately merged into the tree t_{16}, having the union of their edges. Each Merge result is then merged again with all qualifying trees (thus the "agressive" in the algorithm name). For instance, when Grow on t_{20} builds a tree rooted in the PubMedArticle node (not shown; call it t_A), Merge(t_{16}, t_A) is immediately built, and is exactly the answer highlighted with green and blue in Figure 1. Section 5.2 explains the steps of GAM in the context of its parallel version introduced in this paper.

Together, Grow and Merge are guaranteed to generate all solutions. If $m = 2$, Grow alone is sufficient, while $m \geq 3$ also requires Merge. *GAM may build a tree in several ways*, e.g., the answer above could also be obtained as Merge(Merge(t_{15}, Grow(t_{20})), t_5); GAM keeps a history of already explored trees, to avoid repeating work on them. Importantly, GAM can be used with any score function. Its details are described in [3, 13].

4 Use case-driven optimization

In this section, we present an optimization we brought to the graph construction process, guided by our target application.

In the experiments we ran, Named Entity Recognition (NER) took up to 90% of the time ConnectionLens needs to integrate data sources into a graph. The more textual the sources are, the more time is spent on NER. Our application data lead us to observe that:

- Some text nodes ()e.g., those found on the path PubMedArticle.Authors.Author.Name) *always correspond to entities of a certain type* (in our example, Person). If this information is given to ConnectionLens, it can create a Person entity node, like the Alice node in Figure 1, *without calling the expensive NER procedure*.
- Other text nodes may be deemed *uninteresting for extraction* because journalists think no interesting entities appear there. If ConnectionLens is aware of this, it can *skip the NER call on such text nodes*. Observe that the input data, including all its text nodes, is always preserved; we only avoid extraction effort deemed useless (but it can still be applied later if application requirements evolve).

To exploit this insight, we introduced a notion of **context**, and allow users to specify **(optional) extraction policies**. A context is an expression designating a set of text nodes in one or several data sources. For instance, a context specified by the rooted path PubMedArticle.Authors.Author.Name designates all the text values of nodes found on that path in an XML data source; the same mechanism applies to an HTML or JSON data source. In a relational data source containing table R with attribute a, a context of the form $R.a$ designates all text nodes in the ConnectionLens graph obtained from a value of the attribute a in relation R. Finally, an RDF property p used as context designates all the values o such that a triple (s, p, o) is ingested in a ConnectionLens graph.

Based on contexts, an extraction policy takes one of the following form: (i) c *force* T_e, where c is a context and T_e is an entity type, e.g., Person, states that each node designated by the context is exactly one instance of T_e ; (ii) c *skip*, to indicate that NER should not be performed on the text nodes designated by c; (iii) as syntactic sugar, for hierarchical data models (e.g., XML, JSON etc.), c *skipAll* states that NER should not be performed on the text nodes designated by c. This allows larger-granularity control of NER on different portions of the data.

Observe that our contexts (thus, our policies) are specified *within a data model*; this is because the *regularity that allows defining them* can only be hoped for within data sources with identical structure. Policies allow journalists to state what is obvious to them, and/or what is not interesting, in the interest of graph construction

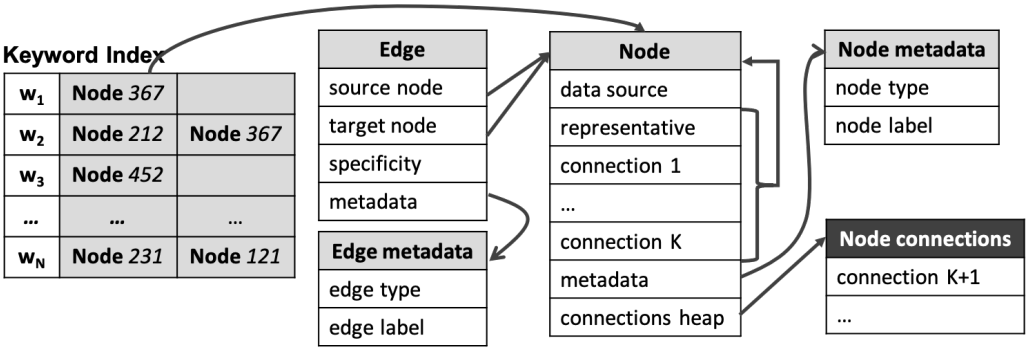


Figure 4: Physical graph layout in memory.

speed. *Force* policies may also improve graph quality, by making sure NER does not miss any entity designated by the context.

5 In-memory parallel keyword search

We now describe the novel keyword search module that is the main technical contribution of this work. A *in-memory graph storage model* specifically designed for our graphs and with keyword search in mind (Section 5.1) is leveraged by a a *multi-threaded, paralell* algorithm, called P-GAM (Section 5.2), which is a parallel extension of our original GAM algorithm outlined in Section 3.2.

5.1 Physical in-memory database design

The size of the main memory in modern servers has grown significantly over the past decade. Data management research has by now led to several DB engines running entirely in main memory, such as Oracle Database In-Memory, SAP HANA, and Microsoft SQL Server with Hekaton. Moving data from hard disks to main memory significantly boosts performance by avoiding disk I/O costs. However, it introduces new challenges on the optimization of the data structures and the execution model for a different bottleneck: memory access [18].

We have integrated P-GAM inside a novel in-memory graph database, which we have built and optimized for P-GAM operations. The physical layout of a graph database is important, given that graph processing is known to suffer from random memory accesses [19–22]. Our design (i) includes all the data needed by applications as described in Section 2, while also (ii) aiming at high-performance, parallel query execution in modern scale-up servers, in order to tackle huge search spaces (Section 3.2).

We start with the scalability requirements. Like GAM, P-GAM also performs Grow and Merge operations (recall Figure 3). To enumerate possible *Grow* steps, P-GAM needs to access all edges adjacent to the root of a tree, as well as the representative (Section 3.1) of the root, to enable growing with an equivalence edge. Further, as we will see, P-GAM (as well as GAM) relies on a simple edge metric, called **specificity**, derived from the number of edges with the same label adjacent to a given node, to decide the best neighbor to Grow to. For instance, if a node has 1 spouse and 10 friend edges, the edge going to the spouse is more specific than one going to a friend. A *Merge* does not need more information than available in its input trees; instead, it requires specific run-time data structures, as we describe below.

In our memory layout, we **split the data required for search, from the rest**, as the former are critical for performance; we refer to the latter as metadata. Figure 4 depicts the memory tables that we use. The *Node* table includes the ID of the data source where the node comes from, and references to each node's: (i) representative, (ii) K neighbors, if they exist (for a fixed K, which is pre-allocated), (iii) metadata, and (iv) other neighbors, if they exist (dynamically allocated beyond K). We separate the allocation of neighbors into static and dynamic,

Algorithm 1: P-GAM

Input: $G = (N, E)$, query Q={w$_1$, . . ., w$_m$}, maximum number of solutions M, maximum time limit
Output: Answer trees for Q on G

```
1 pQueue_i ← new priority queue of (tree, edge) pairs, 1 ≤ i ≤ nt;
2 N_Q ← ∪_{w_i∈Q} keywordIndex.lookup(w_i);
3 for n ∈ N_Q, e edge adjacent to n do
4 |   push (n, e) on some pQueue_j (distribute equally)
5 end
6 launch nt P-GAM Worker (Algorithm 2) threads;
7 return solutions
```

to keep K neighbors in the main Node structure, while the rest are placed in a separate heap area, stored in the *Node connections* table. This way, we can allocate a fixed size to each Node, efficiently supporting the memory accesses of P-GAM. In our implementation, we set $K = 5$; in general, it can be set based on the median degree of the graph vertices. The *Node metadata* table includes information about the type of each node (e.g., JSON, HTML, etc.) and its label, comprising the keywords that we use for searching the graph. The *Edge* table includes a reference to the source and the target node of every edge, the edge specificity, and a reference to the edge metadata. The *Edge metadata* table includes the type and the label of each edge. Finally, we use a *keywordIndex*, which is a hash-based map associating every node with its labels. P-GAM probes the keywordIndex when a query arrives to find the references to the Node table that match the query keywords and start the search from there. The labels are encoded in order to achieve a more compact representation, while also indexed to allow prefix matching, following the work in [23]. Among all the structures, only *Node connections* (singled out by a dark background in Figure 4) is in a dynamically allocated area; all the others are statically allocated.

The above storage is *row* (node) oriented, even though column storage often speeds up greatly analytical processing; this is due to the nature of the keyword search problem, which requires traversing the graph from the nodes matching the keywords, in BFS style. Since we consider ad-hoc queries (any keyword combinations), there are no guarantees about the order of the nodes P-GAM visits. Therefore, in our setting, the vertically selective access patterns, which are exploited by column-stores, do not apply. Instead, the crucial optimization here is to *find the neighbors of every node fast.* This is leveraged by our algorithm, as we explain below.

5.2 P-GAM: parallel keyword query execution

Our P-GAM (Parallel GAM) query algorithm builds a set of data structures, which are exploited by concurrent workers (threads) to produce query answers. We split these data structures to shared and private to the workers. We start with the shared ones. The **history** data structure holds all trees built during the exploration, while **treesByRoot** gives access to all trees rooted in a certain node. As the search space is huge, history and treesByRoot grow very much. Specfically, for history, P-GAM first has to make sure that an intermediate AT has not been considered before (i.e., browse the history) before writing a new entry. Similarly, treesByRoot is updated only when a tree changes its root or if there is a Merge of two trees; however, it is probed several times for Merge candidates. Therefore, we have implemented these data structures as lock-free hash-based maps to ensure high concurrency and prioritize read accesses. Observe that, given the high degree of data sharing, keeping these data structures thread-private would not yield any benefit.

Moving to the thread-private data structures, *each thread*, say number i, has a priority queue **pQueue**$_i$, in which we push (tree, edge) pairs, such that the edge is adjacent to the root of the tree. Priority in this queue is determined as follows: we prefer the pairs *whose nodes match most query keywords*; to break a tie, we prefer *smaller trees*; and to break a possible tie among these, we prefer the pair where the edge has the *highest-specificity*. This is a simple priority order we chose empirically; any other priority could be used, with no change to the

Algorithm 2: P-GAM Worker (thread number i out of nt)

```
1  repeat
2      pop (t, e), the highest-priority pair in pQueue_i (or, if empty, from the pQueue_j having the most
         entries);
3      t_G ← Grow(t, e);
4      if t_G ∉ history then
5          for all edges e' adjacent to the root of t_G, push (t_G, e') in pQueue_i;
6          build all t_M ← Merge(t_G, t') where t' ∈ treesByRoot.get(t_G.root) and t' matches Q keywords
             disjoint from those of t_G;
7          if t_M ∉ history then
8              recursively merge t_M with all suitable partners;
9              add all the (new) Merge trees to history;
10             for each new Merge tree t'', and edge e'' adjacent to the root of t'', push (t'', e'') in pQueue_i;
11         end
12     end
13 until time-out or M solutions are found or all pQueue_j empty, for 1 ≤ j ≤ nt;
```

algorithm.

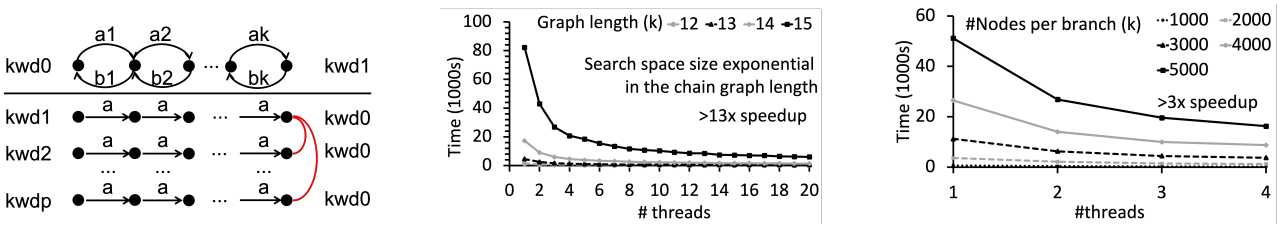
P-GAM keyword search is outlined in Algorithm 1. It creates the shared structures, and nt threads (as many as available based on the availability of computing hardware resources). The search starts by looking up the nodes N_Q matching at least one query keywords (line 2); we create a 1-node tree from each such node, and push it together with an adjacent edge (line 4), in one of the pQueue's (distributing them in round-robin).

Next, nt worker threads run in parallel Algorithm 2, until a global stop condition: time-out, or until the maximum number of solutions has been reached, or all the queues are empty. Each worker repeatedly picks the highest-priority (tree, edge) pair on its queue (line 2), and applies Grow on it (line 3), leading to a 1-edge larger tree (e.g., t_5 obtained from t_4 in Figure 3). Thus, the stack priority *orders* the possible Grow steps at a certain point during the search; it tends to lead to small solutions being found first, so that users are not surprised by the lack of a connection they expected (and which usually involves few connections). If the Grow result tree had not been found before (this is determined from the history), the worker tries to Merge it with all compatible trees, found within treesByRoot (line 6). The Merge partners (e.g., t_5 and t_{15} in Figure 3) should match different (disjoint) keywords; this condition ensures minimality of the solution. Merge results are repeatedly Merge'd again; the thread switches back to Grow only when no new Merge on the same root is possible. Any newly created tree is checked and, if it matches all query keywords, added to the solution set (and not pushed in any queue). Finally, to balance the load among the workers, if one has exhausted his queue, it retrieves the highest-priority (tree, edge) pair from the queue with most entries, pushing the possible results in its own queue.

As seen above, the threads intensely compete for access to history and treesByRoot. As we demonstrate in Section 6.3, our design allows excellent scalability as the number of threads increases.

6 Experimental evaluation

We now present the results of our experimental evaluation. Section 6.1 presents the hardware and data we used. Section 6.2 studies the impact of extraction policies (Section 4). Section 6.3 analyzes the scalability of P-GAM, focusing on its interaction with the hardware, and demonstrates its significant gains with respect to GAM. Section 6.4 demonstrates P-GAM scalability on a large, real-world graph built for our CoI IJ application.



(a) Synthetic graphs: $chain_k$, $star_{p,k}$

(b) Chain graph scaling

(c) Star graph scaling

Figure 5: Synthetic graphs performance

6.1 Hardware and software setup

We used a server with 2x10-core Intel Xeon E5-2640 v4 CPUs clocked at 2.4GHz, and 192GB of DRAM. We do not use Hyper-Threads, and we bind every CPU core to a single worker thread. As shown in Figure 2, we use ConnectionLens (90% Java, 10% Python) to *construct* a graph out of our data sources, and *store* it in PostgreSQL. Following the processing pipeline, we *migrate* the graph to our novel *in-memory graph engine*, which implements P-GAM. The query engine is a NUMA-aware, multi-threaded C++ application.

6.2 Impact of extraction policies

In this experiment, we loaded a set of 20.000 Pubmed XML bibliographic notices (38.4 MB on disk). This dataset inspired the following extraction policy: the text content of any PubMedArticle.Authors.Author.Name is a Person entity, and that extraction is skipped from the article and journal title, as well as from the article keywords. NER is still applied on author affiliations (rich with Organization and Location entities), as well as on the CoIStatement elements of crucial interest in our context.

By introducing the policy, the extraction time went down from 1199s (no policy) to 716s, yielding a speed-up of about 1.67x. The total loading time was reduced from 1461s to 929s, translating to 1.57x speed-up. As a point of reference, we also noted the time to load (and index) the graph nodes and edges in PostgreSQL; extraction strongly dominates the total time, confirming the practical interest of application-driven policies.

6.3 Scalability analysis

The scalability analysis is performed on synthetic graphs, whose size and topology we can fully control. We focus on two aspects that impact scalability: (i) contention in concurrent access to data structures, and (ii) size of the graph (which impacts the search space). To analyze the behavior of concurrent data structures, we use chain_k graphs, because they yield a big number of intermediate results, shared across threads, even for a small graph. This way, we can isolate the size of the graph from the size of the intermediate results. We repeat every experiment five times, and we report the average query execution time.

We use **two shapes of graphs (each with 1 associated query)**, leading to very different search space sizes (Figure 5a). In both graphs, all the kwd_i for $0 \leq i$ are distinct keywords, as well as the labels of the node(s) where the keyword is shown; no other node label matches these keywords. $\textbf{Chain}_k$ has $2k$ edges; on it, $\{\text{kwd}_0, \text{kwd}_1\}$ has 2^k solutions, since any two neighbor nodes can be connected by an a_i or by a b_i edge; further, $2^{k+1} - 2$ partial (non-solution) trees are built, each containing one keyword plus a path growing toward (but not reaching) the other. $\textbf{Star}_{p,k}$ has p branches, each of which is a line of length k; at one extremity each line has a keyword kwd_i, $1 \leq i \leq p$, while at the other extremity, all lines have kwd_0. As explained in Section 3.1, these nodes are equivalent, one is designated their representative (in the Figure, the topmost one), and the others are connected to it through equivalence edges, shown in red. On this graph, the query $\{\text{kwd}_0, \text{kwd}_1, \ldots, \text{kwd}_p\}$ has exactly 1 solution which is the complete graph; there are $O(k+1)2^p$ partial trees.

Graph	chain_{12}	chain_{13}	chain_{14}	chain_{15}	$\text{star}_{4,1000}$	$\text{star}_{4,2000}$	$\text{star}_{4,3000}$	$\text{star}_{4,4000}$	$\text{star}_{4,5000}$
S	4096	8192	16382	32768	1	1	1	1	1
$T_{PGAM}^{1-clean}$	40	92	215	551	34	78	133	196	242
$T_{PGAM}^{1-query}$ (ms)	3	8	17	46	151	693	1957	4711	8592
T_{PGAM} (s)	1	5	17	83	1	4	11	27	51
T^1 (ms)	160	203	234	315	4063	12580	36261	67984	108960
T (s)	674	900	900	900	60	244	900	900	900

Table 1: Single-thread P-GAM vs. GAM performance.

Single-thread P-GAM vs. GAM. We start by comparing P-GAM, *using only 1 thread*, with the (single-threaded) Java-based GAM, accessing graph edges from a PostgreSQL database. We ran the two algorithms on the synthetic graphs and queries, with a *time-out of 15 minutes*; both could stop earlier if they exhausted the search space. Table 1 shows: the number of solutions S, the time $T_{PGAM}^{1-clean}$ (ms) until the internal data structures have been cleaned and properly prepared for queried, the time $T_{PGAM}^{1-query}$ (ms) until the first solution is found by P-GAM and its total running time T_{PGAM} (s) that includes both cleaning and querying for all solutions, as well as the corresponding times T^1 and T for GAM (Java on Postgres). On these tiny graphs, both algorithms found all the expected solutions, however, even without parallelism, P-GAM is $10\times$ to more than $100\times$ faster. Further, on all but the 3 smallest graphs, GAM did not exhaust its search space in 15 minutes. This experiment demonstrates and validates the expected speed-up of a carefully designed in-memory implementation, even without parallelism (since we restricted P-GAM to 1 thread).

Parallel P-GAM. In the following, we omit the time required for cleaning up the data structures after every iteration, as we want to focus on the scalability of the algorithm. Nevertheless, the time for the maintenance of internal data structures takes less than 5% of the query time for large graphs. On the graphs chain_k for $12 \leq k \leq 15$, we report the exhaustive search time (Figure 5b) for query $\{\text{kwd}_0, \text{kwd}_1\}$ as we increase the number of worker threads from 1 to 20. We see a clear speedup as the number of threads increases, which is around 13x for the graph sizes that we report. The speedup is not linear, because as the size of the intermediate results grows, it exceeds the size of the CPU caches, while threads need to access them at every iteration. Our profiling revealed that, as several threads access the shared data structures, they evict content from the CPU cache that would be useful to other threads. Instead, we did not notice overheads from our synchronization mechanisms. Therefore, *we observe that our parallelization approach using concurrent data structures is beneficial for parallel processing, while partitioning-oblivious.*

To study the scalability of the algorithm with the graph size, we use $\text{star}_{4,k}$ for $k \in \{1K, 2K, 3K, 4K, 5K\}$ and the query $\{\text{kwd}_1, \text{kwd}_2, \text{kwd}_3, \text{kwd}_4\}$. Figure 5c shows the exhaustive search time of P-GAM on these graphs of up to 20.000 nodes, using 1 to 4 threads. We obtain an average speed-up of $3.2\times$ with 4 threads, regardless the size of the graph, which shows that P-GAM scales well for different graph models and graph sizes. After profiling, we observed that the size of the intermediate results impacts the performance, similar to the previous case of the chain graph.

In the above $\text{star}_{4,k}$ experiments, we used up to 4 threads since the graph has a symmetry of 4 (however, threads share the work with no knowledge of the graph structure). When keyword matches are poorly connected, e.g., at the end of simple paths, as in our star graphs, P-GAM search starts by exploring these paths, moving farther away from each keyword; if N nodes match query keywords, up to N threads can share this work. In contrast, as soon as these explored paths intersect, Grow and Merge create many opportunities that can be exploited by different threads. On chain_k, the presence of 2 edges between any adjacent nodes multiplies the Grow and Merge opportunities, work which can be shared by many threads. This is why on chain_k, we see scalability up to 20 worker threads, which is the maximum that our server supports.

Data model	$\|E\|$	$\|N\|$	$\|N_P\|$	$\|N_O\|$	$\|N_L\|$
XML	35,318,110	22,204,487	1,561,352	718,434	147,256
JSON	2,800,959	998,013	133,794	147,431	9,822
HTML	232,675	174,849	5,144	4,479	581
Total	38,351,744	23,377,349	1,700,290	870,344	157,659

Table 2: Statistics on Conflict of Interest application graph.

#	Keywords	T^1	T^{last}	T	S	# DS
1	A1, A2	200	4840	4840	1000	1-6, **5**
2	A1, H1	130	615	615	1000	1-7, **7**
3	A3, I1	1263	20547	60000	13	2-4, **2,3**
4	A4, I2	2860	2866	60000	3	2-3, **3**
5	A5, A6, I3	2602	4203	60000	15	6,8, **8**
6	A7, H2, I2	2385	59131	60000	22	5-9, **6**
7	A8, I2, I4	667	51186	60000	63	4-7, **6**
8	A9, H3, I2	264	59831	60000	516	3-8, **5**
9	H2, I1, P1	1267	60212	60000	148	6-8, **6**
10	A5, A10, I2	19077	23160	60000	9	8, **8**
11	A11, I1, I2, P2	4791	54477	60000	9	5,7-8, **8**
12	A9, I1, I4, I5	6327	55762	60000	38	8-9, 11, **8**
13	A7, I1, I6, P1	1857	3057	60000	8	7, 8, **7,8**
14	A12, I1, P2, H3	21031	55221	60000	24	7, **7**
15	A7, A8, I1, I2, I4	3389	28237	60000	4	7-8,11, **11**

Table 3: P-GAM performance on CoI real-world graph.

6.4 P-GAM in Conflict of Interest application

We now describe experiments on actual application data.

The graph. We selected sources based on S. Horel's expertise and suggestions, as follows. (i) We loaded more than **450.000** PubMed bibliographic notices (**XML**), corresponding to articles from 2019 and 2020; they occupy **934 MB** on disk. We used the same extraction policy as in Section 6.2 to perform only the necessary extraction. (ii) We downloaded almost **42,000** PDF articles corresponding to these notices (those that were available in Open Access), transformed them into **JSON** using an extraction script we developed, and preserved only those paragraphs starting with a set of keywords ("Disclosure", "Competing Interest", "Acknowlegments" etc.) which have been shown [1] to encode potentially interesting participation of people (other than authors) and organizations in an article. Together, these JSON fragments occupy **340 MB** on disk. *The JSON and the XML content from the same paper are connected (at least) through the URI of that paper, as shown in Figure 1.* (iii) We crawled 781 **HTML** Web pages from a set of Web sites describing people and organizations previously involved in scientific expertise on sensitive topics (such as tobacco or endocrine disruptors), including: www.desmogblog.com, tobaccotactics.org, www.wikicorporates.org and www.sourcewatch.org. These pages total **24 MB**. Table 2 shows the numbers of edges ($|E|$), of nodes ($|N|$), and of Person, Organization and Location entities ($|N_P|$, $|N_O|$, $|N_L|$), split by the data model, and overall.

Querying the graph. Table 3 shows the results of executing 15 queries, until **1000 solutions** or for at most **1 minute**, using P-GAM. From left to right, the columns show: the query number, the query keywords, the time T^1 until the first solution is found, the time T^{last} until the last solution is found, the total running time T, the number of solutions found, and some statistics on the number of data sources participating in the solutions found ($\#DS$, see below). All times are in milliseconds. We have anonymized the keywords that we use, not to single

out individuals or corporations, and since the queries are selected aiming not at them, but at a large variety of P-GAM behavior. We use the following codes: **A** for author, **H** for hospital, **P** for country, and **I** for industry (company). A $\#DS$ value of the form "2-10, **6**" means that P-GAM found solutions spanning at least 2 and at most 10 data sources, while most solutions spanned over 6 sources.

We make several observations based on the results. The stop conditions were set here based on what we consider as an interactive query response time, and a number of solutions which allow further exploration by the users (e.g., through an interactive GUI we developed). Further, solutions span over several datasets, demonstrating the interest of multi-dataset search enabled, and that P-GAM exploits this possibility. Finally, we report results after performing queries including different numbers of keywords and the system remains responsive within the same time bounds, despite the increasing query complexity.

7 Related Work and Conclusion

In this paper, we presented a complete pipeline for managing heterogeneous data for IJ applications. This innovates upon recent work [3] where we have addressed the problems of integrating such data in a graph and querying it, as follows: (i) we present a complete data science application with clear societal impact, (ii) we show how extraction policies improve the graph construction performance, and (iii) we introduce a parallel search algorithm which scales across different graph models and sizes. Below, we discuss prior work most relevant with respect to the contributions we made here; more elements of comparison can be found in [3].

Our work falls into the *data integration* area [4]; our IJ pipeline starts by ingesting data into an integrated data repository, deployed in PostgreSQL. The first platform we proposed to Le Monde journalists was a mediator [24], resembling polystores, e.g., [7, 25]. However, we found that: (i) their datasets are changing, text-rich and schema-less, (ii) running a set of data stores (plus a mediator) was not feasible for them, (iii) knowledge of a schema or the capacity to devise integration plan was lacking. ConnectionLens' first iteration [26] lifted (iii) by introducing keyword search, but it still kept part of the graph *virtual*, and split keyword queries into subqueries sent to sources. Consolidating the graph in a single store, and the centralized GAM algorithm [3] greatly sped up and simplified the tool, whose performance we again improve here. We share the goal of exploring and connecting data, with *data discovery* methods [10, 27–29], which have mostly focused on tabular data. While our data is heterogeneous, focusing on an IJ application partially eliminates risks of ambiguity, since in our context, one person or organization name typically denote a single concept.

Keyword search has been studied in XML [30, 31], graphs (from where we borrowed Grow and Merge operations for GAM) [32, 33], and in particular RDF graphs [34, 35]. However, our keyword search problem is harder in several aspects: (i) we make no assumption on the shape and regularity of the graph; (ii) we allow answer trees to explore edges in both directions; (ii) we make no assumption on the score function, invalidating Dynamic Programming (DP) methods such as [31] and other similar prunings. In particular, we show in [13] that *edges with a confidence lower than 1*, such as similarity and extraction edges in our graphs, compromise, for any "reasonable" score function which reflects these confidences, the *optimal substructure* property at the core of DP. Works on *parallel keyword search in graphs* either consider a different setting, returning a certain class of subgraphs instead of trees [36] or standard graph traversal algorithms like BFS [37–39]. To the best of our knowledge, GAM is the first keyword search algorithm for the specific problem that we consider in this paper. Accordingly, in this paper we have parallelized GAM, into P-GAM, by drawing inspiration and addressing common challenges raised in graph processing systems in the literature, in particular concerning the CPU efficiency while interacting with the main memory [19–22, 40].

Acknowledgments. The authors thank M. Ferrer and the Décodeurs team (Le Monde) for introducing us, and for many insightful discussions.

References

[1] "European Press Prize: the Monsanto Papers," 2018.

[2] "Offshore Leaks," 2013.

[3] A. G. Anadiotis, O. Balalau, C. Conceição, H. Galhardas, M. Y. Haddad, I. Manolescu, T. Merabti, and J. You, "Graph integration of structured, semistructured and unstructured data for data journalism," *Information Systems*, In Press, 2021.

[4] A. Doan, A. Y. Halevy, and Z. G. Ives, *Principles of Data Integration*. Morgan Kaufmann, 2012.

[5] D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Lembo, A. Poggi, and R. Rosati, "MASTRO-I: efficient integration of relational data through DL ontologies," in *DL Workshop*, 2007.

[6] M. Buron, F. Goasdoué, I. Manolescu, and M. Mugnier, "Obi-wan: Ontology-based RDF integration of heterogeneous data," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 2933–2936, 2020.

[7] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, and S. B. Zdonik, "The BigDAWG polystore system," *SIGMOD*, 2015.

[8] R. Alotaibi, D. Bursztyn, A. Deutsch, I. Manolescu, and S. Zampetakis, "Towards scalable hybrid stores: Constraint-based rewriting to the rescue," in *SIGMOD*, 2019.

[9] A. Quamar, J. Straube, and Y. Tian, "Enabling rich queries over heterogeneous data from diverse sources in healthcare," in *CIDR*, 2020.

[10] M. Ota, H. Mueller, J. Freire, and D. Srivastava, "Data-driven domain discovery for structured datasets," *Proc. VLDB Endow.*, vol. 13, no. 7, pp. 953–965, 2020.

[11] C. Christodoulakis, E. Munson, M. Gabel, A. D. Brown, and R. J. Miller, "Pytheas: Pattern-based table discovery in CSV files," *Proc. VLDB Endow.*, vol. 13, no. 11, pp. 2075–2089, 2020.

[12] F. Nargesian, K. Q. Pu, E. Zhu, B. G. Bashardoost, and R. J. Miller, "Organizing data lakes for navigation," in *SIGMOD*, 2020.

[13] A. G. Anadiotis, M. Y. Haddad, and I. Manolescu, "Graph-based keyword search in heterogeneous data sources," in *Bases de Données Avancés (informal publication)*, 2020.

[14] N. Oreskes and E. Conway, *Merchants of Doubt*. Bloomsbury Publishing, 2012.

[15] S. Horel, *Lobbytomie*. La Découverte, 2018. In French.

[16] S. Horel, "Petites ficelles et grandes manoeuvres de l'industrie du tabac pour réhabiliter la nicotine," 2020. In French.

[17] "Physician Payments Sunshine Act," 2010.

[18] P. A. Boncz, S. Manegold, and M. L. Kersten, "Database architecture optimized for the new bottleneck: Memory access," in *VLDB*, 1999.

[19] N. Elyasi, C. Choi, and A. Sivasubramaniam, "Large-scale graph processing on emerging storage devices," in *USENIX FAST*, 2019.

[20] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *ISCA*, 2015.

[21] A. Roy, I. Mihailovic, and W. Zwaenepoel, "X-stream: edge-centric graph processing using streaming partitions," in *SOSP*, 2013.

[22] S. Hong, S. Depner, T. Manhardt, J. V. D. Lugt, M. Verstraaten, and H. Chafi, "PGX.D: a fast distributed graph processing engine," in *SC*, 2015.

[23] C. Binnig, S. Hildenbrand, and F. Färber, "Dictionary-based order-preserving string compression for main memory column stores," in *SIGMOD*, 2009.

[24] R. Bonaque, T. D. Cao, B. Cautis, F. Goasdoué, J. Letelier, I. Manolescu, O. Mendoza, S. Ribeiro, X. Tannier, and M. Thomazo, "Mixed-instance querying: a lightweight integration architecture for data journalism," *Proc. VLDB Endow.*, vol. 9, no. 13, pp. 1513–1516, 2016.

[25] B. Kolev, P. Valduriez, C. Bondiombouy, R. Jiménez-Peris, R. Pau, and J. Pereira, "Cloudmdsql: querying heterogeneous cloud data stores with a common language," *Distributed Parallel Databases*, vol. 34, no. 4, pp. 463–503, 2016.

[26] C. Chanial, R. Dziri, H. Galhardas, J. Leblay, M. L. Nguyen, and I. Manolescu, "Connectionlens: Finding connections across heterogeneous data sources," *Proc. VLDB Endow.*, vol. 11, no. 12, pp. 2030–2033, 2018.

[27] A. D. Sarma, L. Fang, N. Gupta, A. Y. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu, "Finding related tables," in *SIGMOD*, 2012.

[28] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker, "Aurum: A data discovery system," in *ICDE*, 2018.

[29] R. C. Fernandez, E. Mansour, A. A. Qahtan, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang, "Seeping semantics: Linking datasets using word embeddings for data discovery," in *ICDE*, 2018.

[30] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, "XRANK: ranked keyword search over XML documents," in *SIGMOD*, 2003.

[31] Z. Liu and Y. Chen, "Identifying meaningful return information for XML keyword search," in *SIGMOD*, 2007.

[32] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding top-k min-cost connected trees in databases," in *ICDE*, 2007.

[33] H. He, H. Wang, J. Yang, and P. S. Yu, "BLINKS: ranked keyword searches on graphs," in *SIGMOD*, 2007.

[34] S. Elbassuoni and R. Blanco, "Keyword search over RDF graphs," in *CIKM*, 2011.

[35] W. Le, F. Li, A. Kementsietsidis, and S. Duan, "Scalable keyword search on large RDF data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 11, pp. 2774–2788, 2014.

[36] Y. Yang, D. Agrawal, H. V. Jagadish, A. K. H. Tung, and S. Wu, "An efficient parallel keyword search engine on knowledge graphs," in *ICDE*, 2019.

[37] S. Hong, T. Oguntebi, and K. Olukotun, "Efficient parallel graph exploration on multi-core CPU and GPU," in *PACT*, 2011.

[38] L. Dhulipala, G. E. Blelloch, and J. Shun, "Julienne: A framework for parallel graph algorithms using work-efficient bucketing," in *SPAA*, 2017.

[39] C. E. Leiserson and T. B. Schardl, "A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers," in *SPAA*, 2010.

[40] J. Malicevic, B. Lepers, and W. Zwaenepoel, "Everything you always wanted to know about multicore graph processing but were afraid to ask," in *USENIX ATC*, 2017.

Fact-Checking Statistical Claims with Tables

Mohammed Saeed and Paolo Papotti
EURECOM, France

Abstract

The surge of misinformation poses a serious problem for fact-checkers. Several initiatives for manual fact-checking have stepped up to combat this ordeal. However, computational methods are needed to make the verification faster and keep up with the increasing abundance of false information. Machine Learning (ML) approaches have been proposed as a tool to ease the work of manual fact-checking. Specifically, the act of checking textual claims by using relational datasets has recently gained a lot of traction. However, despite the abundance of proposed solutions, there has not been any formal definition of the problem, nor a comparison across the different assumptions and results. In this work, we make a first attempt at solving these ambiguities. First, we formalize the problem by providing a general definition that is applicable to all systems and that is agnostic to their assumptions. Second, we define general dimensions to characterize different prominent systems in terms of assumptions and features. Finally, we report experimental results over three scenarios with corpora of real-world textual claims.

1 Introduction

Large scale spreading of incorrect information on the internet is a real threat that poses severe societal problems [30]. As no barriers exist for publishing information, it is possible for anyone to diffuse false or biased claims and reach large audiences with ease [6]. This raises the important issue of how to tame the spread of false information, as this has affected public votes[1] and has misinformed people about coronavirus remedies[2] and spread[3]. Accordingly, there has been a great demand for fact-checkers to efficiently verify such news.

Indeed, with the easy accessibility of large social networks and the advent of generating text using recent advances in Natural Language Processing (NLP) [12,32], the surge of false news has overpowered the capabilities of manual fact-checking. Malicious users in social networks are still allowed to profit from misinformation and the affected networks have just started to take effective actions [1]. At the beginning of the COVID-19 pandemic, the spread of false coronavirus news has urged the World Health Organization to spotlight this issue, labeling it as an *infodemic* [2]. One approach to deal with this enormous volume of information is computational fact-checking [34], where parts of or the entire verification pipeline is automated, usually including some ML algorithms [25]. One influential system is ClaimBuster [14], which is an end-to-end fact-checking solution that relies on NLP and supervised algorithms to identify and check factual and false information. Since then, there has been a stream of fact-checking systems.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

[1] https://www.ucf.edu/news/how-fake-news-affects-u-s-elections/
[2] https://fullfact.org/health/honey-ginger-pepper-WHO/
[3] https://fullfact.org/health/indian-variants-sequencing/

Player	Minutes Played	Field Goals	Field Goal Attempts	Assists	Points$^{(3)}$
Courtney Lee	39:08	9	14	3	22
Marc Gasol	35:23	$6^{(2)}$	$12^{(2)}$	6	18
Zach Randolph	29:26	4	9	0	10
Mike Conley	29:13	9	14	$11^{(1)}$	$24^{(4)}$
Tony Allen	23:10	4	6	1	9
Quincy Pondexter	26:43	2	8	0	7
Beno Udrih	18:47	3	6	3	6
Jon Leuer	16:13	1	4	0	2
Kosta Koufos	12:37	0	2	1	0
Vince Carter	9:20	2	5	0	4

Table 4: Statistics of a basketball game.

Some systems verify a certain input by utilizing structured data stored in *knowledge graphs* [5,16] or *relational tables* (or just *relations*) [18, 19], while others rely on unstructured resources such as Wikipedia articles [27, 36]. Other systems have the ability to verify multiple claims occurring within the same input text, such as an entire document [15, 19]. A recent approach also discards all evidence retrieval methods and relies on the implicit knowledge stored in large pre-trained language models (PLM) [20]. Nevertheless, the plethora of different methods calls for a thorough study of their differences with an exploration of the salient aspects related to the design of the systems and how they relate to the fact-checking process. Such study aims to (i) provide readers with a set of dimensions to model this kind of systems, (ii) inspect prominent systems by testing them on various datasets.

We are interested in claims that can be verified by using existing relational tables. Storing data in tables is the go-to format for many applications as it offers declarative querying, scalability, and several other features. For example, reliable statistics for coronavirus are published on a daily basis as relations [4]. The use of such tables supports the verification process and can help in relieving the work done by human fact-checkers. Indeed, manually verifying textual reports, which summarize the most important statistics, is time-consuming and requires automation [3, 25].

In this article, we start with formulating the problem of the computational verification of textual claims by using relational data (Section 2). We discuss four recent systems and highlight their main differences in terms of six generic dimensions (Section 3). We then experimentally compare the systems on several annotated claim corpora (Section 4). Finally, we conclude with some open challenges and future directions for research in this topic (Section 5).

2 Problem Statement

We introduce our problem and related terminology. We assume a scenario where we have a natural language text containing one or more claims to be verified with some relational table(s). Such table(s) are either given as an input or predicted by a system. The following example contains (hypothetical) claims about a basketball game. Table 4[5] contains the information needed to verify such claims. Table values that are used to verify a claim are marked with the same superscript.

Example 1: "Mike Conley had $11^{(1)}$ assists. The field goal percentage for Marc Gasol is $50\%^{(2)}$. The team scored $100^{(3)}$ points in total. Mike Conely scored the $most^{(4)}$ points."

[4] `https://github.com/CSSEGISandData/COVID-19`

[5] Obtained from `https://www.basketball-reference.com/boxscores/201411050PHO.html`

The first claim can be verified with a simple look-up in the table over the **Assists** attribute. The second claim can be verified by computing the ratio of the **Field Goals** to the **Field Goal Attempts** of a certain player; thus, two cell values are needed for verification. The third (false) claim can be checked with an aggregation (summation) over the **Points** column. The fourth claim involves finding the player with the maximum number of **Points**. Other claims might need the involvement of two or more tables.

Definition 1: *Fact-checking* is the process of checking that all facts in a piece of text are correct.

Fact-checking is usually composed of three phases: (i) finding check-worthy claims, (ii) finding the best available evidence for the claim at hand, and (iii) validating or correcting the record by evaluating the claim in light of the evidence [25]. Claims can be provided in a structured form, such as the subject-predicate-object triples in a knowledge graph [5], or in plain text [31], such as the sentences in Example 1. In this article, we assume that the first step (i) has already been executed, and every sentence contains at least one claim.

Definition 2: A *textual claim* is any subset of a natural language input that is to be verified against trustworthy reference sources.

Data in such reference sources can be structured or non-structured. Non-structured data include textual documents while structured data include knowledge graphs, such as DBpedia [21], and relational tables. In this work, we are interested in relational tables as reference data. Specifically, we focus on tables that contain numerical data and on which numerical and Boolean operations can be computed.

Definition 3: A *statistical claim* is any textual claim that can be verified over a trustworthy database by performing a mathematical expression on the database cell values.

The claims in Example 1 are all statistical claims, while a claim such as “Players who commit too many fouls are not allowed to play anymore." is not.

Definition 4: An *explicit claim* is a statistical claim mentioning a number that can be verified by comparing it against the result of a function that takes as parameters some cell values in the input relation.

The first three claims in Example 1 are explicit claims. We assume that symbols `LOOKUP`, `SUM`, and `DIVISION` are defined. `LOOKUP` performs a look-up in a table given a primary key value and an attribute label. `SUM` performs a summation over the values of an attribute. `DIVISION` performs the division of two cell values. The first claim is a simple look-up over the table that could be modeled as `LOOKUP('Mike Conley','Assists')==11`. The second claim requires computing a ratio of Field Goals to Field Goal Attempts for player Marc Gasol. This can be formulated as `DIVISION(a,b)==0.5` where `a=LOOKUP('Marc Gasol','Field Goals')` and `b=LOOKUP('Marc Gasol','Field Goal Attempts')`. The third claim could be modeled as `SUM('Points')==100`.

Definition 5: An *implicit claim* is a statistical claim that does not mention a number and can be verified by a Boolean function that takes as parameters cell values in the input relation.

The last claim in Example 1 is an implicit claim. Assuming MAX has also been defined, it could be modeled as `MAX(Points)=LOOKUP('Mike Conely','Points')`. As we will discuss in the next section, implicit claims are harder to verify and usually require some form of supervised learning, such as the learning of neural representations [19] or the synthesis of a program from the input [8].

Definition 6: Given a text T containing a statistical claim c and a database D, the goal of *Statistical-Claim Fact-Checking* is to verify c with the information in D. Formally, the objective is to find a function $f(T, c, D)$ that successfully maps to one of two labels (**True** or **False**).

This definition is generic enough to model existing fact-checking systems and other systems that can be adapted for this task[6]. It is independent of the different assumptions that apply for the different approaches. For example, multiple systems assume that the input text T contains a single claim [8], thus dropping the need for having the claim c as an input. Also, the database D is often simplified to one relational table given as input [8, 15], while other systems utilize multiple tables [18, 19]. Finally, the definitions above do not cover a notion of explainability of the provided result. Indeed, some systems do not provide any result explanation since the verification process relies on black-box methods, such as deep neural networks [8].

3 Systems

In this section, we study four systems that satisfy Definition 6. We analyze TABLE-BERT [8], TAPAS [15], AGGCHECKER [18], and SCRUTINIZER [19]. In Section 3.1, we describe how each system works and its assumptions. In Section 3.2, we introduce six dimensions to characterize such fact-checking systems.

3.1 Overview of Systems

We introduce the systems, starting from the ones that rely on end-to-end NLP methods, and describe those that use query generation next.

TABLE-BERT [8] models fact-checking of a statistical claim as a Natural Language Inference (NLI) problem [22]. NLI is the task of determining whether a natural language hypothesis h can be inferred from a natural language premise p. In TABLE-BERT, a given table **T** is linearized and fed to the model alongside the natural language hypothesis p. The model consists of a pre-trained BERT model [12] that outputs a sequence-level representation of the input. This representation is then fed into a multi-layer perceptron, which predicts the entailment probability. If the output probability is greater than 0.5, then the hypothesis p is entailed by table **T**.

This system assumes a table as input, i.e., that the reference data is available and already identified. TABLE-BERT comes with a corpus of tables and claims (annotated as true/false) that can be used for fine-tuning. This makes it usable on unseen tables, but our experimental results show that further fine-tuning for the domain at hand is needed to obtain good results. Moreover, TABLE-BERT can be fine-tuned with more examples that contain formulas unseen in the provided corpus. However, the original paper recognizes that the complexity of the formulas that the system can learn is limited and does not support composition of functions [8].

TAPAS [15] can be used to tackle the claim verification as a question-answering problem over an input table [4, 23, 33]. The model takes as input (1) a natural language question **Q** to be answered over (2) the input table **T**. Building on the success of pre-training models on textual data, TAPAS extends this procedure to structured data, by training a BERT [12] model on a large number of natural language questions and Wikipedia tables. This process enables the model to learn correlations between structured and unstructured data. After training, the encoder provides a representation of the input. The output is twofold: the model predicts (1) which *cell values* of the input table are used for answering the question and (2) what *aggregation operation* is performed on such values to produce an answer for the input question.

As with TABLE-BERT [8], TAPAS assumes that the reference table is available, and is linearized in the input. Generating questions from the claim could be done using lexical-based methods as pioneered in ClaimBuster [13], or neural-based methods [7, 35]. While TAPAS has the benefit of being general, i.e., “plug and play" on new domains, it has the limitation that extending it to new formulas or tables requires the full re-training from scratch. Moreover, it is not demonstrated that it could learn complex formulas.

[6]While the definition considers a binary label for the output, it can be extended to multiple labels such as “partially true” or “not enough evidence".

AGGCHECKER [18] takes a relational dataset and a text document as input. It translates the natural language claims in the document into SQL queries that are executed to verify the claims. More specifically, each claim is mapped to a probability distribution over SQL queries. SQL queries are formed by combining query fragments using an iterative expectation-maximization procedure [11]. AGGCHECKER works out of the box on unseen relations and does not assume that training data is available for a new database. The system supports aggregation functions and could be extended to support more. Extending the system is not trivially done by just feeding more binary examples. It needs an update to the information retrieval engine to incorporate new query fragments, and an update to the probabilistic model to account for new SQL query candidates. A module to account for multi-variable formulas is also needed. Similar to TAPAS, the modification needed for this system to account for unseen functions goes beyond examples.

The system benefits from the fact that claims in the same context are often semantically correlated by learning a document-specific prior distribution over queries. As in practice accurate claims are more likely than inaccurate claims, the system increases the likelihood of the query which has a match between the query result and the claim. As multiple candidate queries are to be executed, an execution engine that merges execution of similar queries is used for efficiency.

SCRUTINIZER defines fact-checking as a mixed-initiative verification problem [19]. The approach combines feedback from human workers with automated verification coming from ML classifiers. We neglect the human-in-the-loop part in this article and focus solely on the automatic verification. The system is based on four classifiers that take a statistical claim as input and predict (i) the relation(s) to be used, (ii) the primary key value(s), (iii) the attribute label(s), and (iv) the formula applied on the cells identified by the former three. In contract with other systems, the table is not given as input, but is predicted, and the cell selection is based on the predicted primary key values and attribute labels for such a table. This leaves out the need for inputting the table to the model, but limits the current system only to the table schemas seen during training. After cell selection is done, it can apply the predicted formula and verify the input claim.

SCRUTINIZER can learn any query, including complex formulas, from the training data. However, the price to pay for this generality is that it trains the classifiers, therefore labels for these must be provided, and it does not suffice to have the true/false label for the claim as in TABLE-BERT.

Aside from AGGCHECKER, all the systems use transformer-based language models [12] to encode language knowledge, but only TAPAS requires the expensive pre-training of such models. AGGCHECKER relies on a probabilistic model to map natural language claims to a probability distribution over queries. Others solutions rely on synthesizing a logical program [8], recurrent-based language models [26], reinforcement-learning approaches [37], and graph neural-networks [24].

Type	Dimension	AGGCHECKER	TAPAS	TABLE-BERT	SCRUTINIZER
Input	Implicit Claims		✓	✓	✓
	Schema-Independence	✓	✓	✓̸	
	Multi-variable Formulas	✓̸	✓̸		✓
	Multi-tables	✓̸			✓
Output	Interpretability	✓	✓̸		✓
	Alternative Interpretations	✓			✓

Table 5: Dimensions that characterize the systems (✓̸ denotes partial support).

3.2 System Dimensions

We believe that, given the increasing number of fact-checking systems, it is important to start characterizing them with clear dimensions to enable a more rigorous comparison. We first describe four main dimensions that

characterize the input across the different proposals. Then, we discuss two dimensions that characterize the output. A summary of the dimensions and how systems support them is reported in Table 5.

3.2.1 Input Dimensions

Explicit claims are handled by all fact-checking systems, as they are much easier to deal with. However, the support for **Implicit Claims** requires a deeper understanding of the semantics behind the given sentence. One approach to dealing with this problem is feature-based entity linking where all entities are detected in the input statement and a set of pre-defined trigger-words are used to build programs representing the semantics of the statement [8]. However, such approaches are very sensitive to the error-prone entity linking process. Another approach is to learn such implicit claims in a supervised manner. SCRUTINIZER learns from the classifiers' labels [19]. TAPAS also learns correlations between the text and the table during the pre-training process.

Another dimension is **Schema-Independence**. AGGCHECKER, TABLE-BERT and TAPAS can consume potentially any table with any unseen schema, while SCRUTINIZER is limited to tables whose row index values and attribute labels have been trained on. For SCRUTINIZER, adding new tables requires fine-tuning the classifiers. The operation is not expensive in terms of execution time, because its classifiers are based on a fine-tuning procedure, rather than having to pre-train again from scratch; however, it requires specific annotations that go beyond the true/false binary label. This dimension highlights that SCRUTINIZER is domain-specific and thus has to learn the related tables for the task at hand, while AGGCHECKER and TAPAS try to be agnostic of the table schema, and can handle any table as input. For TABLE-BERT, while it can be used on any unseen schema, our experiments show that it should be trained on the examples at hand in order to obtain good accuracy performance.

In practice, computations involving values of a database go beyond simple look up and aggregation functions such as those reported in Example 1. The function for the verification of a claim can require complex **Multi-variable Formulas**. For example, the Compound Annual Growth Rate[7] is a formula needed to verify a claim in our experiments. SCRUTINIZER handles complex formulas on the condition that they are observed in its classifier-specific training data, and resorts to a brute-force approach to assign predicted values to variables. AGGCHECKER can be extended to handle complex aggregation functions. TAPAS handles aggregate queries with simple functions where the cell values have been selected by the model. It is not clear if and how TAPAS could support functions with more than one variable, and it would require training again the model from scratch such that new functions are learned. Finally, TABLE-BERT has no explicit notion of formulas, as it is a black-box model fine-tuned end-to-end on a binary classification task. According to the original paper and our experiments, TABLE-BERT struggles to learn how to handle formulas with multiple variables.

TABLE-BERT and TAPAS assume that the right table to verify the input claim is also given as input. In practice, many tables can be available and the most likely one for the task at hand is identified by SCRUTINIZER and AGGCHECKER (**Multi-Tables**). Moreover, in some cases more than one table is needed to verify a claim and only SCRUTINIZER supports verification that requires the combination of values from multiple tables. This dimension highlights one of the limits of the methods that rely on the linearized data fed to the transformers, as it is hard to feed multiple tables without hitting the limit on the size of the input.

3.2.2 Output Dimensions

Interpretability is a key dimension supported by methods that output the query used to verify the claim. However, systems using a black-box model to verify claims, such as TABLE-BERT, lack interpretability as an explanation of the prediction is not provided. There do exist methods attempting to explain black-box models which include explanations by simplification [29]. However, there is no consistent method to define how faithful are the

[7] It describes the net gain or loss of an investment over a certain period of time (`https://en.wikipedia.org/wiki/Compound_annual_growth_rate`).

explanations to the model prediction [17]. TAPAS is not fully interpretable since it provides only cell values and, in some cases, the aggregation operation. AGGCHECKER and SCRUTINIZER expose the declarative query used to verify the associated claim. Systems that predict query fragments and combine them, rather than producing an answer in one shot, are easier to interpret [10].

Claims expressed in natural language can be incomplete or ambiguous in many ways. Some systems support **Alternative Interpretations** to clarify how the output changes depending on the details of the verification. Consider a simple claim "Mike scored 30 points", and a table with two players whose first name is "Mike". The claim is true for one player, but false for the other. AGGCHECKER resolves such ambiguities by evaluating multiple queries and soliciting feedback from users. SCRUTINIZER learns ambiguities conditioned that they are represented in the training data. TAPAS and TABLE-BERT do not include any clear means to resolve this kind of ambiguities, as they default to one interpretation in the current architectures.

4 Experimental Evaluation

We evaluated the four systems above by using three datasets with real textual claims manually annotated with the correct checking label. We concatenate the claim to the sentence in case the sentence has multiple claims; otherwise, we only input the sentence. For every system, we report its coverage of the claims, the accuracy of the verification process, and the execution times.

		SCRUTINIZER Labels				TABLE-BERT Label
Sentence	Claim	Table	Attribute Label	Primary Key Value	Formula	Verdict
There were 800 total deaths in China in May 2021.	800 total deaths	total_deaths	May_2021	China	a	False

Table 6: Labeled data for SCRUTINIZER and TABLE-BERT.

4.1 Datasets

Our experiments are based on three use cases: Coronavirus scenario (C19), International Energy Agency (IEA) scenario, and Basketball Data scenario (BBL). Every example contains the textual claim, the relational table to verify it, and the outcome expressed as a binary label True/False. For SCRUTINIZER, the training examples contain also the labels for the four classifiers. For a fair comparison, we fed the associated relation as input to all systems. Given the limitations on the input size in TAPAS, we limit the input for this system to at most a sample of 11 tuples, including the one needed to verify the claim. Without this ad-hoc operation, the system fails with the entire relation as input. An example of our labeled data is shown in Table 6.

For C19, we generated the training data from the relations (3M examples [19]) and used real claims for the testing. We denote the synthetic corpus as $\mathbf{C19_{train}}$. For testing the system with unseen claims, we analyzed the log of more than 30K claims tested by users on a website[8]. We found that more than 60% of the claims in such corpus are statistical and, among those, we have the datasets to verify 70%. From these claims, we randomly selected 55 claims and manually annotated them ($\mathbf{C19_{test}}$).

For IEA, we obtained a document of 661 pages, containing 7901 sentences, and the corresponding corpus of manually checked claims, with check annotations for every claim from three domain experts. The annotations cover 2053 numerical claims, out of which we identified 1539 having a formula that occurs at least five times in the corpus. We denote the resulting dataset as $\mathbf{IEA_{train}}$. After processing the claims, we identify 1791 relations, 830 row indexes, 87 columns, and 413 formulas. Around 50% of the values for all properties appear at most 10

[8] `https://coronacheck.eurecom.fr`

times in the corpus, with the top 5% most frequent formulas appearing at least 8 times. For the test data ($\mathbf{IEA_{test}}$), we randomly selected 20 claims from the most common operations (look up and sum).

For BBL, we use the data in a recent publication [28][9]. We use the 1523 real annotated claims provided in the repository for the testing step and generate ourselves the training data from the tables as in the C19 scenario. We generate an initial dataset of 32.3K samples, where 90% is used for training classifiers and 10% for validation. We use 132 tables for this scenario. The dataset used for bootstrapping is denoted by $\mathbf{BBL_{train}}$ and the test dataset as $\mathbf{BBL_{test}}$. Our datasets ($\mathbf{BBL_{test}}$ and $\mathbf{C19_{test}}$) are publicly available[10].

Table 7: Ratio of supported training claims.

	AGGCHECKER	TAPAS	TABLE-BERT	SCRUTINIZER
$\mathbf{C19_{train}}$	21.49%	21.49%	37.82%	100%
$\mathbf{IEA_{train}}$	33.06%	17.02%	27.28%	74.96%
$\mathbf{BBL_{train}}$	53.00 %	56.17%	56.17%	56.17%

As discussed in Section 5, the tested systems have limitations on the input data and on the space of supported formulas. These limitations are reflected in the percentage of training claims that every system can handle, as reported in Table 7. For example, complex formulas are present in our datasets, with more than 22% of the claims in $\mathbf{IEA_{train}}$ with three or more variables.

4.2 Experimental Results

For TABLE-BERT, we fine-tuned the binary classifier on top of the PLM with the training data after augmenting the data to ensure a balance between classes. For TAPAS, we tried to automatically translate the claims to questions as pioneered by ClaimBuster [13]. However, the precision was not satisfactory, e.g., we could not obtain any questions for 7 out of the 20 IEA test claims. To overcome this issue, we manually translated claims into questions for $\mathbf{IEA_{test}}$ and $\mathbf{C19_{test}}$, and relied on a pattern-based script to generate questions for $\mathbf{BBL_{test}}$. For TAPAS and AGGCHECKER, we did not run any training. For SCRUTINIZER, we provided the examples with the labels for the 4 classifiers, and examples with binary labels for TABLE-BERT. For SCRUTINIZER, we do not rely on the user feedback in this experiment.

Table 8: Verification accuracy on the test datasets.

	AGGCHECKER	TAPAS	TABLE-BERT	SCRUTINIZER
$\mathbf{C19_{test}}$	0.44	0.64	0.76	0.80
$\mathbf{IEA_{test}}$	0.50	0.07	0.58	0.65
$\mathbf{BBL_{test}}$	0.13	0.41	0.17	0.51

Table 8 reports the accuracy results of the experiments with all systems on the test claims. $\mathbf{IEA_{train}}$ is heavily skewed as, for instance, there are formulas such as lookups and summations that are commonly used, unlike formulas comprising multiple variables which are scarce (formulas having ten or more variables form 4.32% of the training data). The formulas in $\mathbf{IEA_{test}}$ are different, as they contain functions supported by all systems

[9] `https://github.com/ehudreiter/accuracySharedTask`; sentences and claims in this corpus are similar to the ones reported in Example 1.

[10] `https://zenodo.org/record/5128604#.YPrSgXUzZuU`

Table 9: Execution time of the test datasets (seconds).

	AGGCHECKER	TAPAS	TABLE-BERT	SCRUTINIZER
C19$_{\text{test}}$	280.41	991.52	23.09	0.03
IEA$_{\text{test}}$	321.53	943.25	18.11	0.68
BBL$_{\text{test}}$	3472.44	12709.45	40.20	236.64

(lookups and summations in this case). For **BBL$_{\text{test}}$**, low results are explained by the fact that all systems have low coverage of the claims in the data and some claims require verification that spans across multiple tables. For **C19$_{\text{test}}$**, the systems do slightly better as the test data comprises lookups and the attribute label and primary key value are usually explicitly stated in the sentence. We observe that the systems perform with mixed results, and none of them can get high accuracy in all cases. We can observe that in most cases the use of training data can lead to the best performance. This is evident for TABLE-BERT, which performs well in two datasets by making use of the true/false labels, and in SCRUTINIZER, which exploits the rich annotations for its classifiers and leads in all scenarios. However, for **C19$_{\text{test}}$**, SCRUTINIZER fails for claims which require formulas that it has not seen in the training data. For running **BBL$_{\text{test}}$** with TAPAS and TABLE-BERT, we replaced abbreviations in the schemas of the table by their proper wording (for example, “PTS" was replaced by “Points"). This has improved the TAPAS accuracy from 0.19 to 0.41 and TABLE-BERT accuracy from 0.09 to 0.17. This is expected as such models, which have been trained on text and table together, correlate better a table schema containing “Points" with the input text compared to the acronym “PTS".

For the **execution times**, we distinguish the training and the testing. Classifier-training time is needed for SCRUTINIZER and TABLE-BERT; however, this is typically negligible (on the studied datasets) with the usage of GPUs. AGGCHECKER and TABLE-BERT, on the other hand, have zero setup time. We report the execution times for all test data in Table 9. TAPAS is the slowest as the model is jointly computing the relevant cells and performing an operation on them, compared to TABLE-BERT that requires a negligible amount of time to perform binary classification. SCRUTINIZER consumes negligible time in classifier predictions, but the brute-force query generation process could potentially take considerable amount of time when multiple combinations are available. AGGCHECKER, although having to perform evaluations of a large number of queries, successfully merges the execution of similar queries to increase efficiency. In summary, all systems are usable in reasonable time in our experience in an entry-level infrastructure with a low-end GPU.

5 Conclusions

We focused our study on the problem of fact-checking a statistical claim with relational tables as reference data and considered four prominent systems. We make a first step towards categorizing fact-checking systems with generic dimensions. We have also experimentally evaluated the four systems on three use cases and gathered many observations on their coverage, their qualitative performance and their execution times.

Our results and the proposed categorization can act as a blueprint when designing a system, as different applications have different requirements. For example, text coming from the basketball data scenario is unlikely ambiguous, so it is safe to neglect ambiguity resolution. However, text related to coronavirus is oftentimes ambiguous and resolution of the ambiguities is a must. Data-driven approaches excel with the provision that sufficient training data is accessible; however, this condition is not always easily met, as manual annotation is costly, especially in scenarios such as IEA where experienced labor is needed. Experiments highlight that training data generated from the tables is a valid solution, but it requires manual work. This aspect is especially important for SCRUTINIZER, which has the highest accuracy, but it is the system that requires most labeling efforts. We also

remark that some systems worked only after pre-processing the input, by rewriting the claim as a question or by limiting and refining the tabular data. We can state that there is no one system that clearly fits for all scenarios. Choosing or designing a system has to be done keeping in mind the scenario(s) at hand.

Finally, we discuss a promising research direction that we identified during the experimental campaign. Given that the systems are getting better at detecting a false claim, is there any hope that they learn how to *repair* a false claim with the correct information?

Consider a basketball data scenario with claim "Vince Carter scored 22 points in 39 minutes." Having a look at Table 4, we see that the player scored 4 points in 9 minutes. A fact-checking system would mark the claims as false. However, we see in the table that another player (Courtney Lee) is the actual fit for the sentence. The sentence is still false, but if we aim at repairing it, the result will be very different. In one case we would change the values and in one case we would repair the claim with a different entity — which one is the correct fix? Looking at this example, someone may argue that the mistake is in the entity, following the principle that it is more likely to have one error rather than two in the same sentence. This is in line with several data cleaning systems for relational data, which repair tuples according to a minimality principle [9]. This aspect of fact-checking is not considered in the design of the current systems, nor is available as a by-product of their results. We believe this is an interesting open problem that can benefit from the experience on cleaning structured data to introduce the concept of "repairing" natural language sentences.

References

[1] Facebook 'still making money from anti-vax sites'. https://www.theguardian.com/technology/2021/jan/30/facebook-letting-fake-news-spreaders-profit-investigators-claim.

[2] Managing the covid-19 infodemic. https://www.who.int/news/item/23-09-2020-managing-the-covid-19-infodemic-promoting-healthy-behaviours-and-mitigating-the-harm-from-misinformation-and-disinformation.

[3] One of the internet's oldest fact-checking organizations is overwhelmed by coronavirus misinformation - and it could have deadly consequences. https://www.businessinsider.fr/us/coronavirus-snopes-misinformation-fact-checking-overwhelmed-deadly-consequences-2020-3.

[4] Rishabh Agarwal, Chen Liang, Dale Schuurmans, and Mohammad Norouzi. Learning to generalize from sparse and underspecified rewards. *CoRR*, abs/1902.07198, 2019.

[5] Naser Ahmadi, Joohyung Lee, Paolo Papotti, and Mohammed Saeed. Explainable fact checking with probabilistic answer set programming. In *Conference for Truth and Trust Online (TTO)*, 2019.

[6] João Pedro Baptista and Anabela Gradim. Understanding fake news consumption: A review. *Social Sciences*, 9(10), 2020.

[7] Max Bartolo, Tristan Thrush, Robin Jia, Sebastian Riedel, Pontus Stenetorp, and Douwe Kiela. Improving question answering model robustness with synthetic adversarial data generation. *CoRR*, abs/2104.08678, 2021.

[8] Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. Tabfact : A large-scale dataset for table-based fact verification. In *ICLR*, April 2020.

[9] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Holistic data cleaning: Putting violations into context. In *ICDE*, pages 458–469. IEEE Computer Society, 2013.

[10] Artur d'Avila Garcez and Luís C. Lamb. Neurosymbolic AI: the 3rd wave. *CoRR*, abs/2012.05876, 2020.

[11] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of the NAACL*, 2019.

[13] Naeemul Hassan, Fatma Arslan, Chengkai Li, and Mark Tremayne. Toward automated fact-checking: Detecting check-worthy factual claims by claimbuster. In *KDD*, 2017.

[14] Naeemul Hassan, Gensheng Zhang, Fatma Arslan, Josue Caraballo, Damian Jimenez, Siddhant Gawsane, Shohedul Hasan, Minumol Joseph, Aaditya Kulkarni, Anil Kumar Nayak, Vikas Sable, Chengkai Li, and Mark Tremayne. Claimbuster: The first-ever end-to-end fact-checking system. *Proc. VLDB Endow.*, 10(12):1945–1948, August 2017.

[15] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. TaPas: Weakly supervised table parsing via pre-training. In *Proc. of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333. ACL, 2020.

[16] Viet-Phi Huynh and Paolo Papotti. A benchmark for fact checking algorithms built on knowledge bases. In *Proc. of CIKM 2019, Beijing, China, November 3-7, 2019*, pages 689–698. ACM, 2019.

[17] Alon Jacovi and Yoav Goldberg. Towards faithfully interpretable NLP systems: How should we define and evaluate faithfulness? In *Proc. of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4198–4205, Online, July 2020. Association for Computational Linguistics.

[18] Saehan Jo, Immanuel Trummer, Weicheng Yu, Xuezhi Wang, Cong Yu, Daniel Liu, and Niyati Mehta. Verifying text summaries of relational data sets. In *SIGMOD*, page 299–316. ACM, 2019.

[19] Georgios Karagiannis, Mohammed Saeed, Paolo Papotti, and Immanuel Trummer. Scrutinizer: Fact checking statistical claims. *Proc. VLDB Endow.*, 13(12):2965–2968, August 2020.

[20] Nayeon Lee, Belinda Z. Li, Sinong Wang, Wen-tau Yih, Hao Ma, and Madian Khabsa. Language models as fact checkers? In *Proceedings of the Third Workshop on Fact Extraction and VERification (FEVER)*, pages 36–41, Online, July 2020. Association for Computational Linguistics.

[21] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, and Christian Bizer. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 6, 01 2014.

[22] Bill MacCartney and Christopher D. Manning. *Natural Logic and Natural Language Inference*, pages 129–147. Springer Netherlands, Dordrecht, 2014.

[23] Sewon Min, Danqi Chen, Hannaneh Hajishirzi, and Luke Zettlemoyer. A discrete hard EM approach for weakly supervised question answering. In *EMNLP-IJCNLP*, pages 2851–2864. ACL, 2019.

[24] Thomas Mueller, Francesco Piccinno, Peter Shaw, Massimo Nicosia, and Yasemin Altun. Answering conversational questions on structured data without logical forms. In *EMNLP-IJCNLP*, pages 5902–5910. ACL, 2019.

[25] Preslav Nakov, David P. A. Corney, Maram Hasanain, Firoj Alam, Tamer Elsayed, Alberto Barrón-Cedeño, Paolo Papotti, Shaden Shaar, and Giovanni Da San Martino. Automated fact-checking for assisting human fact-checkers. In *IJCAI*, pages 4826–4832. ijcai.org, 2021.

[26] Arvind Neelakantan, Quoc V. Le, Martin Abadi, Andrew McCallum, and Dario Amodei. Learning a natural language interface with neural programmer, 2016.

[27] Yixin Nie, Haonan Chen, and Mohit Bansal. Combining fact extraction and verification with neural semantic matching networks. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, pages 6859–6866. AAAI Press, 2019.

[28] Ehud Reiter and Craig Thomson. Shared task on evaluating accuracy. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 227–231. ACL, 2020.

[29] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why Should I Trust You?": Explaining the predictions of any classifier. In *SIGKDD*, KDD '16, page 1135–1144, New York, NY, USA, 2016. ACM.

[30] Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *SIGKDD Explor. Newsl.*, 19(1):22–36, September 2017.

[31] James Thorne and Andreas Vlachos. Automated fact checking: Task formulations, methods and future directions. In *Proc. of the 27th International Conference on Computational Linguistics*, pages 3346–3359. ACL, 2018.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *NIPS*, volume 30. Curran Associates, 2017.

[33] Bailin Wang, Ivan Titov, and Mirella Lapata. Learning semantic parsers from denotations with latent structured alignments and abstract programs. In *EMNLP-IJCNLP*, pages 3774–3785. Association for Computational Linguistics, November 2019.

[34] You Wu, Pankaj K. Agarwal, Chengkai Li, Jun Yang, and Cong Yu. Toward computational fact-checking. *Proc. VLDB Endow.*, 7(7):589–600, March 2014.

[35] Yiben Yang, Chaitanya Malaviya, Jared Fernandez, Swabha Swayamdipta, Ronan Le Bras, Ji-Ping Wang, Chandra Bhagavatula, Yejin Choi, and Doug Downey. Generative data augmentation for commonsense reasoning. In *EMNLP*, pages 1008–1025. ACL, 2020.

[36] Takuma Yoneda, Jeff Mitchell, Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. UCL machine reading group: Four factor framework for fact finding (HexaF). In *Proc. of the First Workshop on Fact Extraction and VERification (FEVER)*, pages 97–102, Brussels, Belgium, November 2018. ACL.

[37] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2SQL: Generating structured queries from natural language using reinforcement learning, 2018.

TFV: A Framework for Table-Based Fact Verification

Mingke Chai, Zihui Gu, Xiaoman Zhao, Ju Fan*, Xiaoyong Du
Renmin University of China
No. 59 Zhongguancun Street, Beijing 100872, China
{cmk,zihuig,xiaomanzhao,fanj,duyong}@ruc.edu.cn

Abstract

Table-based fact verification, which verifies whether a claim is supported or refuted by structured tables, is an important problem with many downstream applications, like misinformation identification and fake news detection. Most existing works solve the problem using a natural language inference approach, which may not be effective to capture structure of the tables. Despite some recent attempts of modeling table structures, the proposed methods are not compared under the same framework and thus it is hard for practitioners to understand their benefits and limitations. To bridge the gap, in this paper, we introduce a framework `TFV`, including pre-trained language models, fine-tuning, intermediate pre-training and table serialization techniques. Based on the framework, we define a space of design solutions for each module in `TFV`, and conduct an empirical study to explore the design space. Through the experiments, we find that structure information is very crucial but yet under-explored. We point out the limitations of the current solutions and identify future research directions. Moreover, we also develop a python package that implements `TFV` and illustrate how it is used for table-based fact verification.

1 Introduction

Fact verification, which examines whether a textual hypothesis (or a *claim*) is supported or refuted by some given evidence, has many downstream applications, such as misinformation identification and fake news detection. As a fundamental task in natural language understanding, fact verification has been extensively studied [2, 6, 13, 22, 27], and most of the existing studies focus on *textual evidence*. For example, given a claim "*solar panels drain the sun's energy*", the approaches find relevant articles as evidence to support or refute the claim [22].

Recently, *table-based fact verification* has been introduced and attracted much attention [3, 9, 14, 31]. Different from traditional fact verification over textual evidence, table-based fact verification aims to classify whether a claim is supported or refuted by a given table. Figure 1 illustrates an example about table-based fact verification. Given a table that contains structured facts of the Turkish cup as evidence, we can verify that the claim C_1 is correct by referring to the table. Similarly, it is obvious that claim C_2 can be refuted. Since tabular data are being created and curated at unprecedented pace and are made available in a variety of data sources, table-based fact verification may become more and more feasible in real-world applications.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Ju Fan is the corresponding author.

A Verified Claim

- C_1: The highest number of winners from a previous round in the Turkish cup was 54 in round 3.

A Refuted Claim

- C_2: The first round in the Turkish cup has involved more clubs than the second round.

Round	Clubs remaining	Clubs involved	Winners from previous round	New entries this round
first round	156	86	none	86
second round	113	108	43	65
third round	59	54	54	none
fourth round	32	32	27	5
fifth round	16	16	16	none

Figure 1: An example of table-based fact verification.

However, compared with fact verification over textual evidence, table-based fact verification is still under-explored. Most recent studies solve the problem using a natural language inference approach [3, 9]. Specifically, the approach first serializes a table into a token sequence and then leverages a pre-trained language model (e.g., BERT [7]) to encode the serialized table and the claim into vectors. One straightforward approach to table serialization is to linearize the table content via horizontal scan, i.e., row-by-row. Unfortunately, the approach may have a limitation because it ignores the inherent *structure* of the tables. For example, the number "54" in Figure 1 is ambiguous if we do not consider the attributes to which it belongs. Similarly, claim C_2 is hard to verify if we ignore the structure of the first and second rows in the table. Although some recent studies consider structure-aware methods [14, 31], as far as we know, the proposed methods were not compared under the same framework. Thus, it is difficult for practitioners to understand the benefits and limitations of these methods.

To address the above problem, this paper introduces a framework `TFV` that unifies the existing solutions to table-based fact verification. Specially, `TFV` consists of four main modules: (1) *Table Serialization* converts a table into a sequence of tokens. (2) *Pre-trained Language Model (LM)* encodes the claim and table sequence into vectors. (3) *Intermediate Pre-training* utilizes masked language modeling (MLM) objective [7] to fine-tune the LM based on training dataset. (4) *Fine-tuning for Verification* takes as input the vectors and returns either support or refutation as the output. Based on the framework, we conduct an empirical study to systemically investigate how to effectively encode structure of tables. We review the existing solutions for each module in our framework, and define a design space by categorizing the solutions. Through exploring the design space, we provide experimental findings on modeling structure of tables. Also, we compare `TFV` with the existing solutions to table-based fact verification, and analyze benefits and limitations of `TFV`. Note that some previous studies have also investigated some individual modules considered in this paper, e.g., structure-aware pre-trained LMs [14] and fine-tuning methods [31], and they reached consistent conclusions with ours regarding the necessity of modeling structure of tables. However, compared with them, we not only further examine the trade-off between different modules, but also evaluate more modules, such as intermediate pre-training and table serialization.

To summarize, we make the following contributions in the paper.

(1) We conduct a comprehensive experimental study on table-based fact verification, with a special focus on investigating how to encode table structures. We formally define the problem and review existing literature (Section 2). We introduce a general framework `TFV`, and define a design space by categorizing the existing solutions in each module of the framework (Section 3).

(2) We empirically evaluate the design space in `TFV` (Section 4). We reveal the insights from our experimental findings to show that modeling table structures can significantly improve the accuracy of fact verification, and also point out research directions for better model design.

(3) We have implemented TFV as a python package (Section 5). We publish all the code at Github[1]. Fed with a collection of structured tables, the package is easy-to-use and offers table-based fact verification to users.

2 Table-Based Fact Verification

2.1 Problem Formalization

This paper considers a structured table T with n attributes (i.e., columns), denoted by $\{A_1, A_2, \ldots, A_n\}$, and m tuples (i.e., rows), denoted by $\{t_1, t_2, \ldots, t_m\}$. In particular, we denote the value of the j-th attribute in tuple t_i as t_{ij}. Moreover, we consider a textual hypothesis (or *claim*), denoted by C. The problem of *table-based fact verification* is, given a pair (C, T) of claim C and table T, to determine whether table T can be used as evidence to *support* or *refute* claim C. Note that claim C may be either as simple as describing one tuple in table T, or as complex as aggregating or comparing multiple tuples in T. Figure 1 provides a running example.

2.2 Related Work

Fact Verification. Fact verification over textual evidence has been extensively studied in the last decade. Some early studies consider a premise sentence as evidence to support or refute a claim [2, 6], while some later works focus on collecting relevant passages from Wikipedia as evidence [13, 22, 27]. These studies rely on techniques including logic rules, knowledge bases and neural networks for verifying claims based on given evidence. Recently, large pre-trained language models (LM) [7,18,29] have been utilized for fact verification [28]. These models are reported to achieve superior performance due to their self-supervised learning from massive text corpora and effective adapting of the resulting models to target fact verification task.

However, most of the existing studies are limited to considering unstructured text as evidence. As verifying claims over structured tables is useful in many applications [5,17], table-based fact verification has been introduced very recently [3, 9, 14, 31]. Chen et al. formalize the problem of table-based fact verification and provide a benchmarking dataset called TabFact [3]. Then, more approaches are proposed [9, 14, 31] and the basic idea of these approach is to serialize the table and feed it into a large-scale pre-trained language model. Unfortunately, these approaches may have a limitation that they do not thoroughly study how to effectively encode table structures. Specifically, compared with natural language, tabular data has its unique structural characteristics. For example, for each cell, the cells in the same column or the same row may provide more contextual information than others. In addition, each token in natural language has its inherent syntactic logic while each cell in a table may not. This limitation motivates us to develop our framework TFV and conduct an experimental study.

Question Answering over Tables. Another line of research closely related to our work is table-based question answering (QA) [16, 19, 20]. Compared with traditional methods based on logical forms [10, 25, 32], methods that utilizes an end-to-end approach to generate answers directly have been used successfully [14, 19]. TAPAS [14] is a representative end-to-end approach. By extending BERT's architecture and using new pre-training tasks [4], TAPAS [14] improves the understanding of tabular data for various QA tasks. In this paper, we borrow some representative techniques, e.g., TAPAS, to solve our table-based fact verification problem. We has an interesting observation that, although designed for QA over tables, these techniques can also improve the performance of fact verification due to its ability of modeling table structures.

3 Overview of Our TFV Framework

Figure 2 shows our general framework TFV for table-based fact verification. It takes as input a structured table T and a textual claim C, and predicts whether C can be supported (label 1) or refuted (label 0) by T. To this

[1] https://github.com/zihuig/TFV

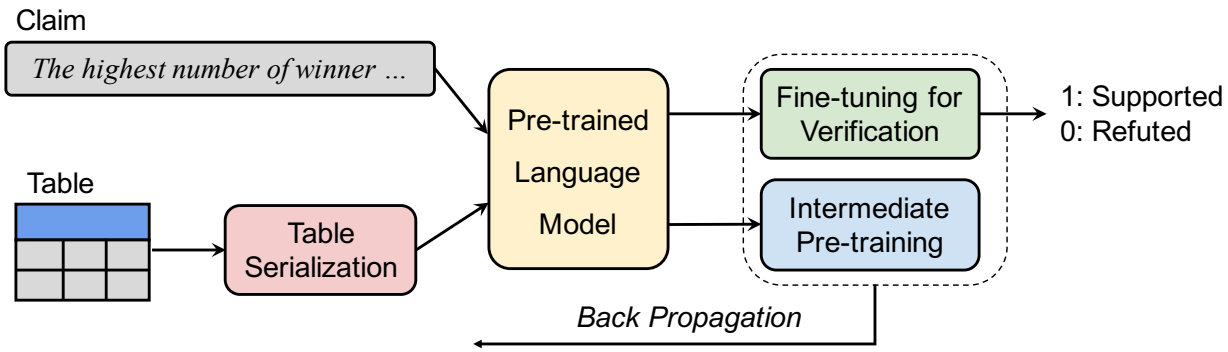


Figure 2: Overview of our TFV framework for table-based fact verification.

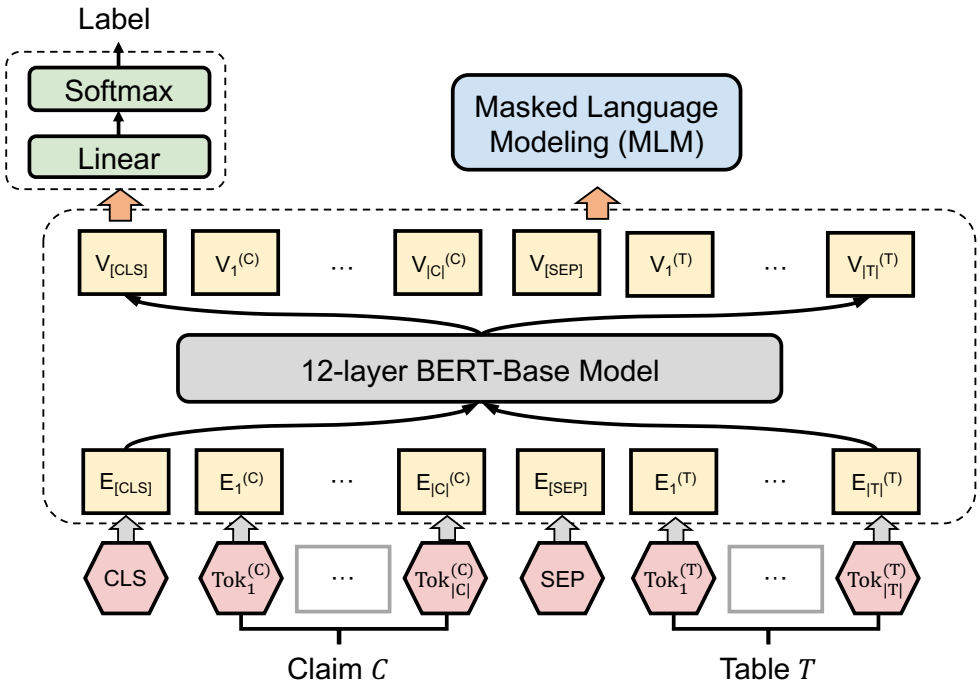


Figure 3: An example of using our TFV framework for table-based fact verification.

end, TFV utilizes the following four main modules. (1) *Table Serialization* converts our structured table T into a sequence of tokens, denoted by $\texttt{seq}_T$. One straightforward serialization method is to horizontally scan table T and concatenate the rows into a sequence. (2) *Pre-trained Language Model (LM)* takes as input the serialized table $\texttt{seq}_T$ and token sequence $\texttt{seq}_C$ of claim C, and produces vector-based representations for the pair (T, C) of table T and claim C. (3) Fed with the vector-based representations, *Fine-Tuning for Verification* produces a predicted verification result on whether C is supported or refuted by T. By considering the classification loss between the predicted result and ground-truth, TFV updates the parameters in the pre-trained LM and the classification model. (4) To further boost the performance, *Intermediate Pre-training* [21] is introduced to utilize training dataset $\{(T, C)\}$ for fine-tuning the pre-trained LM.

Example 1: We use an example to illustrate how TFV works, as shown in Figure 3. We first serialize table T and claim C into a sequence with special tokens, e.g., [CLS] and [SEP], and then we use a pre-trained LM, e.g., BERT [7] to obtain embeddings for all the tokens. Next, we use a fully-connected layer followed by a softmax layer to produce a predicted label, i.e., supported/refuted. Moreover, we also use a masked language modeling

(MLM) task over our training dataset as intermediate pre-training to fine-tune the LM model for improving the token embeddings. Then, we use the LM model after intermediate pre-training to fine-tune for verification.

Next, we present design solutions for each module in our TFV framework. Specifically, we focus on how to encode structural information of tables into pre-trained LM (Section 3.1), fine-tuning for verification (Section 3.2), intermediate pre-training (Section 3.3) and table serialization (Section 3.4) respectively.

3.1 Pre-trained Language Models

Pre-trained LMs have been shown to achieve superior performance in many NLP tasks [7, 18, 29] . The basic idea is to first train an LM on a large unlabeled corpus to learn common representations with unsupervised methods, and then fine-tune the model with a small labeled datasets to fit the downstream tasks. Through pre-training, the LM model can gain better initialization parameters, improved generalization capabilities, faster convergence in downstream tasks, and robustness against over-fitting given small datasets. In our framework TFV, we consider two solutions for pre-trained LMs: (1) the BERT-based LM, and (2) TAPAS [14], an LM pre-trained over tabular datasets.

BERT-based LM. BERT [7] is currently the most popular pre-trained LM. It uses self-attention mechanisms to learn sequence semantic information. BERT is generally pre-trained on a large text dataset of 3.3 billion words, i.e., a concatenation of the BooksCorpus (800 million words) and the English Wikipedia (2.5 billion words) datasets. It leverages two pre-training tasks, namely masked language modeling (MLM) and next sentence prediction (NSP) [7]. Through fine-tuning on different downstream NLP tasks, BERT has achieved the best results so far on the tasks. In our framework, we implement BERT using PyTorch and use "*bert-base-uncased*" as the default BERT setting[2].

TAPAS-based LM. By extending BERT and using *structure-aware* pre-training tasks [4], TAPAS [14] is introduced by Google to improve the understanding of tabular data. Specifically, TAPAS adds additional table-aware positional embeddings for each token, such as column ID, row ID and rank ID, where column/row ID is the index of the column or row that the corresponding token appears in. If the data type of a column is float or date, TAPAS sorts the column values and assigns the values's rank IDs as their numeric ranks. In addition to the additional embeddings that capture tabular structure, TAPAS is pre-trained by using masked language modeling objective over millions of tables crawled from Wikipedia as well as carefully generated claims that associate with the tables. TAPAS can learn correlations between claims and table, and between cells in tables. In TFV, we use the original source code of TAPAS and use "*google/tapas-base*" as the default setting[3].

3.2 Fine-Tuning for Fact Verification

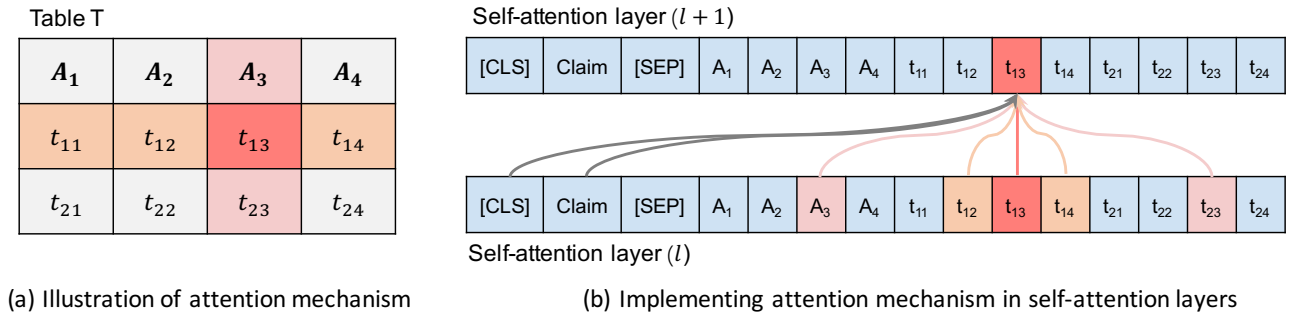
In the fine-tuning step, the LM model is fine-tuned using a labeled dataset denoted as $\{(T, C, l)\}$, where l is the label (supported or refuted). The fine-tuning would make the pre-trained LM more fit the task of fact verification. In our framework TFV, we consider two solutions for fine-tuning: (1) A traditional binary classification model, and (2) a structure-aware binary classification model.

Traditional Classification Model. As illustrated in Figure 3, this solution takes the embedding of token [CLS] as input, and then utilizes a fully connected layer followed by a softmax layer to produce a predicted label. Given the training dataset, it measures the loss between predicted labels and ground-truth to update parameters in both pre-trained LM and the classification model.

Structure-Aware Classification Model. The limitation of the previous fine-tuning method is that it ignores the structure of tables. To address the limitation, Zhang et al. introduce a structure-aware method called SAT for fine-tuning in fact verification [31]. The basic idea of SAT is illustrated in Figure 4(a). SAT aims to capture the

[2]https://pypi.org/project/pytorch-pretrained-bert/

[3]https://github.com/google-research/tapas



(a) Illustration of attention mechanism (b) Implementing attention mechanism in self-attention layers

Figure 4: An illustration of structure-aware model for fact verification fine-tuning.

structure of table T by using an attention mechanism. For example, given a value t_{13}, SAT aims to pay more attention to attribute name A_3 and values t_{11}, t_{12}, t_{14} and t_{23}, which would have more correlation with t_{13}. To realize this attention mechanism, SAT modifies the attention mask in self-attention layers in our pre-trained LM model when fine-tuning the model, as shown in Figure 4(b). For example, for computing representation of t_{13} in the $(l+1)$-th layer, instead of considering all the tokens in the (l)-the layer, SAT only attends to the correlated tokens and masks out the other ones.

3.3 Intermediate Pre-training

In the pre-training step, our model is pre-trained using large-scale general datasets, which learns common language representations. Then, an intermediate pre-training module [8, 15, 23, 24] can be utilized to learn *task-specific* representation using a fact verification dataset. We extend BERT's masked language modeling (MLM) [7] objective to structured data. As in BERT, the training data generator chooses 15% of the tokens of table sequence and claim sequence at random for prediction. For each chosen token, we replace it with "[MASK]", replace it with a random token or keep it intact, with a 80%, 10% and 10% chance, respectively. Note that, when using SAT for fine-tuning, we also use the modified attention mask mechanism for MLM.

3.4 Table Serialization

Table serialization converts our structured table T into a sequence of tokens denoted by $\texttt{seq}_T$. The key challenge here is that there may be a constraint on the maximum length of the sequence. For example, when using BERT as pre-trained LM, one needs to set a hyper-parameter "`max_sequence_length`", which is often 256. However, our table T contains much more tokens. Therefore, it is non-trivial to select the most *valuable* tokens into sequence $\texttt{seq}_T$. In `TFV`, we consider the following heuristic solutions and will study more sophisticated methods in the future work.

No Sampling first serializes table T via horizontal scan, and, when sequence length constrain is met, neglects all remaining tokens. For example, consider table T in Figure 5: given a sequence length constraint, say 15, this strategy only converts attribute names and the first two tuples into the sequence. Obviously, such information is insufficient for verifying our claim C_1, which may degrade the overall performance.

Claim-based Sampling. To address the limitation of the previous strategy, we introduce a claim-based sampling strategy. The basic idea is to selectively sample tokens from table T by considering their correlations with claim C. Figure 5 shows an example: we sample the tokens in attribute names, the third tuple and the fourth columns and convert these tokens into sequence $\texttt{seq}_T$, as these tokens are more related to claim C_1, and thus would benefit the downstream pre-trained LM and fact verification fine-tuning.

To realize claim-based sampling, `TFV` first identifies the cells in T that have overlapping tokens with claim C. For example, cell t_{34} with value 54 can be selected as 54 also occurs in the claim. Then, `TFV` considers two

Round	Clubs remaining	Clubs involved	Winners from previous round	New entries this round
first round	156	86	none	86
second round	113	108	43	65
third round	59	54	54	none
fourth round	32	32	27	5
fifth round	16	16	16	none

Figure 5: An illustration of claim-based sampling for table serialization.

Table 10: Basic statistics of the TabFact dataset.

Type	# Claims	# Tables	Label Ratio
Complex	50,244	9,189	1:1
Simple	68,031	7,392	1:1
All	118,275	16,573	1:1

Table 11: Train/Val/Test splits in TabFact.

Split	# Claims	# Tables	# Rows	# Cols
Train	92,283	13,182	14.1	5.5
Val	12,792	1,696	14.0	5.4
Test	12,779	1,695	14.2	5.4

methods for sampling tokens. (1) Row-based sampling that samples all the rows, in which the identified cells reside, e.g., the attribute row and the 3rd tuple in Figure 5. (2) Cross-based sampling that also considers the columns that contain the identified cells, e.g., the fourth column in Figure 5.

4 Experiments

In this section, we conduct an experimental study to explore the design space in TFV. We present the experiment settings in Section 4.1 and report the main results in Section 4.2. Finally, we summarize the experiment findings and provide insightful takeaways in Section 4.3.

4.1 Experiment Setup

Dataset: We use TabFact [3], the benchmarking dataset for table-based fact verification, for evaluating the design solutions in TFV. This dataset collects tables from Wikipedia and utilizes crowdsourcing to generate claims, where the claims are categorized into *simple claims* and *complex claims*. Specifically, simple claim only describes one tuple in a table, and verifying a simple claim does not involve complex symbolic reasoning [1, 12]. In contrast, complex claims describe multiple tuples in the table, and thus verifying a complex claim needs to consider more complex operations, such as argmax, argmin, count, difference, average, etc. The basic statistics of the TabFact dataset are summarized in Table 10. There are 16573 tables and 118275 claims in the dataset, where each table has on average 14 rows, 5 to 6 columns and 2.1 words in each cell and each table corresponds to 2 to 20 claims. Moreover, the label ratio between supported and refuted claims is 1:1, and the average lengths of positive and negative claims are nearly the same. To evaluate our framework TFV, we divide the dataset into training set, validation set and test set according to the ratio of 8:1:1, where simple and complex claims are stratified in all the three sets. The statistics of these three sets are reported in Table 11.

Evaluation Metric. We use *accuracy*, which is the ratio of correctly predicted supported/refuted labels to all the labels, as the evaluation metric. Specifically, we implement TFV and apply certain design solutions as described above. We utilize the training and validate datasets to train our model for fact verification, and then measure the accuracy of the model on the test dataset.

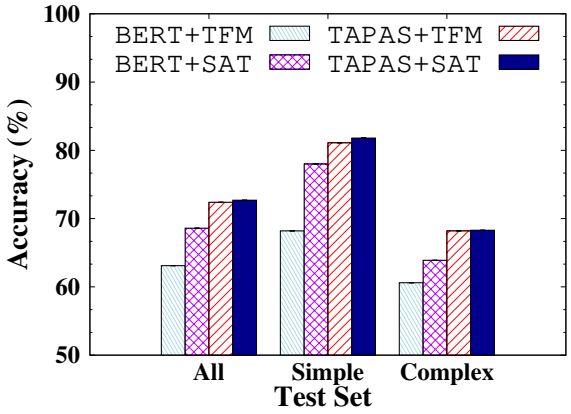


Figure 6: Evaluation of TFV modules.

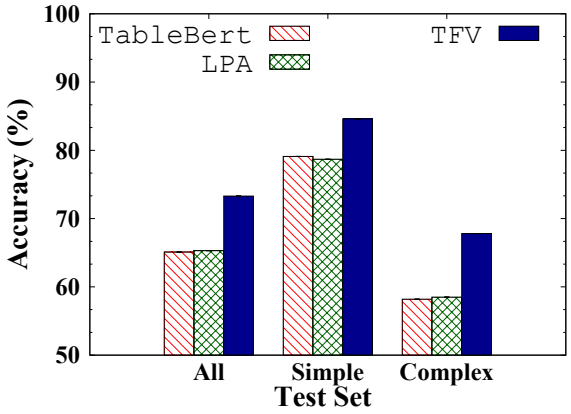


Figure 7: Comparison with existing methods.

Model Parameters: All experiments in this paper use "*bert-base-uncased*" or "*google/tapas-base*" , as described previously, as the pre-trained LM model, where the model has 12 self-attention layers in total. The maximum sequence length of the LM's input, i.e., `max_sequence_length`, is set to 256, and characters exceeding the length will be automatically truncated. Batch size is 32 and the learning rate is 2e-5. We set the maximum number of epochs as 20 and the model usually converges in epochs 15 to 18. The epoch number of every evaluated model is selected as the epoch with the best validation result.

4.2 Experimental Results

We first report our evaluation results on various design solutions in the main modules of TFV. Then, we compare TFV with existing methods for table-based fact verification.

Evaluation of pre-training and fine-tuning. We first evaluate the design solutions of the two main modules in TFV, namely pre-trained LM and fact verification fine-tuning. Figure 6 shows the experiment results, where TFM and SAT represent the traditional and structure-aware classification models respectively. We have two key observations. First, TAPAS-based LM can significantly improve the accuracy on various types of test sets. For example, given traditional fine-tuning method TFM, TAPAS-based LM improves the accuracy by 5.5%, 10% and 3.3% on `All`, `Simple` and `Complex` test sets respectively. This is attributed to the structure-aware pre-training task in TAPAS over millions of tables crawled from Wikipedia and related text segments. The results show that TAPAS can effectively learn correlations between claim C and table T and among cells in table T. Second, the structure-aware fine-tuning method SAT is useful. The main reason is that SAT utilizes the attention mechanism shown in Figure 4 to capture the structure information in tables. We also have an interesting observation that using SAT over the basic LM BERT achieves comparable accuracy with TAPAS, e.g., 68.6% vs. 72.4% on the `All` test set. Because TAPAS needs to pre-train its LM model over millions of tables and text, which is non-trivial and will incur high cost, the alternative of using SAT is more lightweight. Note that, when using TAPAS as the pre-trained LM, the improvement of using SAT is not significant. This is because TAPAS has already captured table structures. In summary, the experiment results show that encoding the structure of tables is beneficial to table-based fact verification.

We also evaluate the effect of intermediate pre-training, and report the results in Table 12. We can see that intermediate pre-training is helpful under different pre-trained LMs and fine-tuning methods. The results show that it is beneficial to utilize the downstream datasets to learn *task-specific* representation before fine-tuning.

Evaluation on table serialization. Next, we evaluate the strategies for table serialization on various types of test sets. For a more comprehensive comparison, we consider two settings of "`max_sequence_length`", namely 128 and 256, in the pre-trained LM model. Table 13 shows the results. First, the performance of

Table 12: Effect of intermediate pre-training for different models (Accuracy %).

Model	All Test		Simple Test		Complex Test	
	w/o training	w/. training	w/o training	w/. training	w/o training	w/. training
BERT+TFM	63.1	66.3 (+3.2)	68.2	74.0 (+5.8)	60.6	62.6 (+2.0)
BERT+SAT	68.6	69.8 (+1.2)	78.0	80.9 (+2.9)	63.9	64.4 (+0.5)
TAPAS+TFM	72.4	72.5 (+0.1)	81.1	81.8 (+0.7)	68.2	67.9 (-0.3)
TAPAS+SAT	72.7	73.3 (+0.6)	81.8	84.6 (+2.8)	68.3	67.8 (-0.5)

Table 13: Comparison of sampling strategies over tables (Accuracy %, Model=BERT+SAT).

Model	All Test		Simple Test		Complex Test	
	max-seq=128	max-seq=256	max-seq=128	max-seq=256	max-seq=128	max-seq=256
NoSample	64.2	70.6	70.7	82.0	61.0	65.0
RwSample	66.5	71.8	74.5	83.4	62.5	65.9
CeSample	**70.1** (+5.9)	**72.5** (+1.9)	**81.7** (+11.0)	**85.9** (+3.9)	**64.4** (+3.4)	**66.1** (+1.1)

NoSample degrades with a large margin, when reducing "max_sequence_length" from 256 to 128, e.g., from 70.6% to 64.2% on the All test set. This result reveals the limitation of existing table serialization techniques for fact verification over large structured tables. Second, claim-based sampling methods, i.e., row-based sampling (RwSample) and cross-based sampling (CeSample) can effectively address the limitation. For instance, when setting "max_sequence_length" as 128, CeSample can outperform NoSample by 5.9% and achieves comparable accuracy with max_sequence_length = 256 (i.e., 72.5%).

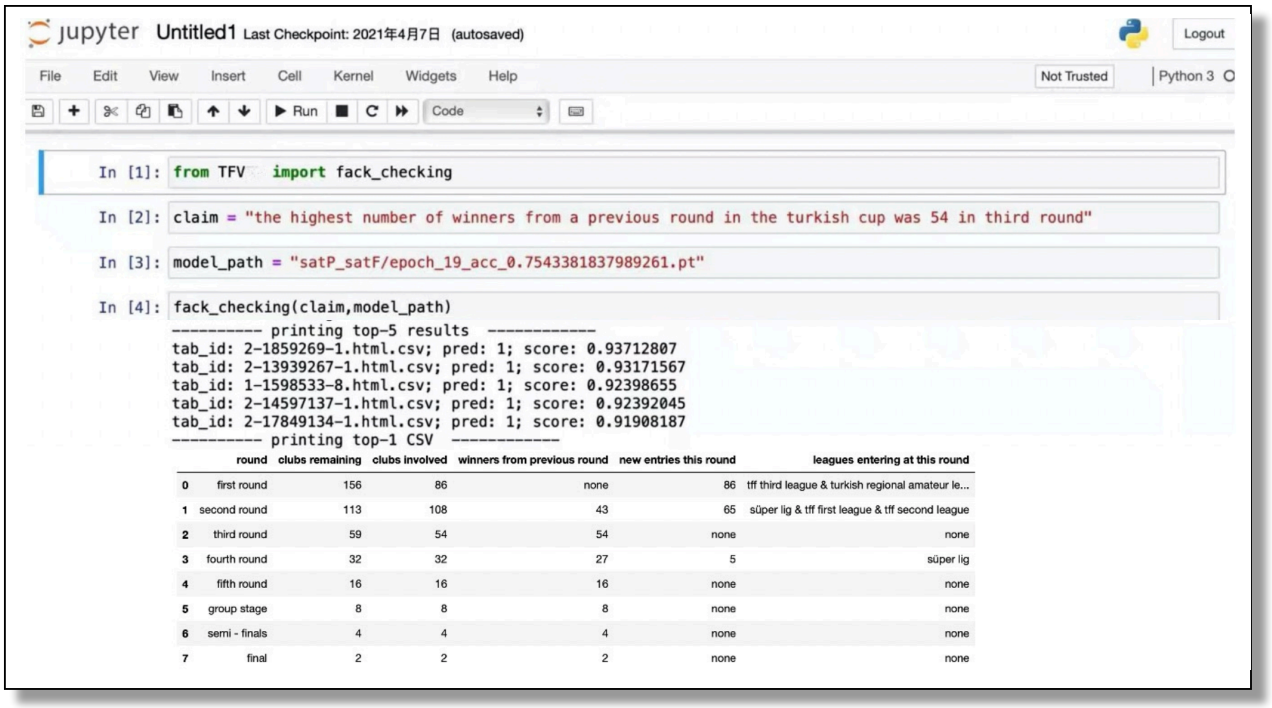
Comparison with existing methods. We compare TFV with the existing methods proposed in TabFact [3]. Specifically, TabFact introduces two solutions: (1) TableBert is the same as our basic setting BERT+TFM with BERT as the pre-trained LM model and basic binary classification for fine-tuning. (2) LPA parses claims into symbolic-reasoning programs and executes the programs over the structured tables to obtain binary verification result. Figure 7 shows the results. We can see that TFV achieves better accuracy than TableBert and IPA. The results show that the current symbolic-reasoning methods, such as IPA, are still under-explored, which may raise new research directions.

4.3 Summary and Takeaways

Based on our experiment findings, we summarize the following key insights that provide guidance to practitioners and researchers on the study of table-based fact verification.

- **Finding 1: Structure information is indispensable for table-based fact verification.** There are two useful solutions to capturing structure information: (i) The first solution utilizes structure-aware pre-trained LM, such as TAPAS, which achieves the best performance but would incur high cost during pre-training. (ii) The second solution relies on more lightweight structure-aware fine-tuning that also achieves comparable accuracy. It calls for more thorough explorations on how to combine the two solutions.

- **Finding 2: Table serialization is important but yet under-explored.** The state-of-the-art benchmarking dataset TabFact [3] only considers small tables from Wikipedia. However, when adopting table-based fact verification in practice, we need to study how to cope with large tables. The preliminary results show that the current table serialization approaches are not effective and simple heuristic sampling solutions are helpful. Therefore, it calls for more theoretical and empirical studies on table serialization.



	round	clubs remaining	clubs involved	winners from previous round	new entries this round	leagues entering at this round
0	first round	156	86	none	86	tff third league & turkish regional amateur le...
1	second round	113	108	43	65	süper lig & tff first league & tff second league
2	third round	59	54	54	none	none
3	fourth round	32	32	27	5	süper lig
4	fifth round	16	16	16	none	none
5	group stage	8	8	8	none	none
6	semi - finals	4	4	4	none	none
7	final	2	2	2	none	none

Figure 8: An example of using our python package of TFV in Jupyter Notebook.

- **Finding 3: Symbolic reasoning approaches are not well studied.** Our experimental results show that the current symbolic reasoning approach IPA [3] achieves inferior accuracy compared with TFV. IPA applies lexical matching to find linked entities in the table and then uses pre-defined templates (e.g., count, argmax, etc.) to generate programs. However, as observed from our example, it may not be easy for linking entities (e.g., "third round" vs. "round 3") and determining correct templates.

5 A Python Package of TFV

We develop a python package[4] that implements TFV. Figure 8 shows an example of using the python package for table-based fact verification in Jupyter Notebook. Fed with a collection of structured tables, we offer fact verification services for users. Note that we do not need the users to provide a specific table T for verification. Instead, we index all the tables and, given a claim provided by a user, we first select candidate tables by applying a keyword matching method based on the TaBERT model [30]. Then, our pre-trained and fine-tuned LM model are then leveraged in the second step to produce the probabilities that support the claim. Finally, we rank the candidate tables according to the supporting probabilities and return the top-k tables.

Moreover, the python package implements representative design solutions for all the modules in Figure 2. Specifically, in each module, such as pre-trained LM, fine-tuning and table serialization, users can choose an appropriate solution to use and combine solutions in multiple modules, so as to easily evaluate various approaches. Note that our framework is extensible, i.e., it is possible to incorporate new modules, new categories, or new methods or variants of existing methods. Note that it is also possible to define the search space from a different angle – we contend that our proposal is rational but may not be the only sensible one.

[4]https://github.com/zihuig/TFV

6 Conclusions and Future Direction

In this paper, we have systemically investigated the problem of table-based fact verification. We have introduced a general framework called TFV and defined a space of solutions for the modules in TFV. We have conducted experiments to test different combinations of methods in the design space with several empirical findings: (1) Structure information is indispensable for table-based fact verification; (2) Table serialization is important but yet under-explored; (3) Symbolic reasoning approaches are not well studied. We have developed a python package that implements TFV and presented its user-friendly features for fact verification.

We also identify several future directions in table-based fact verification that may be worthy of exploration.

- **Benchmarking Datasets.** The state-of-the-art benchmarking dataset TabFact [3] has a limitation that the claims are *not real*. Instead, TabFact solicits crowdsourcing workers to narrate the tables, e.g., describing a single tuple or comparing multiple tuples, to generate claims. Therefore, this dataset may not be effective to tackle claims that people make in natural scenarios, which are more difficult to be aligned to the tables. Thus, it calls for more benchmarking datasets containing real claims.

- **Tabular Data Representation.** Another fundamental problem is whether the current pre-trained LM models [7, 18, 29], which are original designed for natural language, is adequate for tabular data representation. For example, one inherent property of tabular data is *permutation invariant*, i.e., changing the order of rows/columns will not affect the result of fact verification. The current LM models have not considered this property. Therefore, it is desirable to develop new models, such as relational pre-trained transformers [26] and generative adversarial networks [11], for tabular data representation.

- **Symbolic Reasoning.** One experiment finding revealed by this paper is the current symbolic approach achieves inferior performance. However, symbolic reasoning should be indispensable for table-based fact verification, especially for complex claims. Thus, it is desirable to study more effective symbolic reasoning techniques, which may be combined with structure-aware LM models to improve the performance.

References

[1] A. Asai and H. Hajishirzi. Logic-guided data augmentation and regularization for consistent question answering. In *ACL*, pages 5642–5650, 2020.

[2] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. In *EMNLP*, pages 632–642, 2015.

[3] W. Chen, H. Wang, J. Chen, Y. Zhang, H. Wang, S. Li, X. Zhou, and W. Y. Wang. Tabfact: A large-scale dataset for table-based fact verification. In *ICLR*, 2020.

[4] K. Clark, M. Luong, Q. V. Le, and C. D. Manning. ELECTRA: pre-training text encoders as discriminators rather than generators. In *ICLR*, 2020.

[5] P. Clark. Project aristo: Towards machines that capture and reason with science knowledge. In *K-CAP*, pages 1–2, 2019.

[6] I. Dagan, O. Glickman, and B. Magnini. The PASCAL recognising textual entailment challenge. In *MLCW 2005*, volume 3944, pages 177–190, 2005.

[7] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, pages 4171–4186, 2019.

[8] J. Dodge, S. Gururangan, D. Card, R. Schwartz, and N. A. Smith. Show your work: Improved reporting of experimental results. In *EMNLP-IJCNLP*, pages 2185–2194, 2019.

[9] J. M. Eisenschlos, S. Krichene, and T. Müller. Understanding tables with intermediate pre-training. In *EMNLP*, volume EMNLP 2020, pages 281–296. Association for Computational Linguistics, 2020.

[10] J. Fan, G. Li, and L. Zhou. Interactive SQL query suggestion: Making databases user-friendly. In *ICDE*, pages 351–362, 2011.

[11] J. Fan, T. Liu, G. Li, J. Chen, Y. Shen, and X. Du. Relational data synthesis using generative adversarial networks: A design space exploration. *Proc. VLDB Endow.*, 13(11):1962–1975, 2020.

[12] M. Geva, A. Gupta, and J. Berant. Injecting numerical reasoning skills into language models. In *ACL*, pages 946–958, 2020.

[13] A. Hanselowski, H. Zhang, Z. Li, D. Sorokin, B. Schiller, C. Schulz, and I. Gurevych. Ukp-athene: Multi-sentence textual entailment for claim verification. *CoRR*, abs/1809.01479, 2018.

[14] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. M. Eisenschlos. Tapas: Weakly supervised table parsing via pre-training. In *ACL*, pages 4320–4333, 2020.

[15] J. Howard and S. Ruder. Universal language model fine-tuning for text classification. In *ACL*, pages 328–339, 2018.

[16] M. Iyyer, W. Yih, and M. Chang. Search-based neural structured learning for sequential question answering. In *ACL*, pages 1821–1831, 2017.

[17] D. Khashabi, T. Khot, A. Sabharwal, P. Clark, O. Etzioni, and D. Roth. Question answering via integer programming over semi-structured knowledge. In *IJCAI*, pages 1145–1152, 2016.

[18] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.

[19] T. Müller, F. Piccinno, P. Shaw, M. Nicosia, and Y. Altun. Answering conversational questions on structured data without logical forms. In *EMNLP-IJCNLP*, pages 5901–5909, 2019.

[20] P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. In *ACL*, pages 1470–1480, 2015.

[21] R. Peeters, C. Bizer, and G. Glavas. Intermediate training of BERT for product matching. In *DI2KG@VLDB*, volume 2726 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

[22] K. Popat, S. Mukherjee, J. Strötgen, and G. Weikum. Where the truth lies: Explaining the credibility of emerging claims on the web and social media. In *WWW*, pages 1003–1012. ACM, 2017.

[23] Y. Pruksachatkun, J. Phang, H. Liu, P. M. Htut, X. Zhang, R. Y. Pang, C. Vania, K. Kann, and S. R. Bowman. Intermediate-task transfer learning with pretrained language models: When and why does it work? In *ACL*, pages 5231–5247, 2020.

[24] Y. Pruksachatkun, J. Phang, H. Liu, P. M. Htut, X. Zhang, R. Y. Pang, C. Vania, K. Kann, and S. R. Bowman. Intermediate-task transfer learning with pretrained models for natural language understanding: When and why does it work? *CoRR*, abs/2005.00628, 2020.

[25] F. Salvatore, M. Finger, and R. H. Jr. A logical-based corpus for cross-lingual evaluation. In *DeepLo@EMNLP-IJCNLP*, pages 22–30, 2019.

[26] N. Tang, J. Fan, F. Li, J. Tu, X. Du, G. Li, S. Madden, and M. Ouzzani. RPT: relational pre-trained transformer is almost all you need towards democratizing data preparation. *Proc. VLDB Endow.*, 14(8):1254–1261, 2021.

[27] J. Thorne, A. Vlachos, C. Christodoulopoulos, and A. Mittal. FEVER: a large-scale dataset for fact extraction and verification. In *NAACL-HLT*, pages 809–819, 2018.

[28] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. In *NeurIPS*, pages 3261–3275, 2019.

[29] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, pages 5754–5764, 2019.

[30] P. Yin, G. Neubig, W. Yih, and S. Riedel. Tabert: Pretraining for joint understanding of textual and tabular data. In *ACL*, pages 8413–8426. Association for Computational Linguistics, 2020.

[31] H. Zhang, Y. Wang, S. Wang, X. Cao, F. Zhang, and Z. Wang. Table fact verification with structure-aware transformer. In *EMNLP*, pages 1624–1629, 2020.

[32] W. Zhong, D. Tang, Z. Feng, N. Duan, M. Zhou, M. Gong, L. Shou, D. Jiang, J. Wang, and J. Yin. Logicalfactchecker: Leveraging logical operations for fact checking with graph module network. In *ACL*, pages 6053–6065, 2020.

Perturbation-based Detection and Resolution of Cherry-picking

Abolfazl Asudeh
UI Chicago
asudeh@uic.edu

You (Will) Wu
Google Research
wuyou@google.com

Cong Yu
Google Research
congyu@google.com

H. V. Jagadish
University of Michigan
jag@umich.edu

Abstract

In settings where an outcome, a decision, or a statement is made based on a single option among alternatives, it is popular to cherry-pick the data to generate an outcome that is supported by the cherry-picked data but not in general. In this paper, we use perturbation as a technique to design a support measure to detect, and resolve, cherry-picking across different contexts. In particular, to demonstrate the general scope of our proposal, we study cherry picking in two very different domains: (a) political statements based on trend-lines and (b) linear rankings. We also discuss sampling-based estimation as an effective and efficient approximation approach for detecting and resolving cherry-picking at scale.

1 Introduction

Often, an analysis, a decision, or a statement is made or justified based on a possible selection among a collection of valid alternatives. The selection can be a specific piece of data or choice of parameters. Let us consider two very different examples to understand the issues: trendline statements and multi-criteria rankings. Statements made by politicians are often justified based on evidences from data. For example, a politician may compare the unemployment rate on two dates to highlight the success of their policies or to criticize the other parties. As another example, rankings are also used to compare different entities such as universities. Rankings are often generated, using a weight vector that combines a set of criteria into a score, which is then used to sort the entities.

This enables (purposefully or not) cherry-picking to obtain an outcome that is supported by the cherry-picked data but perhaps not in general. In the political statements example, there are plenty of examples cherry-picking factual basis for making misleading conclusions [1]. For example, in his tweet [2] comparing his approval rate with President Obama's, President Trump cherry-picked a single poll source and a specific date which shows the highest approval for him. In such situations, the outcome based on selected data is valid, but the choice of data or parameters can be questioned. In other words, one can ask whether other alternatives *support* the final outcome. Likewise, rankings are both sensitive and have been highly criticized for cherry-picking. College rankings, for example, have a huge presence in Academia but have often been considered harmful [3, 4]. As M. Vardi nicely explains, each ranking is based on a *specific "methodology"* while the choice of methodology is completely arbitrary [4]. A similar concern has been cast by M. Gladwell [3], given that rankings depend on weights chosen for variables.

Our focus in this paper is on how cherry-picking in different settings can be detected, measured, and resolved. In particular, since data/parameters are carefully selected when cherry-picking, we note that *the outcome should*

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

change by perturbing around them. For example, in President Trump's tweet [2], by slightly changing the dates chosen for comparison, the statement that President Trump has a higher approval rate is not longer valid. To this end, we ask what other alternatives could have been chosen for a similar analysis. We can then look at the outcome from all such alternative options. If the outcome is not based on cherry-picking, it should not differ by much from the reported outcome; i.e., it is *stable*. In contrast, a outcome is presumed to be cherry-picked, if it differs greatly from most alternatives considered. Even if it is not intentionally chosen to mislead, there is no question that it does mislead its consumers about the observed trend.

Of course, this begs the question of what alternatives are valid to consider. In the simplest case, valid options are a set of data/parameters to select from. In other cases, a set of constraints may have to hold for an option to be valid. For instance, for a statement comparing the unemployment rate between two US presidents, a pair of dates form a valid trend if each fall in the range of date each president has been in office. We abstract the universe of valid alternatives for generating an outcome as a "*region of interest*".

Following the above argument, we define a notion of "*support*" to measure cherry-picking. That is, given an output O and a region of interest $\mathcal{U}$, we compute its support as the ratio of the valid alternatives in the region of interest that generate the same outcome. Formally,

$$\omega_{\mathcal{U}}(O) = \frac{\big|\{u_i \in \mathcal{U} | O(u_i) \sim O\}\big|}{|\mathcal{U}|} \tag{1}$$

where $O(u_i)$ is the outcome acquired using the option u_i and $O(u_i) \sim O$ indicates that $O(u_i)$ falls in the "acceptance range" of O. The support of a statement shows what portion of the data "agree" with the outcome O. If an outcome has a small support, it has been generated (whether intentionally or not) by cherry-picking. For example, the low support measure for the statement comparing the approval rate of two presidents verifies that it has been cherry-picked, not supported by the rest of the data.

Using the notion of support, our first mission is to detect if an outcome has been cherry-picked. Formally, we define the cherry-picking problem as follows:

Problem 1 (Cherry-picking Detection): Given an output O and a region of interest $\mathcal{U}$, compute $\omega_{\mathcal{U}}(O)$.

Besides detection, the support measure enables to mine data in order to find the most reliable outcome with the maximum support. Formally we define the cherry-picking resolution problem as following:

Problem 2 (Cherry-picking Resolution): Given a region of interest $\mathcal{U}$, find most supported outcome. That is

$$\underset{O \in \{O(u) | u \in \mathcal{U}\}}{\arg\max} \; w_{\mathcal{U}}(O) \tag{2}$$

Cherry-picking and our notion of support for detecting and resolving it are general, not being limited to a specific domain. Still, following the examples provided in this section, we provide a summary of our research findings for Problems 1 and 2 for (i) political statements base on trendlines [5] and (ii) linear rankings [6].

Paper Organization: First, in § 2, we elaborate on the notion of trendlines and carefully provide the formal definitions. We then discuss the design of exact algorithms both for detecting and resolving cherry-picking trendlines. Next, in § 3, we study cherry-picking in our other application domain, linear ranking, and provide exact algorithmic solutions to address Problems 1 and 2 for such rankings. In § 4, we discuss efficient and effective sampling-based approximation techniques for detecting and resolving cherry-picking. Finally, we conclude with brief sections on related work and future work, respectively.

2 Cherry-picked Trendlines

A *trendline* is a common form of statement that appears in many domains, comparing two windows of points in a timestamped data series. Cherry-picked trendlines are prevalent, for example, in politics, among many other different forms of cherry-picking [1]. The partisans on one side of an argument look for statements they can make about trends that support their position [7]. They would like not to be caught blatantly lying, so they cherry-pick the factual basis for their conclusion. That is, the points based on which a *statement* is made may be carefully selected to show a misleading *trendline* that is not a "reasonable" representation of the situation. Comparing with other forms of statements, the simplicity of a trendline may have also contributed to it being a popular form of cherry-picking. In this section, we focus on trendlines derived by comparing a pair of points in data to make a statement. Formally, such a trendline is defined as follows:

Definition 1 (Trendline): For a dataset $\mathcal{D}$, a trendline θ is a defined as a pair of trend points b (the beginning) and e (the end) and their target values in the form of $\theta = \langle (b, y(b)), (e, y(e)) \rangle$.

For example, in a trendline comparing the unemployment rate in two dates d_1 and d_2 is defined as $\theta = \langle (d_1, uemp(d_1)), (d_2, uemp(d_2)) \rangle$ where $uemp(d_i)$ is the unemployment rate at date d_i. We note that trendlines can be defined over based on the aggregate over a window of points, which as explained in [5] can be transform into the standard trendline form after linear preprocessing. Following the definition of trendline, a trendline statement, or simply a statement, is a claim that is made based on the choice of a trendline. Formally,

Definition 2 (Statement): Given a trendline $\theta = \langle (b, y(b)),\ (e, y(e)) \rangle$, a statement is made by proposing a condition that is satisfied by the target values $y(b))$ and $y(e)$. In this paper, we consider the conditions that are made based on the absolute difference between $y(b)$ and $y(e)$. Formally, given the trendline θ, the statement S_θ is a range $(\perp, \top)$ such that $y(e) - y(b) \in (\perp, \top)$.

For instance, the statement "Unemployment decreased" is made by proposing a condition: $(\perp = -\infty, \top = 0)$, which is satisfied by the selected trendline.

Given a statement S, a support region for S, $R_S = (R(b), R(e))$, is defined as a pair of *disjoint* regions, where every trendline θ_i with the beginning and end points b_i and e_i should satisfy the conditions $b_i \in R(b)$ and $e_i \in R(e)$ to be considered for computing the support of S. A support region may naturally be defined by the statement. For instance, for the statement comparing the approval rate of President Trump with President Obama, $R(b)$ (resp. $R(a)$) is any date when President Trump (resp. President Obama) has been in office.

Not all possible trendlines drawn in the support region may be valid or sensible. For example, for a statement comparing the temperature of location/dates, a trendline that compares the temperature of two different locations on different days may not be valid. Depending on the constraints the choice of one trend point enforces on the other, valid trendlines may categorize into unconstrained trendlines and constrained trendlines. In the rest of this section, we show-case our findings for unconstrained trendlines.

2.1 Cherry-picking Detection

Applying Equation 1 on trendlines, given a data set $\mathcal{D}$, a statement $S = (\top, \perp)$, and a support region $R_S = (R(b), R(e))$, the support for S can be computed as

$$\omega_{R_S}(S, \mathcal{D}) = \frac{\text{vol}(\{\text{valid } \langle p \in R(b), p' \in R(e) \rangle \mid y(p') - y(p) \in (\perp, \top)\})}{\text{vol}(\{\text{valid } \langle p, p' \rangle \mid p \in R(b), p' \in R(e)\})} \tag{3}$$

The denominator this equation is the universe of possible valid trendlines from $R(b)$ and $R(e)$. For unconstrained trendlines, this is the product of the "*volume*" of $R(b)$ and that of $R(e)$. Similarly, the numerator can be rewritten

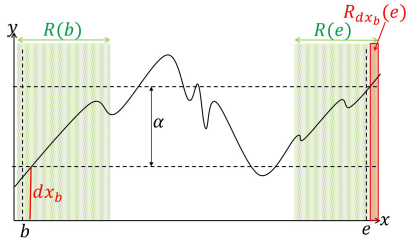


Figure 1: Illustration of a point dx_b and the set of points in $R(e)$ for which $y(dx_e) - y(dx_b) \geq \alpha$.

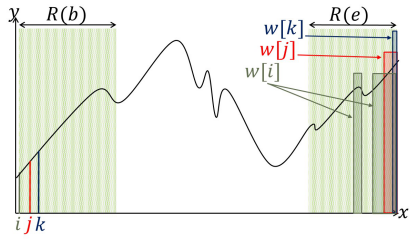


Figure 2: Illustration of weights for three points $dx[i]$, $dx[j]$, and $dx[k]$ in the example of Figure 1.

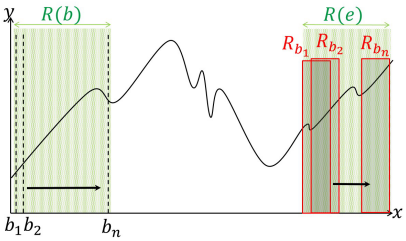


Figure 3: Illustration of the sliding window in $R(b)$ for constrained trendlines.

as a conditional integral as follows:

$$v = \text{vol}(\{\langle p \in R(b), p' \in R(e)\rangle \mid y(p') - y(p) \in (\bot, \top)\}) = \int_{R(b)} \Big(\int_{\{dx \in R(e) \mid y(dx_e) - y(dx_b) \in (\bot,\top)\}} dx_e \Big) dx_b \tag{4}$$

Consider the partitioning of the space into the Riemann pieces (the data records in the dataset $\mathcal{D}$). For a trend point dx_b, let $R_{dx_b}(e)$ be the points in $R(e)$ where $y(dx_e) - y(dx_b) \in (\bot, \top)$. Then, Equation 4 can be rewritten as the sum

$$v = \sum_{\forall dx_b \in R(b)} dx_b \Big(\sum_{\forall dx_e \in R_{dx_b}(e)} dx_e \Big) \tag{5}$$

Consider the example in Figure 1. The horizontal axis shows the trend attribute x while the vertical axis shows y. The trendline of interest is specified by the vertical dashed lines; the left green region identifies $R(b)$ while the one in the right shows $R(e)$, and the curve shows the y values. In this example, the range of the statement S is (α, ∞). A point dx_b in $R(b)$ is highlighted in red in the left of the figure. For dx_b, all points $dx_e \in R(e)$ for which $y(dx_e) - y(dx_b) > \alpha$ support S, forming $R_{dx_b}(e)$ (highlighted in red in the right-hand side of the figure), and therefore, are counted for dx_b. The summation of these counts for all points in $R(b)$ computes the numerator of Equation 5. Following this, the baseline solution sweeps a vertical line from left to right through $R(b)$ and counts the acceptable points in $R(e)$ for each dx_b (similar to highlighted dx_b and R_{dx_b} in Figure 1). For each point in $R(b)$, the baseline algorithm makes a pass over $R(e)$ and, therefore, is *quadratic*: assuming that $|R(e)|$ and $|R(b)|$ are $O(n)$, its run time is $O(n^2)$. In the following, we present an algorithm that pre-processes $R(e)$ in $O(n \log n)$ time, iterates over points in $R(b)$, and utilizes the pre-processed $R(e)$ to compute relevant component of result for each b in $O(\log n)$ time. The overall time complexity is improved significantly to $O(n \log n)$.

Consider Equation 5 once again. For a point $dx[i]$ in $R(b)$, let $w[i]$ be the number of points in $R(e)$ where $y(dx_e) - y(dx[i]) \in (\bot, \top)$, i.e. $\sum_{\forall dx_e \in R_{dx[i]}(e)} dx_e$. Then, Equation 5 can be rewritten as $v = \sum_{\forall dx[i] \in R(b)} w[i]$. For example, in Figure 1, the weight of the point dx_b is the width of the red rectangle $R_{dx_b}(e)$. In the following, we show how the construction of a cumulative function for $R(e)$ enables efficiently finding the corresponding weights for the points in $R(b)$.

In Figure 1, let $dx[1]$ to $dx[n']$ be the set of points in $R(b)$, from left to right. Figure 2 shows three points $dx[i]$, $dx[j]$, and $dx[k]$ where $y(dx[i]) < y(dx[j]) < y(dx[k])$. It also highlights $R_{dx[i]}(e)$, $R_{dx[j]}(e)$, and $R_{dx[k]}(e)$ in the right. Note that $R_{dx[i]}(e)$ consists of two disjoint rectangles. Looking at the figure, one can confirm that $R_{dx[k]}(e)$ is a subset of $R_{dx[j]}(e)$ and $R_{dx[j]}(e)$ is a subset of $R_{dx[i]}(e)$. Since all points in $R_{dx[k]}(e)$ belong to $R_{dx[j]}(e)$ and $R_{dx[i]}(e)$, we do not need to recount those points three time for $dx[i]$, $dx[j]$, and $dx[k]$. Instead, we could start from $dx[k]$, compute its width, move to $dx[j]$, only consider the parts of $R_{dx[j]}(e)$ that is not covered by $R_{dx[k]}(e)$, i.e. $R_{dx[k]}(e) \backslash R_{dx[k]}(e)$, and set $w[j]$ as $w[i]$ plus the width of the uncovered regions by $R_{dx[k]}(e)$. Similarly, in an incremental manner, we could compute $w[i]$, as we sweep over $R(e)$.

Let $dx[1]$ to $dx[n']$ be the set of points in $R(b)$, from left to right. Figure 2 shows three points $dx[i]$, $dx[j]$, and $dx[k]$ where $y(dx[i]) < y(dx[j]) < y(dx[k])$. It also highlights $R_{dx[i]}(e)$, $R_{dx[j]}(e)$, and $R_{dx[k]}(e)$ in the right. Note that $R_{dx[i]}(e)$ consists of two disjoint rectangles. Looking at the figure, one can confirm that $R_{dx[k]}(e)$ is a subset of $R_{dx[j]}(e)$ and $R_{dx[j]}(e)$ is a subset of $R_{dx[i]}(e)$. Since all points in $R_{dx[k]}(e)$ belong to $R_{dx[j]}(e)$ and $R_{dx[i]}(e)$, we do not need to recount those points three time for $dx[i]$, $dx[j]$, and $dx[k]$. Instead, we could start from $dx[k]$, compute its width, move to $dx[j]$, only consider the parts of $R_{dx[j]}(e)$ that is not covered by $R_{dx[k]}(e)$, i.e. $R_{dx[k]}(e) \backslash R_{dx[k]}(e)$, and set $w[j]$ as $w[i]$ plus the width of the uncovered regions by $R_{dx[k]}(e)$. Similarly, in an incremental manner, we could compute $w[i]$, as we sweep over $R(e)$.

Following the above discussion, if we could design a "cumulative" function $F : \mathbb{R} \rightarrow \mathbb{R}$, that for every value y, returns the number of points dx in $R(e)$ where $y(dx) < y$, we could use it to directly compute the weights for the points in $R(b)$. Formally, we seek to design the following function $F = |\{dx \in R(e) \mid y(dx) < y\}|$. Given such a function F, the weight of the point $dx[i] \in R(b)$ can be computed as following:

$$w[i] = F\big(y(dx[i]) + \top\big) - F\big(y(dx[i]) + \bot\big) \tag{6}$$

We use a sorted list $\mathfrak{F}$ as the implementation of F. $\mathfrak{F}$ contains the target values in $R(e)$ such that the i-th element in $\mathfrak{F}$ shows the y value for the i-th largest point in $R(e)$. Having the target values sorted in $\mathfrak{F}$, in order to find $F(y)$, it is enough to find index i for which $\mathfrak{F}[i] < y$ and $\mathfrak{F}[i+1] \geq y$. Then, $F(y) = i$. That is because, for all $j \leq i$: $\mathfrak{F}[j] < y$, while for all $j > i$: $\mathfrak{F}[j] \geq y$. Therefore, the number of points for which $y(x) < y$ is equal to i. Also, since the values in $\mathfrak{F}$ are sorted, we can use binary search for finding the index i.

Having the sorted list $\mathfrak{F}$ constructed, the weight of a point $dx \in R(b)$ can be computed using Equation 6 by applying two binary searches over $\mathfrak{F}$. Then making a pass over $R(b)$, we can compute the nominator of Equation 3 as $v = \sum_{\forall dx[i] \in R(b)} w[i]$. The sum is then is used to calculate $\omega(S, R_S)$. Considering $O(n)$ points in each region, the algorithm conducts $O(\log n)$ for each point in $R(b)$ for binary searches, and hence takes $O(n \log n)$ time.

2.2 Cherry-picking Resolution

An immediate question after detecting an statement based on cherry-picked trendlines is *if not this, what is the right statement supported by the data?* For instance, consider a fantastical statement that, cherry-picking a summer day and a winter day, claims in 2012 Summer was colder than winter in Northern Hemisphere. Apparently, using the 2012 weather data, this statement has very low support. Then, a natural question would be: what is the fair statement supported the most by data? For example, considering a 5 degrees Celsius range for the statement, is summer typically warmer than winter by 20-25 degrees Celsius, is it 15-20 degrees, or is it something else? How representative can it be if we would like to make such a statement with a 5 degree difference?

Formally speaking, adjusting Problem 2 for trendlines, given a dataset $\mathcal{D}$, a value d, and a support region R_S, we want to find the statement $S = (\bot, \bot + d)$ with the maximum support. Finding most supported statements (MSS) is challenging. That is because a brute force solution needs to generate *all* possible statements and check the support for each using the techniques provided in the previous sections. Let y_{min} and y_{max} be $\min(\ y(R(e)))$ and $\max(y(R(e)))$ respectively. For MSS, $(y_{max} - y_{min})$ provides a lower bound for $\bot$ and $(y_{max} - y_{min} - d)$ is an upper bound for it. The brute-force algorithm can start from the lower bound, check the support of $S(\bot, \bot + d)$, increase the value of $\bot$ by a small value ϵ, check the support of the new statement, repeat this process until $\bot$ reaches the upper bound, and return the statement with the maximum support. Note that in addition to the efficiency issue, this algorithm cannot guarantee the discovery of the optimal solution, no matter how small ϵ is.

Instead, we first create the "sorted distribution of trendlines." That is, we create a sorted list ℓ (from smallest to largest) where every value is the difference between the target values of a valid trendline. Constructing ℓ requires passing over the pairs of trendlines and then sorting them. Given that the number of pairs is $O(n^2)$, constructing the ordered list takes $O(n^2 \log n)$ time.

Having ℓ constructed, finding the MSS requires a single pass. Recall that every value in ℓ represents the target-value difference of a valid trendline. For a fixed statement range, the support window should contain all

trendlines that their target-value differences belong to the statement range; hence, the window size is variable. The algorithm for finding MSS starts from the beginning of ℓ the algorithm sweeps a window over ℓ. At every step i, it increases the value of j until it finds the index where $(\ell[j] - \ell[i]) \leq d$ while $(\ell[j+1] - \ell[i]) > d$. The support of the statement identified by the current window is $(j-i)/|\ell|$. In the end, the window with the maximum size (therefore maximum support) is returned. Note the values of i and j only get increased during the algorithm until they reach to the end of the list ℓ. As a result, after constructing the sorted list ℓ, the algorithm requires $O(n^2)$ to find the MSS.

3 Cherry-picked Rankings

Compared with trendlines, ranking is commonplace yet challenging, especially when there are multiple criteria to consider. When there is more than one attribute to be considered for ranking, it is common to use a weight vector to linearly combine the criteria into a score that is used for sorting the items. While complex function can also be used for scoring, in this section we will focus on linear ranking functions that are often used in human-designed rankers such as U.S. News and World Report, Times Higher Education, the National Research Council, etc.

Rankings are important as they may have a significant impact on individuals and society, when it comes to, for example, college admissions, employment, university ranking, sports teams/players ranking, etc. Many sports use ranking schemes. An example is the FIFA World Ranking of national soccer teams based on recent performance. FIFA uses these rankings as "a reliable measure for comparing national A-teams" [8]. Despite the trust placed by FIFA in these rankings, many critics have questioned their validity. University rankings is another example that is both prominent and often contested [3]: various entities, such as U.S. News and World Report, Times Higher Education, and QS, produce such rankings. Similarly, many funding agencies compute a score for a research proposal as a weighted sum of scores of its attributes. These rankings are, once again, impactful, yet heavily criticized.

Example 1: Consider a real estate company with 5 agents that would like to rank them (for promotion) based on two criteria, x_1 : `customer satisfaction` and x_2 : `sales`. Figure 4 shows the candidates as well as their (normalized) values for x_1 and x_2. Claiming that the company values `sales` slightly more than `customer satisfaction`, they use the weight vector $\vec{w} = \langle 1.1, 1.3 \rangle$, computed as $f(t) = 1.1x_1 + 1.3x_2$ for ranking the agents. The scores generated for each agent is shown in the last column of Figure 1.

The ranking generated in Example 1 has been generated using the weights selected in an ad-hoc manner, while the outcome heavily depend in the selection of weights [3]. In other words, unstable outcomes can be generated by *cherry picking the weights*. In particular, as we shall evaluate it next, it turns out the selected ranking has a low support value, indicating that it (whether intentionally or not) has been cherry-picked.

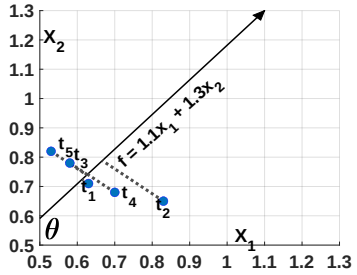
Following Example 1, in the rest of this section we limit our scope to the 2D ranking functions. First, in the following, we provide some background about the geometry of rankings. Next, adjusting the notions of support and region of interest for linear rankings, we propose two algorithms for detecting and resolving cherry-picking. Later in § 4, we will provide a sampling-based approximation approach for MD cases where there are more that than two criteria for ranking.

3.1 Geometry of Rankings

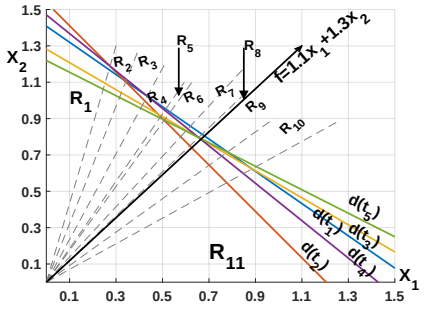
In the popular geometric model for studying data, each attribute is modeled as a dimension and items are interpreted as points in a multi-dimensional space (Figure 4b). This is called the *primal space* where a scoring function is modeled as an origin starting ray and the ranking of items based on it is determined by their projection on the line, as shown in Figure 4b. We transform this primal space into a *dual space* [9], in order to identify regions that help detecting cherry-picking. In the dual space, in $\mathbb{R}^2$, every item t is a line given by $\mathsf{d}(t) : t[1]x_1 + t[2]x_2 = 1$

$\mathcal{D}$			f
id	x_1	x_2	$1.1x_1 + 1.3x_2$
t_1	0.63	0.71	1.34
t_2	0.83	0.65	1.48
t_3	0.58	0.78	1.36
t_4	0.7	0.68	1.38
t_5	0.53	0.82	1.35

(a) A sample database, $\mathcal{D}$, of items with scoring attributes x_1 and x_2; and the result of scoring function $f = 1.1x_1 + 1.3x_2$.



(b) *Original space:* each item is a point. A scoring function is a ray which induces a ranking of the items by their projection.



(c) *Dual space:* items are the lines. Within a region bounded by the intersections of dual lines, all functions induce the same ranking.

Figure 4: A sample database and its geometric interpretation in the original space and dual space.

(Figure 4c). In the dual space, a scoring function f based on the vector $\vec{w}$ translates to an origin starting ray that passes through the point $\vec{w}$. For example, the function f with the weight vector $\vec{w} = \langle 1.1, 1.3 \rangle$ in Example 1 is drawn in Figure 4c as the origin-starting ray that passes through the point $[1.1, 1.3]$.

Every scoring function can then be identified by the angle θ it makes with the x-axis. For example, the function f in Figure 4c is identified by the angle $\theta = \arctan(\frac{1.3}{1.1})$. In other words, there is a one-to-one mapping between possible values for angle θ and the set of possible scoring functions in 2D. This observation enables extending the notion of support for rankings.

The ordering of the items based on a function f is determined by the ordering of the intersection of the hyperplanes with the vector of f. The closer an intersection is to the origin, the higher its rank. For example, in Figure 4c, the intersection of the line t_2 with the ray of $f = 1.1x_1 + 1.3x_2$ is closest to the origin, and t_2 has the highest rank for f.

One observation from the dual space is that the intersections between the dual lines of the items partition the space of possible scoring functions (different values of θ) into discrete regions, called *ranking regions*, where (a) all scoring functions in each region generate the same ranking and (b) no two regions generate the same ranking. In order words, there is a one-to-one mapping between possible rankings and the ranking regions. Formally, let $\mathfrak{R}_{\mathcal{D}}$ be the set of rankings over the items in $\mathcal{D}$ that are generated by at least one choice of weight vector. For a ranking $\mathfrak{r} \in \mathfrak{R}_{\mathcal{D}}$, we define its region, $R_{\mathcal{D}}(\mathfrak{r})$, as the set of functions that generate $\mathfrak{r}$:

$$R_{\mathcal{D}}(\mathfrak{r}) = \{f \mid \nabla_f(\mathcal{D}) = \mathfrak{r}\} \tag{7}$$

Figure 4b shows the boundaries (as dotted lines) of the regions for our sample database, one for each of the 11 feasible rankings. We use the ranking regions in order to extend the notion of *support* for rankings. Looking at the figure, one can observe that the weight vector $\vec{w} = \langle 1.1, 1.3 \rangle$ in Example 1 belongs to a narrow ranking region (R_8) and by slightly changing it, the ranking changes. In other words, it is evident from the figure that the ranking has been cherry-picked.

Every ranking region in 2D can be identified by the two angles in its boundary. Let $\theta_b(R)$ and $\theta_e(R)$ be the beginning and the end angles for the region R. We define the volume of the region, $vol(R) = \theta_e(R) - \theta_b(R)$ to measure the bulk of the region. Similarly, a region of interest in 2D, $\mathcal{U}$, is identified by two angles demarcating the edges of the pie-slice, i.e., $\mathcal{U} = \langle \theta_b, \theta_e \rangle$. For example, let a region of interest be defined by the set of constraints $\{w_1 \leq w_2, \sqrt{3}w_1 \geq w_2\}$. This defines the set of functions above the line $w_1 = w_2$ and below the line $\sqrt{3}w_1 = w_2$, limiting the region of interest to the angles in the range $[\pi/4, \pi/3]$. Similarly, a region defined

around $f = x_1 + x_2$ with the maximum angle $\pi/10°$ corresponds to the angles in the range $[3\pi/20, 7\pi/20]$. The volume of the region of interest can be computed as $\text{vol}(\mathcal{U}) = \theta_e - \theta_b$.

Following Equation 1, the support of a ranking $\mathfrak{r}$ can be measured as the ratio of the volume of its region to the volume of the region of interest: (Equation 1) as following:

$$\omega_{\mathcal{U}}(\mathfrak{r}, \mathcal{D}) = \frac{\text{vol}(R_{\mathcal{D}}(\mathfrak{r}))}{\text{vol}(\mathcal{U})} = \frac{\theta_e(R_{\mathcal{D}}(\mathfrak{r})) - \theta_b(R_{\mathcal{D}}(\mathfrak{r}))}{\theta_e - \theta_b} \tag{8}$$

We note that sometimes in practice not all parts of a ranking are important for studying the support. For example, if the end goal of a ranking is done to select the top-k items, the support value shall be defined on possible (unordered) top-k sets, not the rankings. Similarly, a partial ranking may only look into the top-k (or bottom-k) items. As a generalization of both examples above, inversions at different positions of a ranking may be of different levels of interest/importance (e.g., inverting 10th and 11th items still matters, but not as much as inverting the 1st and the 2nd), and a ranking can be considered as supporting another if the cumulative importance of their inversions is within an acceptable range. [6] elaborates on how to extend the notions of ranking region and support, as well as the detection and resolution algorithms for some of these cases.

3.2 Cherry-picking Detection

The intersections between the lines of items in the dual space, called *ordering exchanges*, are the key in identifying the ranking regions. Consider a ranking $\mathfrak{r}$. For a value of $i \in [1, n)$, let t and t' be the i-th and $(i+1)$-th items in $\mathfrak{r}$. If t dominates t' (i.e., $t[1] > t'[1]$ and $t[2] > t'[2]$) the dual lines $\mathsf{d}(t)$ and $\mathsf{d}(t')$ will not intersect. Otherwise, using the equations of dual lines, the ordering exchange between t and t' can be computed as:

$$\theta_{t,t'} = \arctan \frac{t'[1] - t[1]}{t[2] - t'[2]} \tag{9}$$

If $t[1] < t'[1]$ (resp. $t[1] > t'[1]$), all functions with angles $\theta < \theta_{t,t'}$ (resp. $\theta > \theta_{t,t'}$) rank t higher than t'. The reason is that if $t[1] > t'[1]$, $t[2]$ should be smaller than $t'[2]$, otherwise t dominates t'. Hence $\frac{t[1]}{t[2]} > \frac{t'[1]}{t'[2]}$, i.e. the dual line $\mathsf{d}(t)$ has a larger slope than $\mathsf{d}(t')$, and intersects the rays in range $[0, \theta_{t,t'})$ closer to the origin.

We use this idea for computing the support (and the region) of a given ranking $\mathfrak{r}$. The cherry-picking detection algorithm uses the angle range (θ_1, θ_2), where $0 \leq \theta_1 < \theta_2 \leq \pi/2$, for specifying the region of $\mathfrak{r}$. For each value of i in range $[1, n)$, the algorithm considers the items t and t' to be the i-th and $(i+1)$-th items in $\mathfrak{r}$, respectively. If t' dominates t, the ranking is not valid. Otherwise, if t does not dominate t', the algorithm computes the ordering exchange $\theta_{t,t'}$ and, based on the values of $t[1]$ and $t'[1]$, decides to use it for setting the upper bound or the lower bound of the ranking region. After traversing the ranked list $\mathfrak{r}$, the algorithm returns $(\theta_{min}, \theta_{max})$ as the region of $\mathfrak{r}$, and $\frac{\theta_{max} - \theta_{min}}{\theta_2 - \theta_1}$ as the support of $\mathfrak{r}$. Since the algorithm scans the ranked list only once, computing the support of a ranking in 2D takes $O(n)$ time.

3.3 Cherry-picking Resolution

Similar to other cherry-picking problems such as cherry-picked trendlines, a natural question followed by the detection problem is to find the most supported ranking. This should help the producers of rankings to reveal the ranking that is not just supported by a single function, but the one that has the most support among all possible rankings generated by the functions in the region of interest. Formally, for a dataset $\mathcal{D}$ with n items over d (here $d = 2$) scoring attributes, a region of interest $\mathcal{U}$ (in 2D, $\mathcal{U} = (\theta_b, \theta_e)$), find the ranking $\mathfrak{r}$ with maximum support.

Following the notion of ranking regions, we propose a *ray sweeping* algorithm for finding the most supported ranking, that is, the ranking with the largest region. Let $\mathcal{U} = (\theta_b, \theta_e)$ be the region of interest. The algorithm starts from the angle θ_b and, while sweeping a ray toward θ_e, uses the dual representation of the items for computing the ordering exchanges and finding the ranking regions.

To do so, the algorithm starts by ranking $\mathfrak{r}_{\theta_b}$ the items based on θ_b. It uses the fact that at any moment, an adjacent pair in the ordered list of items exchange ordering, and, therefore, computes the ordering exchanges between the adjacent items in the ordered list. The intersections that fall into the region of interest are added to a min-heap data structure that serve as the sweeper.

Next, it removes the ordering exchange with the minimum angle $\theta_{t,t'}$ from the heap, which together with θ_b for the first ranking region $R(\mathfrak{r}_{\theta_b})$. The support of the first region is $\omega_{\mathcal{U}}(\mathfrak{r}, \mathcal{D}) = (\theta_{t,t'} - \theta_b)/(\theta_e - \theta_b)$, which is the maximum support discovered so far. That is $\omega_{max} = \omega_{\mathcal{U}}(\mathfrak{r}, \mathcal{D})$. After identifying the first region, the algorithm updates its ranking by changing the order between t and t' in it list. The new ranking adds two new ordering exchanges between t and t' and their new neighbors in the ranking, which are added to the sweeper's heap. The algorithm then pops the next ordering exchange from the heap to identify the next ranking region; computes its support; and updates ω_{max} if the new region has a higher support than the best known solution. The algorithm stops when the heap is empty and returns the corresponding ranking with ω_{max} as the output. It also returns the region of the ranking, along with a scoring function that generates the ranking.

The maximum number of ranking regions is $O(n^2)$, since there are at most $\binom{n}{2}$ ordering exchanges between the items. Adding or removing an item from the sweeper's list takes $O(\log n)$, hence the complexity of the cherry-picking resolution algorithm is $O(n^2 \log n)$.

4 Sampling-based Approximation

In large-scale settings where perturbation space, i.e. $\mathcal{U}$, is sizable, it is challenging to either detect or resolve cherry-picking at interactive speed. That is because in such cases even a linear scan over the region of interest to consider all possible cases is time consuming. Consider cherry-picking trendlines as an example. In very large settings where the number of points in $R(b)$ and $R(e)$ is significant, or in the absence of explicit target values where acquiring the data is costly, exact algorithms may not be efficient. The situation is even worse for ranking. So far in this paper, we only considered 2D scoring functions that use two criteria for ranking. In practice, however, there often are more than two criteria for ranking. FIFA rankings, for example uses 4 criteria to rank the national soccer teams [8]. In such cases, due to the curse of dimensionality, the size of the region of interest exponentially grows with d (the number of criteria) and exact algorithms are no longer efficient (please refer to [6] for more details).

On the other hand, approximate estimations of support may often be enough to give the user a good idea about cherry-picking. Hence, a user may prefer to quickly find such estimates, rather than spending a significant amount of time for finding out the exact values. Sampling-based approached, in particular Monte-Carlo methods [10, 11] turn out to be both efficient and accurate for such approximations.

Monte-Carlo methods use repeated sampling and the central limit theorem [12] for solving deterministic problems. Based on the law of large numbers [12], the mean of independent random variables can serve for approximating integrals. That is because the expected number of occurrence of each observation is proportional to its probability. At a high level, the Monte-Carlo methods work as follows: first, they generate a large enough set of random inputs based on a probability distribution over a domain; then they use these inputs to estimate aggregate results.

An important observation is that *uniform sampling from a region of interest $\mathcal{U}$ allows sampling output O based on its support value*. This enables both detection and resolution of cherry-picking by observing different outputs based on the samples and estimating their supports. As a specific topic for the explanation, let us once again consider cherry-picking trendlines. Consider a statement $S = (\bot, \top)$ with the region of interest $R_S = \langle R(b), R(e) \rangle$. The universe of possible trendlines from $R(b)$ to $R(e)$ is the set of valid pairs $\langle p, p' \rangle$ where $p \in R(b)$ and $p' \in R(e)$. Let ω be the support of S in the region R_S, i.e., $\omega(S, R_S)$. For each uniformly sampled pair $\langle p, p' \rangle$, let the random Bernoulli variable $x_{\langle p,p' \rangle}$ be 1 if $y(p') - y(p) \in (\bot, \top)$, 0 otherwise. The probability

distribution function (pdf) of the Bernoulli variable x is:

$$p(x) = \begin{cases} \omega & x = 1 \\ 1 - \omega & x = 0 \end{cases} \tag{10}$$

The mean of a Bernoulli variable with the success probability of x is $\mu = \omega$ and the variance is $\sigma^2 = \omega\,(1-\omega)$. For every set ξ of N iid (independent and identically distributed) samples taken from the above binary variable x, let m_ξ be the random variable showing the average of ξ. Using the central limit theorem, m_ξ follows the Normal distribution $\mathcal{N}\big(\mu, \frac{\sigma}{\sqrt{N}}\big)$ – with the mean μ and standard deviation $\frac{\sigma}{\sqrt{N}}$. Given a confidence level α, the confidence error e identifies the range $[m_\xi - e, m_\xi + e]$ where

$$p(m_\xi - e \leq \mu \leq m_\xi + e) = 1 - \alpha$$

Using the Z-table,

$$e = Z(1 - \frac{\alpha}{2})\frac{\sigma}{\sqrt{N}}$$

For a large enough value of N, we can estimate σ as $\sqrt{m_\xi\,(1-m_\xi)}$. Hence, the confidence error can be computed as:

$$e = Z(1 - \frac{\alpha}{2})\sqrt{\frac{m_\xi\,(1-m_\xi)}{N}} \tag{11}$$

Following the above discussion, the algorithm to estimate the support $\omega(S, R_S)$ uses a budget of N sample trendlines from R_S. The algorithm computes m_ξ by ratio of samples that support S. It then computes the confidence error e, using Equation 11 and returns m_ξ and e. It is easy to see that, since the algorithm linearly scans over N samples, its running time is $O(N)$.

Similarly, the samples can be used to identify the most supported outcome. To see a different application, let us now consider cherry-picking in ranking. In MD where there are $d > 2$ criteria for ranking, every item is represented with a hyperplane $\mathsf{d}(t) : \sum_{i=1}^{d} t[i]x_i = 1$. A scoring function remains as a origin-anchored ray in $\mathbb{R}^d$, identified by $d-1$ angles. In such cases, a region of interest can be described as an origin-anchored hyper-spherical cone, identified by a cosine similarity around an original scoring function. Taking unbiased samples from such an environment becomes challenging, in particular when the region of interest is narrow. Such a sampler is provided in [6, 13]. Having the sampler designed, we can design a Monte-carlo method for identifying the most supported ranking. The algorithm uses a hash data structure that contains the aggregates of the rankings it has observed so far. Upon calling the algorithm, it first draws N sample functions from the region of interest $\mathcal{U}$. For each sampled scoring function, the algorithm finds the corresponding ranking and checks if it has previously been discovered. If not, it adds the ranking to the hash and sets its count as 1; otherwise, it increments the count of the ranking. The algorithm then chooses the ranking that has the maximum count. It computes the support and confidence error of the ranking (using Equation 11) and returns it. Note that following the Monte-carlo method, the algorithm approximately estimates the support of each region and, hence, may miss to return the actual ranking with the maximum support, especially when the number of ranking regions is not small and their supports are close to each other. Still, following the bounds provided by the confidence error, the algorithm guarantees a (user-controllable) upper bound on the difference between the actual maximum support and the algorithm's selection. Considering a budget of N samples while finding the ranking for each sample, the algorithm runs in $O(Nn \log n)$ time. This method has been used in our demo system [14] for responsible ranking design.

5 Related Work

Cherry-picking detection and resolution is closely related to, but not limited to *computational fact checking*, which originated in journalism, with an aim to detect fake news by comparing of claims extracted from the news content against the existing facts [15–22]. The initial fact checking efforts included manual methods based on the domain knowledge of human expert and crowdsourcing [18, 20]. Manual fact checking efforts, however, are not scalable and may not make full use of relevant data. As a result, computational approaches have emerged, with the "Holy Grail" being a platform that can automatically "evaluate" a claim in real-time [17]. Computational fact checking harnesses on techniques from various areas of research such as natural language processing [23, 24], information retrieval [25, 26], and graph theory [15], and spurred novel research including but not limited to multi-source knowledge extraction [27–29], data cleaning and integration [30–32], and credibility evaluation [33, 34]. Existing work also includes style-based [35–38], propagation-based [39–41], and credibility-based [40, 42–45] study of fake news. Further information about fake news and the detection mechanisms can be found in a comprehensive literature survey by Zhou and Zafarani [46].

Using perturbations for studying uncertainty has been studied in different context in data management [47–49]. Perturbation is an effective technique for studying the robustness of query outputs. For example, [6, 14, 50] use function perturbation for verifying the stability of ranking queries, as well as discovering fair and stable rankings. Query perturbation has also been used for retrieving more relevant query results [51–54].

The idea of query perturbation has also seen its applications in the context of the computational journalism, in both fact-checking [21] and lead-finding [55]. Compared with [21], whose focus is more on the modeling of a generic framework for perturbation-based fact-checking, we drill down on two common types of statements—trendlines and linear rankings. On the mining aspect, while [55] studied the representative points to capture the high-value regions of a complex surface, we have treated all points in the support region indifferently, proposed and studied the notion of "support," which is a natural measure that can be defined within the framework and complementary to those defined in [21].

6 Final Remarks and Future Work

In this article, we proposed a measure of support, based on perturbation, to detect and resolve cherry-picking in different contexts. We have demonstrated cherry-picking detection and resolution in two representative types of statements, namely trendlines and linear rankings, with applications in various domains, including but not limited to politics, environment, education, sports, and business intelligence. Besides the exact algorithms, we proposed sampling-based and Monte-Carlo methods as effective approximations for detecting and resolving cherry-picking at scale.

We only focused on the algorithmic aspect of cherry-picking in this paper, which simplifies the problem by assuming the existence of data and a query. Any successful attempt as a real-world system for detection and resolution of cherry-picking needs to address the challenges associated with such assumptions. In the context of trendlines, for example, the first challenge is to translate the (informal) human-language statements to formal trendline statement queries. This requires efficient interaction with human experts for statement formation or (semi-)automatic methods. The next major challenge is to discover the relevant data for evaluating the support of the statement. Discovering relevant data or unbiased samples that can be used for studying cherry-picking is often challenging for real-world scenarios. Fortunately, there have been extensive efforts in the database community, both in designing interactive query systems [56–58] as well as data discovery [59–62], which can be extended for the context of cherry-picking.

Acknowledgments

This research is supported in part by NSF 2107290, 1741022, 1934565, and the Google Research Scholar Award.

References

[1] L. Jacobson. The age of cherry-picking. PolitiFact, Feb. 5, 2018.

[2] L. Jacobson. Donald trump tweet on 50% approval cherry-picks polling data. PolitiFact, June 19, 2017.

[3] M. Gladwell. The order of things: What college rankings really tell us. The New Yorker Magazine, Feb. 14, 2011.

[4] M. Y. Vardi. Academic rankings considered harmful! *Communications of the ACM*, 59(9), 2016.

[5] A. Asudeh, H. V. Jagadish, Y. Wu, and C. Yu. On detecting cherry-picked trendlines. *PVLDB*, 13(6):939–952, 2020.

[6] A. Asudeh, H. Jagadish, G. Miklau, and J. Stoyanovich. On obtaining stable rankings. *PVLDB*, 12(3), 2019.

[7] C. Wardle. Fake news. it's complicated. First Draft News, Feb. 16, 2017.

[8] Fifa/coca-cola world ranking procedure. The Fédération Internationale de Football Association, 28 March 2008.

[9] H. Edelsbrunner. *Algorithms in combinatorial geometry*, volume 10. Springer Science & Business Media, 2012.

[10] C. P. Robert. *Monte carlo methods*. Wiley Online Library, 2004.

[11] F. J. Hickernell, L. Jiang, Y. Liu, and A. B. Owen. Guaranteed conservative fixed width confidence intervals via monte carlo sampling. In *Monte Carlo and Quasi-Monte Carlo Methods 2012*, pages 105–128. Springer, 2013.

[12] R. Durrett. *Probability: theory and examples*. Cambridge university press, 2010.

[13] A. Asudeh and H. Jagadish. Responsible scoring mechanisms through function sampling. *CoRR, abs/1911.10073*, 2019.

[14] Y. Guan, A. Asudeh, P. Mayuram, H. Jagadish, J. Stoyanovich, G. Miklau, and G. Das. Mithraranking: A system for responsible ranking design. In *SIGMOD*, pages 1913–1916. ACM, 2019.

[15] S. Cohen, J. T. Hamilton, and F. Turner. Computational journalism. *CACM*, 54(10):66–71, 2011.

[16] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu. Toward computational fact-checking. *PVLDB*, 7(7):589–600, 2014.

[17] N. Hassan, B. Adair, J. T. Hamilton, C. Li, M. Tremayne, J. Yang, and C. Yu. The quest to automate fact-checking. In *Computation+Journalism Symposium*, 2015.

[18] N. Hassan, F. Arslan, C. Li, and M. Tremayne. Toward automated fact-checking: Detecting check-worthy factual claims by claimbuster. In *SIGKDD*, pages 1803–1812. ACM, 2017.

[19] N. Hassan, G. Zhang, F. Arslan, J. Caraballo, D. Jimenez, S. Gawsane, S. Hasan, M. Joseph, A. Kulkarni, A. K. Nayak, et al. Claimbuster: the first-ever end-to-end fact-checking system. *PVLDB*, 10(12):1945–1948, 2017.

[20] N. Hassan, C. Li, and M. Tremayne. Detecting check-worthy factual claims in presidential debates. In *CIKM*, 2015.

[21] Y. Wu, P. K. Agarwal, C. Li, J. Yang, and C. Yu. Computational fact checking through query perturbations. *TODS*, 42(1):4, 2017.

[22] 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*, 7(13):1557–1560, 2014.

[23] Y. Li, I. Chaudhuri, H. Yang, S. Singh, and H. Jagadish. Danalix: a domain-adaptive natural language interface for querying xml. In *SIGMOD*, pages 1165–1168. ACM, 2007.

[24] Y. Li, H. Yang, and H. Jagadish. Constructing a generic natural language interface for an xml database. In *ICDT*, pages 737–754. Springer, 2006.

[25] P. A. Bernstein and L. M. Haas. Information integration in the enterprise. *CACM*, 51(9):72–79, 2008.

[26] A. Doan, A. Halevy, and Z. Ives. *Principles of data integration*. Elsevier, 2012.

[27] S. Pawar, G. K. Palshikar, and P. Bhattacharyya. Relation extraction: A survey. *CoRR, abs/1712.05191*, 2017.

[28] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *SIGKDD*, pages 601–610. ACM, 2014.

[29] R. Grishman. Information extraction. *IEEE Intelligent Systems*, 30(5):8–15, 2015.

[30] R. C. Steorts, R. Hall, and S. E. Fienberg. A bayesian approach to graphical record linkage and deduplication. *Journal of the American Statistical Association*, 111(516):1660–1672, 2016.

[31] A. Magdy and N. Wanas. Web-based statistical fact checking of textual documents. In *SMUC*. ACM, 2010.

[32] Y. Altowim, D. V. Kalashnikov, and S. Mehrotra. Progressive approach to relational entity resolution. *PVLDB*, 7(11):999–1010, 2014.

[33] D. Esteves, A. J. Reddy, P. Chawla, and J. Lehmann. Belittling the source: Trustworthiness indicators to obfuscate fake news on the web. *CoRR, abs/1809.00494*, 2018.

[34] X. L. Dong, E. Gabrilovich, K. Murphy, V. Dang, W. Horn, C. Lugaresi, S. Sun, and W. Zhang. Knowledge-based trust: Estimating the trustworthiness of web sources. *PVLDB*, 8(9):938–949, 2015.

[35] G. D. Bond, R. D. Holman, J.-A. L. Eggert, L. F. Speller, O. N. Garcia, S. C. Mejia, K. W. Mcinnes, E. C. Ceniceros, and R. Rustige. 'lyin'ted','crooked hillary', and 'deceptive donald': Language of lies in the 2016 us presidential debates. *Applied Cognitive Psychology*, 31(6):668–677, 2017.

[36] S. Volkova, K. Shaffer, J. Y. Jang, and N. Hodas. Separating facts from fiction: Linguistic models to classify suspicious and trusted news posts on twitter. In *ACL-IJCNLP (Volume 2: Short Papers)*, pages 647–653, 2017.

[37] M. Potthast, J. Kiesel, K. Reinartz, J. Bevendorff, and B. Stein. A stylometric inquiry into hyperpartisan and fake news. *CoRR, abs/1702.05638*, 2017.

[38] D. Pisarevskaya. Deception detection in news reports in the russian language: Lexics and discourse. In *EMNLP Workshop*, pages 74–79, 2017.

[39] J. Ma, W. Gao, and K.-F. Wong. Rumor detection on twitter with tree-structured recursive neural networks. In *ACL-IJCNLP (Volume 1: Long Papers)*, pages 1980–1989, 2018.

[40] K. Wu, S. Yang, and K. Q. Zhu. False rumors detection on sina weibo by propagation structures. In *ICDE*, 2015.

[41] S. Vosoughi, D. Roy, and S. Aral. The spread of true and false news online. *Science*, 359(6380):1146–1151, 2018.

[42] Z. Jin, J. Cao, Y. Zhang, and J. Luo. News verification by exploiting conflicting social viewpoints in microblogs. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[43] M. Gupta, P. Zhao, and J. Han. Evaluating event credibility on twitter. In *ICDM*, pages 153–164. SIAM, 2012.

[44] J. Zhang, L. Cui, Y. Fu, and F. B. Gouza. Fake news detection with deep diffusive network model. *CoRR, abs/1805.08751*, 2018.

[45] K. Shu, S. Wang, and H. Liu. Exploiting tri-relationship for fake news detection. *CoRR, abs/1712.07709*, 2017.

[46] X. Zhou and R. Zafarani. Fake news: A survey of research, detection methods, and opportunities. *CoRR, abs/1812.00315*, 2018.

[47] C. C. Aggarwal. *Managing and mining uncertain data*, volume 35. Springer Science & Business Media, 2010.

[48] R. Jampani, F. Xu, M. Wu, L. Perez, C. Jermaine, and P. J. Haas. The monte carlo database system: Stochastic analysis close to the data. *TODS*, 36(3):18, 2011.

[49] N. N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.

[50] A. Asudeh, H. Jagadish, J. Stoyanovich, and G. Das. Designing fair ranking schemes. In *SIGMOD*, 2019.

[51] S. Chaudhuri. Generalization and a framework for query modification. In *ICDE*, pages 138–145. IEEE, 1990.

[52] J.-L. Koh, K.-T. Chiang, and I.-C. Chiu. The strategies for supporting query specialization and query generalization in social tagging systems. In *DASFAA*, pages 164–178. Springer, 2013.

[53] S.-Y. Huh, K.-H. Moon, and H. Lee. A data abstraction approach for query relaxation. *IST*, 42(6):407–418, 2000.

[54] C. S. Jensen and R. Snodgrass. Temporal specialization and generalization. *TKDE*, 6(6):954–974, 1994.

[55] Y. Wu, J. Gao, P. K. Agarwal, and J. Yang. Finding diverse, high-value representatives on a surface of answers. *PVLDB*, 10(7):793–804, 2017.

[56] C. Mishra and N. Koudas. Interactive query refinement. In *ICDT*, 2009.

[57] C. Wang, A. Cheung, and R. Bodik. Interactive query synthesis from input-output examples. In *SIGMOD*, 2017.

[58] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: An automatic query steering framework for interactive data exploration. In *SIGMOD*, pages 517–528, 2014.

[59] R. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker. Aurum: A data discovery system. In *ICDE*, pages 1001–1012. IEEE, 2018.

[60] R. Fernandez, E. Mansour, A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In *ICDE*, 2018.

[61] Y. Shen, K. Chakrabarti, S. Chaudhuri, B. Ding, and L. Novik. Discovering queries based on example tuples. In *SIGMOD*, pages 493–504, 2014.

[62] W.-C. Tan. Deep data integration. In *SIGMOD*, 2021.

WebChecker: Towards an Infrastructure for Efficient Misinformation Detection at Web Scale

Immanuel Trummer
Cornell University
itrummer@cornell.edu

Abstract

We focus on scaling up fact checking to very large document collections. Given a computational cost budget, our goal is to find a maximal number of instances of misinformation. We present the WebChecker, a platform that leverages indexes, cheap filters, and matching methods with various cost-accuracy tradeoffs to maximize fact checking efficiency. It uses a reinforcement learning based optimizer to find optimal checking plans. In our experiments, we use an early prototype to find misinformation on the Web, exploiting Google search to retrieve Web documents and pre-trained language models to identify problematic text snippets within them. WebChecker finds significantly more matches per time unit, compared to naive baselines, and reliably identifies near-optimal checking plans within its plan space.

1 Introduction

Misinformation on the Web can have disastrous consequences [32]. Major Web companies such as Google, Facebook, and Twitter have recently taken steps to protect their user base from being served misleading content [17, 44]. However, those efforts still rely significantly on teams of human fact checkers who are all too often overwhelmed by the scale at which misinformation propagates [36, 46]. Over the past years, this has motivated significant research on automated fact checking methods [10], in industry as well as in academia.

At its core, automated fact checking requires analyzing natural language text (for instance, in order to verify whether a given claim is equivalent to a previously verified one). The area of natural language processing (NLP) has recently seen significant progress. In particular, large pre-trained models based on the Transformer [43] architecture have advanced the state of the art in multiple sub-areas of NLP [16, 47] and achieve near human-level performance for various NLP tasks [5, 28].

The increased accuracy of recent NLP approaches comes, however, at a price. State-of-the-art performance often requires large neural networks with hundreds of millions [7] to hundreds of billions of parameters [9]. Inference via such models is costly. Applying such models naively to large collections of documents (for fact checking or other tasks) is prohibitively expensive. Prior work on automated fact checking has often focused on verifying single claims. Here, given methods to verify single claims, we are interested in scaling up automated checking to large collections of documents, even up to Web scale. In order to do so, we leverage classical techniques from the database community, in particular query planning [37] and adaptive processing [3].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

We describe and evaluate a first prototype of the WebChecker, a platform aimed at misinformation detection in very large document collections. Of course, finding all mistakes in a sufficiently large set of documents (say, the Web) is not realistic. Hence, our goal is to find the maximal number of instances of misinformation under a fixed, computational budget instead. As an extension, we may assign different weights to different instances of misinformation and prioritize search accordingly.

As one possible use case, we envision WebChecker as a means to trigger corrective actions quickly if misinformation is detected. For instance, WebChecker could notify Wikipedia administrators if factual mistakes appear on pages that fall within their core domain. Alternatively, WebChecker could create tags requesting verification by volunteers from the Wikipedia community. Similarly, WebChecker could be applied to verify content on large Web platforms such as Twitter, notifying internal fact checker teams of potentially problematic content (triggering platform-specific counter-measures). Note that new content is generated at a regular pace in all of the aforementioned use cases. This means that fact checking is a continuous process, rather than a one-time operation. This makes computational efficiency, our focus, key.

The WebChecker takes as input a repository containing misinformation and a collection of documents. Its goal is to match text snippets in documents to entries in the repository. To do so efficiently, WebChecker uses a collection of fact checking methods, including high and low precision matching methods, cheap, heuristic filters, and indexes. A-priori, it is unclear which combination of methods maximizes the number of matches found within the cost budget. WebChecker therefore uses an adaptive processing approach, switching between different processing plans to sample their quality. To decide which plans to try next, WebChecker uses reinforcement learning methods.

We evaluate an early prototype of WebChecker experimentally (source code for this prototype is available at `https://github.com/itrummer/WebChecker`). We show that plan choices have significant impact on the number of matches found (up to two orders of magnitude). Furthermore, we show that WebChecker successfully converges to near-optimal plans via reinforcement learning. At the same time, our experimental results point at important avenues for improvements.

The remainder of this paper is organized as follows. We introduce our problem model formally in Section 2. In Section 3, we give a high-level overview of the WebChecker prototype. In Section 4, we describe the space of misinformation detection plans considered by the prototype. Then, in Section 5, we describe the adaptive processing approach used to identify promising plans. We present first experimental results in Section 6. Finally, we discuss related work in Section 7 and conclude with future work plans in Section 8.

2 Problem Model and Terminology

Our goal is to find misinformation in a document collection. We detect misinformation at a fine granularity, verifying specific text snippets (as opposed to classifying entire documents as fake news). We consider large document collections (up to "Web-scale") where verifying each single text snippet naively is prohibitively expensive. Hence our focus on maximizing detection efficiency. To identify misinformation, we compare against entries in a so-called anti-knowledge base [24] (also known as "negative knowledge" [2]), defined next.

Definition 1 (Anti-Knowledge Base, Anti-Fact): An Anti-Knowledge Base (AKB) is a collection A of Anti-Facts, each one representing a piece of information known to be wrong. Each anti-fact $a \in A$ is expressed as a subject-predicate-object triple $a = \langle S, P, O \rangle$. It expresses a relationship between subject and object that does not hold. In addition, each anti-fact may be associated with one or multiple tags (expressing for instance the source from which the anti-fact was mined).

We assume that anti-facts are indexed by tags such that anti-facts with specific tags can be retrieved efficiently. Our goal is to match AKB entries to text snippets in the aforementioned document collection.

Definition 2 (AKB Match): An AKB Match is described as a pair $\langle a, s \rangle$ where $a \in A$ is an anti-fact from the AKB, $s \in D$ is a text snippet (e.g., a single sentence) from the document collection D. Snippet and anti-fact are related in that the text describes the anti-fact, i.e. the text snippet contains misinformation.

Identifying matches between AKB entries and text snippets requires natural language analysis. This field has advanced quickly in recent years, seeing widespread adoption of novel and promising methods such as the Transformer architecture [43]. Nevertheless, analysis precision is still not perfect, leading to erroneous matches.

Definition 3 (Matching Precision, False Positive): With regards to a set of (potential) matches $M = \{\langle a_i, s_i \rangle\}$, we call the ratio of true matches the Matching Precision. We call any pair $\langle a, s \rangle \in M$ where s does not express the anti-fact a a False Positive. We compare different matching methods by their (expected) matching precision.

Setting a different precision target allows adapting WebChecker to different scenarios (e.g., increasing recall at the expense of precision if potential matches are shared with a large crowd for verification while preferring few, high-precision matches if results are forwarded to a small team of fact checkers).

Given many documents and AKB entries, it is often not realistic to find all instances of misinformation. Hence, we must focus our efforts on high-priority instances. To do so, we introduce a weighted recall metric.

Definition 4 (Match Weighting Function): The detection process can be configured by providing a Match Weighting Function. This function assigns each AKB match to a weight, expressing importance of the match. The match weighting function $w_M : M \mapsto \mathbb{R}^+$ assigns AKB matches $m \in M$ to a numerical, positive weight. The weight of a match $\langle a, s \rangle$ is the product of two weight functions, w_A and w_S, assigning a weight for the AKB entry and the text snippet respectively (i.e., $w_M(\langle a, s \rangle) = w_A(a) \cdot w_S(s)$).

The weighting function w_A allows assigning higher weights for misinformation that relates to specific topics (e.g., for anti-facts about the Coronavirus that may accelerate the spread). Weighting function w_S allows assigning higher weights for specific Web sites (e.g., to prioritize highly visible Web sites where misinformation could be particularly harmful). WebChecker prioritizes highly weighted matches in its search. More precisely, it processes detection tasks as defined next.

Definition 5 (Detection Task, Detection Result): A Detection Task is defined by a tuple $\langle A, D, t, w_M, b \rangle$. Here, A is an anti-knowledge base and D a collection of documents. Threshold t lower-bounds expected matching precision (thereby restricting the selection of matching methods) while w_M assigns weights to matches. Finally, b is a cost budget limiting computational overheads. The Detection Result is a set M of matches, ideally maximizing $\sum_{m \in M} w_M(m)$ under cost budget b while respecting precision threshold t.

3 System Overview

Figure 1 shows a high-level overview of the WebChecker system. Next, we describe its context and components quickly. In the following sections, we discuss specific components in more detail.

WebChecker accesses two large repositories: an anti-knowledge base (AKB), containing known misinformation, and a large document collection. WebChecker searches for text snippets in the document collection that match entries in the AKB. More precisely, it processes detection tasks, specified by a user. A detection task is characterized by a matching precision threshold, a function assigning weights to matches, and a cost budget (see Definition 5 for further details). The output is a set of matches, linking text snippets to AKB entries.

The goal of WebChecker is to find a set of matches that maximize accumulated weight, while staying within the cost budget. Furthermore, the result matches must be verified by a verification method that complies with the precision constraints. WebChecker uses several techniques to find high-quality matches efficiently.

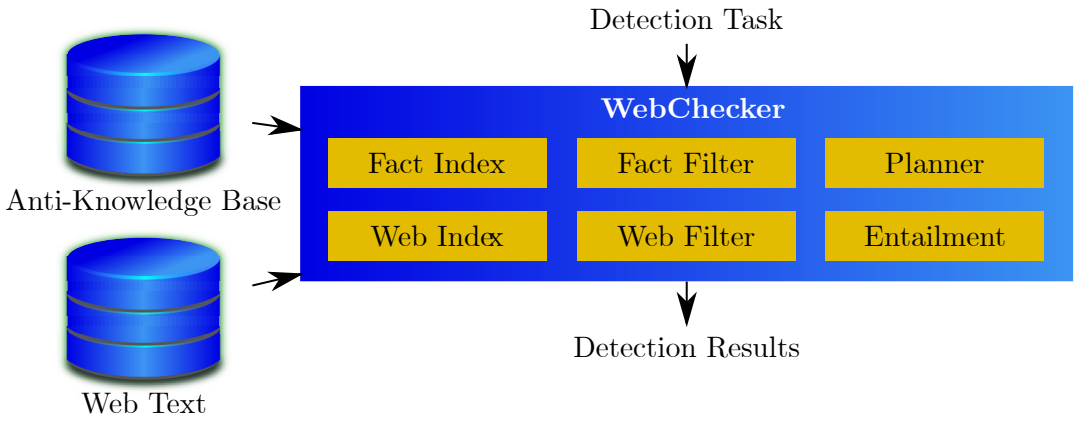


Figure 1: Overview of WebChecker system architecture.

First, WebChecker uses indexes to access subsets of anti-facts and documents efficiently. AKB entries are associated with tags, indicating for instance the source from which AKB entries have been extracted or the type of claim (e.g., distinguishing claims about numerical properties from claims about entity-to-entity relationships [24]). AKB entries are indexed, making it efficient to retrieve all entries associated with specific tags. On the other side, we assume that documents are indexed by keywords (e.g., by a Web search engine). This allows WebChecker to efficiently retrieve documents likely to contain matches for specific anti-facts. Second, WebChecker may use heuristic filters to narrow down the set of anti-facts and text snippets. Doing so is beneficial if it reduces the number of (more expensive) follow-up operations.

Ultimately, all result matches must satisfy an entailment check. Here, WebChecker verifies via natural language analysis whether specific text snippets entail AKB entries. If so, the corresponding snippet contains misinformation. Checking entailment via large neural networks (e.g., via Roberta [30]) is computationally expensive. Hence, WebChecker may use cheap but less precise checks to narrow down the focus for expensive checks. For instance, those checks may include simple checks for word overlap between AKB entry and text snippet. Alternatively, they may include checks via size-reduced language models that trade accuracy for overheads (e.g., via approaches such as Albert [26] and Distilbert [34]).

WebChecker has various options to process a given detection task. For instance, WebChecker can select the anti-facts for which to search matches via Web search (the optimal choice depends on the anti-fact weight as well as on the prevalence). Also, Web Checker may decide whether or not to activate heuristic filters and which sequence of entailment checks to perform. To make theses and other optimization choices, WebChecker uses a planning component.

Optimal planning decisions depend on factors that are unknown a-priori. For instance, it is difficult to estimate the prevalence of specific anti-facts. WebChecker does not assume that such statistics are available before processing starts. Instead, it learns to make near-optimal planning decisions via trial and error. The WebChecker planner uses an iterative algorithm. In each iteration, it uses a fixed percentage of the processing budget to find a maximal number of AKB matches. The accumulated weight of all matches retrieved in a single iteration serves as reward function in a reinforcement learning framework [40]. Thereby, existing algorithms from the reinforcement learning domain can be used to converge to optimal planning decisions (associated with maximal reward). Planning is described in more detail in Section 5.

4 Detection Plans

WebChecker processes detection plans to identify misinformation. We discuss their structure next.

Our goal is to match anti-facts in an anti-knowledge base A to text snippets in a document collection D. We

assume an entailment check e is available that checks if a given text snippet entails a given anti-fact (if so, the text snippet contains misinformation). In its simplest form, a detection plan may be executed as a join between anti-facts and snippets, using the entailment check as join condition, i.e.

$$A \bowtie_e D.$$

The type of entailment check is restricted by the user-specified precision threshold t. For a low-precision threshold, relatively cheap entailment checks (e.g., via distilled language models such as DistilBERT [34]) can be used. Also, instead of a single entailment check, we can use a sequence of more and more expensive entailment checks, reserving expensive checks to matches that pass the cheaper tests. In doing so, we may reduce recall (due to false negatives of cheaper filters) but not precision (as long as the final filter satisfies the precision threshold). If cheap filters are sufficiently selective, overall efficiency increases.

We can use indexes on anti-facts as well as text documents to speed up matching. We consider scenarios where the document collection to analyze is extremely large. It is not realistic to create specific indexes for WebChecker on the entire Web. Fortunately, large parts of the Web have been indexed for keyword search by search engine providers such as Google or Microsoft. We assume that such an index is used to retrieve relevant documents for a given anti-fact. We denote the function that maps an anti-fact a to a keyword query as $i_D(a)$. For instance, we can form this keyword query by concatenating subject, predicate, and object keywords from the triple associated with a. Alternatively, we can use keywords from only a subset of those three components. WebChecker may create one or multiple indexes on anti-facts, indexing them for instance by mining source or topic. By i_A, we denote a condition on anti-facts that can be evaluated using an index on anti-facts. For instance, we can retrieve anti-facts from a specific source or of a specific type. In summary, plans with index usage can be described by the following expression:

$$\sigma_{i_A}(A) \bowtie_e \sigma_{i_D(a)}(D).$$

They are evaluated by retrieving single anti-facts satisfying condition i_A, then retrieving matching documents from D for each anti-fact a.

Finally, we can exploit cheap heuristics to filter out anti-facts and text snippets. We are interested in filtering out anti-facts that are unlikely to generate matches. Also, we want to filter text snippets that are unlikely to contain misinformation. We denote by h_A a heuristic filter on anti-facts and by h_D a heuristic filter on text snippets. For instance, we can exploit previously proposed methods for identifying check-worthy claims [15] to filter text snippets. For filtering anti-facts, we can exploit insights on which anti-fact properties make them likely to propagate (e.g., we show in prior work that mistakes are more likely for certain types of entities and predicates [24]). In summary, plans using heuristics and indexes can be written as

$$\sigma_{h_A}(\sigma_{i_A}(A)) \bowtie_e \sigma_{h_D}(\sigma_{i_D(a)}(D)).$$

WebChecker evaluates such plans by first using the index on A to retrieve anti-facts satisfying i_A and then filtering anti-facts via h_A. For each resulting anti-fact a, WebChecker uses the document index (i.e., the search engine) to retrieve documents satisfying i_D (up to a maximal number of documents) before filtering associated text snippets using heuristic h_D. Finally, the entailment checks e are executed between a and each remaining text snippet. Each snippet that entails a is added to the set of result matches.

5 Adaptive Planning and Processing

We discussed the general structure of detection plans in the previous section. That structure leaves open several optimization decisions. First, we can choose which filters (if any) to use on anti-facts and text snippets (heuristic filters and index-based filters). We assume that a set of alternatives is available for each type of filter (we discussed

Algorithm 3: Adaptive processing algorithm used by WebChecker.

```
1 WebChecker(A, D, t, w_M, b) begin
2     M ← ∅;                                   // Initialize AKB matches
3     while Cost budget b not reached do
4         p ← SelectPlan(A, D, t);             // Select plan via RL
5         N ← EvalOnSample(p, t);              // Evaluate new plan
6         r ← Σ_{n∈N} w_M(n);                  // Calculate reward value
7         UpdateStats(p, r);                   // Update RL statistics
8         M ← M ∪ N;                           // Add new matches
9     return M;                                // Return detection result
```

some alternatives in Section 4). Second, we can choose the sequence of entailment checks. Of course, any filter (or additional entailment check) tends to decrease the total number of matches if processing all anti-facts and text snippets. However, it is not realistic to search the entire Web. Instead, our goal is to maximize the number of matches found until the cost budget runs out. Given the cost budget, filters can increase the number of matches found as they help us focus our efforts on the most relevant AKB entries and text snippets.

Following Definition 5, the best plan maximizes the accumulated weight of all matches found under the cost budget. To select optimal plans, we would need to know the cost and accuracy of filter operations and entailment checks, as well as the frequency at which certain types of misinformation appear on the Web. In particular, having the latter kind of information about a large and dynamic document collection such as the Web does not seem realistic. This motivates a more flexible processing strategy that adapts to information gained at run time. More precisely, WebChecker implements the adaptive processing strategy described as Algorithm 3.

Given an anti-knowledge base A, a document collection D, a precision threshold t, a match weighting function w_M, and a cost budget b as input, WebChecker iterates until the cost budget is depleted (our current implementation measures cost as run time while other metrics, such as monetary processing fees, could be used instead). In each iteration, WebChecker picks a detection plan p and uses it for a limited amount of processing (in our current implementation, we limit ourselves to one AKB entry per iteration and 30 seconds of Web search time). The Web matches resulting from processing are added to the result set.

In each iteration, WebChecker picks plans based on statistics collected while processing the current detection task. In doing so, it faces the so-called exploration-exploitation dilemma: we have a tension between selecting plans that have worked well so far and selecting plans about which little is known (e.g., since they have not been tried yet). Reinforcement learning is the classical framework used to resolve this kind of tension in a principled manner. Hence, we apply a reinforcement learning algorithm to select the next plan to try in each iteration. As reward function (quantifying the quality of a plan), we use the sum of weights over all matches found in a given iteration. The reinforcement learning algorithm will therefore converge towards plans that return more and higher weighting matches.

More formally, the environment for a reinforcement learning algorithm is typically described as a Markov Decision Process (MDP). An MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$ of states $\mathcal{S}$, actions $\mathcal{A}$, transitions $\mathcal{T}$, and reward function $\mathcal{R}$. In our case, states correspond to vectors $\langle v_1, \ldots, v_n \rangle \in \mathbb{N}^n$ where each component $v_1, ..., v_{n-1}$ represents a plan property and v_n represents the plan property to decide next. Actions represent alternative values for the next plan property. The transition function maps a state $\langle v_1, \ldots, v_i, \ldots, v_{n-1}, i \rangle$ and an action a to the state $\langle v_1, \ldots, a, \ldots v_{n-1}, i+1 \rangle$ (i.e., we set the value specified by the action and advance to the next plan property). After determining the last plan property, the current episode ends and the specified plan is evaluated. The reward for any transition is zero except for the last transition in an episode. For the last transition, the reward corresponds to the sum of weights over all matches collected by the specified plan in the

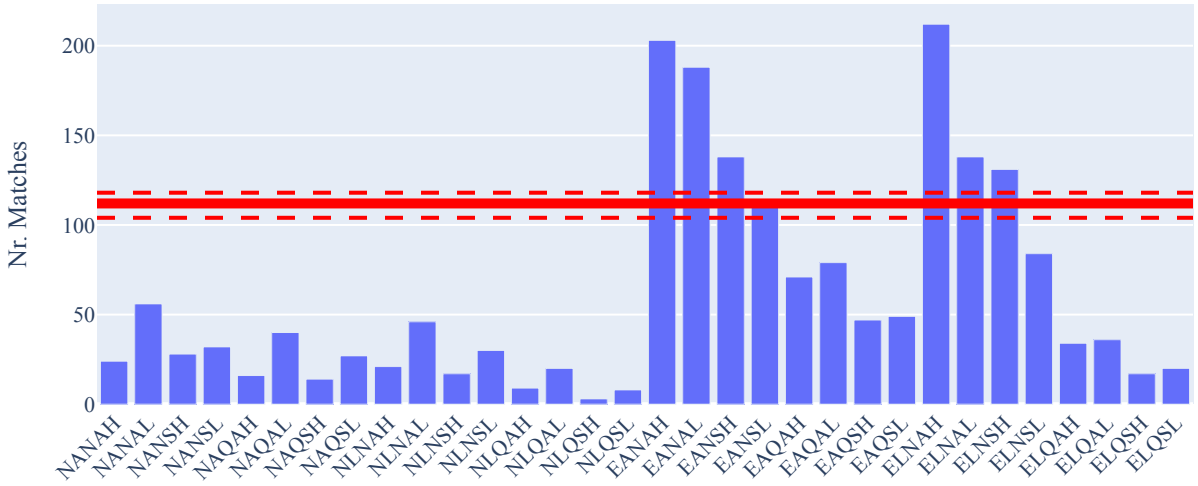


Figure 2: Number of misinformation matches found on the Web during one hour of processing time for 32 different plans. The red lines mark the number of matches found when optimizing the plan via reinforcement learning in three consecutive runs (solid line marks median).

current iteration. The transition function is deterministic while the reward function is stochastic (since using the same detection plan on different AKB entries in different iterations may yield a different number of matches).

6 Experimental Results

We report first experimental results for an early prototype of WebChecker. The prototype is implemented in Python 3. The source code is publicly available (`https://github.com/itrummer/WebChecker`).

The prototype uses an AKB consisting of over 100,000 entries (publicly available online at `https://github.com/cornelldbgroup/AntiKnowledgeBase`). Those entries were mined from Wikipedia update logs using the process described by Karagiannis et al. [24]. For querying the Web, the prototype uses Google's programmable search engine API [12]. For natural language processing, it uses Transformer models via the Huggingface Transformers library [47]. The prototype uses Stable Baselines 3 [39] for reinforcement learning. The experiments were executed on a p3.2xlarge EC2 instance with Tesla V100 GPU.

The prototype considers the following plan space properties. First, AKB entries are indexed by their type, distinguishing mistakes with incorrect numbers (e.g., incorrect birth dates) from mistakes with incorrect entities (e.g., an incorrect location for the headquarters of a company). The prototype may choose between those two categories to maximize detection yield. Second, the prototype may choose to filter AKB entries to the ones with long text (the intuition being that Web search results tend to be more specific and relevant for such triples). Third, the prototype may choose how to construct Web queries for a given AKB triple. In the current version, we consider two simple variants (either using quotes around triple subject and triple object or no quotes at all). Fourth, the WebChecker prototype may choose to filter Web results to Web sites with relatively small text content. Intuitively, more small Web sites can be processed for a fixed time limit, reducing the risk of investing significant computational resources into a Web site that turns out to be irrelevant. Finally, the prototype considers two different sequences of entailment checks. The first version directly applies a large Roberta model [30], pre-trained on MNLI (`https://huggingface.co/roberta-large-mnli`), to check whether a Web sentence entails the concatenation of components of an AKB triple. The second version performs a cheaper check first (testing for a sufficient ratio of overlapping words between triple and sentence). It applies the model-based test only if the cheaper one succeeds.

AKB Entry (with Correction)	Web Match (with URL)
MCS6501 microprocessors, was in, July 24 1974 issue of Electronics magazine **[Correct: July 24 1975]**	One of the earliest was a full-page story on the MCS6501 and MCS6502 microprocessors in the July 24, 1974 issue of Electronics magazine [https://en-academic.com/dic.nsf/enwiki/12304]
Biraj Bahu, is, 1955 Hindi film **[Correct: 1954]**	... and Biraj Bahu (1955), which won her the Filmfare Best Actress Award in 1955 [https://en.wikipedia.org/wiki/Kamini_Kaushal]
Haflioi Hallgrimsson, currently lives in, Edinburgh **[Correct: Bath]**	Haflioi Hallgrimsson, an Icelandic composer, currently living in Edinburgh [https://en.wikipedia.org/wiki/Icelandic_diaspora]
Acorda Therapeutics, is, biotechnology company based in Hawthorne **[Correct: Ardsley]**	Acorda Therapeutics (NASDAQ: ACOR, FWB: CDG) is a biotechnology company based in Hawthorne, New York [https://www.morebooks.de/store/fr/book/acorda-therapeutics/isbn/978-620-1-18842-6]

Table 14: Examples of Web matches found by WebChecker (all links were verified on 6/10/2021).

Considering binary choices for all five plan properties, the prototype considers a plan space of size 32 in total. In a first test, we ran all detection plans with a timeout of one hour per plan (we set a timeout of 30 seconds for finding matches for a single AKB entry and restrict the number of considered AKB entries to 120). The goal was to verify that planning choices have significant impact on the number of matches found. In a second test, we use the AC2 [31] reinforcement learning algorithm to make planning choices automatically (using the number of Web matches found as reward function). Note that we use uniform weights for those initial experiments (i.e., every Web match counts equally). Also, we do not explicitly consider precision thresholds (all possible plans use the same, final entailment check via the Roberta Transformer). Adding more fine-grained precision control is one of the future work avenues discussed in Section 8.

Figure 2 shows the results of both experiments. The y axis represents the number of Web matches found within one hour. The x axis describes plans in short notation (each letter describes one plan property in the following order): N for number mistakes versus E for entity mistakes, A for all triples versus L for long triples, N for non-quoted versus Q for quoted Web queries, A for all versus S for short Web sites, and H for (direct) high precision entailment checks versus L for an initial low-precision check. The solid red line represents the median number of matches found in three consecutive runs when using reinforcement learning for planning (dashed lines represent the number of matches for the other two runs).

The number of matches found by different plans differs by two orders of magnitude (ranging from three matches for plan NLQSH to 212 for plan ELNAH). For instance, AKB entries indexed as entity-related mistakes tend to produce more Web matches, compared to number-related mistakes (24 versus 98 matches when averaging over all plans). As another example, leaving out quotes in Web queries tends to increase recall significantly (91 versus 31 matches in average over all plans). Note, however, that different planning choices are dependent on each other and cannot be easily separated (e.g., enabling low-precision checks increases recall in many cases but not in all). This motivates the use of sophisticated planning approaches.

Reinforcement learning consistently finds plans with a quality (i.e., number of matches) significantly above average. In all three runs, the plans produced via reinforcement learning were better than 25 out of 32 plans with a median of 112 matches (compared to a median of 35 matches over all plans). This shows that reinforcement learning is suitable for finding good plans within the detection plan space.

Table 14 shows example matches found by WebChecker. Clearly, matches found cover a variety of topics (reflecting the diversity of content of the AKB we use). The current prototype uses the same final entailment check for all candidate matches. Nevertheless, planning choices may influence not only recall but also precision of the final result. We hand-verified entailment for a sample of 20 matches from the plan generating most matches (ELNAH) and hand-verified all matches for the plan generating the least matches (NLQSH). We found a precision of 100% for the latter but only a precision of 24% for the former. While our sample is small, this indicates that more work is needed to optimize precision during retrieval. We discuss first ideas in Section 8.

7 Related Work

Our work connects to a large body of recent work that exploits machine learning for automated fact-checking [6, 18, 22, 27, 29], including work focused on the platforms we cite as motivating use cases [19, 35]. In particular, our research connects to prior work using language models for fact checking [6, 27]. We refer to recent surveys for a detailed overview of corresponding approaches [4, 38]. What distinguishes our work from prior contributions is our focus on computational efficiency, exploiting ideas from the database community to scale up automated fact checking.

We use Web search for automated fact checking. That connects our work to prior checking methods based on Web search engines [8, 11, 45]. However, prior work typically uses Web search to retrieve relevant documents for verifying a given claim. Instead, we use it to retrieve instances of misinformation from the Web. In that, our work is similar to the one by Elyashar et al. [8]. Here, the focus is on collecting fake news documents via Web search. However, our work differs as Web search is only one step in a multi-step verification pipeline. Our contribution is in an infrastructure that takes planning decisions maximizing misinformation retrieval efficiency.

Multiple branches of prior work could be used to extend WebChecker. For instance, a popular branch of fact checking research focuses on identifying check-worthy claims [13–15, 33]. If sufficiently efficient, such methods can be used as filters on Web results before applying more expensive verification methods. Also, we currently identify misinformation by comparing against entries in an AKB [1, 2, 24]. In future work, we may consider other verification sources such as knowledge graphs [29] or relational databases [13, 20, 22, 23].

Finally, we use techniques from the database area to make large-scale fact checking efficient. In particular, we exploit adaptive processing and query planning techniques [3, 41, 42]. Reinforcement learning has been used for optimizing adaptive query plans before [42]. However, our plan space, processing engine, and reward function are specific to the domain of fact checking. Our goal is not to process all input data for a given query. Instead, it is to produce the maximal number of matches until a time budget runs out. More broadly, our work is part of a research direction that transfers techniques from the database community to new use cases (e.g., recent work exploiting declarative query optimization to make visual data processing more efficient [21, 25]). In this case, we transfer ideas such as query planning to the domain of automated fact checking.

8 Conclusion and Outlook

Our goal is to make large-scale, automated fact checking efficient. We present a first prototype of WebChecker, a system that searches for misinformation in a large document collections while minimizing computational overheads. To do so, WebChecker exploits a diverse set of operators and an adaptive optimizer to choose between them.

We observe the following in our experiments. First, planning choices can significantly impact efficiency in automated fact checking. Comparing best and worst plans, we find a two orders of magnitude difference in computational overheads per (misinformation) detection. Second, we find that adaptive processing and reinforcement learning are effective at identifying good detection plans. While less efficient than the optimum by a factor of about two (rate of detection), learned plans outperform the majority of plans in the plan space.

Our current prototype does not yet support user-defined precision constraints (we outline in Section 2 why such constraints are required to adapt WebChecker to different scenarios). Also, our experiments indicate that output precision depends on various plan properties (beyond the final entailment check). In a first approach, we plan to consider operator-specific accuracy statistics (gained, for instance, via small-scale experiments on pieces of misinformation for which manually generated ground truth is available) during planning. Alternatively, we plan to explore approaches that use feedback by crowd workers on result samples to evaluate plan precision. Beyond precision, we plan to increase computational efficiency by expanding WebChecker's set of operators (e.g., by adding entailment checks by smaller models [26, 34] as optional filters). Also, we plan to study parallel processing as a means to increase verification throughput.

Acknowledgement

This project is supported by a Google Faculty Research Award ("Mining an Anti-Knowledge Base for Fact Checking from Wikipedia Updates"). We thank the editors for helpful suggestions and comments on an earlier version of this paper.

References

[1] H. Arnaout, S. Razniewski, and G. Weikum. Negative Statements Considered Useful. 2020.

[2] H. Arnaout, S. Razniewski, G. Weikum, and J. Z. Pan. Negative Knowledge for Open-world Wikidata. *Wikimedia Workshop*, 2, 2021.

[3] R. Avnur and J. Hellerstein. Eddies: continuously adaptive query processing. In *SIGMOD*, pages 261–272, 2000.

[4] A. Bondielli and F. Marcelloni. A survey on fake news and rumour detection techniques. *Information Sciences*, 497:38–55, 2019.

[5] B. Byrne, K. Krishnamoorthi, S. Ganesh, and M. S. Kale. TicketTalk: Toward human-level performance with end-to-end, transaction-based dialog systems. 2020.

[6] G. Demartini, S. Mizzaro, and D. Spina. Human-in-the-loop Artificial Intelligence for Fighting Online Misinformation: Challenges and Opportunities. (September):65–74, 2020.

[7] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, volume 1, pages 4171–4186, 2019.

[8] A. Elyashar, M. Reuben, and R. Puzis. Fake News Data Collection and Classification: Iterative Query Selection for Opaque Search Engines with Pseudo Relevance Feedback. 2020.

[9] L. Floridi and M. Chiriatti. GPT-3: Its Nature, Scope, Limits, and Consequences. *Minds and Machines*, 30(4):681–694, 2020.

[10] Fullfact. The state of automated factchecking. Technical report, 2016.

[11] D. Gerber, D. Esteves, J. Lehmann, L. Bühmann, R. Usbeck, A. C. Ngonga Ngomo, and R. Speck. DeFacto - Temporal and multilingual deep fact validation. *Journal of Web Semantics*, 35:85–101, 2015.

[12] Google. https://programmablesearchengine.google.com/about/, 2021.

[13] N. Hassan, F. Arslan, C. Li, and M. Tremayne. Toward automated fact-checking: detecting check-worthy factual claims by ClaimBuster. In *SIGKDD*, pages 1803–1812, 2017.

[14] N. Hassan, C. Li, and M. Tremayne. Detecting check-worthy factual claims in presidential debates. *International Conference on Information and Knowledge Management, Proceedings*, 19-23-Oct-2015:1835–1838, 2015.

[15] N. Hassan, G. Zhang, F. Arslan, J. Caraballo, D. Jimenez, S. Gawsane, S. Hasan, M. Joseph, A. Kulkarni, A. K. Nayak, V. Sable, C. Li, and M. Tremayne. ClaimBuster: the first-ever end-to-end fact-checking system. *VLDB*, 10(7):1–4, 2017.

[16] J. Howard and S. Ruder. Universal Language Model Fine-tuning for Text Classification. In *ACL*, pages 328–339, 2018.

[17] T. Hughes, J. Smith, and A. Leavitt. Helping People Better Assess the Stories They See in News Feed with the Context Button. *Facebook*, 2018.

[18] K. Hunt, P. Agarwal, and J. Zhuang. Monitoring Misinformation on Twitter During Crisis Events: A Machine Learning Approach. *Risk Analysis*, 00(00), 2020.

[19] S. Jain, V. Sharma, and R. Kaushal. Towards automated real-time detection of misinformation on Twitter. *2016 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2016*, pages 2015–2020, 2016.

[20] S. Jo, I. Trummer, W. Yu, X. Wang, C. Yu, D. Liy, and N. Mehta. AggChecker: a fact-checking system for text summaries of relational data sets. *VLDB*, 12(12):1938–1941, 2019.

[21] D. Kang, P. Bailis, and M. Zaharia. Blazelt: Optimizing declarative aggregation and limit queries for neural networkbased video analytics. *VLDB*, 13(4):533–546, 2019.

[22] G. Karagiannis, M. Saeed, P. Papotti, and I. Trummer. Scrutinizer: A mixed-initiative approach to large-scale, data-driven claim verification. *VLDB (Conditionally Accepted - Preprint: https://arxiv.org/abs/2003.06708)*, 2020.

[23] G. Karagiannis, M. Saeed, P. Papotti, and I. Trummer. Scrutinizer: a mixed-initiative approach to large-scale, data-driven claim verification [Extended Technical Report]. Technical report, 2020.

[24] G. Karagiannis, I. Trummer, S. Jo, S. Khandelwal, X. Wang, and C. Yu. Mining an "anti-knowledge base" from Wikipedia updates with applications to fact checking and beyond. *VLDB*, 13(4):561–573, 2020.

[25] S. Krishnan, A. Dziedzic, and A. J. Elmore. DeepLens: Towards a visual data management system. *CIDR 2019 - 9th Biennial Conference on Innovative Data Systems Research*, 2019.

[26] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. pages 1–17, 2019.

[27] N. Lee, B. Li, S. Wang, W.-t. Yih, H. Ma, and M. Khabsa. Language Models as Fact Checkers? pages 36–41, 2020.

[28] A. H. Li and A. Sethy. Knowledge Enhanced Attention for Robust Natural Language Inference. 2019.

[29] J. Liu, C. Wang, C. Li, N. Li, J. Deng, and Z. Pan. DTN: Deep Triple Network for Topic Specific Fake News Detection. In *Web Semantics: Science, Services and Agents on the World Wide Web*, page 100646. Elsevier B.V., 2021.

[30] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv*, (1), 2019.

[31] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016*, 4:2850–2869, 2016.

[32] E. O. Nsoesie and O. Oladeji. Identifying patterns to prevent the spread of misinformation during epidemics. *Harvard Kennedy School Misinformation Review*, 2020.

[33] S. Rosenthal and K. McKeown. Detecting opinionated claims in online discussions. *Proceedings - IEEE 6th International Conference on Semantic Computing, ICSC 2012*, pages 30–37, 2012.

[34] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. pages 2–6, 2019.

[35] A. Sathe, S. Ather, T. M. Le, N. Perry, and J. Park. Automated fact-checking of claims from wikipedia. *LREC 2020 - 12th International Conference on Language Resources and Evaluation, Conference Proceedings*, (May):6874–6882, 2020.

[36] M. Scott. 'It's overwhelming': On the frontline to combat coronavirus 'fake news'. *Politico*, pages 1–16, 2020.

[37] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD*, pages 23–34, 1979.

[38] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu. Fake News Detection on Social Media: A Data Mining Perspective. *SIGKDD Explorations*, 19(1):22–36, 2017.

[39] Stable Baselines. https://github.com/DLR-RM/stable-baselines3.

[40] R. R. S. Sutton and A. G. A. Barto. *Reinforcement learning: an introduction*. 1998.

[41] I. Trummer, S. J. Mosley, J. Antonakakis, and S. Jo. SkinnerDB : regret-bounded query evaluation via reinforcement learning. *VLDBJ*, 11(12):2074–2077, 2018.

[42] I. Trummer, J. Wang, D. Maram, S. Moseley, S. Jo, and J. Antonakakis. SkinnerDB: regret-bounded query evaluation via reinforcement learning. In *SIGMOD*, pages 1039–1050, 2019.

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-Decem(Nips):5999–6009, 2017.

[44] K. Vault, H. Rank, and S. By. Google ' s Knowledge Vault Helps Rank Sites By Accuracy. *Popular Science*, pages 1–4, 2015.

[45] X. Wang, C. Yu, S. Baumgartner, and F. Korn. Relevant Document Discovery for Fact-Checking Articles. In *WWW*, pages 525–533, 2018.

[46] C. Wardle and E. Singerman. Too little, too late: Social media companies' failure to tackle vaccine misinformation poses a real threat. *The BMJ*, 372:1–3, 2021.

[47] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-Art Natural Language Processing. pages 38–45, 2020.

It's FREE to join!

TCDE

tab.computer.org/tcde/

The Technical Committee on Data Engineering (TCDE) of the IEEE Computer Society is concerned with the role of data in the design, development, management and utilization of information systems.

- Data Management Systems and Modern Hardware/Software Platforms
- Data Models, Data Integration, Semantics and Data Quality
- Spatial, Temporal, Graph, Scientific, Statistical and Multimedia Databases
- Data Mining, Data Warehousing, and OLAP
- Big Data, Streams and Clouds
- Information Management, Distribution, Mobility, and the WWW
- Data Security, Privacy and Trust
- Performance, Experiments, and Analysis of Data Systems

The TCDE sponsors the International Conference on Data Engineering (ICDE). It publishes a quarterly newsletter, the Data Engineering Bulletin. If you are a member of the IEEE Computer Society, you may join the TCDE and receive copies of the Data Engineering Bulletin without cost. There are approximately 1000 members of the TCDE.

Join TCDE via Online or Fax

ONLINE: Follow the instructions on this page:

www.computer.org/portal/web/tandc/joinatc

FAX: Complete your details and fax this form to **+61-7-3365 3248**

Name _____

IEEE Member # _____

Mailing Address _____

_____

Country _____

Email _____

Phone _____

TCDE Mailing List

TCDE will occasionally email announcements, and other opportunities available for members. This mailing list will be used only for this purpose.

Membership Questions?

Xiaoyong Du
Key Laboratory of Data Engineering and Knowledge Engineering
Renmin University of China
Beijing 100872, China
duyong@ruc.edu.cn

TCDE Chair

Xiaofang Zhou
School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, QLD 4072, Australia
zxf@uq.edu.au

IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1314

Non-profit Org.
U.S. Postage
PAID
Los Alamitos, CA
Permit 1398