



Cardiff University
School of Computer Science and Informatics
CM3203 ONE SEMESTER INDIVIDUAL PROJECT REPORT

Implementation and Analysis of Truth Discovery Algorithms

Joe Singleton

Supervisor: **Dr Richard Booth**

Moderator: **Dr Federico Cerutti**

May 2019

Abstract

With the vast amount of data available in today's world, particularly on the web, it is common to find conflicting information from different sources. Given an input consisting of conflicting claims from multiple sources of unknown trustworthiness and reliability, truth discovery algorithms aim to evaluate which claims should be believed and which sources should be trusted. The evaluations of trust and belief should cohere with one another, so that a claim receives a high belief ranking if it is backed up by trustworthy sources and vice versa.

This project investigates truth discovery from practical and theoretical perspectives. On the practical side, a number of algorithms from the literature are implemented in software and analysed. On the theoretical side, a formal framework is developed to study truth discovery from a general point of view, allowing results to be proved and comparisons made between truth discovery and related areas in the literature. Desirable properties of truth discovery algorithms are defined in the framework, and we consider whether they are satisfied by a particular real-world algorithm, *Sums*.

Contents

Contents	iii
List of Figures	vi
1 Introduction	1
2 Background	3
2.1 Related Areas	4
2.1.1 Resolving Conflicts in Data	4
2.1.2 Trust Analysis	5
2.2 Truth Discovery Background	5
2.3 Existing Work	7
2.3.1 Software Implementations	7
2.3.2 Theoretical Work	9
3 Software Implementation	11
3.1 Specification and Design	11
3.1.1 Datasets	14
3.1.2 Algorithms	22
3.1.3 Results and Evaluation	28
3.1.4 Visualisation	31
3.1.5 User Interfaces	34
3.2 Implementation	39
3.2.1 Programming Languages	40

3.2.2	Third-party Libraries Used	41
3.2.3	Truth Discovery Algorithms	42
3.3	Results and Evaluation	50
3.3.1	Real-World Dataset Demonstration	50
3.3.2	Synthetic Data Accuracy Experiments	52
3.3.3	Convergence Analysis	56
3.3.4	Time Complexity Analysis	58
3.3.5	User Interfaces	59
3.3.6	Testing	64
3.3.7	Evaluation against Requirements	68
3.4	Future Work	70
4	Theoretical Analysis	72
4.1	Approach	72
4.1.1	Overview of Approach	74
4.2	A Framework for Truth Discovery	75
4.2.1	Standard Definitions and Notation	75
4.2.2	Truth Discovery Definitions	76
4.2.3	Axioms	78
4.2.4	Iterative Truth Discovery Operators	88
4.3	Evaluation	93
4.4	Future Work	97
4.4.1	Unfinished Business	97
4.4.2	Future Directions	98
5	Conclusions	101
6	Reflection on Learning	103
	References	105
A	Web-interface Screenshots	111
B	Test Results	119
C	Proofs	124
C.1	Proof of proposition 1	124
C.2	Proof of proposition 2	125
C.3	Proof of proposition 3	126

C.4 Proof of proposition 4	126
C.5 Proof of proposition 5	127
C.6 Proof of lemma 1	127
C.7 Proof of theorem 1	129
C.8 Proof of theorem 2	134

List of Figures

3.1	Example truth discovery dataset as a list of claim tuples	15
3.2	Matrix representation of the dataset shown in figure 3.1.	16
3.3	CSV representation of the dataset shown in figure 3.1.	18
3.4	UML class diagram showing high-level design for datasets.	21
3.5	UML class diagram for algorithms and iterators.	27
3.6	Example of results of an algorithm as key-value mappings.	28
3.7	UML class diagram for results.	31
3.8	Results from figure 3.6 represented graphically	32
3.9	UML class diagram for graphical representations.	34
3.10	Command-line interface example	36
3.11	Command-line YAML output example	37
3.12	UML class diagram for user interfaces.	39
3.13	Python code for loading the stock dataset.	51
3.14	Results for the stock dataset for each algorithm.	52
3.15	Source trust distribution experiment on synthetic datasets.	53
3.16	Claim probability experiment on synthetic datasets.	54
3.17	Domain size experiment on synthetic datasets.	55
3.18	Convergence experiment on a large synthetic dataset.	57
3.19	Algorithm run time experiments on synthetic datasets.	58
3.20	Python code demonstrating simple use of the API.	60
3.21	Example output of the code in figure 3.20.	61
3.22	Example of CLI interface for running an algorithm.	62
3.23	Example of a Python unit test.	66

3.24	Status of the requirements in the finished implemetion.	69
4.1	Example of two equivalent truth discovery networks	80
4.2	Network where we do not wish for an IIA-type axiom to hold. . .	85
4.3	Network where some notion of independence may be applied. . . .	86
4.4	A network in which <i>Sums</i> does not rank all sources equally. . . .	92
4.5	A counterexample to independence for <i>Sums</i>	93
A.1	Basic view of the input form in the web interface.	112
A.2	CSV dataset entry in the web interface.	113
A.3	Advanced algorithm options in the web interface.	114
A.4	Example results of a single algorithm run via the web interface. .	115
A.5	Example results of several algorithms run via the web interface. .	116
A.6	Graph representation of results in the web interface.	117
A.7	Frame from an animation of the results of an algorithm.	118

Chapter 1

Introduction

There is an increasing amount of data available in today's world, particularly from the vast number of pages on the web, user-submitted content on social media platforms and blogs, and sensor data from scientific instruments. Data can be found in many different formats, from structured datasets to natural language articles, and from many disparate sources, e.g. news outlets, scientific institutions, and members of the public.

With this abundance of data, it is common to find information about a single object from multiple sources. An inherent problem faced when using such data is that different sources may provide *conflicting information* for the same object.

Conflicts have a variety of causes, including out of date information, poor quality data sources, errors in information extraction (when parsing natural language text, for example), and deliberate spread of misinformation. When it comes to finding information about an object with conflicting reports, the question is this: *which* information should we accept as correct?

Truth discovery has emerged as a topic aiming to tackle this problem of determining what to believe by considering also the *trustworthiness* of sources. A main principle in many approaches is that believable claims are those that are made by trustworthy sources, and that trustworthy sources generally make believable claims.

This project has both practical and theoretical components. On the practical side, we develop a robust software framework in Python for truth discovery, which supports running truth discovery algorithms on real-world datasets and evaluating their performance. A few popular algorithms from the literature are implemented, although the framework aims to be easily extendible, well-documented and well-tested so as to allow more algorithms and features to be implemented in the future. The framework will provide a uniform interface for users and researchers in truth discovery to run different algorithms on datasets, and compare behaviour between algorithms. Additionally, it could be used to aid development of new algorithms by evaluating performance against a number of existing algorithms. To demonstrate its capabilities, some basic analysis and evaluation of the implemented algorithms is performed.

For the theoretical component, we define a formal mathematical framework for truth discovery, highlighting parallels with other areas in the literature, especially the theory of voting in social choice [41]. Following the axiomatic approach used in social choice, we look for axioms (desirable properties) of truth discovery algorithms expressed in the formal framework, and consider which axioms are satisfied by a particular algorithm, namely *Sums* [27]. As well as providing some immediate results, this provides foundations for further theoretical work on truth discovery.

Chapter 2

Background

The fundamental problem of truth discovery is to resolve conflicts in a set of claims from different sources. A naïve approach is to apply a *majority vote*, where the claim made by the largest number of sources is accepted. Unfortunately, this is prone to yield poor results when the sources are not all equally trustworthy. For example, the study in [31] found that false information on Twitter is shared more quickly and more widely than true information. Applying a majority vote in the Twittersphere would therefore, in many cases, select the *false* information as correct.

The problem with the majority vote is that all sources are treated identically: a claim from one source carries as much weight as a claim from any other. This is contrary to how we judge the veracity of statements in everyday life, where claims from trusted colleagues have considerably more weight than claims from unknown persons (and especially people known to be *untrustworthy*). Trust can therefore be a valuable tool in resolving conflicting information, since one expects that trustworthy sources are more likely to make accurate claims than untrustworthy sources are.

Truth discovery therefore has two components: determining *trust* and *belief* in sources and claims, and resolving conflicts in data. These are tightly linked, since the trust evaluation is based on the claims in the input data, and the claims to accept are based on the trust evaluation.

Presently we outline related areas in the literature that deal with re-

solving conflicts and trust analysis, before providing some background on truth discovery itself. Then we review existing work relevant to the practical and theoretical components of the project.

2.1 Related Areas

2.1.1 Resolving Conflicts in Data

There are numerous areas in the existing literature that deal with resolving conflicts in data. In data mining, *data fusion* considers aggregating data from different sources into a single representation. Various approaches have been suggested (see [8] for a review): for example, taking a majority vote (as discussed above), taking the most recent value as correct, or ignoring objects entirely when a conflict exists.

In collective annotation [17], multiple individuals assign labels (annotations) to objects, which are to be aggregated into a single collective annotation. Different individuals may not agree on the appropriate labels for a given object; *aggregation functions* consider how to obtain the collective annotation given these conflicts.

Belief revision [28] is set out in a logical framework, and considers how to update a knowledge base upon receiving new information that could cause the knowledge base to be inconsistent.

Argumentation theory [7] takes an abstract view and considers a set of ‘arguments’ which conflict with each other (known as ‘attacking’). The structure of the arguments is abstracted away, and only the network of which arguments attack each other is considered. The aim is then to find sets of arguments that are acceptable and consistent.

In a more general sense, social choice also deals with conflicts in data. Here voters express preferences for a number of ‘alternatives’ (e.g. candidates for an election), and a social ordering of the alternatives is sought that reflects the will of the voters. Difficulties may arise when there is no consensus among the voters – e.g. one voter’s favourite outcome could be another’s least favourite. Although the notion of fairness present here is not applicable to truth discovery, the issue of conflicts in the preferences of voters is relevant.

2.1.2 Trust Analysis

Trust has been studied in many different domains for different applications (see [24] for a survey). In the social sciences and economics, trust between humans has been considered for the effects on economic transactions. E-commerce sites such as eBay use the concept of trust and reputation of sellers to inform buyers.

In wireless sensing, P2P networks and ad-hoc mobile networks, nodes are required to behave cooperatively for the network to function. Various approaches have been suggested for analysing the trust of nodes in these scenarios (e.g. see section 4.4 in [24]) to mitigate the effects of misbehaviour, e.g. due to hardware problems or malicious interference from an adversary.

It should be noted that some of these approaches to evaluating trust compute ‘local’ measures of trust from the perspective of a particular node. For example, a sensor in a network may evaluate the trust of its neighbours based on its interactions with them. The trust assigned to a given node may therefore vary as it is evaluated from different perspectives. This is not the case with truth discovery, where we seek an objective ‘global’ notion of trust in sources based only on the claims made.

Other works do not aim to *compute* trust, but instead use trust relationships between ‘agents’ in a multi-agent system for some purpose. One example is trust-based recommendation systems [5], where an agent is given a recommendation for an item of interest based on the recommendations of the agents it trusts. Other examples include personalised ranking systems [2], trust-based argumentation [29] and trust-based belief revision [9].

The trust considered here is again from the perspective of a given agent. Nevertheless ideas from these areas still may be relevant to truth discovery.

From a theoretical point of view, Marsh [23] provides a formal model for trust in multi-agent systems.

2.2 Truth Discovery Background

In the preceding sections truth discovery has been discussed only informally. Here we outline more precisely the main concepts in the basic

form of truth discovery, and discuss the various extensions to this basic form that have been addressed in the literature. More information can be found in survey papers on truth discovery methods [15, 21].

A *source* is an entity that provides (or *claims*) information, called *facts*, about *objects*.¹ A source may claim at most one fact for a given object. Different sources may provide different facts for the same object; a fact need not be ‘true’. It is often assumed that for each object there is a unique true fact. In the basic form of truth discovery, the nature of the facts is irrelevant, and they are treated as categorical values.²

A truth discovery algorithm takes the sources, facts, objects and the facts claimed by each source as its input, and outputs a measure of the *trustworthiness* of sources and the *believability* of facts. In an application of truth discovery for finding true facts, the fact determined to be most believable for a given object is taken to be the truth.

The precise definitions of input and output vary across the popular truth discovery literature. Definitions of input are generally compatible with each other, in the sense that they can be restated in terms of sources, objects and facts (as above) without changing the essence of the approach.

For example, in *Sums*, *Average-Log*, *Investment* and *PooledInvestment* [27] there is no concept of objects, but instead of *mutual exclusion sets* of facts which cannot simultaneously be true. However the mutual exclusion sets themselves can be seen as the objects linking related facts together.

Cosine, *2-Estimates* and *3-Estimates* [14] also have no concept of objects, and sources may make *negative claims* where they state a fact is *false*. However one may view the ‘facts’ (in the sense of [14]) as objects, and ‘true’ and ‘false’ as the only two facts associated with each object.

The form of the output varies more widely. The treatment of sources is generally the same: each source is assigned a *trust score*, (usually a number in $[0, 1]$) with higher scores indicating more trustworthy sources. Differences appear for the treatment of facts, however. The two main ideas are to assign each fact a *belief score* [27, 35, 36, 37], or to select a single ‘true’ fact for each object [34, 38, 40]. Another view, taken in [22] for example, is to produce a single value for each object to represent

¹Note that the terminology is not uniform across the literature; e.g. the survey in [15] refers to sources as ‘providers’, and the facts in [27] are referred to as ‘claims’ (i.e. the word ‘claim’ is used as a noun, whereas we use it as a verb).

²Some approaches instead make use of the specific data types of the facts in their calculations (e.g. [22]).

the true fact, but where the true value may not have been claimed by any sources (for example, a weighted average could be applied when facts are numeric values, and could lead to this situation).

Note that selecting a single fact for each object can be seen as a special case of assigning each fact a score; e.g. the true facts receive a score of 1 and all others receive 0.

In practise, most algorithms operate *iteratively*, computing trust and belief scores (or finding ‘true’ facts) until convergence or some stopping criterion is satisfied. For algorithms that assign trust and belief scores, facts are first assigned initial belief scores. At each iteration, the trust scores for sources are updated based on the current belief scores for the facts they claim; then the belief scores are updated based on the current trust scores for the sources. This mutual dependence of trust and belief scores is hoped to encode the idea that trustworthy sources are ones that claim believable facts, and believable facts are those claimed by trustworthy sources.

This basic formulation is a simple representation of the real world, and a number of extensions have been addressed to deal with more complex situations. Some approaches extend the basic model by requiring more information in the input (e.g. a set of ‘ground truths’ for semi-supervised truth discovery), whereas others keep the same basic input but consider more nuanced issues (e.g considering copying amongst sources). Extensions include: implications between facts [35]; heterogeneous data [22]; correlations between objects [34]; hardness of facts [14]; incorporating prior domain knowledge [27]; sparsity in claims [37]; semi-supervised truth discovery [36]; copying between sources [11]; time-varying truth [11]; and streaming data [39].

2.3 Existing Work

2.3.1 Software Implementations

Due to the wide range of domains in which truth discovery may be applied and the variety of approaches and additional considerations beyond the basic model (some of which are domain specific), there is not likely to be a single algorithm that is best suited for all applications.

Instead, for a given problem it is necessary to try several algorithms, or even tailor a bespoke one, to achieve the best results. Note that even evaluation may be domain specific: run time and memory efficiency may be critical in some cases (e.g. when dealing with large volumes of data in a scenario where real-time results are desired), whereas some applications may be insensitive to long run times but require precise accuracy.³

For this reason, there is a need for an openly available and extendible software framework for truth discovery, which allows different algorithms to be evaluated and compared in a uniform environment. When faced with a particular truth discovery problem, users may then run many algorithms on their data without additional effort for each algorithm. An easily extendible framework will allow them to define their own metrics for evaluation that are suitable for the dataset in question, and even modify algorithms to suit the type of data if required.

Despite the wide interest in truth discovery in research papers, there are few open source software implementations. One such implementation is *spectrum*⁴, available on GitHub. This library implements some algorithms from the literature, but lacks proper documentation, does not provide a uniform interface for getting results across algorithms, and does not support evaluating results for datasets for which some true values are already known. It also does not provide fine-tuned control for the running of algorithms, such as the threshold for determining when trust and belief scores have converged.

Another open source library is DAFNA-EA⁵, which is the implementation behind the comparative study in [18]. Whilst extensive in the number of algorithms implemented, its capability for evaluating performance and generation of *synthetic data*, it lacks documentation to allow its code to be extended (i.e. for users to define their own evaluation metrics or algorithms), and is not geared towards real-world applications of truth discovery. Additionally, a web interface is reportedly available, which may improve accessibility for non-technical users, but is not operational at the time of writing.

Other truth discovery implementations are available on GitHub and elsewhere,⁶ but these are generally repositories containing code used in

³ The meaning of ‘accuracy’ for truth discovery algorithms will be discussed in section 3.1.

⁴<https://github.com/totucuong/spectrum>

⁵<https://github.com/daqcri/DAFNA-EA>

the production of a research paper rather than general purpose libraries.

To produce a software framework for truth discovery that achieves the goals stated above and which addresses the deficiencies of the existing frameworks, this project will implement a selection of algorithms from the literature in a uniform way with a focus on extendability, which will include comprehensive documentation of both the code and the user interface. Fine-tuned control of parameters for initialising and running algorithms will be available. It will also identify standard metrics for evaluating algorithms (including generation of synthetic datasets), and provide means of comparing algorithms with respect to those metrics.

2.3.2 Theoretical Work

Many of the truth discovery algorithms in the literature are supported by some form of theoretical analysis. For example, the survey in [15] analyses the time complexity of various algorithms, the convergence of *semi-supervised truth finder* is proved in [36], and the approach in [33] is proven to converge to find the true facts under assumptions on the distributions of source reliability. In [32], a probabilistic framework termed *Unified Truth Discovery* is developed, and theoretical properties of this framework are proved under certain mild assumptions.

A limitation of this kind of theoretical analysis is that the results apply only to a single algorithm or class of algorithm. These results, whilst important in their own right, are not general enough to apply to truth discovery as a whole, and depend on the specific ideas and approaches in use.

As far as I am aware, at the time of writing there is no general theoretical framework capable of modelling *all* truth discovery algorithms, which allows general properties of such algorithms to be studied. Developing such a framework would allow algorithms to be compared with respect to their theoretical properties as opposed to purely practical ones, something which has not been done in the literature to date.

A general theory of truth discovery would also facilitate deeper comparison between truth discovery and other areas in the literature. For example, it was noted above that truth discovery bears similarity to be-

⁶ e.g. <https://github.com/lvlingyu11/Truth-Discovery-for-Crowdsourcing-Data> and <https://github.com/MengtingWan/KDEm>

belief revision, argumentation theory and, in a more general sense, social choice. Each of these areas have rich formal foundations which have provided useful results for both theoretical and practical purposes; developing similar foundations for truth discovery may reveal deeper similarities and allow results in these areas to be applied to truth discovery.

An approach that has seen great success in social choice is the *axiomatic method*, where axioms (desirable properties) of voting rules are stated, and rules are compared with respect to these properties. Such analysis can provide deep results; for example, K. Arrow's famous impossibility theorem [6] shows that it is impossible for a voting rule to simultaneously satisfy a few simple axioms that one would reasonably expect of a fair voting rule, thus proving a fundamental limitation of voting rules in general. The axiomatic approach has also been successfully applied in the related areas of ranking [2, 3], recommendation [5, 19], collective annotation [17] and belief revision [1].

This project will aim to define a theoretical framework for truth discovery that is general enough for any algorithm to be modelled, independent of the algorithm's specific approach. Following the axiomatic method employed in social choice and related areas, axioms for truth discovery algorithms will be developed to encode desirable properties. To demonstrate the framework's suitability as a tool for analysing real-world truth discovery algorithms, *Sums* [27] will be defined formally and analysed with respect to the developed axioms.

Chapter 3

Software Implementation

This chapter describes the software framework for truth discovery developed for the practical component of this project. First, high level requirements and the design of the system are discussed and justified.

3.1 Specification and Design

The broad goals and aims for the practical component of this project were outlined in section 2.3.1. In this section, these ideas are developed and made into precise requirements for the software. This is accompanied by a high-level description of how the software is designed to meet these requirements.

The primary use case for the system is applying truth discovery algorithms to real-world datasets to tackle real-world truth discovery problems. For example, a user may have collected information from various websites and wishes to use truth discovery to determine, as far as possible, which information is true and the trustworthiness of the websites. To distinguish between other types of users, we shall call such users *truth discovery practitioners*. To determine the best algorithm to use for their specific purpose, these users will also be interested in *evaluating* algorithms in various ways, such as time and memory complexity. Due to the

variety of diverse domains in which truth discovery can be applied, practitioners may also wish to define their own methods of evaluation specific to their type of data.

Another use case is algorithm development. Developing and testing a new truth discovery algorithm requires a lot of supporting infrastructure, such as methods for loading datasets and user interfaces. Additionally, one will often look to compare the new algorithm to existing ones, in order to determine in what sense the new algorithm is an improvement; this requires implementing existing algorithms and methods for evaluation.

Algorithm developers would therefore benefit from a library for truth discovery that provides the necessary supporting code, allowing them to focus solely on the development of the algorithm itself. Evaluating the existing and new algorithms with the same library also ensures comparisons are fair.

Both the ‘practitioner’ and algorithm developer roles require evaluating algorithms in some sense. An important measure of an algorithm’s performance is its *accuracy*, defined as the proportion of cases where the algorithm predicts the true fact for an object [20, 27]. In much of the truth discovery literature, accuracy is calculated by running an algorithm on a dataset for which true values are already known for some objects. We will refer to such datasets as *supervised* datasets. Supervised datasets are often constructed *synthetically*, where sources, facts and claims are generated randomly according to some statistical model, due to the difficulty in determining with confidence the true facts in real-world datasets [14, 22, 27, 35].

Accordingly, our system should provide methods for loading supervised datasets, both from real-world data and by synthesis, and for evaluating accuracy with respect to such datasets.

The final major use for the system is to be a tool for theoretical work. Users considering theoretical aspects of truth discovery, who we shall refer to as *theorists*, will often need to construct simple instances of truth discovery problems for examples and counterexamples, and run algorithms on these instances. They may also use a software library to empirically verify or disprove properties of certain algorithms.

Having outlined the target audience for the software implementation, their goals and the tasks they wish to perform, we can identify distinct components of the system and high-level requirements for each.

- **Datasets:** Datasets need to be loaded from suitable formats. This includes loading files on disk for large real-world datasets, and creating small examples by hand. Additionally, supervised and synthetic data generation should be supported.
- **Algorithms:** A selection of algorithms from the literature should be implemented. Users should have control over any parameters available, such as stopping criterion for iterative algorithms. The implementation should be extensible so that new algorithms can be developed without reimplementing code, to support the ‘algorithm developer’ use case.
- **Results and evaluation:** Results from algorithms should be presented to the user in a suitable format that reflects the aims of the user – the format may differ between use cases. To support evaluation, information such as run time, memory usage and the number of iterations should also be returned. It should also be possible for users to extend the code to define their own metrics, to support the ‘practitioner’ use case when the evaluation of an algorithm is domain-specific.
- **Visualisation:** For the ‘theorist’ use case, it is important that the system allows simple examples to be created by hand, and for these to be analysed in some way. Visual representations such as images and animations could be used to analyse results for small input datasets.
- **User interfaces:** Each of the main user types defined above have different reasons for using the system and may perform different kinds of tasks. A suitable user interface (or interfaces) should be available that reflect the aims of the user and allow the required tasks to be carried out as simply as possible.

Detailed requirements and design for datasets, algorithms and results will depend on the model of truth discovery that is adopted, since this will dictate the form of input and output. For our software implementation, we aim for a model that is general enough to support a wide range of

current and future algorithms, but does not stray too far from the model actually used in the definition of the algorithms we wish to implement.¹

For input, we shall use the sources, facts and objects model already described; it was noted that this approach is widely applicable to popular algorithms in the literature. However, we will use different terminology. It is anticipated that facts for objects will commonly represent *numeric values* for a *variable*, e.g. a source may claim the temperature in Celsius for the weather on the 27th April in Cardiff is 14°C. In such cases it is more natural to call ‘temperature’ a *variable* instead of an object, and to call ‘14°C’ a *value* instead of a fact. Whilst we will not actually restrict users to numeric values only, this terminology will occasionally be used instead of sources/objects/facts.² Since the same value may be used for multiple objects, the notion of a ‘fact’ is replaced by a variable-value pair.

For output, trust and belief scores for each source and value-variable pair will be used, as this is the more general form of output discussed in section 2.2.

We now describe the components listed above in more detail, set out precise requirements, and discuss high-level aspects of the design. Non-functional requirements, which describe how the system should *be* as opposed to what it should *do*, are also be discussed.

3.1.1 Datasets

Format

The key parts of a truth discovery dataset are the sources, the variables, and the values claimed by sources for these variables. In a real application of truth discovery, one wishes to use an algorithm to determine true values for the variables and to analyse the trustworthiness of sources; the sources and variables therefore need to be *labelled* in some way so they can be referred to in the results. Labels will most commonly be strings (e.g. ‘Met Office’, ‘Humidity’), but other data types such as integers and floating point numbers should be allowed too.

¹ The requirements for a model here differ from those in the theoretical part of the project (discussed in section 4.1), where we are less concerned with the model matching the definition of specific algorithms.

² The decision to use this alternate terminology was made early on in the project. Whether it actually improves or worsens clarity is up for debate.

```
(weather.com, 0.50, Humidity),  
(weather.com, 1014.2 mb, Pressure),  
(weather.com, 14°C, Temperature),  
(weather.com, 16.1 km, Visibility),  
(Met Office, 0.56, Humidity),  
(Met Office, 1014 mb, Pressure),  
(Met Office, 11°C, Temperature),
```

Figure 3.1: Example of a small truth discovery dataset expressed as a list of claim tuples. The data was collected manually from two weather forecasts for Cardiff on the 27th April 2019.

The first assumption made in this project is that these labels are unique identifiers for sources and variables. With this assumption, a claim consists of only three parts: a source label, a value, and a variable label. There is no need to define the sources and variables separately, since the whole set of sources and variables can be constructed from a list of claims. A dataset will therefore be represented as a *list of claim tuples*. An example is shown in figure 3.1. This format is simple to understand and work with by hand and in code, and leads to the first requirement.

Requirement 1. *Datasets can be loaded from a list of (source, value, variable) tuples. Each component of the tuple can be any reasonable data type, such as a string, integer, or floating point number.*

For use cases other than real-world applications of truth discovery, such as evaluation of algorithms for theoretical analysis and algorithm development, one is not interested in the results of truth discovery for its own sake, but for other aspects such as run time and accuracy on supervised datasets. In this case the labels for the sources and variables are irrelevant – the conflicts in the claims between sources is the only important detail. Labels can also be irrelevant even when one *is* interested in the results; for example when constructing a simple ‘fake’ example dataset by hand to analyse the results of an algorithm.

Having to create artificial labels in these cases would be inconvenient, so a format with ‘anonymous’ sources and variables is desired. One simple way to achieve this is to have a *matrix* with rows corresponding to sources, columns corresponding to variables, and claimed values as the

$$\begin{bmatrix} 0.50 & 1012.2 \text{ mb} & 14^\circ\text{C} & 16.1 \text{ km} \\ 0.56 & 1012 \text{ mb} & 11^\circ\text{C} & - \end{bmatrix}$$

Figure 3.2: Matrix representation of the dataset shown in figure 3.1.

entries of the matrix. Sources can then be referred to by their row number if necessary, but no explicit labels need to be given. An example is shown in figure 3.2. Note that the matrix may contain ‘empty’ cells when a source does not make a claim for all variables, such as the second source (‘Met Office’) for the final variable (‘Visibility’) in figure 3.2.

Requirement 2. *Datasets can be loaded from a matrix format with a row for each source and a column for each variable, where the entry at row i , column j is the value the i -th source claims for the j -th variable, or a special value denoting an empty cell.*

In all but trivial cases, datasets need to be stored and loaded from files on disk. Due to the limited existing software for truth discovery, there is no standardised format for storing truth discovery datasets. For this reason we consider ‘bespoke’ formats for the claim tuple and matrix formats described above.

The claim tuple format (requirement 1) is designed with real-world applications of truth discovery in mind. Creating a real-world dataset from scratch is a difficult process which requires considerable time and effort. To create a dataset of a reasonable size the process must be automated, which presents challenges for identifying the variables and claims, especially when dealing with unstructured sources of information such as web pages. Furthermore, different sources may use different names for the same variables (e.g. ‘Wind’ and ‘Wind Speed’) which must be consolidated – this is called *schema matching* [8] – and values in different formats (e.g. 14mph and 22.5kmph) need to be recognised and converted to a common form.

As such, it is expected that users will more often use data from existing work that they *did not collect themselves* – such data can be readily found on the web³ – and so will have no control over the format. With so many different formats in use (e.g. CSV files, JSON files, relational databases), there is no single format that can be used in all cases. Defining a fixed

file format for our system would therefore require users to first write ‘conversion code’ to re-write the data in the chosen format; this would take up additional storage space, require data to be processed twice, and add complexity for users.

Clearly some amount of code from the user is required, however, to load data from an unknown format. Ideally the user would write as little code as possible, specifying only how to extract (`source`, `value`, `variable`) tuples, with other implementation details abstracted away.

To achieve this, base classes will be provided that implement the core functionality for loading datasets, but leave the method of extracting claim tuples from the file undefined. Users will then extend these classes to implement this functionality according to the particular format in use.

This approach allows files from any format to be ready directly, as the conversion to claim tuple format can be done on the fly.

Requirement 3. *Users can load data from any custom format by implementing a function that extracts (`source`, `value`, `variable`) tuples from an input file.*

The situation is not the same for the matrix format (requirement 2), which is intended for evaluation and analysis of algorithms on synthetic and hand-made datasets. In this case the datasets are more likely to be used within the scope of this software only, and will be created by users themselves, so the issues with defining a fixed file format described above do not apply.

We opt to use a plain text CSV (comma separated values) format for matrix datasets. CSV files are widely used in many domains, can be created and edited easily in a text editor or in spreadsheet software, and naturally represent a matrix: each line in the file represent a row, and values within a row are separated by commas. Note that empty cells can be conveniently denoted by the ‘empty string’. Figure 3.3 shown an example dataset in CSV format.

Requirement 4. *Datasets can be loaded from a CSV file representing a matrix.*

³ For example, there are five real-world datasets available to download at <http://lunadong.com/fusionDataSets.htm>.


```
0.50,1012.2mb,14°C,16.1km  
0.56,1012mb,11°C,
```

Figure 3.3: CSV representation of the dataset shown in figure 3.1.

Supervised and Synthetic Data

Supervised data is often used to evaluate the accuracy of truth discovery algorithms by considering the proportion of variables for which the predicted value is correct. Supervised datasets can be obtained by manually determining true variable values in a real-world dataset, or by creating the dataset synthetically. Both approaches are useful in their own ways: real-world supervised data may be a realistic indicator of accuracy, while synthetic data is easier to obtain and allows parameters of the dataset to be precisely controlled (e.g. the number of sources and variables, the proportion of true claims, the variation in claimed values etc.).

Note that supervised data is still useful if true values are only known for a *subset* of variables. Running an algorithm on the full input data provides more information for the algorithm to base its judgment on, and the results can be evaluated based only on the subset. In any case, for real-world datasets with hundreds of variables it is infeasible to manually find true values for *all* variables with a high degree of confidence.

A supervised dataset therefore consists of a regular dataset, as defined in requirements 1 or 2, along with a subset of variables and associated true values.

For datasets in claim tuple form, true values can be represented simply as a list of (*true_value*, *variable*) tuples. For the matrix format, they can be represented in an additional row.

Requirement 5. *Supervised datasets can be loaded from the following formats:*

- *a list of (*source*, *value*, *variable*) tuples together with a list of (*true_value*, *variable*) tuples for a subset of the variables;*
- *a matrix of claimed values, where the first row contains the known true values.*

For generating synthetic data, various methods have been used in the literature.

In some cases synthetic data is not completely artificial, but generated based on real-world true facts [22, 27]. The number of artificial sources is fixed and *reliability levels* set, either by random selection [27] or provided as input to process [22]. Claims are then generated according to the source reliability levels. In some methods, source reliability is the *probability* that any given claim from a source is true; sources choose the correct value with this probability and choose uniformly from a set of pre-generated false values otherwise [27]. Other methods do not explicitly generate false facts, but generate a source’s claims by adding different levels of *noise* to the true values, where the amount of noise is inversely proportional to the source reliability [22].

There are also papers describing entirely artificial data generation [14, 35]. These approaches are broadly similar to the ones described above, but the ‘true values’ for variables are created randomly first.

Note that sources do not make a claim for *every* variable in the generated set of claims, since this would not reflect the sparse nature of real-world datasets. The number of sources for a variable can be determined first and the appropriate number of sources selected randomly [27, 35], or one can specify the probability of a source making a claim on any particular variable and make a probabilistic choice for each source-variable pair [14].

For our truth discovery implementation, we aim for a simple and general method that can be used with a variety of algorithms. A simple approach will provide value initially, and may be extended in future work to model more specific qualities of truth discovery datasets. Taking a more complex and opinionated view from the outset would instead make it difficult to adapt to future use cases, and risks tailoring the synthetic data generation to a particular algorithm or type of data.

We will therefore generate synthetic data wholly artificially, i.e. not based on real-world truths, using a fixed set of categorical values for variable values. The user will specify the number of sources and variables to generate, a reliability value for each source, the probability that a source makes a claim for any particular variable, and the number of possible values for variables.

For each variable, a uniform random number in $\{0, \dots, d - 1\}$ will be generated as the ‘true’ value, where d is the number of possible values (specified by the user). Source reliability will have the probabilistic interpretation mentioned above: a source with reliability p chooses the ‘true’

value with probability p , and any other value with probability $\frac{1-p}{d-1}$; that is, the sources chooses a false value uniformly.

Having the source reliabilities specified by the user instead of generated randomly allows for different scenarios to be considered – for example one can investigate the difference in the behaviour of algorithms when most sources are trustworthy and when most are untrustworthy. In any case, the user is free to generate source reliabilities randomly themselves beforehand.

Requirement 6. *Synthetic datasets can be generated based the approach described above, using user-supplied parameters.*

In practise, synthetic data is used to evaluate and compare different algorithms. It is important that the algorithms are run on exactly the same dataset, particularly when random chance is involved. The software must therefore support saving synthetic datasets to files; a natural choice is to use the same CSV format proposed for supervised data to avoid introducing an extra format.

Requirement 7. *Synthetic datasets can be saved in the CSV format for supervised data defined in requirement 5.*

Large Data

Real-world truth discovery datasets are often very large. For example, consider the *stock dataset* curated by the authors in [20]. Information about 1,000 stock symbols from 55 sources was collected for each week day in July 2011. For each stock symbol, 16 attributes for stocks were identified. Data was obtained for 21 days, resulting in $21 \times 16 = 336$ variables for each stock, and 336,000 variables in total. In my own experiments, I found that not all sources made a claim for all variables; even so, across the 55 sources there are 2,843,803 claims. Since this dataset only covers a period of one month, one may imagine that even larger datasets are possible.

Indeed, the authors in [36] performed experiments on a highly diverse web-scale dataset consisting of 65.7 million variables, 591 million claims, and 33,000 sources.

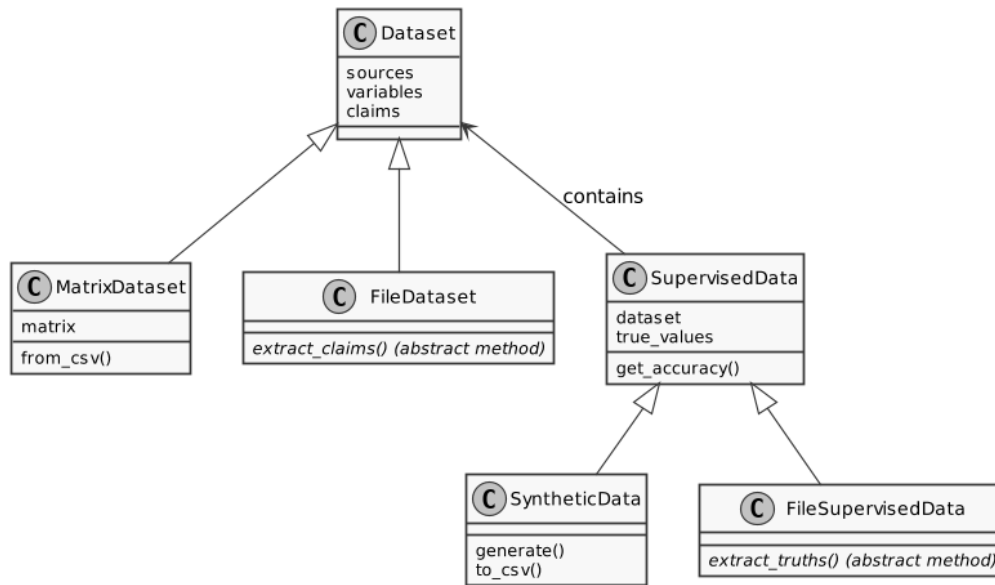


Figure 3.4: UML class diagram showing high-level design for datasets.

Any serious truth discovery implementation must therefore be able to handle ‘large’ datasets. Clearly ‘large’ is not a precise term, but nevertheless we state the following non-functional requirement.

Requirement 8. *The system supports loading and running algorithms on large scale datasets without significant slow-down.*

Design

With requirements for datasets defined, the main classes required for dealing with input data can be identified. The core class, called `Dataset`, will implement loading data from the claim tuples format defined in requirement 1. This is the most general form of input: the matrix form in requirement 2 can be reduced to this form by labelling the sources and variables according to their respective row and column numbers. A class `MatrixDataset` will therefore be a specialisation of `Dataset`.

Supervised data consists of a dataset and true values. In order to use any kind of dataset (`Dataset` or `MatrixDataset`), the `SupervisedData` class will hold a `Dataset` object; i.e. composition is used instead of in-

heritance. Synthetic datasets are special cases of supervised data, so a `SyntheticData` class will be a specialisation of `SupervisedData`.

For loading data from custom file formats, as set out in requirement 3, classes `FileDataset` and `FileSupervisedData` will implement the necessary functionality, except the format-specific details. These will be *abstract classes*; they must be sub-classed for the unimplemented methods to be defined.

A UML class diagram [30] depicting the class hierarchy is shown in figure 3.4.

3.1.2 Algorithms

The core ingredient for a truth discovery library is the implementation of truth discovery algorithms, but so far the actual selection of algorithms that will be implemented has not been discussed. In this section we briefly describe the various kinds of algorithms defined in the literature, and set out which ones will be implemented. Other aspects concerning their implementation will then be considered.

Algorithm Selection

Many truth discovery algorithms have been described in the literature, with wide variety in the approach and methodology used. The authors in [20] define five categories: *baseline*, *web-link based*, *IR based*, *Bayesian based*, and *copying affected*. Baseline methods are basic data fusion methods such as majority voting. These methods are not intended to be serious contenders for truth discovery, but are useful for evaluation of ‘real’ algorithms by comparison; a truth discovery algorithm should surely perform better than majority voting to warrant the additional computational cost.

Web-link based algorithms use the structure of links between sources, facts and objects to iteratively compute trust scores for sources and belief scores for facts. Examples include *Sums*, *Average-Log*, *Investment* and *PooledInvestment* [27]. They are inspired by algorithms that measure authority of web pages based on hyperlink structure, such as *Hubs and Authorities* [16] and *PageRank* [25].

IR (information retrieval) based methods use similarity measures popular in information retrieval, such as cosine similarity, to measure similar-

ity between sources' claims, and use this to infer trust and belief. Examples include *Cosine*, *2-Estimates* and *3-Estimates* [14].

Bayesian based methods employ Bayesian probability and statistics to, roughly speaking, determine the *probability* that sources will claim true facts. Examples include *TruthFinder* [35], *LDT* [38], *BCCTD* [12], *PTDCorr* [34], and the algorithms defined in [33].

Copying aware algorithms consider copying relationships between sources, and aim to reduce the trustworthiness of sources that copy from others. Examples include the algorithms defined in [11].

The survey in [21] notes an additional class of algorithms that define truth discovery as an optimisation problem. Sources are assigned weights (i.e. trust scores), and variables assigned 'true' values. The objective function, to be minimised, is the sum of the distances between 'true' values and source claims, with each distance weighted by the source weight. Distances must be measured using a metric suitable for the data type of the variables.

In terms of implementation, most algorithms operate in an iterative fashion, and alternate between source trustworthiness and true fact inference steps. The inference steps vary from simple update rules (e.g. *Sums*, *Average-Log* and *friends*), to complex operations involving several hyperparameters (e.g. Bayesian statistical methods).

Ideally, the implementation for this project would cover a range of algorithms from each of the above categories, including both simple and complex ones. However there is limited time available, and other aspects of the work besides the implementation of algorithms to consider. For this reason we will require only that some of the simpler algorithms are implemented.

The simplest method to implement is undoubtedly the baseline majority voting method. Voting will also be useful for evaluation of algorithms, which is an important consideration for this project. For non-baseline methods, it is my view that the web-link based algorithms proposed in [27] are the simplest to implement, along with the more straightforward Bayesian methods such as *TruthFinder*. Implementing 'simple' algorithms only still provides a useful framework, in which more advanced algorithms can be considered in future work.

Requirement 9. *The algorithms implemented include baseline majority voting, Sums, Average-Log, Investment, PooledInvestment and TruthFinder.*

Parameters

Many algorithms have depend on various parameters that control their operation. Some parameters are specific to particular algorithms, whereas others apply widely to whole classes of algorithms.

It is important that the system allows users to fine-tune these parameters. This is particularly important for evaluation use cases, where one may be interested in not just comparing different algorithms against each other, but comparing instances of the same algorithm with different parameters. Additionally, some parameters have no semantic meaning (e.g. g for *Investment* and *PooledInvestment*), so that experimentation may be the only sensible way to choose values for them.

All the algorithms listed in requirement 9 (except baseline voting) operate *iteratively* and *recursively*, updating trust and belief scores based on the scores in the previous iteration. There are two parameters that apply globally to this class of algorithm: the mode of iteration and *priors*.

The recursion aspect requires that initial trust or belief scores are specified. Four of the algorithms listed use fact beliefs as the initial values (called *priors* in [27]), so we will do the same. Users should be able to select the method of assigning these initial scores; for example all facts could receive the same score (referred to later as *fixed priors*), or belief could be distributed evenly amongst facts for the same object (*uniform priors*).

Requirement 10. *The method of assigning prior belief scores can be specified when running an algorithm.*

The iterative aspect means that algorithms run until a pre-defined stopping criterion is satisfied. The stopping criterion is clearly an important parameter, as the results may be greatly affected by it – both in terms of how well truths are discovered in the data, and in terms of run time.

A basic method of iteration is to simply perform a fixed number of iterations in all cases. More commonly though, algorithms iterate until ‘convergence’ of source trust scores, i.e. until trust scores settle down to fixed values.

In a truth discovery problem with m sources, the set of trust scores can be seen as a vector in \mathbb{R}^m , which is a *metric space* when equipped with a suitable metric. Convergence of a sequence in a metric space (X, d) is

defined as follows: a sequence $(x_n)_{n \in \mathbb{N}}$ in X converges to $y \in X$ iff for any $\epsilon > 0$ there exists $N \in \mathbb{N}$ such that $d(x_n, y) < \epsilon$ for all $n > N$.

The convergence (or otherwise) of a sequence in this sense cannot be determined via purely computation means, since infinitely many terms would have to be considered.⁴ A common heuristic is to instead iterate until the distance between successive terms passes below a fixed (small) threshold δ .

For the metric d , common choices for \mathbb{R}^m include the metrics induced by the ℓ_p norm (for $p \geq 1$) and the *infinity norm*, which are $d(\vec{x}, \vec{y}) := (\sum_{i=1}^m |x_i - y_i|^p)^{\frac{1}{p}}$ and $d(\vec{x}, \vec{y}) := \max_{1 \leq i \leq m} |x_i - y_i|$ respectively. The cases $p = 1$ and $p = 2$ of the ℓ_p norm, called the *Manhattan* and *Euclidean* norms respectively, are widely used.

Some authors consider a looser sense of convergence, where a function that does not qualify as a metric is used to measure distances between iterations. For example, the authors of *TruthFinder* use cosine distance to determine when iteration should stop, but cosine distance does not satisfy the triangle inequality and is therefore not a metric.⁵

To facilitate such approaches, we will use the term *distance measure* to mean any function $\mathbb{R}^m \times \mathbb{R}^m \rightarrow [0, \infty)$ that is used to compare the trust scores in successive iterations.

Two parameters required for convergence until iteration are therefore the distance measure d and the threshold distance δ . It is possible that the distance always remains above δ ; to avoid an infinite loop a maximum iteration count should also be given.

Requirement 11. *The mode of iteration can be specified when running an iterative algorithm. The mode may be one of:*

- **Fixed iteration**, where a fixed number of iterations are performed;
- **Convergence iteration**, where iteration continues until the distance between trust scores in successive iterations, measured by a user-specified

⁴ This issue is discussed in more detail in section 4.2.4.

⁵ Cosine distance is defined as 1 minus cosine similarity:

$$d(\vec{x}, \vec{y}) := 1 - \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|}$$

where \cdot denotes the dot product and $\|\cdot\|$ the Euclidean ℓ_2 norm.

distance measure, becomes smaller than a user-specified threshold level, or a maximum number of iterations are performed. The available distance measures include ℓ_1 , ℓ_2 , ℓ_∞ and cosine distance.

Finally, particular algorithms may have their own parameters, such as the ‘dampening factor’ in *TruthFinder*. Such parameters should also be possible for the user to define.

Requirement 12. *Algorithm-specific parameters can be specified by the user when running an algorithm.*

Development

For the development use case, users need to extend code and define their own algorithms. This clearly requires some interaction with the code base, but should not require users to know implementation details of other (unrelated) areas of the software, such as loading datasets and user interfaces. Even some algorithm-related functionality should be handled external to the user’s code, such as checking the stopping criterion and reading parameters from user input.

To achieve this we aim to isolate the algorithm-specific code so that it can be overridden without needing to reimplement generic functionality. Doing so lowers the barrier to entry for algorithm developers, requiring less time investment to start working with the library. We state a non-functional requirement to capture these ideas.

Requirement 13. *Users can implement new algorithms without needing to reimplement or know implementation details of unrelated parts of the software.*

Design

As with the design for datasets in section 3.1.1, the class hierarchy for the implementation of algorithms can now be set out so as to meet the requirements defined above.

It was noted above that the implementation of an algorithm includes both generic functionality and the core steps the algorithm itself. An abstract base class `BaseAlgorithm` will contain the generic functionality, and each individual algorithm will be implemented as a sub-class.

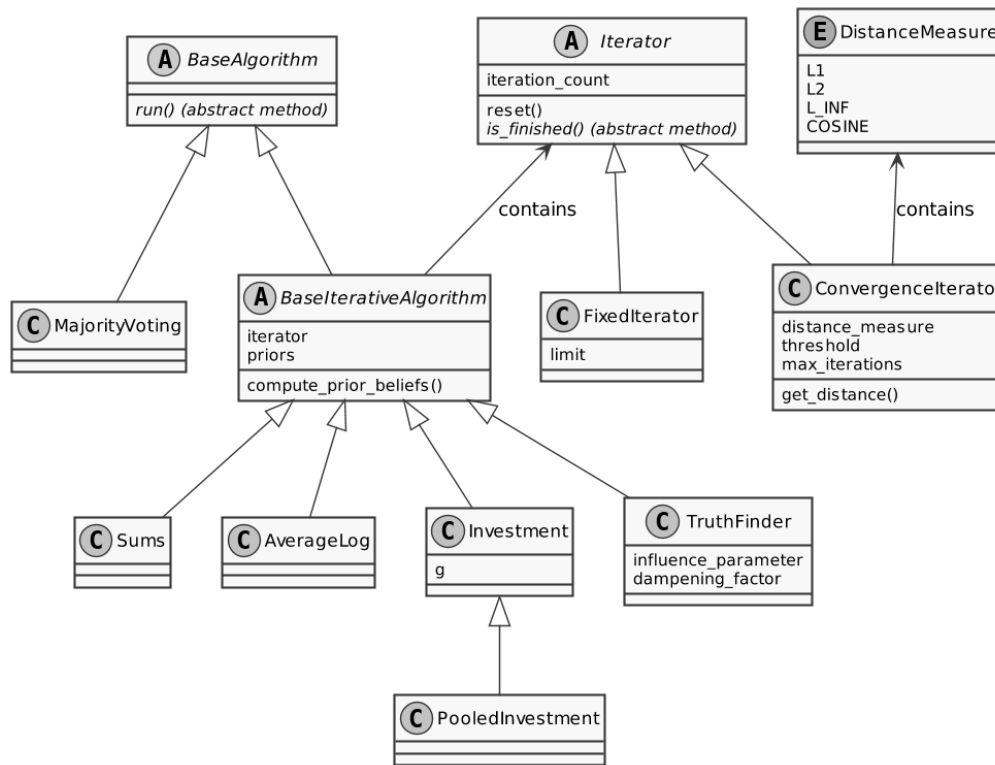


Figure 3.5: UML class diagram showing high-level design for algorithms and iterators.

Iterative algorithms have yet more generic functionality, such as initialising iteration and prior beliefs; this will be implemented in a `BaseIterativeAlgorithm` class.

For iteration, it should be possible to use either fixed or convergence iteration with any algorithm. As such the iteration logic itself – particularly the stopping criterion for convergence – should be defined outside the algorithm classes themselves. This avoids repetition of code in pursuit of requirement 13. A base class `Iterator` will implement any common functionality, with `FixedIterator` and `ConvergenceIterator` sub-classes providing the actual logic. Each iterative algorithm object will then have an `Iterator` instance as a parameter.

Finally, we note that `PooledInvestment` is a specialisation of `Investment`, and so will be implemented as a sub-class.

```
Trust: {
  "source 1": 1.0,
  "source 2": 0.5,
  "source 3": 0.5,
  "source 4": 0.75,
  "source 5": 0.1
}
Belief: {
  "var 1": {
    "7": 0.9, "8": 0.3, "43": 0.01
  },
  "var 2": {
    "green": 0.7, "red": 0.9, "dark red": 0.93
  }
}
```

Figure 3.6: Example of the raw results of a truth discovery algorithm as key-value mappings.

These classes and their relationships are illustrated in the UML class diagram in figure 3.5.

3.1.3 Results and Evaluation

Having run a truth discovery algorithm, users need to obtain the results in some way. Results include the raw trust and belief scores as produced by the algorithm, and information about the running of the algorithm itself, such as time/memory usage and the number of iterations.

Evaluation of algorithms – which is a key task for all the use cases defined at the start of this chapter – will be based solely on the results they return. As such the requirements for evaluation may shape the form results are presented in, so we consider results and evaluation together in this section.

For the ‘practitioner’ use case, where algorithms are run on real-world datasets, users clearly need to access the trust and belief scores given by the algorithm. For trust scores, a key-value mapping that maps source labels to trust scores is a convenient representation. For belief scores, each variable has a number of proposed values, which in turn have belief

scores. A two-level mapping can be used here: keys at the outer level are the variable labels, and the proposed values are mapped to belief scores at the inner level. An example is shown in figure 3.6.

There are a number of derived results that can be obtained from the raw trust and belief scores, including the value for each variable with maximum belief score (these are often taken as the ‘true values’ in applications), and statistics regarding the trust and belief scores, e.g. the mean and standard deviation. For convenience, these calculations will be implemented in a method that users can call as required.

Finally, recall that in a real application there may be hundreds or thousands of sources and variables. It may be that only a subset variables are of interest, so it should be possible for users to ‘query’ their results and only include specified sources and variables.

Requirement 14. *Results of an algorithm are given in the key-value mapping format shown in figure 3.6. Methods are available to obtain the fact for a given variable with maximum belief score, and the mean and standard deviation of trust and belief scores. Results can also be limited to a specified set of sources and variables.*

When it comes to evaluation of algorithms, two important metrics are time and memory usage. Time usage is straightforward to calculate, but memory usage needs to be defined precisely. For example, it could be interpreted as the maximum memory in use at any given time during the algorithm’s operation, or the total memory allocated; these measurements could be vastly different. We defer the precise meaning to the implementation.

Time and memory usage are useful for real-world uses, where one may compare different algorithms on the same data to determine which is more efficient, but also for algorithm development and theoretical purposes, where the asymptotic complexity can be estimated by running an algorithm on datasets of increasing size (such datasets could be generated synthetically).

For evaluation of iterative algorithms, analysing the behaviour of convergence is also important. This includes the total number of iterations taken, and the distances between trust scores in successive iterations over time. For example, does the distance decrease linearly, or does the convergence ‘slow down’ as iteration progresses? Users may answer such questions if given detailed information on convergence is provided.

Requirement 15. *Time, memory usage and iteration statistics (where applicable) are returned alongside the raw results of an algorithm.*

Another important aspect of evaluation is accuracy on supervised datasets. As mentioned previously, this is usually defined as the proportion of variables where the value with highest belief score is the correct one according to the supervised data. However, the precise accuracy calculation may vary depending on the type of dataset in use: consider a case where the true value of a variable is ‘8’, but the most likely value according to an algorithm is ‘8.00’. Whether this is acceptable as a correct answer depends on the context in which truth discovery is being applied.

Nevertheless, a basic implementation of accuracy can be provided that compares values exactly (i.e. ‘8’ would be considered distinct from ‘8.00’). This will be useful in most cases, and users may extend this implementation for their own more specific accuracy calculations if required.

Requirement 16. *Users can calculate the accuracy of a set of results with respect to a supervised dataset.*

Finally, the ‘theorist’ use case involves analysing theoretical properties of algorithms. One interesting task is to compare the results of an algorithm between two datasets, i.e. to study the effects of a small change in the input data.⁶ This leads to the final requirement for results and evaluation.

Requirement 17. *Two sets of results can be compared to view*

- *changes in trust and belief scores (for sources and facts present in both results);*
- *differences in time and memory usage, and the number of iterations taken (where applicable)*

Design

The class structure for results and related functionality is illustrated in figure 3.7. The `Result` class will represent the results of an algorithm, and has fields for trust and belief scores (in the key-value format described

⁶ See the *Monotonicity* axiom in section 4.2.3 for an example of the type of change that could be studied.

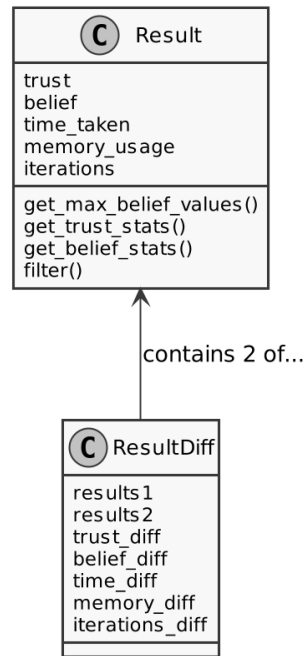


Figure 3.7: UML class diagram showing high-level design for results.

above), time and memory usage, and the number of iterations. Note that the number of iterations is not applicable to all algorithms – it does not apply to majority voting – yet we include it in the results class. This is because we expect that voting is an outlier in this respect, and that almost all future algorithms to be implemented *will* be iterative.

Comparing results will be done using the `ResultDiff` class, which stores the differences between the fields in its two component `Result` objects.

3.1.4 Visualisation

For the ‘theorist’ use case defined previously, we stated that users will be interested in creating small examples of truth discovery problems to analyse the behaviour of algorithms. This involves inspecting results, and comparing results in different cases. Whilst this is possible using the key-value form of results defined in the previous section, a visual representation is more suitable.

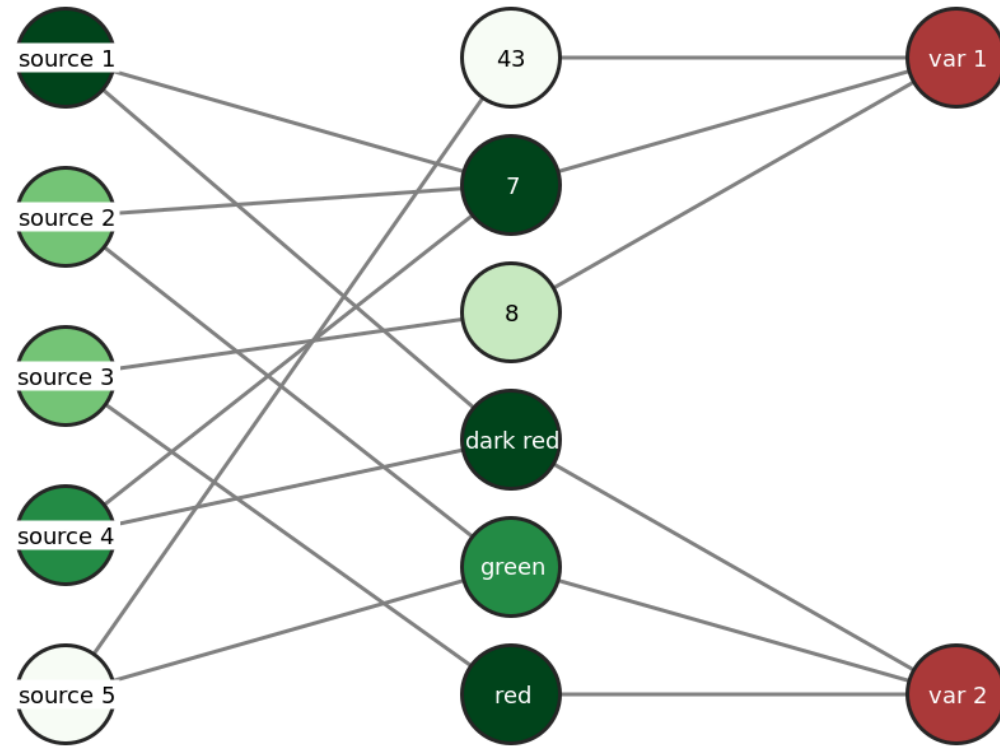


Figure 3.8: Results from figure 3.6 represented graphically. Darker colours correspond to higher trust and belief scores.

For example, consider the set of results that were given in the textual key-value pairs format in figure 3.6. They may alternatively be represented by the colour-coded graph, shown in figure 3.8, where darker colours correspond to higher trust and belief scores.

It is my view that the visual representation is much more useful for extracting key information at a glance. For example, one immediately sees which sources are most and least trustworthy, and which values are most and least believable. It can also aid in developing intuition for what is going on, which is important for theoretical work. Note that this only applies to small datasets: any more than a handful of sources, values and variables would cause the graphs to become overly crowded.

Graphs can also be useful for visualising datasets alone, i.e. without colouring the nodes according to the results of some algorithm. For ex-

ample, chapter 4 includes several figures that demonstrate datasets with particular properties, and datasets that provide counterexamples for particular properties of algorithms.

Requirement 18. *A dataset can be represented visually as a graph with nodes for sources, variables and values, and edges indicating connections between them.*

The results of an algorithm can additionally be shown in the graph by colouring sources and values according to their respective trust and belief scores.

We note that the graph representation corresponds to the definition of input to a truth discovery problem in the theoretical analysis of chapter 4.

In addition to visualising the end results of an algorithm, one may wish to visualise the *convergence* of results. Requirement 15 partly addresses this by ensuring users can access statistics regarding the convergence of trust scores for all sources as a whole. However for small datasets where a graphical representation is feasible, we can do even better, and visualise the convergence of scores for individual sources by means of an *animation*. This will simply be a sequence of graph images coloured-coded according to trust and belief scores as described above.

Requirement 19. *Animations can be generated that show the results of an iterative algorithm at each iteration as a colour-coded graph.*

Design

A UML diagram showing the design and class hierarchy for graphing datasets and results is shown in figure 3.9. Note that this diagram references classes first defined in previous sections, namely `Result`, `Dataset` and `BaseIterativeAlgorithm`.

The core class for producing graphs will be `GraphRenderer`. It will contain a `Dataset` object and a `ColourScheme` object for specifying the colour palette; other graphical settings (node size, border widths etc. . .) are encapsulated in the `display_settings` field in the diagram for brevity. The `ResultsGradientColourScheme` class will implement colouring nodes according to their scores in the results of an algorithm, and is a specialisation of `ColourScheme`.

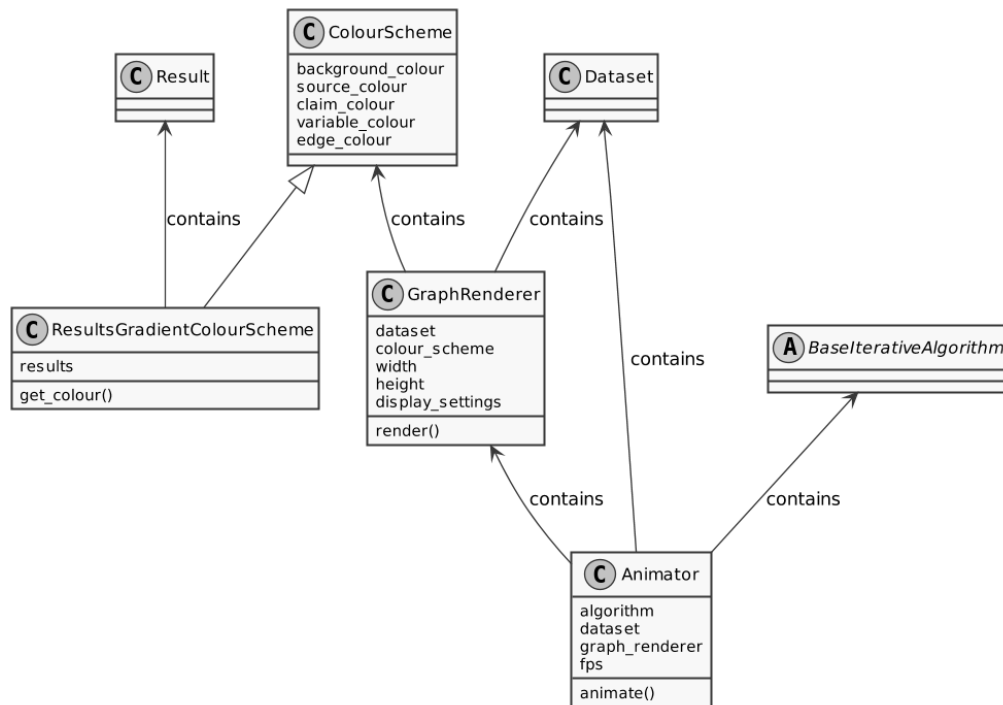


Figure 3.9: UML class diagram showing high-level design for graphical representation of datasets and results.

For animations, an `Animator` class will take dataset and algorithm objects as input, and create an image for each partial result as the algorithm progresses. Individual frames will be rendered via a `GraphRenderer` to avoid duplication of functionality.

3.1.5 User Interfaces

So far we have discussed the form of users' interactions with the system, but not the specific interface they will use. In this section we consider appropriate user interfaces for the use cases and tasks described.

Since the use cases have sometimes widely varying goals and constraints, there is no single interface that will be suitable for all purposes. For example, using the software as a tool for theoretical analysis requires building datasets by hand and detailed inspection of results. By contrast, real-world applications need to load datasets from files, and the scale of

the data makes it impractical to look at detailed results for each source and variables.

There is also variation in aims for the same broad use case: e.g. algorithm development clearly requires interacting with the code itself on the one hand (i.e. using an API interface from code), but may also require investigating results on smaller datasets to get a feel for its behaviour. The latter could also be done from code, but a graphical interface could allow for simpler data entry and visualising results.

With this in mind, we propose three separate user interfaces:

- **API:** a simple and well-documented Python API⁷ covering the entire codebase will support algorithm development, loading datasets from bespoke formats and integrating the library with other code. Additionally, an all-encompassing API ensures that all functionality is available to users in at least some form, even if it is not implemented in other more accessible interfaces.
- **Command-line:** this will be suitable for tasks involving large datasets – where graphical representations are infeasible – and for interfacing with other code at a higher level (for example, to be used with programming languages other Python). Also, well designed command-line interfaces can often be simpler to use than graphical ones, particularly when there are many sub-commands and options available.
- **Web-based:** for small-scale datasets and non-technical users, a web-based interface will be provided. This will make it easy for new users to try out the software without learning a command-line interface or API. It also allows the graphical representations discussed above to be presented in a simple way.

API

In this context, by API we simply mean a set of public-facing classes for users to interact with in their own code. These classes should allow the user to have full control over operation of algorithms, datasets etc..., whilst providing a simple interface that does not require knowledge of

⁷ Justification for using Python as the programming language will be provided in section 3.2.

```
truthdiscovery run --algorithm sums --dataset synth-data.csv \  
  --supervised --output time accuracy trust
```

Figure 3.10: Example of command-line arguments for running an algorithm on synthetic data.

unrelated implementation details. In particular, the user should be able to treat the system as ‘black box’, providing their input and receiving output without consideration for how the implementation actually works.

The design of the classes that comprise the software should therefore consider the tasks users wish to perform, which details are relevant to them, and what should be hidden as an implementation details.

Beside the code itself, thorough documentation and a suite of examples are essential for a successful API.

Requirement 20. *A Python API is available that allows all features of the system to be accessed without detailed knowledge of the system as a whole.*

Detailed documentation and examples of API usage are also provided.

Command-Line

A command-line interface is suitable for many of the tasks described throughout this chapter, particularly those for which input and output needs to be stored in files, where data is large and cannot be reasonable represented visually, and where few steps of interaction is required. It will also allow the library to be used programmatically with other projects, particularly if a machine-readable output format is used.

As an example, consider evaluating accuracy, run-time and trust scores for a particular algorithm on a synthetic dataset. The use supplies a few simple parameters, including the file to read the CSV dataset from, and receives some simple output. An example of how this could look, following the established conventions for command-line programs in the UNIX world, is shown in figure 3.10.

Here `truthdiscovery` is the name of the program, `run` is the relevant sub-command, and options are given in the long format used by many GNU utilities.

Output should be printed to `stdout` – this complies with established conventions and allows users to either inspect results by eye in their ter-

```
sums:
  accuracy: 0.625
  time: 0.1234
  trust:
    0: 0.2
    1: 0.4
    2: 1.0
```

Figure 3.11: Hypothetical YAML output for the command shown in figure 3.10.

minal for small scale datasets, or use output redirection functionality from their shell to save output to a file or pipe to other programs.

In terms of the format of the output, YAML will be used.⁸ YAML is a data serialisation format that is both human and machine readable, with implementations available for wide range of programming languages. A hypothetical example for the output of the command in figure 3.10 is shown in figure 3.11.

The full list of tasks that will be available in the command-line interface are as follows.

- Running algorithms on CSV datasets, specifying the output fields and parameters for algorithms;
- Generating synthetic data;
- Producing visual graph representations of datasets, and saving these as images to files.

Requirement 21. *A command-line interface is available that implements the tasks listed above. Its form complies with conventions for command-line programs. Output is given in YAML format where appropriate.*

Web-based

A web-based interface will provide a graphical front end to the system. This is well-suited for casual and non-technical users, as it requires no installation and virtually all users are familiar with using the web.

⁸ <https://yaml.org/>

A task that will greatly benefit from a graphical interface is creating example datasets by hand, as required for the ‘theorist’ use case, and to some extent algorithm development. Recall that the matrix form of input (requirement 2) is most suitable in these cases. A simple approach to inputting such matrices in a graphical context is to display an empty matrix in which users can click the cells to interactively provide values. This is less error prone than manual CSV entry, and arguably more user-friendly.

Viewing the results of algorithms for small datasets will also be possible in the web interface. There is some overlap with the command-line client in this sense, but graphical elements such as colour, font size and text styles can be used in a web page to enhance the output.

Finally, visualising datasets and results through images and animations, as per section 3.1.4, is simple in a web interface. When one is interested in simply viewing such imagery, as opposed to saving to a file for later use, it is much more convenient to have the images displayed immediately alongside the results.

It is worth noting that the web interface has its limitations, and is not suitable for all use cases. For example, large datasets will be difficult to input, and the bespoke formats in which many real-world datasets are stored cannot be used. The interface also cannot be accessed programmatically, and making the site available to the public requires dedicated hosting which may cost real money. Nevertheless, it will complement the API and command-line interfaces and provide value for its intended use cases.

Requirement 22. *A web interface is available that provides a graphical front end to the system. Users can interactively input a dataset in the matrix format described in requirement 2, and select algorithms to run. Results can be viewed, and images and animations as per section 3.1.4 are shown.*

Design

The UML class diagram in figure 3.12 shows the high-level design for the implementation of the user-interfaces discussed above. The API interface is not represented by a single class: the API in fact consists of *all* the classes defined throughout this chapter. The two more concrete interfaces, command-line and web-based, are represented by classes `CommandLineClient` and `WebClient` respectively. Some common functionality is required in

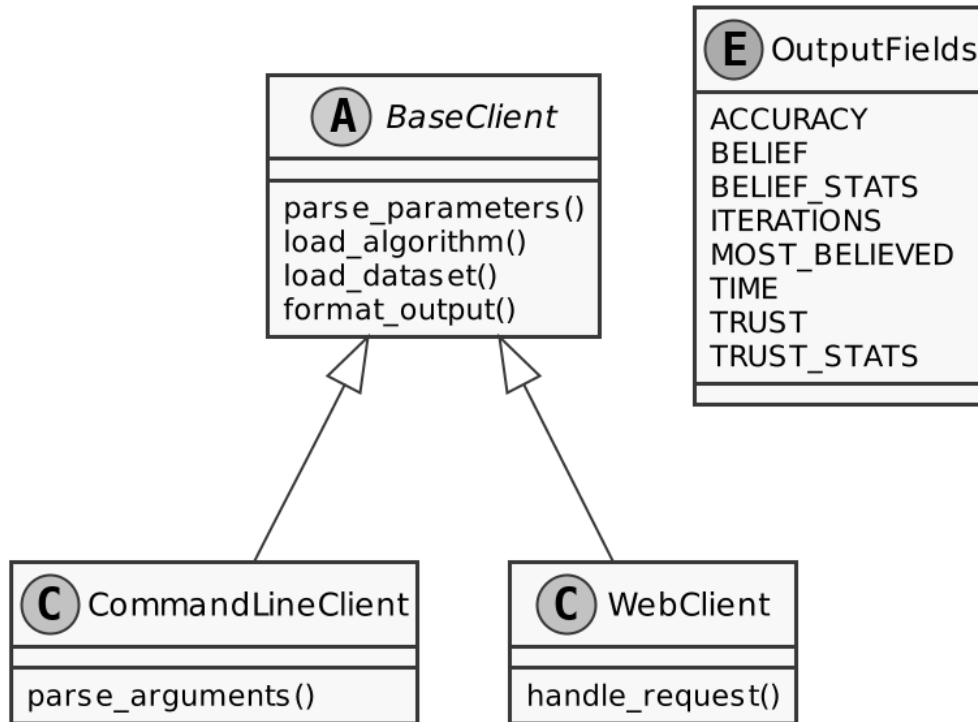


Figure 3.12: UML class diagram showing high-level design for user interfaces.

both interfaces: e.g. parsing algorithm names and parameters and datasets from user-supplied strings to their object representations. Such functions will be implemented in a base class `BaseClient`.

Users will also be able to specify which output fields they wish to receive, as was illustrated in the command-line example in figure 3.10. An enumeration `OutputFields` lists the available fields.

3.2 Implementation

This section discusses the software implementation at a lower level, covering how the requirements of the preceding section were actually implemented in code. Justification for the programming languages and libraries used is given first, followed by detailed descriptions of the algorithms identified in requirement 9.

3.2.1 Programming Languages

Python was chosen as the core programming language for this project – specifically Python 3. Development and testing was performed with version 3.6.7, but it is expected that it will work with versions 3.3 and newer.⁹

There were several reason for choosing Python. Firstly, it is an extremely popular language – according to the TIOBE index it is fourth most popular worldwide, as of April 2019.¹⁰ This is an important consideration, since a major goal for the software is to be extendible to allow for new algorithms to be developed, and for it to integrate nicely with other code. An unpopular or obscure programming language would go against this goal.

Due to its popularity, Python has a rich ecosystem of libraries and packages surrounding it. This allows external libraries to be used during development to provide functionality that would otherwise take too long to implement in the time available. For example, I was able to use the `cairo`¹¹ library to produce graphics and `Flask`¹² to implement a web server – both tasks could have constituted entire projects in their own right if no suitable libraries were available. Such libraries are invariably more comprehensive than a ‘home-grown’ solution would be anyway, since they are dedicated projects aimed specifically at graphics and web frameworks respectively.

Whilst third-party libraries are not unique to Python, they are particularly plentiful in the Python world, which makes it a suitable choice.

Python also has strong object-oriented programming capabilities, which were useful for creating a clean and straightforward API. Code for different portions of the code can be easily compartmentalised into separate classes, making it so that users only need to be aware of the specific classes relevant to their use case.

Finally, Python is the language I personally have the most experience with. Using it for this project allowed me to get started quickly, without

⁹ To the best of my knowledge, the newest language feature used is the `yield` from expression, which was added in version 3.3. See <https://docs.python.org/3/whatsnew/3.3.html#pep-380> for details.

¹⁰ <https://www.tiobe.com/tiobe-index/>.

¹¹ <https://github.com/pygoject/pycairo>

¹² <http://flask.pocoo.org>

having to learn a new language. This meant that more features could be developed in the limited time available.

For the web interface to the software, an interactive web page was required: e.g. for inputting data, selecting which results to view and controlling animations. Such interactive elements require the use of JavaScript. The JavaScript code interacts with the Python server via HTTP.

3.2.2 Third-party Libraries Used

The main purpose of the practical component of this project is to implement truth discovery algorithms. These algorithms often involve lots of numerical computations, which can be represented conveniently in terms of matrix operations on large matrices. The de-facto standard library in the Python world for numerical computing is `numpy`.¹³ Amongst other things, `numpy` has a highly efficient n -dimensional array implementation, written in C, which supports various operations including matrix multiplication (the case $n = 2$ of an n -dimensional array is a matrix). Another library `scipy`¹⁴ – which is often used in conjunction with `numpy` – implements *sparse arrays*, which are optimised to store large arrays with few non-zero entries in a memory efficient way. Operations can also be performed on sparse arrays efficiently without converting to a ‘dense’ format. Sparse arrays were essential in this project for representing large real-world datasets, and `numpy` and `scipy` were used extensively throughout.

For generating graphics, `cairo` was used. The graphs described in requirement 18 only require simple drawing, such as circles, lines, rectangles and basic text rendering. `cairo` was more than perfectly sufficient for this, and provides a simple and straightforward API. For generating animations as per requirement 19, `imageio`¹⁵ was used to combine PNG images produced by `cairo` into an animated GIF.¹⁶

As briefly mentioned already, `Flask` was used to provide the back-end server for the web interface. Testing was performed with the help of `pytest`,¹⁷

¹³<https://www.numpy.org/>

¹⁴<https://www.scipy.org/>

¹⁵<https://imageio.github.io/>

¹⁶ Note that `cairo` and `imageio` are not used for the images and animations shown in the web interface: JavaScript canvas drawing is used instead.

¹⁷ <https://docs.pytest.org/en/latest/>

For the web front-end, I opted to use AngularJS,¹⁸ a JavaScript library designed to simplify the development of web applications and encourage rapid development. In particular, Angular removed the need for large amounts of simple and tedious code in my application, and its simplicity allowed me to implement features that I may not have bothered with otherwise. For the visual aspect of the web interface, I used a CSS framework called `Spectre.css`,¹⁹ which handles visual styling of elements on the page and provides responsive and mobile-friendly layouts.

Sphinx was used for documentation.²⁰ Sphinx compiles reStructuredText sources to HTML, and can automatically parse Python source files to document classes and methods. For this project the documentation includes a user guide and examples of using the API and CLI client, with links to the class API documentation as appropriate.

3.2.3 Truth Discovery Algorithms

In this section we describe in detail the algorithms implemented for this project – namely majority voting, *Sums*, *Average-Log*, *Investment*, *Pooled-Investment* and *TruthFinder* – and how they were implemented in code.

The format of truth discovery datasets was discussed from a user’s perspective in section 3.1.1. In terms of actual implementation, yet another representation is most appropriate: the dataset is represented by two matrices (distinct from the matrix format described in requirement 2), trust and belief scores are stored as *vectors*, and scores are updated at each iteration using matrix operations.

Some notation is required to properly define the algorithms above. Consider a fixed dataset of m sources labelled $1, \dots, m$ and n distinct facts labelled $1, \dots, n$. Note that the labelling is arbitrary. We denote the set of sources claiming fact j by $\text{src}(j)$, the set of facts claimed by source i by $\text{facts}(i)$, and the object relating to fact j by $\text{obj}(j)$.²¹ The set of facts about the same object as j – the facts mutually exclusive with j – is $\text{mut}(j) = \{k : \text{obj}(k) = \text{obj}(j)\}$.

The trust scores for an algorithm at iteration $t \in \mathbb{N}$ will be denoted $T^t \in \mathbb{R}^m$, and the belief scores $B^t \in \mathbb{R}^n$. Subscripts will denote the entries

¹⁸<https://angularjs.org/>

¹⁹<https://picturepan2.github.io/spectre/>

²⁰<http://www.sphinx-doc.org/en/master/>

in these vectors; i.e. T_i^t is the trust score for source i at iteration t .

The following matrices can be used to represent the dataset and perform the trust/belief score updates.

- **Source-claims matrix:** this binary $m \times n$ matrix indicates which claims are made by which sources, and will be denoted here by M . It is defined as:

$$[M]_{ij} = \begin{cases} 1 & \text{if } i \in \text{src}(j) \\ 0 & \text{otherwise} \end{cases}$$

i.e. 1 if source i claims fact j , and 0 otherwise.

- **Mutual exclusion matrix:** this $n \times n$ matrix, denoted X , indicates which facts relate to the same object, i.e. which facts are mutually exclusive:

$$[X]_{kj} = \begin{cases} 1 & \text{if } \text{obj}(k) = \text{obj}(j) \\ 0 & \text{otherwise} \end{cases}$$

Note that X is symmetric.

In practise, both M and X are often extremely sparse, in that most entries are 0. Note that a truth discovery dataset is uniquely determined by M and X , up to the ordering of the sources and facts.

Using this notation, each algorithm listed above is specified by three components:

- the prior belief scores B^0 , which may depend on M and X ;
- the formulae for obtaining T^{t+1} and B^{t+1} from B^t , T^t , M and X ;
- the stopping criterion.

Note that we do not claim *all* iterative truth discovery algorithms are determined by these three factors, merely that the ones implemented in this project are. However, this structure is common across the literature.

Since it was stated in requirement 11 that the user should have full control over the stopping criterion, we do not consider it as part of the algorithm here.

²¹ This notation is similar to that which will be adopted in the theoretical work in section 4.2, but adapted to the set up here.

Prior Belief Scores

It was stated in requirement 10 that the user should be able to specify how the prior belief scores are assigned to facts. We choose the priors defined by Pasternack and Roth in [27] for the available options: *fixed*, *uniform* and *voted*.

- **Fixed:** each fact is assigned a score of 0.5:

$$B^0 = 0.5e_n$$

where $e_n = [1, \dots, 1] \in \mathbb{R}^n$ is the vector consisting of n ones.

- **Uniform:** each object is allocated unit belief, which is distributed evenly amongst objects, i.e. the score for fact j is $\frac{1}{|\text{mut}(j)|}$. Note that $|\text{mut}(j)|$ is the sum of the j -th row of X , which is given by the j -th entry of Xe_n . We may therefore write

$$B^0 = \frac{1}{Xe_n}$$

where the division is performed entry-wise.

- **Voted:** the belief in a fact is proportional to the number of sources claiming it, with the scores scaled such that the total belief across each object is 1:

$$B_j^0 = \frac{|\text{src}(j)|}{\sum_{k \in \text{mut}(j)} |\text{src}(k)|}$$

Note that $|\text{src}(j)|$ is the sum of the j -th column of M , i.e. the j -th entry of $M^T e_m$. Set $v = M^T e_m$. Also note that $k \in \text{mut}(j)$ iff $[X]_{jk} = 1$ and $[X]_{jk} = 0$ otherwise. The denominator above is therefore

$$\sum_{k \in \text{mut}(j)} |\text{src}(k)| = \sum_{k=1}^n x_{jk} v_k = [Xv]_j$$

Performing entry-wise division of vectors, we may write

$$B^0 = \frac{v}{Xv}$$

numpy supports matrix multiplication and entry-wise division as used above, which makes computing prior beliefs extremely simple when expressed in this form.

We can now define the truth discovery algorithms in terms of matrix operations in a similar way.

Sums

Inspired by the *Hubs and Authorities* [16] for ranking web pages based on the hyperlink structure of the web, *Sums* sets the trust score for a source to the sum of the belief scores of its facts, and vice versa:

$$T_i^{t+1} = \sum_{j \in \text{facts}(i)} B_j^t$$

$$B_j^{t+1} = \sum_{i \in \text{src}(j)} T_i^{t+1}$$

Sums has a natural matrix representation, which is also given in the original *Hubs and Authorities* paper. Note that $j \in \text{facts}(i)$ iff $i \in \text{src}(j)$ iff $[M]_{ij} = 1$, and $[M]_{ij} = 0$ otherwise. Hence

$$T_i^{t+1} = \sum_{j=1}^n [M]_{ij} B_j^t = [MB^t]_i$$

and similarly

$$B_j^{t+1} = [M^T T^{t+1}]_j$$

so the trust and belief updates are simply

$$T^{t+1} = MB^t$$

$$B^{t+1} = M^T T^{t+1}$$

To prevent the trust and belief scores growing without bound, T^{t+1} and B^{t+1} are normalised by dividing by $\max_i T_i^{t+1}$ and $\max_j B_j^{t+1}$ respectively after the above operations.

AverageLog

Sums allows sources to inflate their trust score by simply making many claims, which is potentially undesirable. The number of claims should not be ignored, however: a source with 90% accuracy over a hundred claims is surely more trustworthy than one with 90% accuracy over 10 [27]. *Average-Log* attempts a compromise by setting the trust score for a source to the average belief score of its claims, multiplied by the logarithm of the number of facts it claims.

$$T_i^{t+1} = \log(|\mathbf{facts}(i)|) \cdot \frac{\sum_{j \in \mathbf{facts}(i)} B_j^t}{|\mathbf{facts}(i)|}$$

Taking a logarithm ensures that making many trivial claims has diminishing returns for sources. Belief score update is the same as for *Sums*.

Observe that $|\mathbf{facts}(i)|$ is the sum of the i -th row of M , which is the i -th entry of Me_n . It was shown for *Sums* that the sum of the belief scores is the i -th entry of MB^t . Set

$$w = \frac{\log(Me_n)}{Me_n}$$

where the log and division are taken entry-wise. Then

$$T^{t+1} = w \circ MB^t$$

where \circ denotes entry-wise multiplication of vectors (also called the *Hadamard* product).

As with *Sums*, normalisation is performed to prevent numerical overflow.

Investment

In *Investment*, sources invest their trust among their claims, belief scores are grown non-linearly, and sources receive trust returns proportional to the their investment in the next iteration (relative to other sources). More detail on the intuition behind the algorithm is given in [27]. The trust and

belief updates are as follows.

$$T_i^{t+1} = \sum_{j \in \text{facts}(i)} B_j^t \cdot \frac{T_i^t}{|\text{facts}(i)| \cdot \sum_{r \in \text{src}(j)} \frac{T_r^t}{|\text{facts}(r)|}}$$

$$B_j^{t+1} = \mathcal{G} \left(\sum_{i \in \text{src}(j)} \frac{T_i^{t+1}}{|\text{facts}(i)|} \right)$$

where $\mathcal{G}(x) = x^g$. The parameter g is set to 1.2 in [27]. In our implementation the g value can be specified by the user, but defaults to 1.2.

Each source invests their trust score evenly amongst its facts: the investment amount is $\frac{T_i^t}{|\text{facts}(i)|}$ for source i . Let S^t be the vector of investment amounts at iteration t , i.e.

$$S^t = \frac{T^t}{Me_n}$$

Then we have

$$\begin{aligned} T_i^{t+1} &= \sum_{j=1}^n [M]_{ij} \cdot B_j^t \cdot \frac{S_i^t}{\sum_{r=1}^m [M]_{rj} \cdot S_r^t} \\ &= S_i^t \cdot \sum_{j=1}^n [M]_{ij} \cdot B_j^t \cdot \frac{1}{[M^T S^t]_j} \\ &= S_i^t \cdot \sum_{j=1}^n \frac{[M]_{ij}}{[M^T S^t]_j} \cdot B_j^t \end{aligned}$$

Define an $m \times n$ matrix N by $[N]_{ij} = \frac{[M]_{ij}}{[M^T S^t]_j}$; then the sum is the i -th entry of NB^t , so

$$T_{t+1} = S^t \circ NB^t$$

For belief update, we have

$$\begin{aligned} B_j^{t+1} &= \mathcal{G} \left(\sum_{i=1}^m [M]_{ij} \cdot [S^{t+1}]_i \right) \\ &= \mathcal{G} ([M^T S^{t+1}]_j) \end{aligned}$$

so B^{t+1} is obtained by applying \mathcal{G} to each entry in $M^T S^{t+1}$.

Again, normalisation is performed by dividing by the maximum trust and belief scores.

PooledInvestment

PooledInvestment uses the same trust update as *Investment*, but belief scores are linearly scaled after applying \mathcal{G} so that the total across each object is preserved. We defer the detailed interpretation to [27]. The belief update is as follows: with $H^{t+1} \in \mathbb{R}^n$ defined by $H_j^{t+1} = \sum_{i \in \text{src}(j)} \frac{T_i^{t+1}}{|\text{facts}(i)|}$:

$$B_j^{t+1} = H_j^{t+1} \cdot \frac{\mathcal{G}(H_j^{t+1})}{\sum_{k \in \text{mut}(j)} \mathcal{G}(H_k^{t+1})}$$

Note that H_j^{t+1} is the total amount ‘invested’ in fact j before applying \mathcal{G} in *Investment*, so we have $H^{t+1} = M^T S^{t+1}$ with S^{t+1} defined as above.

Write $\tilde{H}^{t+1} = \mathcal{G}(H^{t+1})$, where \mathcal{G} is applied entry-wise. Then the sum in the denominator is

$$\sum_{k \in \text{mut}(j)} \mathcal{G}(H_k^{t+1}) = \sum_{k=1}^n [X]_{jk} \tilde{H}_k^{t+1} = [X \tilde{H}^{t+1}]_j$$

Hence we have

$$B^{t+1} = H^{t+1} \circ \frac{\tilde{H}^{t+1}}{X \tilde{H}^{t+1}}$$

Normalisation is performed after the trust and belief updates.

TruthFinder

TruthFinder, by Yin et. al. [35], was one of the first truth discovery algorithms to be introduced. In contrast to the algorithms above, belief scores in *TruthFinder* have pseudo-probabilistic interpretation: B_j^t is (an estimate for) the probability that fact j is correct. *TruthFinder* also uses prior *trust* scores instead of prior belief scores; trust for each source is set to a fixed initial value t_0 , which can be specified by the user in our implementation but defaults to 0.9.

TruthFinder also considers *implications between claims* for cases where confidence in one fact should increase (or decrease) the confidence in another. For each pair of facts f_1, f_2 relating to a common object, an implication value $\text{imp}(f_1 \rightarrow f_2) \in [-1, 1]$ describes the level of implication: 1 for strong positive implication, -1 for strong negative implication, and 0 for no implication. The specific method of assigning implication values is domain-specific.

An example from [35] is as follows: suppose a fact f_1 states that the author of a particular book is ‘Jennifer Widom’, and a second fact f_2 gives the authors of the same book as ‘Jennifer Widom and Stefano Ceri’. If we have high confidence in f_2 , then f_1 is incomplete and thus should receive low confidence: the implication $imp(f_2 \rightarrow f_1)$ should be low. On the other hand, it is common for sources to list only the first author of a book, even when multiple authors exist. If we are confident about f_1 , then we should also be confident about f_2 , because f_2 is consistent with f_1 . Thus $imp(f_1 \rightarrow f_2)$ should be high. This example illustrates that the implication values are asymmetric.

As for the actual definition of *TruthFinder*, it is already given in terms of matrix operations in the original paper, using an $m \times n$ matrix U and $n \times m$ matrix V defined as follows.²²

$$[U]_{ij} = \begin{cases} 1/|\mathbf{facts}(i)| & \text{if } j \in \mathbf{facts}(i) \\ 0 & \text{otherwise} \end{cases}$$

$$[V]_{ji} = \begin{cases} 1 & \text{if } j \in \mathbf{facts}(i) \\ \rho \cdot imp(f_k \rightarrow f_j) & \text{if } k \in \mathbf{facts}(i) \text{ for some } k \in \mathbf{mut}(j) \\ 0 & \text{otherwise} \end{cases}$$

where $\rho \in [0, 1]$ is a parameter controlling the influence of implications between facts. It defaults to 0.5 in our implementation.

The definition of trust and belief updates make use of additional vectors $\tau \in \mathbb{R}^m$ and $\sigma \in \mathbb{R}^n$; these represent trust and belief scores scaled from $[0, 1]$ to $[0, \infty)$ by taking a logarithm. The precise definitions are as follows (as usual, logarithms, scalar addition etc. for vectors are taken entry-wise).

$$\begin{aligned} \tau^{t+1} &= -\log(1 - T^t) \\ \sigma^{t+1} &= V\tau^{t+1} \\ B^{t+1} &= \frac{1}{1 + \exp(-\gamma \cdot \sigma^{t+1})} \\ T^{t+1} &= UB^{t+1} \end{aligned}$$

$\gamma \in (0, 1)$ is a parameter called the *dampening factor*; details can be found in the original paper. It defaults to 0.3 in our implementation.

²² U and V are called A and B in the original paper.

3.3 Results and Evaluation

Having described the requirements for the truth discovery software framework in section 3.1 and aspects of its implementation in section 3.2, we come to demonstrating exactly what was implemented and evaluating the system against the requirements. The demonstrations will cover each of the use cases identified, including real-world truth discovery applications and evaluation of algorithms.

3.3.1 Real-World Dataset Demonstration

The main use case for the system is to run truth discovery algorithms on real-world datasets. The *stock dataset* briefly described in section 3.1.1 provides a suitable example of ‘large’ real-world data to test with: it contains 2,843,803 claims covering 336,000 variables from 55 sources.²³

The claims relate to 1,000 distinct stocks. For 100 of these, data was manually obtained from NASDAQ by the original authors, and these values are taken to be ground truths. This means we can not only run algorithms on the large dataset, but also load *supervised data* and assess accuracy with respect to the NASDAQ ground truths.

Additionally, the data is in a TSV (tab separated values) format, which allows us to demonstrate the base classes `FileDataset` and `FileSupervisedData` for loading data from custom formats.

The Python classes are shown in figure 3.13. Note that no code is given to actually open the files and construct the matrices required for running algorithms: this is handled automatically by the `FileDataset` and `FileSupervisedData` base classes.

A small script was created to run each algorithm on the dataset with, initialised with its default parameters. The results are shown in figure 3.14.

From these results we see that *TruthFinder* is by far the quickest (excluding majority voting), but achieves poor accuracy. It should be noted that *TruthFinder* was run without considering implications between claims, which may explain this. We also see that *Sums* and *PooledInvestment* are the only algorithms that perform better than naïve majority voting.

²³ The dataset was collected by the authors in [20], and is available to download at <http://lunadong.com/fusionDataSets.htm>.

```
1 class StockBase:
2     """
3     Base class to provide functions required for parsing both data and truth
4     files
5     """
6     def get_tsv_reader(self, fileobj):
7         return csv.reader(fileobj, delimiter="\t")
8
9     def get_variable_name(self, symbol, date, attr_index):
10        return "{}-{}-attr-{}".format(symbol, date, attr_index)
11
12    def clean_value(self, value):
13        bad_chars = ("$", "%", "+", "-")
14        for char in bad_chars:
15            value = value.replace(char, "")
16        return value.strip()
17
18    def get_var_vals(self, row, date):
19        """
20        Yield tuples of the form (var, val) for each value in the given row.
21        Note that ``row`` should not include the source or date.
22        """
23        symbol, *attributes = row
24        for i, attr in enumerate(attributes):
25            var = self.get_variable_name(symbol, date, i)
26            val = self.clean_value(attr)
27            if val:
28                yield (var, val)
29
30
31 class StockDataset(FileDataset, StockBase):
32     def get_tuples(self, fileobj):
33         for row in self.get_tsv_reader(fileobj):
34             date, source, *rest = row
35             for var, val in self.get_var_vals(rest, date):
36                 yield (source, var, val)
37
38
39 class SupervisedStockData(FileSupervisedData, StockBase):
40     def get_pairs(self, fileobj):
41         for row in self.get_tsv_reader(fileobj):
42             date, *rest = row
43             yield from self.get_var_vals(rest, date)
```

Figure 3.13: Python code for loading the stock dataset.

```
loading data...
  loaded in 208.617 seconds
loading true values...
  loaded in 0.119 seconds

dataset has 55 sources, 2843803 claims, 336000 variables
running MajorityVoting...
  0.280 seconds, 0.641 accuracy
running Sums...
  11.998 seconds, 0.646 accuracy
running AverageLog...
  12.172 seconds, 0.641 accuracy
running Investment...
  31.216 seconds, 0.423 accuracy
running PooledInvestment...
  18.545 seconds, 0.671 accuracy
running TruthFinder...
  1.598 seconds, 0.439 accuracy
```

Figure 3.14: Results for the stock dataset for each algorithm.

3.3.2 Synthetic Data Accuracy Experiments

Another key use case for the system is evaluation of algorithms with respect to their accuracy on synthetic datasets. Unlike real-world datasets, synthetic datasets are easy to obtain, allow accuracy to be calculated, and their parameters can be precisely controlled. One can also generate multiple datasets with varying parameters to study the effects of changes in certain parameters.

Recall the parameters available in this project: the source reliability scores (interpreted as probabilities), the number of variables to generate, the probability that a source will make a claim for a given variable (referred to as the *claim probability*), and the domain size for each generated variable. We will explore this parameter space in a number of directions to demonstrate the types of analysis that can be performed with the system.

The first experiment studies the effects of the distribution of source reliability scores. We consider three distributions:

- **Mostly bad:** a third of sources have reliability score 0.75, and the

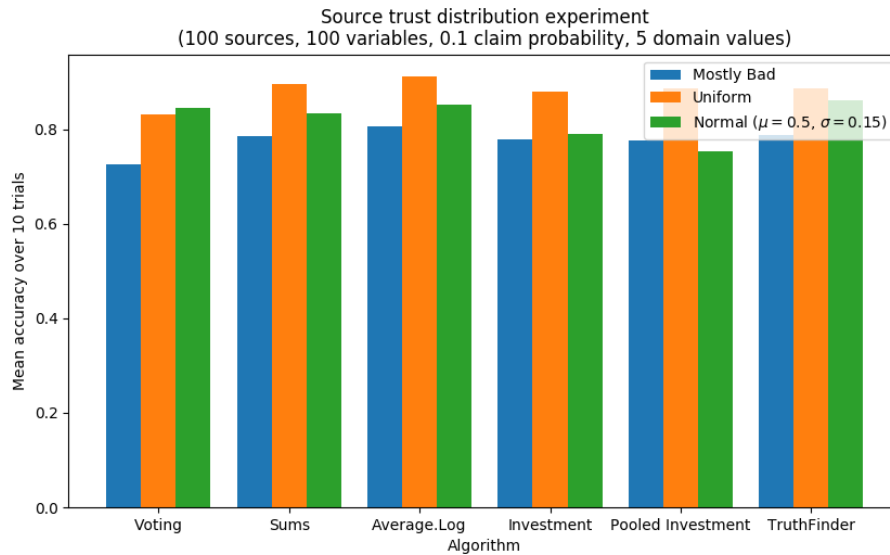


Figure 3.15: Source trust distribution experiment on synthetic datasets.

remaining two-thirds have score 0.25, i.e. most sources are correct only 25% of the time.

- **Uniform:** reliability scores are drawn from a uniform distribution on $[0, 1]$.
- **Normal:** reliability scores are drawn from a normal distribution with mean 0.5 and standard deviation 0.15. It is possible for scores to be less than 0 or greater than 1: they are clipped to 0 or 1 in such cases.

For each of these distributions, source reliability scores were generated and a synthetic dataset produced with 100 sources, 100 variables, 0.1 claim probability and 5 domain values. Each algorithm was then run (on the same dataset), and its accuracy computed. This process was repeated 10 times for each distribution in an attempt to cancel out random effects of the reliability score and claim generation.

The mean accuracy scores were then taken as the final results, which are shown in figure 3.15.

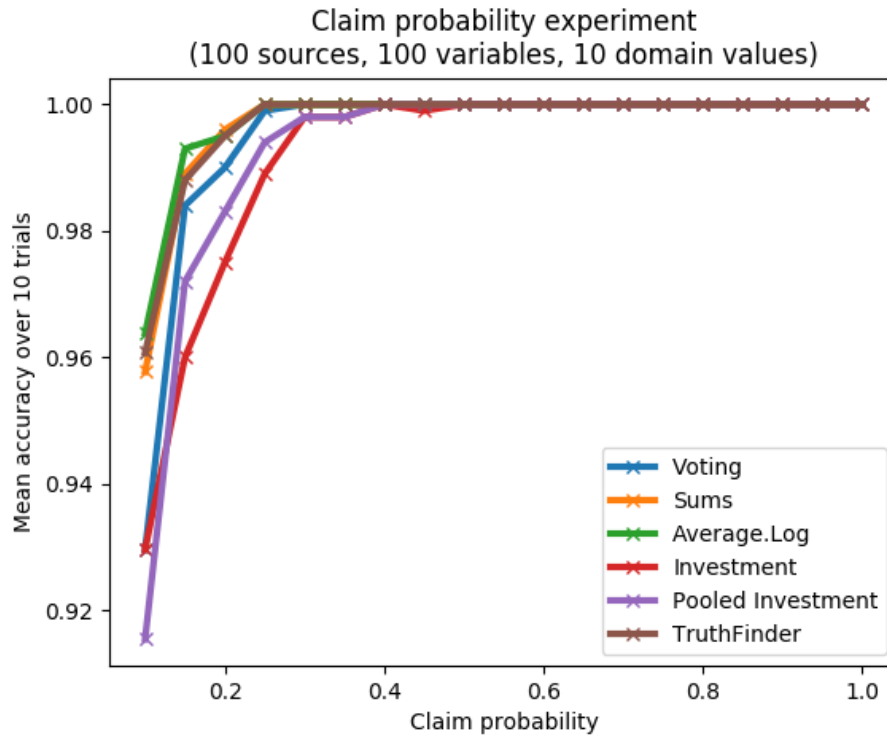


Figure 3.16: Claim probability experiment on synthetic datasets.

As one might expect, the ‘Mostly bad’ distribution leads to the poorest accuracy for all algorithms, with the exception of *PooledInvestment* where ‘Normal’ gave slightly worse accuracy.

For the next experiment, the effects of the claim probability p_c on accuracy were investigated. Recall that for each artificial source and variable, a claim is made with probability p_c . In the extreme case $p_c = 1$, every source claims a value for every variable. The dataset becomes more ‘sparse’ as p_c decreases.

In the experiment, claim probability was increased from 0.1 to 1 in increments of 0.05. For each value, 10 synthetic datasets were generated with a uniform trust distribution over 100 sources, 100 variables, and 10 domain values. The mean accuracy of each algorithm across the 10 datasets is shown in figure 3.16.

The results are again unsurprising. For low claim probabilities, the

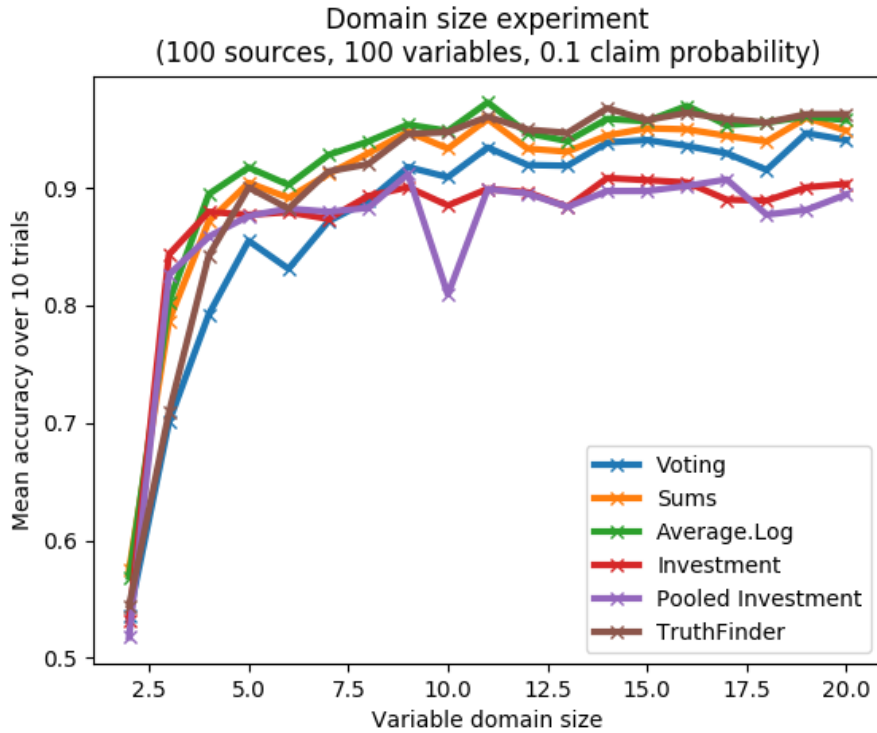


Figure 3.17: Domain size experiment on synthetic datasets.

expected number of claims for each variable is small. This means it is less likely for multiple sources to agree on the value for a variable – but this is one of the main ways in which truth discovery algorithms determine the true values. As a result we see accuracy rise dramatically as the claim probability increases. All algorithms achieve the maximal accuracy score of 1 for p_c greater than around 0.5.

The final experiment concerns the domain size, which is the number of values the artificial sources choose from for their claims – in this project the domain of possible values is the same for each artificial variable. The minimum domain size is 2; values from 2 to 20 were tested for this demonstration. Other parameters were kept at the same values as in the claim probability experiment, with claim probability itself set to 0.1. Mean accuracy scores for each variable are shown in figure 3.17.

The shape of the graph is similar to the claim probability experiment;

that is, accuracy increases sharply as domain size increases. Note that the increase is more dramatic here: mean accuracy increases from around 0.55 to near 1, whereas in the claim probability experiment the minimal accuracy was already fairly high at around 0.9.

Recall that in the model of synthetic data adopted in this project, a source s will choose the correct value with probability p_s – their reliability score – when making a claim for a variable, and choose one of the $d - 1$ incorrect values each with probability $\frac{1-p_s}{d-1}$.

Since each incorrect value is chosen with equal probability, the probability of choosing any particular value decreases as the domain size d is increased. This means that for larger d , it is less likely that sources will agree on incorrect values. Stated another way, when d is large agreements between sources are likely to correspond to the true values, as opposed to multiple sources making the same mistake. Agreements between sources is a key indicator for true values, at least as far as the algorithms considered in this project are concerned, and thus we see higher accuracy as domain size increases.

3.3.3 Convergence Analysis

Continuing with the evaluation of truth discovery algorithms using the developed software, we analyse the convergence behaviour of each algorithm. To do this, a synthetic dataset consisting of 1000 sources and 1000 variables was created. Each iterative algorithm was then run for 100 iterations, and the distance between the trust score vectors in successive iterations measured in the ℓ_2 norm, i.e. we measure $\|T^t - T^{t-1}\|_2$ for each time $t \in \{2, \dots, 100\}$.²⁴ If the trust scores converge – in the sense of a limit in a metric space as discussed earlier – we should see this distance become arbitrarily small, and remain so as $t \rightarrow \infty$.

Figure 3.18 shows the results. Many algorithms, particularly *Sums*, *Average-Log* and *TruthFinder*, appeared to converge exponentially quickly. A logarithmic scale is therefore used on the vertical axis to show more clearly the convergence behaviour.

Unfortunately, it was only possible to run *TruthFinder* for 3 iterations. After this the trust score T_i^3 for one of the sources i became sufficiently

²⁴ Distance measures other than ℓ_2 – namely L_1 , L_∞ and cosine distance – were also experimented with. The graphs for each looked almost identical to the L_1 graph, so they are not included here.

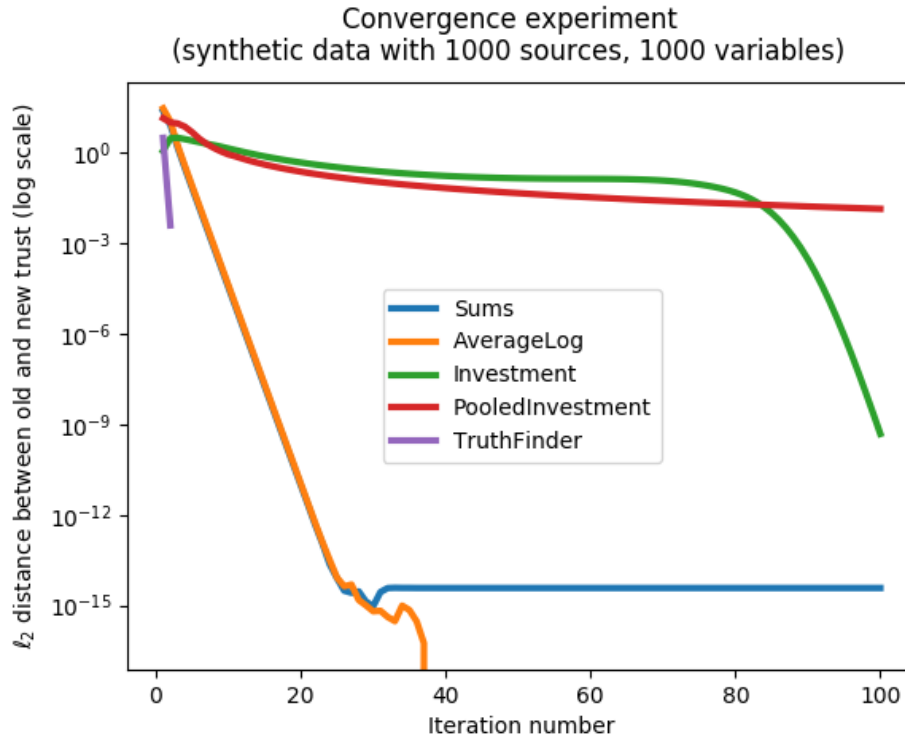


Figure 3.18: Convergence experiment on a large synthetic dataset.

close to 1 that $1 - T_i^3$ becomes 0 due to a rounding error; we then get an undefined result when computing $\log(1 - T_i^3)$ in the next iteration as per the *TruthFinder* algorithm. More work needs to be done to determine whether this is an implementation issue or a limitation of *TruthFinder* itself. However, in only three iterations the distance between successive trust scores becomes close to 10^{-3} , which is small enough that one may consider it to have converged reasonably well.

Another interesting point is the distances for *Sums* and *Average-Log*. They are almost identical up to around 25 iterations, where they start to diverge. *Average-Log* then stabilises so that the trust scores remain constant, and the distance is therefore 0 for the remainder.

In contrast, the distances for *Sums* remain more or less constant after 30 iterations, but are not 0. Upon closer inspection, the trust scores repeatedly oscillate between two (very close together) vectors. This could

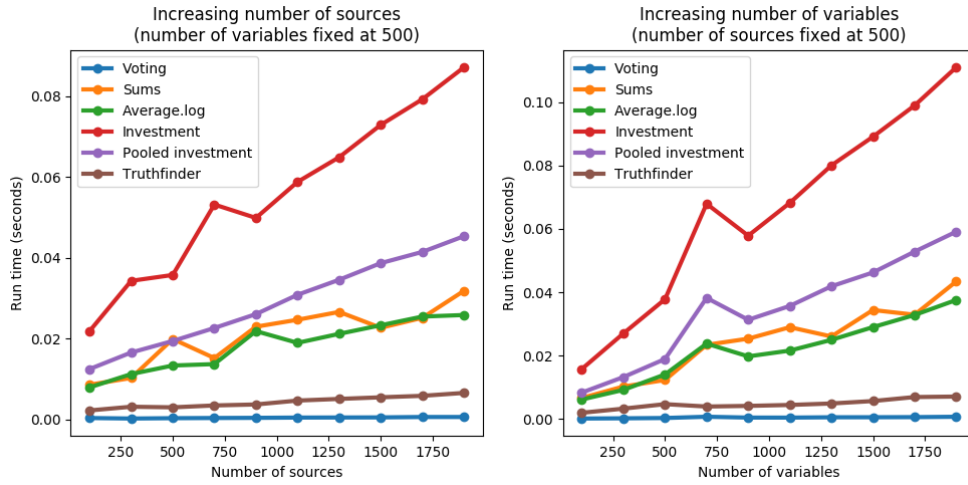


Figure 3.19: Algorithm run time experiments on synthetic datasets.

be caused by numerical instability in the algorithm, or its implementation in this project. In practical terms this is not problematic, since the distance between these two vectors in the ℓ_2 norm is as small as 10^{-14} .

Finally, observe that *Investment* and its friend *PooledInvestment* converge much slower than the other algorithms. The reason for the sharp drop off in distance for *Investment* at around 80 iterations is not clear. Running beyond 100 iterations showed this behaviour continuing; unlike *Sums*, the distance appears to decrease consistently towards 0.

3.3.4 Time Complexity Analysis

To conclude the analysis of truth discovery algorithms, we consider algorithm run time as the size of the input dataset varies. Synthetic datasets were again used, due to the ease of creating datasets with fixed sizes. ‘Size’ has at least three components here: the number of sources, the number of variables, and the number of claims.

The effect of the number of claims on *accuracy* was shown above. Its effect on run time was observed to be marginal, so it is not considered in detail.

Instead, this test investigates the effects of varying the number of sources and variables. In principle an algorithm may scale differently

with respect to these parameters, so they are changed independently in two separate tests.

The methodology was as follows. First, a large synthetic dataset with 2,000 sources, 2,000 variables and uniform trust distribution was created. For each parameter (number of sources and number of variables), sizes from 100 to 2,000 in increments of 200 were tested, whilst the other parameter was fixed at 500. The large $2,000 \times 2,000$ dataset was then reduced to the correct size by taking the first n sources and first m variables in each case. Figure 3.19 shows the results.

Subsetting a large dataset for each trial, as opposed to generating a new dataset, is hoped to reduce any random effects on timing that may arise due to the random nature of synthetic datasets. It also allowed larger sizes to be tested in a reasonable time, since generating many random claims is time consuming.

In terms of the results, we see that most algorithms exhibit linear growth with respect to both the number of sources and variables. The exception is *Voting*, which is not an iterative algorithm and thus does not depend in a major way on the size of the dataset. One might expect the matrix operations involved to take longer for larger matrices, but this effect is negligible compared to other algorithms.

Note that *TruthFinder* run time is particularly quick, and grows very slowly. This may be due to it running for only a few iterations due to the numerical problems mentioned above.

3.3.5 User Interfaces

As set out in section 3.1.5, three user interfaces were developed for the system: a documented Python API, a command-line interface, and a web-based interface.

Python API

A simple example of API usage is shown in figure 3.20. This example creates a synthetic dataset, runs an algorithm with particular parameters, and inspects the results. Output of this script is shown in figure 3.21.

Note that the API is the most comprehensive interface, in the sense that all the implemented functionality is available and can be used in any combination. For the CLI and web interfaces, some functionality has to be

```
1 import numpy as np
2
3 from truthdiscovery import *
4
5 # Create a synthetic dataset
6 synth = SyntheticData(
7     trust=np.random.uniform(size=(10,)),
8     num_variables=7,
9     claim_probability=0.44,
10    domain_size=9
11 )
12
13 # Use Investment with custom parameters, and iterate till convergence
14 alg = Investment(
15     g=1.4,
16     priors=PriorBelief.FIXED,
17     iterator=ConvergenceIterator(DistanceMeasures.L_INF, 0.00123)
18 )
19
20 # Get results
21 results = alg.run(synth.data)
22
23 print("Got results in {:.3f} seconds, {} iterations"
24       .format(results.time_taken, results.iterations))
25
26 best_source = max(results.trust, key=lambda s: results.trust[s])
27 print("Most trustworthy source is {}".format(best_source))
28
29 values = results.get_most_believed_values(4)
30 print("Most probable value(s) for variable 4 are: {}".format(list(values)))
31
32 csv_dataset = synth.to_csv()
33 print("Dataset was:")
34 print(csv_dataset)
35 with open("/tmp/synth_data.csv", "w") as csvfile:
36     csvfile.write(csv_dataset)
```

Figure 3.20: Python code demonstrating simple use of the API.

```

Got results in 0.012 seconds, 23 iterations
Most trustworthy source is 5
Most probable value(s) for variable 4 are: [3.0]
Dataset was:
0.0,0.0,0.0,7.0,3.0,7.0,5.0
3.0,0.0,,,,2.0
0.0,0.0,0.0,,,5.0
,2.0,7.0,,,0.0
,,7.0,3.0,,
1.0,1.0,5.0,,4.0,,
0.0,,,,3.0,,
,,,2.0,2.0,
,,,5.0,,6.0
,,,4.0,
0.0,6.0,,3.0,,0.0

```

Figure 3.21: Example output of the code in figure 3.20.

left out in the interest of presenting a simple and fit for purpose interface. For example, it is not possible in the CLI or web interfaces to instantiate multiple algorithms with separate parameters; this is a niche requirement and allowing for it would complicate parameter specification for the most common use cases. However, this is trivial to achieve when using the API.

Further examples of API usage include the code for each of the algorithm analysis experiments discussed throughout this section. Each experiment uses the ‘external’ API interface without requiring modifications to the core code; this emulates the level of access an end user would have if using the API interface. These examples can be found in the `examples` directory in the source code repository.

Command-Line Interface

In the command-line interface, users can run algorithms on datasets in CSV format, generate synthetic data, and create graph representations of datasets. Figure 3.22 shows an example of running an algorithm; this uses the same dataset, algorithm and parameters as the code of figure 3.20. One can confirm that the results coincide with those shown in figure 3.21.

```

$ truthdiscovery run -a investment -f <(tail -n+2 /tmp/synth_data.csv) -p g=1.4
priors=fixed iterator=l_inf-convergence-0.00123 -o time iterations trust belief
--variables 4
investment:
belief:
  4:
    2.0: 5.36634601026388e-218
    3.0: 1.0
    4.0: 5.36634601026388e-218
    5.0: 5.36634601026388e-218
iterations: 23
time: 0.011137723922729492
trust:
  0: 9.557208353611696e-27
  1: 3.543582884052153e-08
  2: 5.34920635294263e-17
  3: 0.0010965736074619982
  4: 1.2839976861703537e-155
  5: 1.0
  6: 6.419988430851768e-156
  7: 6.419988430851768e-156
  8: 3.209994215425884e-156
  9: 0.000668287527796295
$ █

```

Figure 3.22: Example of CLI interface for running an algorithm.

Note that the `tail`²⁵ command is used to remove the first line of the synthetic data CSV, which contains the ‘true’ values and is thus not part of the data itself. This demonstrates a strength of the command-line interface: it can easily be used in conjunction with other programs using standard shell features such as input and output redirection and variable substitution.

Not all functionality of the command-line interface can be demonstrated here. A full description of the available features and options is given in the help output, which can be shown with `truthdiscovery --help` and `truthdiscovery <cmd> --help` for each of the available sub-commands, namely `run`, `synth` and `graph`.

²⁵ <http://man7.org/linux/man-pages/man1/tail.1.html>

Web Interface

The finished web interface is shown in figures [A.1](#) to [A.7](#) in appendix [A](#). These screenshots are included in a separate appendix due to their size and number.

Figure [A.1](#) shows the basic view presented when a user first loads the page. At the top, the user selects one or more algorithms from the list. By default a small dataset is already loaded. Datasets are represented in the the matrix form described in requirement [2](#).

Users may edit the cells by simply clicking them, and buttons are available to add and remove sources and variables. The ‘trash’ icon in the lower right allows all entries to be cleared.

This interactive matrix format has proved to be successful for manual entry of small datasets. For example, I used it extensively when analysing the behaviour of algorithms for the theoretical work of chapter [4](#). It is not suitable for datasets of any significant size, however; it is particularly limited by the number of variables (displayed horizontally) that can comfortably be shown at one time. It also becomes fiddly to enter many entries when one has to click each individual cell. Future work could improve this by implementing keyboard shortcuts to navigate the matrix.

It is also possible to load datasets from CSV format, as shown in figure [A.2](#). The reverse conversion – exporting the constructed matrix to CSV – was not implemented; this is a clear improvement that can be made in future work.

As a final point on data entry, note that there is a dropdown menu for ‘preset’ datasets. I created four small datasets exhibiting different properties: a ‘typical’ dataset with mixture of agreements and disagreements, one where all sources are in agreement bar one, one consisting of two disjoint sets of sources whose claims do not overlap, and one where no sources agree on any variable. This allows users to quickly get started with the site without having to construct their own dataset.

Another aspect of user input is algorithm parameters. Parameter options are hidden behind an ‘advanced options’ checkbox, since it is expected that most users will use the default parameters. Figure [A.3](#) shows the options displayed when this box is checked. Algorithm parameters are specified by a free-text field using the same format as for the command-line interface. This is a weakness of the interface; a graphical view would be more appropriate here.

Once the algorithms, dataset and parameters are chosen, the user runs the algorithms by clicking the ‘Run’ button. An example of how results are displayed is shown in figure A.4.

First, the run-time and number of iterations are displayed. Beneath this there are four sections: source trust scores, claim belief scores, graph representation and animation. Each section can be collapsed and expanded by clicking its title.

The trust and belief score sections are straightforward: they are displayed in a tabular format, with the maximum scores displayed in bold. Note that the rows can be sorted by score (ascending or descending) or source/variable ID by clicking the table headings.

Note that each score has an additional number in brackets beside it. This shows the *change* in trust/belief score compared to the previous dataset. Displaying these changes is optional, and can be controlled by the checkbox labelled ‘Compare against previous results’ which can be seen in figure A.1. This was particularly useful when experimenting with algorithms on hand-crafted datasets to see how they react to small changes in the dataset, and also to see how different algorithms compare for the same dataset. Note that comparison is not available when running multiple algorithms simultaneously, since it is not clear *which* results are being compared against.

The results of running multiple algorithms is shown in figure A.5. Observe that a tabbed interface is used, so that only one set of results can be seen at a time. This was useful for comparing results between algorithms – this was a major use case identified in section 3.1. One may collapse the sections of results that are not of interest and quickly change between the tabs to see any changes.

As for the graph and animation sections in the results, examples are shown in figures A.6 and A.7 respectively. A slider and buttons beneath the animation frame allows the user to control playback; arrow keys can also be used. Future work could implement automatic playback of animations – presently the user must control the slider themselves to see a smooth animation.

3.3.6 Testing

Testing was an important consideration throughout the practical component of this project. Several use cases involve users extending or otherwise

interacting with the code, which becomes extremely difficult if the code is buggy or unreliable. Thorough testing is therefore required to ensure, as far as possible, that the code behaves correctly.

The most basic form of testing involved manually checking the behaviour of the code during development. For example, after implementing running algorithms until convergence, I experimented with the threshold level and checked that the number of iterations performed changed accordingly.

Clearly this kind of ad-hoc testing is not sufficient. It is impractical to perform such tests for all areas of the software after *every* code change; this means it is easy for bugs to creep in when making future changes.

Moreover, manual inspection is not always enough to spot errors in output. For the truth discovery algorithms implemented here, it is surely impossible for one to know the expected results of an algorithm before running it through software. In this case looking at the results by eye will most likely not provide any useful information regarding the correctness of the implementation. Even if one could predict the results of an algorithm, the trust and belief scores are often floating point numbers with an excess of 10 digits after the decimal point: checking these numbers manually would be extremely challenging and error-prone.

Automated testing was used heavily to address these issues. This allowed many tests covering broad parts of the code to be run quickly without manual intervention, and allowed the tests to be far more comprehensive than manual testing could be. For these reasons and others, automated tests are now standard practise in software development.

I intended to adopt a *test-driven development* workflow, where tests are developed *before* real code. This is hoped to force one to write tests that cover the actual functionality required, rather than testing the specific implementation. I achieved this semi-successfully, often writing a basic test first and spotting gaps in the test coverage after the code implementation.

At any rate, the project benefits from a comprehensive suite of 150 unit tests covering all aspects of the Python code. Test coverage, which is the percentage of lines of code that are run during the execution of the tests, is 100% as measured using the `coverage` library²⁶ (after manually excluding lines that ought *not* to run during tests, such as starting the web server, reading command-line arguments etc.). Note that 100% test coverage does not necessarily mean that the tests are sufficient to find all


```

1 class TestDataset:
2     @pytest.fixture
3     def data(self):
4         triples = (
5             ("john", "wind", "very windy"),
6             ("paul", "wind", "not very windy"),
7             ("george", "wind", "very windy"),
8             ("ringo", "wind", "not very windy at all"),
9
10            ("john", "rain", "dry"),
11            ("george", "rain", "wet"),
12
13            ("john", "water", "wet"), # re-use value
14            ("paul", "water", "drink"),
15            ("george", "water", "drink"),
16
17            # Mix up the order of variables
18            ("ringo", "rain", "dry"),
19        )
20        return Dataset(triples)
21
22    def test_claims_matrix(self, data):
23        expected_claim_mat = np.array([
24            [1, 0, 0, 1, 0, 1, 0],
25            [0, 1, 0, 0, 0, 0, 1],
26            [1, 0, 0, 0, 1, 0, 1],
27            [0, 0, 1, 1, 0, 0, 0],
28        ])
29        assert data.sc.shape == expected_claim_mat.shape
30        assert np.array_equal(data.sc.toarray(), expected_claim_mat)

```

Figure 3.23: Example of a unit test for constructing the source-claims matrix for a dataset.

potential bugs in the code, but it is nonetheless a desirable statistic.

As mentioned, `pytest` was used to implement and run the tests. The complete test output is shown in [B](#).

Most of the automated tests can be described as *unit tests*, which check the behaviour of a small part of the code (a ‘unit’) in isolation. In my case this usually involved writing multiple tests for each class and method. An example is shown in [figure 3.23](#). This test checks that the source-claims matrix M is constructed correctly when a dataset is loaded.

A pattern I used throughout development was to first write tests for the expected behaviour of a class with ‘correct’ inputs, then consider ‘edge cases’, and finally consider invalid inputs to check that error checking is performed as appropriate.

²⁶ <https://coverage.readthedocs.io/en/v4.5.x/>

The example in figure 3.23 illustrates this to some extent. The first six claims follow a predictable format: they deal with one variable at a time ('wind' for the first four, 'rain' for the next two), with a mixture of repeated values and distinct ones. These constitute the obviously 'correct' parts of the input. Edge cases are thrown in for the remaining claims. First, the value 'wet' is repeated, but for a different variable; this is perfectly valid input, but a naïve implementation could incorrectly fail to distinguish between the claims in this case. The variables are also introduced out of order with the final claim. Again, this is an attempt to 'trick' the code in order to catch errors that may occur if one does not consider these edge cases.

Erroneous input is handled in a separate test which is not shown in figure 3.23. Test cases here include invalid datasets where a source makes multiple claims for a variable.

In addition to unit tests, *regression tests* were used for testing the implementation of algorithms. Regression tests are designed to ensure code behaves correctly *over time* as new developments are made. This was particularly important for the truth discovery algorithms themselves, where it is not at all obvious if an algorithm has been implemented correctly by looking at its results against a dataset.

To create the regression tests, each algorithm was run against a large synthetic dataset as soon as it was implemented. The results were then stored in a file, and a test created to re-run the algorithm on the same dataset and compare against the stored results. Any inadvertent changes resulting in a change in behaviour for a particular algorithm are then caught the next time the tests are run. Note that the regression tests rely on the algorithms being correct at the time of their initial implementation; this was checked separately in unit tests.

Indeed, this proved useful on a few occasions throughout development, where accidental errors in the algorithm code caused non-obvious errors in results. These were fortunately caught by the regression tests and quickly rectified.

Besides the Python code, JavaScript was used for the interactive elements of the web interface. Unfortunately, *no* automated tests were produced for the JavaScript code. Manual testing was performed during development and a final run through and test of all functionality was performed on completion, but this was the extent of testing for the web interface. On the one hand, the core backend Python code *is* thoroughly

tested, so the web interface testing would only need to cover the interactive user interface aspects, such as dataset entry, displaying of results and so on. Automated testing for such graphical components is much more challenging than for the Python code, and would have required significant time investment. On the other hand, AngularJS, the JavaScript framework used, is built with testability in mind, and various tools exist to simplify automated testing of Angular applications. Tests could be added in future work to address this.

3.3.7 Evaluation against Requirements

To conclude this section, the software as a whole is evaluated with respect to the requirements and use cases of section 3.1.

The table in figure 3.24 shows the status of each requirement in the finished system. Some requirements are tested in several ways; for example generating animations is covered by unit tests, but the animations also inspected visually. Requirements with a subjective component (e.g. support for ‘large’ datasets) are indicated with asterisks.

The table shows that most requirements were indeed met, and for the most part verified by unit tests. Note that most of the unit tested requirements have multiple associated tests, and that the unit tests go far beyond the high-level requirements listed.

Requirement 8, which relates to the handling of ‘large’ datasets, was not met. Whilst it was possible to run algorithms on a reasonably large dataset (see section 3.3.1), it took far too long for the dataset to load. It is unclear where the bottleneck in dataset loading lies; this could be investigated in future work.

Requirement 15 stated that time, memory and iteration statistics should be returned with the results of an algorithm. This is only partially complete in the finished system: no memory usage information is returned. Defining memory usage proved to be challenging, and from initial investigations it seemed that profiling memory usage whilst an algorithm runs would have a negative impact on performance. Memory usage was omitted for these reasons. Note that the other aspects of the requirement, namely time and iteration statistics, *were* met and verified with unit tests.

Naturally requirement 17, which requires that time, memory and iteration statistics can be compared between results, was also only partially met.

Req. number	Unit tested	Manually tested	Partially met	Not met
1	✓			
2	✓			
3	✓			
4	✓			
5	✓			
6	✓			
7	✓			
8				✓*
9	✓			
10	✓			
11	✓			
12	✓			
13		✓*		
14	✓			
15	✓		✓	
16	✓			
17	✓		✓	
18	✓	✓		
19	✓	✓		
20		✓*		
21	✓	✓*		
22		✓		

Figure 3.24: Status of each of the requirements in the finished implementation.

The use cases identified at the start of this section are supported to varying degrees in the final implementation. The main goals for the ‘practitioner’ use case were to run algorithms on real-world datasets and evaluate them with respect to their performance. It has been seen that large real-world datasets are *not* well-supported in the system due to the excessive time it takes to load them. This limits the system’s usefulness in practical applications of truth discovery, especially if real-time results are required.

Evaluation is supported reasonably well; the demonstrations throughout this section show how the system can be used to evaluate and compare algorithms with respect to various metrics, such as accuracy on synthetic

datasets and run-time. One shortcoming is that analysis of memory usage is not possible.

The ‘algorithm developer’ use case requires users to be able to easily extend the codebase to implement new algorithms with ease. This is a subjective goal, and it is difficult for the author of some software to impartially comment on its usability from the perspective of others. Nevertheless, I feel that this goal was mostly achieved, due to the clean separation in the code between the public API and implementation details. The code is also well-documented, well-tested, and includes numerous examples of its usage.

Finally, the system was intended to be helpful as a tool for theoretical truth discovery work. The web interface proved particularly useful for this in my own theoretical work (see chapter 4), where I often needed to run a particular algorithm on small datasets to get a feel for its behaviour in different scenarios. The counter-example used to prove that *Sums* does not satisfy a certain independence property (see theorem 2) was found using the web interface, and the figures demonstrating it were generated using the Python API.

3.4 Future Work

This section discusses unrealised ideas and potential future work for the software framework.

Perhaps the most obvious area for improvement is the selection of algorithms available. Truth discovery has been studied extensively in the literature and many algorithms proposed, yet only five are implemented here. It would be interesting to implement algorithms of different types; for example, algorithms based on statistical models (e.g. [33, 38]) are absent.

Real-valued variables could also be handled in future work. This implementation treats values categorically – 3.001 is completely distinct from 3.002. In practise, however, datasets often contain real-valued variables whose claimed values may be close but not exactly equal. In this case the values can be ‘quantised’ to discrete intervals before truth discovery is applied. Future work could implement this inside the framework to reduce the effort required from end users.

Another area that could be greatly expanded upon is synthetic data generation. The model of synthetic data adopted here is a simple one, and could be extended in various ways. For example, each variable could have its own domain of possible values, instead of a fixed constant domain; this would more accurately reflect real-world datasets, where variables have their own unique characteristics.

The distribution of claimed values could also vary between variables: currently all source incorrect values uniformly at random. This artificially prevents certain algorithms performing as well as they might do on real data, where similarity between claimed values can be used to great effect – for example, implications between claims in *TruthFinder*.

Chapter 4

Theoretical Analysis

This chapter presents and analyses a formal theoretical framework for truth discovery. First, the approach taken is discussed and justified.

4.1 Approach

In the previous chapters, we have motivated the need for a general theoretical framework for truth discovery. To work towards actually constructing one, it is necessary to set out exactly what such a framework will consist of, and what features and properties are required for it to be useful.

The main goal of developing the framework is to set out rigorous definitions for what truth discovery is, which allows the current situation to be modelled whilst also permitting a more general view. The key definitions will therefore be

- What is the ‘input’ to truth discovery? The input has been described in terms of sources, facts, objects and conflicting claims, but this needs to be formulated mathematically.
- What is the ‘output’? We have stated that the output is most commonly trust and belief scores for sources and facts, according to ex-

isting work in the literature. However our aim is to study truth discovery in full generality, and not just the algorithms already in existence. Therefore a more general view could be taken if desired, so long as it can still model existing algorithms.

With these definitions in place, a truth discovery algorithm is simply a mapping from the space of inputs to outputs. This abstracts away the *process* of performing truth discovery so that ‘algorithm’ is not the correct term to use. We opt for *truth discovery operator* to describe a mapping from inputs to outputs.

There are several criteria against which to judge the usefulness of the developed framework.

- **Ability to model existing approaches:** We aim to find a unified framework that allow as many existing algorithms in the literature as possible to be represented.
- **Simplicity:** the key definitions should be easy to interpret, and should relate to intuitive notions of truth discovery in a clear way.
- **Flexibility:** we wish to prove properties of operators, compare different operators, and develop axioms, so the framework should be easy and flexible to work in.
- **Generality:** the framework should be general and ‘unopinionated’ enough to be useful as foundations for future work, i.e. it should not rely on specific ideas and approaches to performing truth discovery. It should also be general in the sense of facilitating easy comparison between truth discovery and related areas in the literature. This will allow ideas in these areas to be applied to truth discovery, e.g. many axioms from social choice could be translated to truth discovery.

Once the framework has been established, we aim to develop axioms for operators. In line with axiomatic foundations for other problems, the axioms should represent intuitively desirable properties that a ‘reasonable’ operator should satisfy. The power of the axiomatic approach is to then consider multiple axioms together; the types of results attained include *impossibility results*, where it is proved that no operator¹ can satisfy a set of axioms, and *representation theorems*, where a set of sound and complete axioms are found for a particular operator. For example, in the context of

ranking systems, the authors in [3] show that two seemingly complementary and desirable axioms cannot be satisfied simultaneously, which has implications when deciding which ranking system to use in practise.

Requirements for ‘good’ axioms include having simple interpretations and representing desirable properties in some way or another.

4.1.1 Overview of Approach

We now give an overview and justification of the approach to developing the theoretical framework, before the formal definitions are given in section 4.2.

For input to a truth discovery problem, it was noted in section 2.2 that the following form is applicable to many approaches in the literature: we have a set of sources, facts and objects, and sources claim facts for objects.

To represent this formally, a graph-theoretic representation is chosen. Nodes will be sources, facts and objects. Edges between nodes represent the obvious relations: an edge from a source to a fact represents the source claiming that fact, and an edge from a fact to an object indicates the object the fact relates to. Setting this up in graph theory allows for simple interpretation, and allows concepts in graph theory to be usefully applied to describe properties of the input (for example, the notion of *connected components* is key in axiom 6).

Using a well established tool such as graph theory is hoped to also provide flexibility for future refinements to consider more complex problems: notions such as weighted edges, annotated nodes etc. could be used to conveniently describe additional properties of the input.

Finally, a graph representation is already used in the related area of ranking systems [2, 3]. Using a similar set up allows comparison between the two areas.

For output, consider the two main ideas discussed in section 2.2: assigning each fact a score and selecting a single true fact for each object. Since the latter is a special case of the former, the former is more suitable for a general theory of truth discovery.

Note that assigning a numeric score to each source and fact in particular induces a *ranking* of the sources and facts. We argue that the essence

¹ We use ‘operator’ as a blanket term to refer to social choice functions, ranking systems, annotation aggregators etc.

of truth discovery lies more in this induced ranking than the particular numerical scores, and will therefore define the output of truth discovery as a pair of rankings (precisely, *total preorders*) on the set of sources and facts.

Indeed, when applying truth discovery methods to determine which sources to trust and which facts to believe, one is interested in which sources/facts are *more believable than others*, not in the particular numeric values produced by an algorithm. Additionally, the numeric values produced often do not have any semantic meaning [27], which prevents inter-algorithm comparison. The induced rankings can therefore act as a bridge between results from different algorithms.

A similar view is also taken in social choice and the axiomatic theory of ranking systems, where rankings instead of numeric scores are the main objects of interest. Taking the same approach here highlights the similarities between truth discovery and these areas, and allows concepts in these areas to be carried over into truth discovery.

Nevertheless, to model in their entirety the algorithms that produce numeric scores, it will be possible to define such operators in the framework as more general objects, but restrict our attention mainly to the ranking-output operators.

4.2 A Framework for Truth Discovery

In this section we define formally the graph-theoretic framework for truth discovery, and set out the central problem of truth discovery via the definition of a *truth discovery operator*. We then develop axioms (desirable properties) for such operators, and prove some basic properties regarding them. Finally, we consider how to represent real-world truth discovery algorithms in the developed framework, focussing specifically on *Sums* [27].

First we state some standard definitions and notation.

4.2.1 Standard Definitions and Notation

Definition 1. A *preorder* on a set X is a binary relation \preceq that is reflexive and transitive:

1. $x \preceq x$ for all $x \in X$ (reflexivity)

2. If $x \preceq y$ and $y \preceq z$, then $x \preceq z$ for all $x, y, z \in X$ (transitivity)

A *total preorder* is a preorder that is complete: for all $x, y \in X$, $x \preceq y$ or $y \preceq x$. $\mathcal{L}(X)$ is the set of all total preorders on X .

The *strict order* induced by \preceq is \prec where $x \prec y$ if and only if $x \preceq y$ but not $y \preceq x$. Note that \prec is irreflexive, transitive (and hence asymmetric), and is not complete.

The *equality predicate* associated with \preceq is \simeq , where $x \simeq y$ if and only if $x \preceq y$ and $y \preceq x$. Note that \simeq is an equivalence relation on X .

Definition 2. A *permutation* of a set X is a bijective mapping $X \rightarrow X$. We use cyclic notation for permutations: $\pi = (a, b, c)$ is the mapping $\pi(a) = b$, $\pi(b) = c$, $\pi(c) = a$, and $\pi(x) = x$ for $x \notin \{a, b, c\}$. Juxtaposition of cycles denotes function composition.

Definition 3. Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there is a bijective mapping $\phi : V \rightarrow V'$ such that $(u, v) \in E \iff (\phi(u), \phi(v)) \in E'$.

Definition 4. Let $G = (V, E)$ be an undirected graph, and define a relation \sim on V by $u \sim v$ iff there is a path from u to v in G (including the zero-length path when $u = v$). It is easily checked that \sim is an equivalence relation. A *connected component* of G is an induced subgraph of an equivalence class of \sim .

Notation. For sets X and Y , Y^X denotes the set of all functions $X \rightarrow Y$.

4.2.2 Truth Discovery Definitions

We consider fixed finite and mutually disjoint sets \mathcal{S} , \mathcal{F} and \mathcal{O} , called the *sources*, *facts* and *objects* respectively. All definitions and axioms will be stated with respect to these sets.

Definition 5. A *truth discovery network* is a directed graph $N = (V, E)$ where $V = \mathcal{S} \cup \mathcal{F} \cup \mathcal{O}$, and $E \subseteq (\mathcal{S} \times \mathcal{F}) \cup (\mathcal{F} \times \mathcal{O})$ satisfies the following properties:

1. Each $f \in \mathcal{F}$ has a unique successor node in \mathcal{O} , denoted $\text{obj}(N, f)$ (i.e. each fact relates to a single object).

2. For $s \in \mathcal{S}$ and $o \in \mathcal{O}$, there is at most one directed path from s to o (i.e. sources can only claim one fact per object).
3. $(\mathcal{S} \times \mathcal{F}) \cap E$ is non-empty (i.e. at least one claim is made).

We will say that s *claims* a fact f when $(s, f) \in E$. Let \mathcal{N} denote the set of all truth discovery networks.

Remark. Note that the definition above does not rule out a source s making no claims, a fact f being claimed by no sources, or an object o having no associated facts or sources.

In the special case where each object has exactly two associated facts, the objects can be seen as binary variables taking one of two values, e.g. true or false. The truth discovery network is then similar to a set of judgements in judgement aggregation [13] for an agenda consisting only of propositional variables.

Notation. For convenience, for a network $N = (V, E)$, define:

$$\begin{aligned} \text{facts}(N, s) &= \{f \in \mathcal{F} : (s, f) \in E\} \\ \text{facts}(N, o) &= \{f \in \mathcal{F} : (f, o) \in E\} \\ \text{src}(N, f) &= \{s \in \mathcal{S} : (s, f) \in E\} \\ \text{src}(N, o) &= \{s \in \mathcal{S} : \exists f \in \mathcal{F} : (s, f), (f, o) \in E\} \end{aligned}$$

Definition 6. A truth discovery operator T is a mapping $T : \mathcal{N} \rightarrow \mathcal{L}(\mathcal{S}) \times \mathcal{L}(\mathcal{F})$, i.e. T assigns to each truth discovery network N a pair of total preorders $T(N) = (\sqsubseteq_N^T, \preceq_N^T)$ on the sets \mathcal{S} and \mathcal{F} respectively.

$s_1 \sqsubseteq_N^T s_2$ means s_2 is ranked as *more trustworthy* than s_1 in the network N according to T ; $f_1 \preceq_N^T f_2$ means f_2 is ranked as *more believable* than f_1 .

In practise, real-world truth discovery algorithms do not usually output a ranking of sources and facts directly, but instead assign each source a numeric *trust score*, and each fact a *belief score*. This is captured in the following definition.

Definition 7. A numerical truth discovery operator T is a mapping $T : \mathcal{N} \rightarrow \mathbb{R}^{\mathcal{S}} \times \mathbb{R}^{\mathcal{F}}$, i.e. T assigns to each truth discovery network N functions $t_N : \mathcal{S} \rightarrow \mathbb{R}$ (referred to as the *source trust mapping*) and $b_N : \mathcal{F} \rightarrow \mathbb{R}$ (referred to as the *fact belief mapping*).

Remark. Any numerical truth discovery operator T naturally induces a truth discovery operator T' , where for any truth discovery network N we define

$$\begin{aligned} s_1 \sqsubseteq_N^{T'} s_2 &\iff t_N(s_1) \leq t_N(s_2) \\ f_1 \preceq_N^{T'} f_2 &\iff b_N(f_1) \leq b_N(f_2) \end{aligned}$$

for $s_1, s_2 \in \mathcal{S}$ and $f_1, f_2 \in \mathcal{F}$.

In this work we deal primarily with truth discovery operators as defined in definition 6, instead of working directly with numeric trust and belief scores as in definition 7. This is due to the reasons discussed in section 4.1.1; namely that, from a theoretical point of view, we are interested in the qualitative ranking of sources and facts rather than quantitative values.

One disadvantage to this approach is that whilst we can tell whether or not s_1 is more trustworthy than s_2 , we cannot tell by *how much*. For example, consider two numerical operators T and T' and $\mathcal{S} = \{s_1, s_2\}$ such that $t_N(s_1) = 0.5, t_N(s_2) = 0.51$, and $t'_N(s_1) = 0.01, t'_N(s_2) = 0.99$. Both operators induce the same ranking on \mathcal{S} , yet T considers the two sources to have similar trust values while T' considers s_2 to be much more trustworthy than s_1 .

4.2.3 Axioms

The fact-believability component of truth discovery can be seen as a special case of voting in the theory of social choice [41], where agents are sources and alternatives are facts. Each source then ranks the facts it claims above all other facts, and ranks its claimed facts equally.² Several axioms for voting rules from this theory can be adapted to truth discovery, and we do so presently.

Symmetry and Dictatorship

Definition 8. Two truth discovery networks N and N' are *equivalent* if there is a graph isomorphism π between them that preserves sources, facts and objects:

² Note that the formulation of social choice must allow for agents to have *weak* preferences for alternatives, where ties are allowed.

1. $\pi(s) \in \mathcal{S}$ for all $s \in \mathcal{S}$
2. $\pi(f) \in \mathcal{F}$ for all $f \in \mathcal{F}$
3. $\pi(o) \in \mathcal{O}$ for all $o \in \mathcal{O}$

In such case we write $\pi(N)$ for N' .

Figure 4.1 shows an example of two equivalent networks.

The first axiom states that the ordering of sources and facts should not depend on the ‘names’ of the sources, facts and objects in the input.

Definition 9. Let T be a truth discovery operator. T satisfies *symmetry* if for any equivalent truth discovery networks N and $N' = \pi(N)$, we have

$$s_1 \sqsubseteq_N^T s_2 \iff \pi(s_1) \sqsubseteq_{N'}^T \pi(s_2)$$

and

$$f_1 \preceq_N^T f_2 \iff \pi(f_1) \preceq_{N'}^T \pi(f_2)$$

T satisfies *source-symmetry* if both the above statements hold in cases where π only permutes sources, i.e. $\pi(f) = f$ and $\pi(o) = o$ for all $f \in \mathcal{F}$ and $o \in \mathcal{O}$. *Fact-symmetry* and *object-symmetry* are defined similarly.

Axiom 1 (Symmetry). *An operator T should satisfy symmetry.*

Source-symmetry is analogous to *anonymity* in classical social choice, where all voters are treated identically, and fact and object symmetry are analogous to *neutrality*, where the alternatives being voted on are treated identically [41].

Note that source-symmetry does not mean that sources are treated *equally* per se, since some sources are presumed to be more trustworthy than others (this is more or less the central premise of truth discovery). Instead it means that sources are judged solely by the facts that they claim, not their identities.

Proposition 1. *T satisfies symmetry if and only if it satisfies source, fact and object symmetry.*

Proposition 2. *If T satisfies source-symmetry and $\mathit{facts}(N, s_1) = \mathit{facts}(N, s_2)$ in some network N , then $s_1 \simeq_N^T s_2$.*

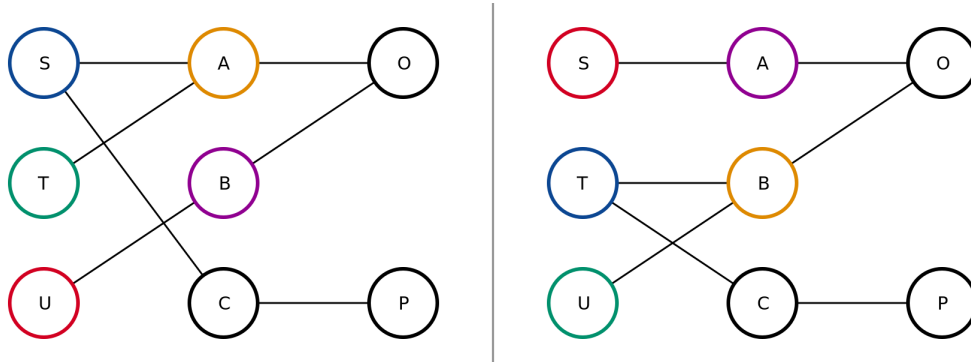


Figure 4.1: Example of two equivalent truth discovery networks. Here the isomorphism between the left and right networks is $\pi = (S, T, U)(A, B)$. The colours show that the structure of the right network is the same as the left, just with different 'names' for the nodes.

Similarly, if T satisfies fact-symmetry and $\text{src}(N, f_1) = \text{src}(N, f_2)$, $\text{obj}(N, f_1) = \text{obj}(N, f_2)$, then $f_1 \approx_N^T f_2$.

That is, any two sources making identical claims are ranked equally, and any two facts for the same object with identical support from sources are ranked equally.

All missing proofs are presented in appendix C.

In some sense the opposite of source-symmetry, where the identities of the sources are irrelevant and only the structure of the truth discovery network is important, is a situation where *only* the identities of the sources are considered.

Definition 10. A source $s^* \in S$ is *authoritative* for a network $N = (V, E)$ with respect to an operator T if $s \sqsubseteq_N^T s^*$ for all $s \in S$, and $(s^*, f^*) \in E$ implies $f \preceq_N^T f^*$ for all $f \in \mathcal{F}$.

In other words, s^* is more (or equally) trusted than all other sources, and its facts are more (or equally) believable than all others.

We also define a strict version: s^* is *strictly authoritative* if additionally $s \sqsubset_N^T s^*$ for all $s \neq s^*$, and $f \prec_N^T f^*$ for all $f, f^* \in \mathcal{F}$ such that $(s^*, f^*) \in E$ and $(s^*, f) \notin E$.

An operator T is a *dictatorship* if there is a source $s^* \in S$ (the dictator) that is authoritative for all networks, and T is a *strict dictatorship* if there is a source $s^* \in S$ that is strictly authoritative for all networks.

Axiom 2 (Non-dictatorship). *An operator T should not be a dictatorship.*

As noted above, source-symmetry and dictatorship are conceptually at odds with one another. This is expressed formally in the following proposition, which essentially shows that only trivial operators can satisfy both properties.

Proposition 3. *If an operator T is both source-symmetric and a dictatorship, then for any network N :*

1. *All sources are ranked equally*
2. *If f_1 is claimed by at least one source in N , then $f_2 \preceq_N^T f_1$ for all facts f_2 .*

In particular, there is no operator that is both source-symmetric and a strict dictatorship.

Example 1. A trivial operator that satisfies symmetry and dictatorship is one that always ranks all sources and facts equally: $T_{triv}(N) = (\mathcal{S}^2, \mathcal{F}^2)$.

If we restrict \mathcal{N} to those networks where all facts are claimed by at least one source, then proposition 3 shows that T satisfies source-symmetry and dictatorship if and only if $T = T_{triv}$.

Without this restriction, facts not claimed by any source may be ranked strictly below other facts. Indeed, consider T defined as follows. For any network N write $F_+ = \{f \in \mathcal{F} : \text{src}(N, f) \neq \emptyset\}$, and define T by $s_1 \simeq_N^T s_2$ for all $s_1, s_2 \in \mathcal{S}$, and

$$f_1 \preceq_N^T f_2 \iff f_2 \in F_+ \text{ or } f_1 \notin F_+ \quad (f_1, f_2 \in \mathcal{F})$$

T is trivially a dictatorship for any $s^* \in \mathcal{S}$. It can be easily checked that \preceq_N^T is a well-defined total preorder, and that T is also symmetric. However any fact in $\mathcal{F} \setminus F_+$ ranks strictly below any fact in F_+ .

Dictatorship requires there to be a fixed source that is authoritative in all networks. A weaker form of dictatorship, which is more compatible with symmetry, is where the authoritative source may depend on N .

Definition 11. A truth discovery operator T is a *generalised dictatorship* if for every network N there exists a source $s_N \in \mathcal{S}$ that is authoritative for N with respect to T . A *generalised strict dictatorship* is defined similarly.

Clearly a dictatorship is also a generalised dictatorship.

Example 2. An operator that is both symmetric and a generalised dictatorship is the numerical operator T defined as follows. For any truth discovery network N , let $Q_N = \{s \in \mathcal{S} : |\mathbf{facts}(N, s)| = \max_{x \in \mathcal{S}} |\mathbf{facts}(N, x)|\}$ be the set of sources making the maximal number of claims, and set

$$t_N(s) = \begin{cases} 1 & \text{if } s \in Q_N \\ 0 & \text{otherwise} \end{cases}$$

$$b_N(f) = \begin{cases} 1 & \text{if } \mathbf{src}(N, f) \cap Q_N \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

Clearly any source in Q_N is authoritative.

To show symmetry, let N and $\pi(N)$ be equivalent networks. Let $s \in \mathcal{S}$. First note that $f \in \mathbf{facts}(N, s)$ iff $\pi(f) \in \mathbf{facts}(\pi(N), \pi(s))$ by definition of equivalent networks, and in particular the restriction of π to $\mathbf{facts}(N, s)$ is a bijection into $\mathbf{facts}(\pi(N), \pi(s))$; hence $|\mathbf{facts}(N, s)| = |\mathbf{facts}(\pi(N), \pi(s))|$. Also, since π restricted to \mathcal{S} is a bijection into \mathcal{S} , we have

$$\begin{aligned} \max_{x \in \mathcal{S}} |\mathbf{facts}(N, x)| &= \max_{x \in \mathcal{S}} |\mathbf{facts}(\pi(N), \pi(x))| \\ &= \max_{x \in \mathcal{S}} |\mathbf{facts}(\pi(N), x)| \end{aligned}$$

and so

$$\begin{aligned} s \in Q_N &\iff |\mathbf{facts}(N, s)| = \max_{x \in \mathcal{S}} |\mathbf{facts}(N, x)| \\ &\iff |\mathbf{facts}(\pi(N), \pi(s))| = \max_{x \in \mathcal{S}} |\mathbf{facts}(\pi(N), x)| \\ &\iff \pi(s) \in Q_{\pi(N)} \end{aligned}$$

We see that $t_N(s) = t_{\pi(N)}(\pi(s))$ for any $s \in \mathcal{S}$.

Now let $f \in \mathcal{F}$. Note that $s \in \mathbf{src}(N, f)$ iff $\pi(s) \in \mathbf{src}(\pi(N), \pi(f))$. Using this fact and $s \in Q_N \iff \pi(s) \in Q_{\pi(N)}$, it is easy to see that $\mathbf{src}(N, f) \cap Q_N \neq \emptyset$ iff $\mathbf{src}(\pi(N), \pi(f)) \cap Q_{\pi(N)} \neq \emptyset$, i.e. $b_N(f) = b_{\pi(N)}(\pi(f))$.

Finally this means, for any $s_1, s_2 \in \mathcal{S}$ and $f_1, f_2 \in \mathcal{F}$:

$$\begin{aligned} s_1 \sqsubseteq_N^T s_2 &\iff t_N(s_1) \leq t_N(s_2) \\ &\iff t_{\pi(N)}(\pi(s_1)) \leq t_{\pi(N)}(\pi(s_2)) \\ &\iff \pi(s_1) \sqsubseteq_{\pi(N)}^T \pi(s_2) \end{aligned}$$

and similarly $f_1 \preceq_N^T f_2$ iff $\pi(f_1) \preceq_{\pi(N)}^T f_2$. Hence T is symmetric.

Note that to be a generalised dictatorship, an operator needs only to rank facts claimed by the most trusted source(s) above all other facts. One may argue that this is not necessarily an undesirable property, since the most trusted source presumably claims believable facts, which *should* rank highly.

However, the operator in example 2 has the additional (perhaps undesirable) property that the ranking is ‘binary’: it is two-level ranking where all non-authoritative sources rank equally to each other and strictly below the authoritative ones. This behaviour is captured in the following definition.

Definition 12. A truth discovery operator T is a *binary generalised dictatorship* if for every network N there is a set of sources $Q_N \subseteq \mathcal{S}$ such that, with

$$t_N(s) = \begin{cases} 1 & \text{if } s \in Q_N \\ 0 & \text{otherwise} \end{cases}$$

$$b_N(f) = \begin{cases} 1 & \text{if } \text{src}(N, f) \cap Q_N \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

it holds that

$$s_1 \sqsubseteq_N^T s_2 \iff t_N(s_1) \leq t_N(s_2)$$

$$f_1 \preceq_N^T f_2 \iff b_N(f_1) \leq b_N(f_2)$$

Remark. If T is a binary generalised dictatorship, it clear that for each network N , each source in Q_N is authoritative.

In such case the orderings \sqsubseteq_N^T and \preceq_N^T are fully determined by the choice of Q_N . Therefore a binary generalised dictatorship can be identified with a mapping $\mathcal{N} \rightarrow 2^{\mathcal{S}}$ that selects the authoritative sources for each truth discovery network.

Axiom 3 (Non- binary generalised dictatorship). *An operator T should not be a binary generalised dictatorship.*

Proposition 4. *Non-dictatorship and non- binary generalised dictatorship are independent.*

Unanimity

The next axioms formalise the idea that if all sources are in agreement about the status of a fact, then a truth discovery operator should respect this in its verdict. Two obvious ways in which sources can be in agreement are when *all* sources believe a fact is true, and when *no* sources believe a fact is true.

Axiom 4 (Unanimity). *For any truth discovery network N , $\text{src}(N, f) = \mathcal{S}$ implies $f' \preceq_N^T f$ for all $f' \in \mathcal{F}$.*

Axiom 5 (Groundedness). *For any truth discovery network N , $\text{src}(N, f) = \emptyset$ implies $f \preceq_N^T f'$ for all $f' \in \mathcal{F}$.*

That is, a fact cannot do better than to be claimed by all sources when T satisfies unanimity, and cannot do worse than to be claimed by no sources when T is grounded.

Note that we do not require strict inequalities here, so as to not be too restrictive. For unanimity in particular, requiring f to rank strictly above all other facts would require T to choose a highest-ranking fact arbitrarily in the case where there are multiple facts claimed by all sources.

Unanimity is similar to the *weak Paretian* property [10] in social choice, which states that whenever each individual prefers an alternative a over b , the social preference order prefers a over b also. It can also be compared to unanimity in judgement aggregation [13].

Axioms similar to groundedness have been proposed for collective annotation (e.g. see *groundedness* in [17])

Example 3. The *majority voting* operator, which ranks a fact by the number of sources claiming it, satisfies unanimity and groundedness. Indeed, define T_{vote} by $s_1 \simeq_N^{T_{\text{vote}}} s_2$ for all $s_1, s_2 \in \mathcal{S}$, and

$$f_1 \preceq_N^{T_{\text{vote}}} f_2 \iff |\text{src}(N, f_1)| \leq |\text{src}(N, f_2)|$$

If $\text{src}(N, f) = \mathcal{S}$ then for all f' we have $\text{src}(N, f') \subseteq \mathcal{S} = \text{src}(N, f)$, so $f' \preceq_N^T f$. Also, if $\text{src}(N, f) = \emptyset$ then $\text{src}(N, f) \subseteq \text{src}(N, f')$ for all f' , so $f \preceq_N^T f'$. Hence T_{vote} is unanimous and grounded.

A consequence of groundedness is that any fact ranking strictly above all others must have been claimed by at least one source (assuming $|\mathcal{F}| > 1$).

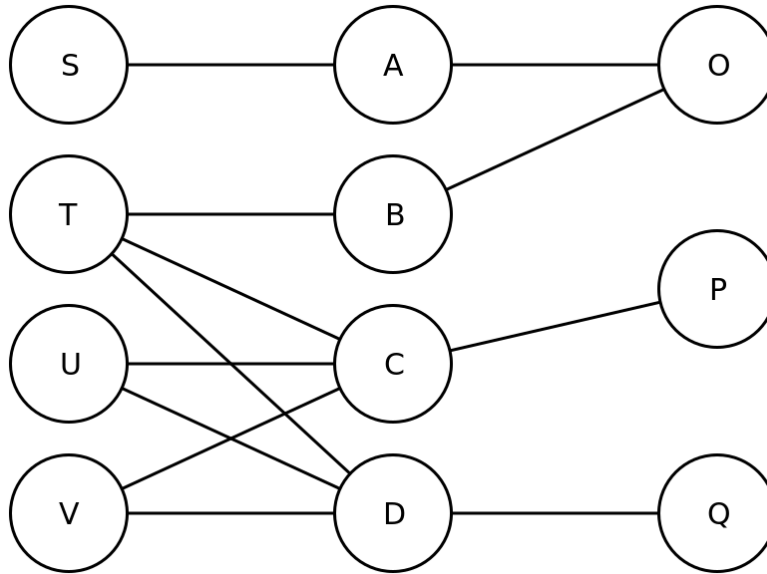


Figure 4.2: Network demonstrating a case where we do not wish for an IIA-type axiom to hold.

Proposition 5. *Unanimity and groundedness are independent.*

Independence

In social choice, the ‘Independence of Irrelevant Alternatives’ (IIA) axiom [6] requires that the relative ranking of two alternatives A and B depends only on the individual rankings of A and B , and not on any ‘irrelevant’ alternative C . That is, if the individual voter preferences are changed such that the ranking of A versus B remains the same for each voter, the ranking of A and B in the social ranking remains unchanged.

To consider whether a similar axiom should be adopted for truth discovery, consider facts A and B in the network shown in figure 4.2. A has support from source S only, who is not in agreement with any other sources, whilst B has support from T , who agrees with both U and V on facts C and D . For this reason, it may be reasonable to expect that T is more trustworthy than S , and therefore B is more believable than A .

Directly translating IIA to this situation, we would require that the ranking of A and B is unchanged if, say, we removed T ’s claims for C and

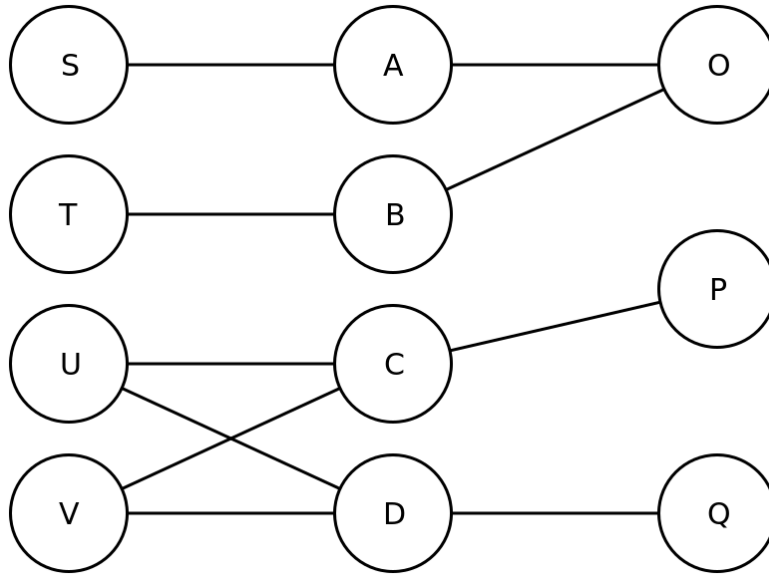


Figure 4.3: Network where some notion of independence may be applied.

D (which are ‘irrelevant’), and instead had S make these claims. However the intuition above suggests that the ranking of A and B should actually be *reversed* in this case, despite the individual judgements on A and B remaining unchanged. For this reason, we argue that a more subtle notion of independence is required.

The issue in figure 4.2 is that C and D are not entirely irrelevant to A and B , since they are connected indirectly via sources that make claims for both objects O and P (namely, source T). Consider removing these indirect links, as show in figure 4.3. In this case it can be argued that C and D truly are irrelevant to A and B , and so changes to the network outside of S, T, A, B and O should not affect the ranking of A and B .

This idea can be generalised by noting that two nodes are ‘relevant’ to each other (perhaps indirectly) if they lie in the same *connected component* of the network (where we consider the connected components of the undirected version of the graph). A suitable independence axiom is therefore to require that changes outside a connected component do not affect the ranking of sources and facts within that component. A precise statement is given below.

Axiom 6 (Independence Of Irrelevant Stuff). *For any truth discovery networks N_1, N_2 with a common connected component G , the restrictions of $\sqsubseteq_{N_1}^T$ and $\sqsubseteq_{N_2}^T$ to $G \cap \mathcal{S}$ are equal, and the restrictions of $\preceq_{N_1}^T$ and $\preceq_{N_2}^T$ to $G \cap \mathcal{F}$ are equal.*

Independence of irrelevant stuff requires that $s_1 \sqsubseteq_{N_1}^T s_2$ if and only if $s_1 \sqsubseteq_{N_2}^T s_2$. A weaker version is to require only that the ranking of s_1 and s_2 in N_1 is *not reversed* in N_2 , not necessarily that the ranking is the same (for example, a strict inequality in N_1 may become weak in N_2).

Axiom 7 (Weak Independence Of Irrelevant Stuff). *For any truth discovery networks N_1, N_2 with a common connected component G and for any $s_1, s_2 \in G \cap \mathcal{S}$ and $f_1, f_2 \in G \cap \mathcal{F}$:*

$$\begin{aligned} s_1 \sqsubseteq_{N_1}^T s_2 &\implies s_1 \sqsubseteq_{N_2}^T s_2 \\ f_1 \prec_{N_1}^T f_2 &\implies f_1 \preceq_{N_2}^T f_2 \end{aligned}$$

Clearly axiom 6 implies axiom 7.

Coherence

A guiding principle of many truth discovery approaches is that facts claimed by trustworthy sources should receive high belief, and sources claiming high belief facts should be seen as trustworthy – the trust and belief rankings should cohere with one another in this sense. The following axiom aims to formalise this in a specific case where it is possible to compare facts for two sources in a straightforward way (and similarly for facts). Its form is inspired by transitivity axioms for ranking systems [3].

Axiom 8 (Coherence). *Suppose N is a truth discovery network and $s_1, s_2 \in \mathcal{S}$ are such that there is a bijective mapping $\phi : \text{facts}(N, s_1) \rightarrow \text{facts}(N, s_2)$ with $f \preceq_{N_1}^T \phi(f)$ for all $f \in \text{facts}(N, s_1)$. Then $s_1 \sqsubseteq_N^T s_2$.*

That is, if the facts claimed by two sources can be paired up such that the fact claimed by s_1 always ranks beneath the fact claimed by s_2 , then s_1 ranks beneath s_2 .

Similarly, if there are facts $f_1, f_2 \in \mathcal{F}$ and a bijection $\phi : \text{src}(N, f_1) \rightarrow \text{src}(N, f_2)$ such that $s \sqsubseteq_N^T \phi(s)$ for all $s \in \text{src}(N, f_1)$, then $f_1 \preceq_N^T f_2$.

Monotonicity

The axioms considered so far have largely dealt with the output of a truth discovery operator for one input network at a time, or for two networks which are structurally similar. Another dimension to the axiomatic approach is to consider how the output of an operator is effected by a *change* in the input to modify it in a particular way.

The following axiom considers what should happen if a network is changed by adding additional support for a particular fact. Intuitively, this should be seen as additional evidence that the fact is true, and an operator should rate it no worse than it did before.

Axiom 9 (Monotonicity). *Let $N = (V, E)$ be a truth discovery network, and $f \in \mathcal{F}$, $s \in \mathcal{S}$ such that $(s, f) \notin E$. Write $o = \text{obj}(N, f)$. Consider the network $N' = (V, E')$ where s claims f , i.e.*

$$E' = \{(s, f)\} \cup E \setminus \{(s, f') : f' \neq f, (f', o) \in E\}$$

Then $f' \preceq_N^T f$ implies $f' \preceq_{N'}^T f$ for all $f' \in \mathcal{F}$.

That is, if f receives additional support from a new source s , its ranking should not get worse.

4.2.4 Iterative Truth Discovery Operators

Real-world algorithms for truth discovery generally compute numerical trust and belief scores, as per definition 7. Additionally, most operate in an *iterative* manner, computing trust and belief scores recursively from one another until the respective scores (hopefully) converge to fixed values.

In this section we define the concept of iterative truth discovery operators to represent and reason about such real-world algorithms.

Definition 13. An *iterative truth discovery operator* is a sequence $I = (T_n)_{n \in \mathbb{N}}$ of numerical truth discovery operators, i.e. a sequence of mappings $T_n : \mathcal{N} \rightarrow \mathbb{R}^{\mathcal{S}} \times \mathbb{R}^{\mathcal{F}}$.

For a network N and $n \in \mathbb{N}$ we will write $T_n(N) = (t_N^n, b_N^n)$ to refer directly to the source trust and claim belief mappings for the n -th iteration (but note that this notation does not make explicit the dependence of t and b on the sequence I).

I is said to *converge* to a numerical operator T^* if $t_N^n \rightarrow t_N^*$ and $b_N^n \rightarrow b_N^*$ pointwise as $n \rightarrow \infty$ for each network N .

Remark. Recall that a numerical operator T^* naturally induces a (non-numerical) operator by rankings sources according to t_N^* and facts according to b_N^* . We may therefore identify a convergent iterative operator I with the operator induced by its limit T^* , and write \sqsubseteq_N^I and \preceq_N^I for the source and fact rankings. To be explicit:

$$s_1 \sqsubseteq_N^I s_2 \iff \lim_{n \rightarrow \infty} t_N^n(s_1) \leq \lim_{n \rightarrow \infty} t_N^n(s_2)$$

and similarly for facts.

Note that this mapping is not injective: there may be many iterative operators with the same limit T^* ; furthermore two distinct numerical operators T^* and T^{**} may induce the same non-numerical operator.

Given a real-world algorithm in practise, one usually aims to determine whether it converges by iterating until the distance (measured in some suitable way) between trust (or belief) scores in consecutive iterations becomes smaller than a fixed threshold. This is of course only a heuristic, since it is not possible to determine whether a sequence converges by considering only finitely many terms. Moreover even if the difference between subsequent trust/belief scores were to become *arbitrarily* small (i.e. smaller than *any* threshold), one still cannot guarantee convergence.³ This means that it may not be trivial to define a real-world algorithm as a truth discovery operator, since it may not be clear whether the trust and belief scores converge in all cases. Nevertheless, in this work we will assume that the iteration *does* converge in all cases and consider which axioms of section 4.2.3 are satisfied given this assumption.

To check whether a convergent operator satisfies the axioms, it will be convenient to have sufficient conditions for some of the axioms that refer to the numeric trust and belief scores directly.

For convenience we assume that trust and belief scores are in the range $[0, 1]$, as this is generally the case in practise.

Lemma 1. *Let I be a convergent iterative truth discovery operator with limit T^* . Suppose that $t_N^n(s) \in [0, 1]$, $b_N^n(f) \in [0, 1]$ for all N, s, f and n . Then:*

1. $t_N^*(s) \in [0, 1]$ and $b_N^*(f) \in [0, 1]$

³ For an example of a sequence exhibiting such behaviour, consider the partial sums of the *Harmonic series* $\sum_{j=1}^{\infty} \frac{1}{j}$, which is divergent. The difference between the $(n+1)$ -th and n th terms is $\frac{1}{n+1}$ which converges to 0 as $n \rightarrow \infty$, yet the series does not converge.

2. If for any equivalent networks N and $N' = \pi(N)$ it holds that

$$t_N^n(s) = t_{\pi(N)}^n(\pi(s))$$

and

$$b_N^n(f) = b_{\pi(N)}^n(\pi(f))$$

for all N, n, s, f , then I satisfies symmetry (axiom 1).

3. If for any network N and $f \in \mathcal{F}$,

$$\text{src}(N, f) = \mathcal{S} \implies b_N^n(f) = 1 \text{ for sufficiently large } n \in \mathbb{N}$$

then I satisfies unanimity (axiom 4).

4. If for any network N and $f \in \mathcal{F}$,

$$\text{src}(N, f) = \emptyset \implies b_N^n(f) = 0 \text{ for sufficiently large } n \in \mathbb{N}$$

Then I satisfies groundedness (axiom 5).

5. If for any networks N_1, N_2 with a common connected component G it holds that $t_{N_1}^n(s) = t_{N_2}^n(s)$ and $b_{N_1}^n(f) = b_{N_2}^n(f)$ for $s \in G \cap \mathcal{S}$ and $f \in G \cap \mathcal{F}$, then I satisfies independence of irrelevant stuff (axiom 6).

6. If for any networks N_1, N_2 with a common connected component G there are sequences of non-negative numbers $(\alpha_n)_{n \in \mathbb{N}}, (\beta_n)_{n \in \mathbb{N}}$ such that, for all $n \in \mathbb{N}, s \in G \cap \mathcal{S}$ and $f \in G \cap \mathcal{F}$,

$$t_{N_2}^n(s) = \alpha_n \cdot t_{N_1}^n(s)$$

$$b_{N_2}^n(f) = \beta_n \cdot b_{N_1}^n(f)$$

then I satisfies weak independence of irrelevant stuff (axiom 7).

Sums

Sums [27] is an iterative algorithm for truth discovery based on the *Hubs and Authorities* [16] algorithm for the ranking of web pages based on the hyperlink structure of the web. The trust score for a source at a given iteration is computed as the sum of the current belief scores of its claimed facts, and the belief score for a fact is given by the sum of its sources trust scores.

The trust/belief scores are normalised at each iteration by dividing by the maximum score; this prevents the scores growing without bound to ensure convergence.

Definition 14 (Sums). *Sums* is the iterative truth discovery operator I_{sums} defined for any network N as follows, where we write t_n for t_N^n for brevity:

$$t_1(s) = \frac{1}{2}, \quad b_1(f) = \frac{1}{2}$$

and for $n > 1$:

$$\begin{aligned} \hat{t}_n(s) &= \sum_{f \in \text{facts}(N,s)} b_{n-1}(f) \\ \hat{b}_n(f) &= \sum_{s \in \text{src}(N,f)} \hat{t}_n(s) \\ t_n(s) &= \frac{\hat{t}_n(s)}{\max_{x \in \mathcal{S}} \hat{t}_n(x)} \\ b_n(f) &= \frac{\hat{b}_n(f)}{\max_{y \in \mathcal{F}} \hat{b}_n(y)} \end{aligned}$$

Note that \hat{t} and \hat{b} are only used to define t and b , and are not part of the definition of *Sums* itself.

If a source s makes no claims in N (i.e. $\text{facts}(N, s) = \emptyset$), we follow the convention that an empty sum is 0 and set $\hat{t}_N^n(s) = 0$ (similar for a fact without sources).

Remark. *The normalisation ensures that trust and belief scores always lie in $[0, 1]$. Note that any source that makes at least one claim has strictly positive trust score for all n , and any fact with at least one source has strictly positive belief score. Since any network N must contain at least one claim, this ensures that the maximum in the denominator for t_n and b_n is non-zero.*

Theorem 1. *If Sums is convergent, it satisfies symmetry (axiom 1), non-dictatorship (axiom 2), unanimity (axiom 4), groundedness (axiom 5) and weak independence of irrelevant stuff (axiom 7).*

The proof of theorem 1 uses lemma 1, and can be found in appendix C. For non-dictatorship, it is sufficient by proposition 3 and symmetry to find a single truth discovery network in which *Sums* does not rank all sources equally. Figure 4.4 shows such a network: in this network A ranks strictly above B and C , which are ranked equally.

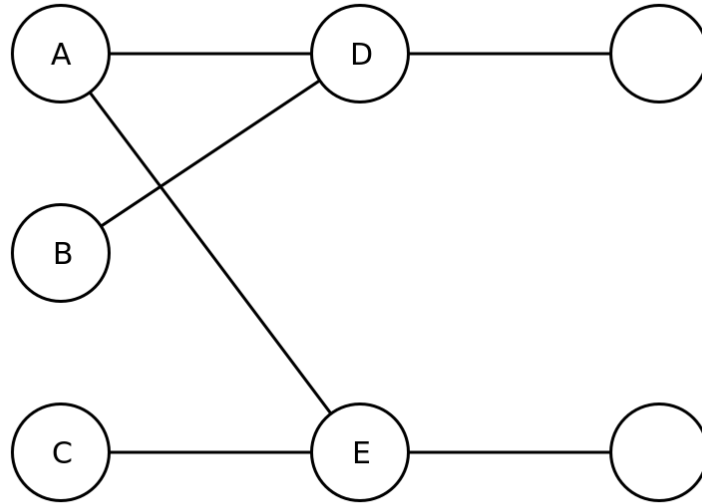


Figure 4.4: A network in which *Sums* does not rank all sources equally.

Theorem 2. *Sums* does not satisfy Independence of Irrelevant Stuff (axiom 6).

Figures 4.4 and 4.5 provide an example of *Sums* failing to satisfy independence. The details can be found in the proof in appendix C.

We conjecture that *Sums* is indeed convergent for any input network. Indeed, it is easy to see that *Sums* is closely related to *Hubs and Authorities* [16], where trust scores correspond to hub scores, and belief scores correspond to authority scores. There are only two differences: the initial scores ($\frac{1}{2}$ in *Sums* and 1 in *Hubs and Authorities*), and the method of normalisation (*Sums* ensures the maximum score is 1, whereas *Hubs and Authorities* ensures the sum of the squares of the scores is 1). In [16] it is proved that *Hubs and Authorities* always converges using techniques from linear algebra. The difference in normalisation only amounts to using a different norm to measure convergence (namely $\|\cdot\|_\infty$ in *Sums* instead of $\|\cdot\|_2$), so it is hoped that the proof can be modified to work for *Sums*.

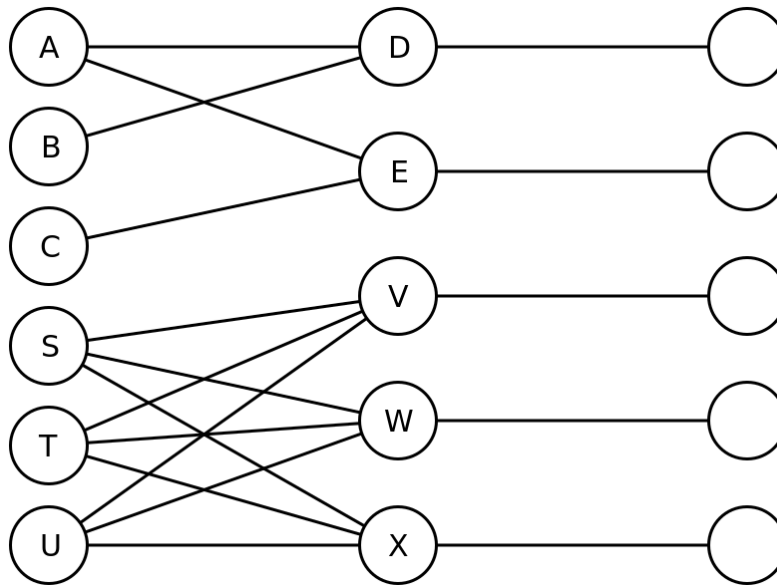


Figure 4.5: A counterexample to independence for *Sums*: this network contains the one shown in figure 4.4 as a connected component, but A is ranked equal to B here whereas it ranks strictly above B in figure 4.4.

4.3 Evaluation

In this section, the framework and results of the previous section are evaluated. In particular, the framework is evaluated with respect to the criteria outlined in section 4.1.

Ability to model existing approaches

The main definitions are that of a truth discovery network and a truth discovery operator. For the framework to be useful as tool for analysing truth discovery, these definitions should be compatible with the existing ideas and approaches to truth discovery, in the sense that it should be possible to define existing algorithms within the framework.

For the input network definition, it is easily verified that the definition given is capable of modelling the input required for many algorithms proposed in the literature. Indeed, since there is little disagreement on the

form on input across algorithms, there are several possible choices for the exact form of the input, and the one we make is sufficient.

For the operator definition, we must consider whether the output of an operator, namely a pair of total preorders on the set of sources and facts, is sufficient to model the existing approaches. As mentioned previously, output usually consists of numeric trust scores for each source, and either numeric belief scores for facts or a single ‘true’ fact for each object.

Whilst neither of these options involve rankings of sources and facts directly, they both *induce* such rankings, allowing both forms of output to be reduced to a common form. Indeed, it was already noted that the numeric scores induce rankings by simply sorting sources and facts by their score in ascending order.

When given instead an identified ‘true’ fact for each object, a ranking is induced by having the identified true facts rank equally to each other and strictly above non-true facts. For some algorithms, the identified true fact may not have been claimed by any source; this is not a problem in the framework since we permit a network to contain facts with no associated sources. One may simply take the set of facts for an object to be *all* permitted values for the object.⁴

The definition of an operator can therefore model many existing algorithms. However it neglects an important characteristic of many algorithms in practise, which is that they operate *iteratively*, running until the results converge in some sense. For this reason, we defined an *iterative operator*.

This allowed a real-world algorithm, *Sums*, to be defined and analysed in the framework. Due to time constraints, no other algorithms were realised. However it is clear that many other algorithms can be defined in a similar way. This is immediate for algorithms similar to *Sums*, such as *Average-Log*, *Investment* and *PooledInvestment* [27], since they only differ in their formulae for trust and belief score updates; their fundamental method of operation is the same.

⁴We make the assumption that the domain of all possible values is well-defined as a set.

Simplicity

Simplicity is naturally a subjective aim, since what appears simple to the author may not appear so to others. Nonetheless, we argue that the framework achieves its goal of expressing ideas as simply as possible.

For example, one of the key definitions is that of truth discovery network. Adopting a graph-theoretic approach, the definition (including the constraints on the graph) is easy to understand for those familiar with the basics of graph theory, and even lends itself to pictorial representations of truth discovery networks.

The next main definition is that of a truth discovery operator. This is defined simply as a mapping from a space of inputs, denoted \mathcal{N} , to a space of outputs, denoted $\mathcal{L}(\mathcal{S}) \times \mathcal{L}(\mathcal{F})$. The definition of an iterative operator extends the non-iterative one in a natural way, by defining it simply as a sequence of non-iterative operators.

Whilst the notation for the rankings for a particular operator and particular network may appear crowded at first, it expresses all the components of the ranking without having to introduce additional notation prior to its use each time. It is inspired by the notation introduced by Altman & Tennenholtz [3] for ranking systems.

We also believe that the axioms are expressed as simply as possible. Where the formalities become tedious, plain-English explanations are provided to give insight into the intuition backing them.

Flexibility

Flexibility is also not something that can be objectively verified. Nevertheless, we were able to express a variety of ideas in the framework without excessive complexity, and the basic results shown have simple proofs.

Generality

By and large, the framework is neutral with respect to any specific idea or approach for truth discovery. A possible exception is perhaps the definition of an iterative operator; this is defined as a sequence of *numerical* operators, whereas in principle an iterative algorithm need not compute numerical scores. Indeed, algorithms such as CRH [22] operate in an iterative manner, yet do not assign belief scores to facts.

However, the definition could easily be generalised to a sequence of non-numerical operators, and a separate definition given for numerical iterative operators. The definition as given was chosen to reduce the number of definitions required and improve the clarity of the work, since the only algorithm actually discussed *does* in fact use numeric scores.

Another aim for the framework was to permit comparison between truth discovery and related areas in the literature. The framework is general enough for this; whilst clearly being a framework for truth discovery, one may easily see truth discovery networks and operators from the perspective of social choice and ranking systems. For example, it is easily seen that truth discovery networks form a particular class of graphs, and a truth discovery operator is essentially a ranking system defined on this class of graphs. The similarity is also demonstrated empirically by the fact that many of the developed axioms are directly inspired by axioms in these areas, but still have intuitive interpretations in terms of truth discovery. However, the similarities do not extend to areas less influenced by social choice, such as argumentation theory and belief revision.

Having evaluated the definitions comprising the framework, we turn to the work carried out inside it, namely the development of axioms and analysis of operators with respect to these axioms.

Axioms and Results

Several axioms covering a range of ideas were defined, each accompanied by a description of the intuition backing them. It is hoped that the axioms represent ‘desirable’ properties for operators, although of course desirability is a subjective property.

However, little work was done beside stating the axioms. An important aspect of the axiomatic approach is to analyse the *implications* of axioms, and to consider interactions between them (e.g. impossibility and representation results, or interesting properties entailed by a combination of axioms). In section 4.2 only very simple results regarding the axioms were proved, such as the independence of similar axioms and incompatibility of source-symmetry and dictatorship. More work is required to fully study the developed axioms.

In terms of analysis of operators with respect to the axioms, a set of sound axioms for *Sums* was obtained. Whilst I expect that these axioms are not *complete*, this was not considered in section 4.2.

A clear weakness of the analysis is that only one real-world algorithm is considered. One of the aims for the framework was a unified model that can represent many different algorithms – defining only a single algorithm does not demonstrate this particularly well.

As such, there is no comparison of the theoretical properties *between operators*. An interesting task would be to find axioms that distinguish between operators, i.e. axioms that one operators satisfies but another does not. This would provide insight into meaningful differences between operators, which is hard to glean from the definitions in terms of an iterative procedure. Knowledge of the differences in terms of simple desirable properties could be helpful in deciding which algorithm to use in practise for real applications of truth discovery.

It was noted above that a strength of the framework is the scope for comparison between truth discovery and other areas. A weakness of the analysis is that no such comparison was *actually carried out*, besides the casual observations linking truth discovery to social choice and ranking systems. To make the links more concrete, one could consider whether social welfare functions, ranking systems, annotation aggregators etc. can be formulated as truth discovery operators, or vice versa.

4.4 Future Work

This section discusses possible future work for the theoretical part of the project.

4.4.1 Unfinished Business

Due to time constraints, there are some elements of the work left in a semi-finished state. The most obvious example is regarding the convergence of *Sums*. At the end of section 4.2.4, we conjecture that the trust and belief scores of *Sums* convergence in *any* input network. As mentioned there, a proof might be obtained by modifying the proof of the convergence of *Hubs and Authorities*, to which *Sums* is closely related.

The lack of a proof of the convergence in all cases (or otherwise, as the case may be) leaves the analysis of *Sums* in the unsatisfactory state where it is unclear if *Sums* properly defines a truth discovery operator in

the sense of definition 6. This also means that theorems 1 and 2 must include the hypothesis that *Sums* is actually convergent.

Another unfinished element of the *Sums* analysis is that we make no determination (or even conjecture) on whether it satisfies the monotonicity and coherence axioms. Monotonicity and coherence are also not addressed in lemma 1.

4.4.2 Future Directions

In terms of future directions for the framework, two aspects to consider are *problems* that could be fixed in future work, and *new ideas* that could be explored. New ideas can be further split into ideas for the framework itself (i.e. new or modified definitions and concepts) and ideas for new results that one could attempt to prove.

A first problem is in the role of *objects*. Whilst objects are an important part of truth discovery in many approaches (particularly those which output an identified true fact for each object), they do not play much of a role in the framework of section 4.2, beside the constraint that sources make at most one claim per object in definition 6.

Future work could address this by developing axioms that consider objects directly. For example, one could consider what happens if two objects are ‘merged’, i.e. facts from each combined under object in a new network. This would presumably have little or no effect in algorithms such as *Sums* and *Average-Log* [27] which do not use objects in their calculations, but would affect algorithms such as *Investment* [27] and *TruthFinder* [35] where the role of objects is important.⁵

In many of the definitions and axioms, there is redundancy where we state a property for source rankings, and then an almost identical one for fact rankings. Similarly, the proofs often prove a result for source rankings, and the result for facts follows by an identical argument. This may lead one to wonder whether truth discovery as defined here is just an instance of ranking k groups of nodes in a $k + 1$ -partite graph for the special case $k = 2$ (plus one to account for the objects). A similar idea is considered in the PhD thesis of Pasternack ([26], section 4.5.3) to represent ‘groups’ of sources.

⁵ Objects are termed *mutual exclusion sets* in [27].

Taking this more general view would highlight the symmetry between the groups (sources and facts in our case), where symmetry exists, and remove redundancy from definitions and proofs. However, this new problem may no longer represent truth discovery in the same way, and would need careful interpretation. In particular, it is not clear how objects would fit into this approach. Nevertheless, it could be something to consider in future work.

There are numerous possible extensions to the framework that could be made. One example is the definition of an iterative truth discovery operator (definition 13). Most real-world algorithms operate not only iteratively but *recursively*, updating trust and belief scores based on the scores in the previous iteration. However, the recursion aspect is not captured in any definition in this work. Making such a definition could lead to a simpler representation of these algorithms in terms of the update rules. This would provide a more general set up for studying recursively iterative algorithms; for example one could consider the effects of making *changes* to the update rules. It could also provide a method for comparing different algorithms, by comparing their update rules.

More potential extensions to the framework come from the numerous extensions to the basic truth discovery model, some of which were listed in section 2.2. This would allow for consideration of more specific sub-problems in truth discovery and lead to a richer theory.

Future work on the axioms themselves could involve *generalising* axioms where possible. For example, one may note that unanimity and groundedness (axioms 4 and 5) are special cases of the following axiom.

Axiom 10. *For any truth discovery network N and facts $f_1, f_2 \in \mathcal{F}$, $\text{src}(N, f_1) \subseteq \text{src}(N, f_2)$ implies $f_1 \preceq_N^T f_2$.*

That is, facts with ‘more’ sources rank higher than those with less.⁶ Unanimity is the case where $\text{src}(N, f_2) = \mathcal{S}$, and groundedness is the case where $\text{src}(N, f_1) = \emptyset$.

Future work could investigate this axiom in detail. For example, it would be interesting to find operators that satisfy unanimity and/or groundedness, but *not* this more general axiom.

Also relating to axioms, it was mentioned in the text preceding the monotonicity axiom (axiom 9) that most of the axioms deal with the prop-

⁶ ‘More’ here is in the sense of set inclusion, not the *number* of sources

erties of rankings in a static network. Monotonicity, however, deals with *modifications* to a network and the resulting effect on rankings. These kind of axioms, where input is changed in some way, are used to great effect in social choice and related areas; a notable example is the axiomatisation of *PageRank* [25] in [4]. Developing more axioms of this type may therefore be useful in seeking sound and complete axioms for truth discovery algorithms.

Finally, the approach taken throughout the theoretical work was largely to apply ideas from social choice to truth discovery. Future work could explicitly incorporate ideas from other related areas such as argumentation theory and belief revision.

In terms of new results, a major aspect of future work will be to formulate more results involving the axioms. As alluded to in the evaluation in section 4.3, this could include implications of the axioms (particularly the implications of an operator satisfying multiple axioms simultaneously), impossibility results, and finding sound and complete axioms for a particular algorithm or class of algorithms. More truth discovery operators could also be defined to provide examples of axioms failing to hold.

Chapter 5

Conclusions

Truth discovery has been explored in this project from a practical perspective in chapter 3, and a theoretical perspective in chapter 4. The two halves have been largely disjoint, although it is noted that the software implementation was useful in constructing examples for the theoretical work and verifying results empirically.

The practical aspect involved implementing truth discovery algorithms in Python and making them accessible through command-line and web-based user interfaces. Methods for evaluating and comparing algorithms were provided, and some basic algorithm analysis was demonstrated in section 3.3. Additionally, visual representation of truth discovery datasets and results were produced, which were useful for demonstration of concepts throughout the theoretical chapter.

For the theoretical component, we set out a formal framework in which to study truth discovery; the key definitions being that of a *truth discovery network* and *truth discovery operator*. Graph-theoretic and ordinal representations were used, highlighting the similarities with related areas in the literature such as ranking systems and social choice.

Following the approach commonly taken in social choice, *axioms* were defined to encode desirable properties of operators. The axioms presented are largely inspired by existing axioms from other areas in the literature, adapted to the truth discovery framework. Some basic results regarding

these axioms were shown, including a proof that *Sums* satisfies some but not all of our axioms when viewed as an operator within the framework.

Chapter 6

Reflection on Learning

To conclude this report, we reflect on the skills developed and what was learned throughout the project. There were several aspects to the project, each requiring a unique set of skills. In particular, there were elements of *research*, *software development*, *theoretical work*, and *project management*.

Research was required throughout the project, and especially in its early stages. Understanding the cutting-edge approaches and ideas in the truth discovery world required careful reading of numerous published papers, something which I had little experience with before this project. With time I was able to efficiently parse the relevant information from these articles, which often have broad scope and contain sections irrelevant to my work. Extracting the important details from large bodies of text in this way is an important skill that will undoubtedly be useful in future work.

Beyond just extracting these details, I had to constantly relate them back to my own project to consider how they were relevant and if they would influence my approach. This was particularly challenging when reading papers relating to other fields, such as social choice and ranking systems. As a result of this I am now more confident in combining ideas from existing papers and keeping a mental map of information across the literature.

Reading published papers also improved my writing skills, as I became

more accustomed to the type of language and phrasing appropriate for technical writing.

In terms of software development, the project called for a fairly large-scale system to be designed, developed and tested. This contrasted with previous programming work I had done, which for the most part only involved writing code to satisfy fixed requirements. Aspects of the design process, such as considering relevant use cases and user interfaces, were not specific to truth discovery; this allowed me to develop general software design skills that will be useful in future projects.

Writing documentation is another general skill that the project provided ample opportunity to practise. In addition to documenting the code throughout development, I wrote a user guide explaining the concepts of the system and how it can be used. This required looking critically at the software as a whole and understanding how it can be broken down logically in a clear way.

The theoretical work was largely separate from the programming, and required very different skills and techniques. Transferable skills developed here include formulating mathematically rigorous definitions, theorems and proofs, mathematical writing, and laying out a mathematical paper. I also had to think carefully about the level of generality and the type of results desired; this involved many iterations before the final framework was decided on. I am better equipped to approach future theoretical work having gone through the process in this project.

Finally, this was a long-running individual project which required time management and organisational skills. In particular, I learned through trial and error what sort of time schedule works well for me personally; this will be carried forward to future projects in order to use my time as efficiently as possible. I also learned suitable methods for organising my workload, finding *Issues* on GitHub particularly useful, for example.¹

¹ <https://help.github.com/en/articles/about-issues>

References

- [1] Carlos E. Alchourrón, Peter Gärdenfors, and David Makinson. “On the Logic of Theory Change: Partial Meet Contraction and Revision Functions”. In: *The Journal of Symbolic Logic* 50.2 (1985), pp. 510–530. ISSN: 00224812. URL: <http://www.jstor.org/stable/2274239>.
- [2] Alon Altman and Moshe Tennenholtz. “An Axiomatic Approach to Personalized Ranking Systems”. In: *J. ACM* 57.4 (May 2010), 26:1–26:35. ISSN: 0004-5411. DOI: [10.1145/1734213.1734220](https://doi.org/10.1145/1734213.1734220). URL: <http://doi.acm.org/10.1145/1734213.1734220>.
- [3] Alon Altman and Moshe Tennenholtz. “Axiomatic Foundations for Ranking Systems”. In: *J. Artif. Int. Res.* 31.1 (Mar. 2008), pp. 473–495. ISSN: 1076-9757. URL: <http://dl.acm.org/citation.cfm?id=1622655.1622669>.
- [4] Alon Altman and Moshe Tennenholtz. “Ranking Systems: The PageRank Axioms”. In: *Proceedings of the 6th ACM Conference on Electronic Commerce. EC '05*. Vancouver, BC, Canada: ACM, 2005, pp. 1–8. ISBN: 1-59593-049-3. DOI: [10.1145/1064009.1064010](https://doi.org/10.1145/1064009.1064010). URL: <http://doi.acm.org/10.1145/1064009.1064010>.
- [5] Reid Andersen et al. “Trust-based Recommendation Systems: An Axiomatic Approach”. In: *Proceedings of the 17th International Conference on World Wide Web. WWW '08*. Beijing, China: ACM, 2008, pp. 199–208. ISBN: 978-1-60558-085-2. DOI: [10.1145/1367497.1367525](https://doi.org/10.1145/1367497.1367525). URL: <http://doi.acm.org/10.1145/1367497.1367525>.

- [6] Kenneth J. Arrow. “Social Choice and Individual Values”. In: *Ethics* 62.3 (1952), pp. 220–222.
- [7] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. “Abstract Argumentation Frameworks and Their Semantics”. In: *Handbook of Formal Argumentation*. Ed. by Pietro Baroni et al. College Publications, 2018. Chap. 4.
- [8] Jens Bleiholder and Felix Naumann. “Data Fusion”. In: *ACM Comput. Surv.* 41.1 (Jan. 2009), 1:1–1:41. ISSN: 0360-0300. DOI: [10.1145/1456650.1456651](https://doi.org/10.1145/1456650.1456651). URL: <http://doi.acm.org/10.1145/1456650.1456651>.
- [9] Richard Booth and Aaron Hunter. “Trust as a Precursor to Belief Revision”. In: *J. Artif. Intell. Res.* 61 (2018), pp. 699–722.
- [10] Felix Brandt et al. “Introduction to Computational Social Choice”. In: *Handbook of Computational Social Choice*. Ed. by Felix Brandt et al. 1st. New York, NY, USA: Cambridge University Press, 2016. Chap. 1.
- [11] Xin Luna Dong, Laure Berti-Equille, and Divesh Srivastava. “Truth Discovery and Copying Detection in a Dynamic World”. In: *Proc. VLDB Endow.* 2.1 (Aug. 2009), pp. 562–573. ISSN: 2150-8097. DOI: [10.14778/1687627.1687691](https://doi.org/10.14778/1687627.1687691). URL: <https://doi.org/10.14778/1687627.1687691>.
- [12] Y. Du et al. “Bayesian Co-Clustering Truth Discovery for Mobile Crowd Sensing Systems”. In: *IEEE Transactions on Industrial Informatics* (2019), pp. 1–1. ISSN: 1551-3203. DOI: [10.1109/TII.2019.2896287](https://doi.org/10.1109/TII.2019.2896287).
- [13] Ulle Endriss. “Judgment Aggregation”. In: *Handbook of Computational Social Choice*. Ed. by Felix Brandt et al. 1st. New York, NY, USA: Cambridge University Press, 2016. Chap. 17.
- [14] Alban Galland et al. “Corroborating Information from Disagreeing Views”. In: *Proceedings of the Third ACM International Conference on Web Search and Data Mining*. WSDM ’10. New York, New York, USA: ACM, 2010, pp. 131–140. ISBN: 978-1-60558-889-6. DOI: [10.1145/1718487.1718504](https://doi.org/10.1145/1718487.1718504). URL: <http://doi.acm.org/10.1145/1718487.1718504>.

- [15] Manish Gupta and Jiawei Han. “Heterogeneous Network-based Trust Analysis: A Survey”. In: *SIGKDD Explor. Newsl.* 13.1 (Aug. 2011), pp. 54–71. ISSN: 1931-0145. DOI: [10.1145/2031331.2031341](https://doi.org/10.1145/2031331.2031341). URL: <http://doi.acm.org/10.1145/2031331.2031341>.
- [16] Jon M. Kleinberg. “Authoritative Sources in a Hyperlinked Environment”. In: *J. ACM* 46.5 (Sept. 1999), pp. 604–632. ISSN: 0004-5411. DOI: [10.1145/324133.324140](https://doi.org/10.1145/324133.324140). URL: <http://doi.acm.org/10.1145/324133.324140>.
- [17] Justin Kruger et al. “Axiomatic Analysis of Aggregation Methods for Collective Annotation”. In: *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems. AAMAS '14*. Paris, France: International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 1185–1192. ISBN: 978-1-4503-2738-1. URL: <http://dl.acm.org/citation.cfm?id=2617388.2617437>.
- [18] Dalia Attia Waguih and Laure Berti-Equille. “Truth Discovery Algorithms: An Experimental Evaluation”. In: *CoRR* abs/1409.6428 (2014). URL: <http://arxiv.org/abs/1409.6428>.
- [19] Omer Lev and Moshe Tennenholtz. “Group Recommendations: Axioms, Impossibilities, and Random Walks”. In: *TARK*. 2017.
- [20] Xian Li et al. “Truth finding on the deep web: is the problem solved?” In: *Proceedings of the 39th international conference on Very Large Data Bases. PVLDB'13*. Trento, Italy: VLDB Endowment, 2013, pp. 97–108. URL: <http://dl.acm.org/citation.cfm?id=2448936.2448943>.
- [21] Yaliang Li et al. “A Survey on Truth Discovery”. In: *SIGKDD Explor. Newsl.* 17.2 (Feb. 2016), pp. 1–16. ISSN: 1931-0145. DOI: [10.1145/2897350.2897352](https://doi.org/10.1145/2897350.2897352). URL: <http://doi.acm.org/10.1145/2897350.2897352>.
- [22] Yaliang Li et al. “Conflicts to Harmony: A Framework for Resolving Conflicts in Heterogeneous Data by Truth Discovery”. In: *IEEE Transactions on Knowledge and Data Engineering* 28.8 (Aug. 2016), pp. 1986–1999. ISSN: 1041-4347. DOI: [10.1109/TKDE.2016.2559481](https://doi.org/10.1109/TKDE.2016.2559481).
- [23] Stephen Marsh. “Formalising Trust as a Computational Concept”. PhD thesis. University of Stirling, 1994.

- [24] Mohammad Momani and Subhash Challa. “Survey of trust models in different network domains”. In: *CoRR* abs/1010.0168 (2010).
- [25] Lawrence Page et al. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report 1999-66. Previous number = SIDL-WP-1999-0120. Stanford InfoLab, Nov. 1999. URL: <http://ilpubs.stanford.edu:8090/422/>.
- [26] Jeff Pasternack. “Knowing Who to Trust and What to Believe in the Presence of Conflicting Information”. PhD thesis. University of Illinois at Urbana-Champaign, 2011. URL: <http://hdl.handle.net/2142/29516>.
- [27] Jeff Pasternack and Dan Roth. “Knowing What to Believe (when You Already Know Something)”. In: *Proceedings of the 23rd International Conference on Computational Linguistics*. COLING ’10. Beijing, China: Association for Computational Linguistics, 2010, pp. 877–885. URL: <http://dl.acm.org/citation.cfm?id=1873781.1873880>.
- [28] Gärdenfors Peter. *Belief Revision*. Cambridge Univserity Press, 2003.
- [29] Yuqing Tang et al. “Using argumentation to reason about trust and belief”. In: *Journal of Logic and Computation* 22.5 (Oct. 2012), pp. 979–1018. ISSN: 0955-792X. DOI: [10.1093/logcom/exr038](https://doi.org/10.1093/logcom/exr038).
- [30] *Unified Modeling Language (Version 2.5)*. Mar. 2015. URL: <http://www.omg.org/spec/UML/2.5>.
- [31] Soroush Vosoughi, Deb Roy, and Sinan Aral. “The spread of true and false news online”. In: *Science* 359.6380 (2018), pp. 1146–1151. ISSN: 0036-8075. DOI: [10.1126/science.aap9559](https://doi.org/10.1126/science.aap9559). eprint: <https://science.sciencemag.org/content/359/6380/1146.full.pdf>. URL: <https://science.sciencemag.org/content/359/6380/1146>.
- [32] Houping Xiao. “Multi-sourced Information Trustworthiness Analysis: Applications and Theory”. PhD thesis. University at Buffalo, State University of New York, 2018.

- [33] Houping Xiao et al. “A Truth Discovery Approach with Theoretical Guarantee”. In: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, 2016, pp. 1925–1934. ISBN: 978-1-4503-4232-2. DOI: [10.1145/2939672.2939816](https://doi.org/10.1145/2939672.2939816). URL: <http://doi.acm.org/10.1145/2939672.2939816>.
- [34] Yi Yang, Quan Bai, and Qing Liu. “A probabilistic model for truth discovery with object correlations”. In: *Knowledge-Based Systems* 165 (2019), pp. 360–373. ISSN: 0950-7051. DOI: <https://doi.org/10.1016/j.knosys.2018.12.004>. URL: <http://www.sciencedirect.com/science/article/pii/S0950705118305914>.
- [35] Xiaoxin Yin, Jiawei Han, and Philip S. Yu. “Truth Discovery with Multiple Conflicting Information Providers on the Web”. In: *IEEE Transactions on Knowledge and Data Engineering* 20.6 (June 2008), pp. 796–808. ISSN: 1041-4347. DOI: [10.1109/TKDE.2007.190745](https://doi.org/10.1109/TKDE.2007.190745).
- [36] Xiaoxin Yin and Wenzhao Tan. “Semi-supervised Truth Discovery”. In: *Proceedings of the 20th International Conference on World Wide Web*. WWW '11. Hyderabad, India: ACM, 2011, pp. 217–226. ISBN: 978-1-4503-0632-4. DOI: [10.1145/1963405.1963439](https://doi.org/10.1145/1963405.1963439). URL: <http://doi.acm.org/10.1145/1963405.1963439>.
- [37] Daniel Yue Zhang et al. “On robust truth discovery in sparse social media sensing”. In: *2016 IEEE International Conference on Big Data (Big Data)*. Dec. 2016, pp. 1076–1081. DOI: [10.1109/BigData.2016.7840710](https://doi.org/10.1109/BigData.2016.7840710).
- [38] Liyan Zhang et al. “Latent Dirichlet Truth Discovery: Separating Trustworthy and Untrustworthy Components in Data Sources”. In: *IEEE Access* 6 (2018), pp. 1741–1752. ISSN: 2169-3536. DOI: [10.1109/ACCESS.2017.2780182](https://doi.org/10.1109/ACCESS.2017.2780182).
- [39] Zhou Zhao, James Cheng, and Wilfred Ng. “Truth Discovery in Data Streams: A Single-Pass Probabilistic Approach”. In: *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. CIKM '14. Shanghai, China: ACM, 2014, pp. 1589–1598. ISBN: 978-1-4503-2598-1. DOI: [10.1145/2661829.2661892](https://doi.org/10.1145/2661829.2661892). URL: <http://doi.acm.org/10.1145/2661829.2661892>.

- [40] Shi Zhi et al. “Modeling Truth Existence in Truth Discovery”. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '15. Sydney, NSW, Australia: ACM, 2015, pp. 1543–1552. ISBN: 978-1-4503-3664-2. DOI: [10.1145/2783258.2783339](https://doi.org/10.1145/2783258.2783339). URL: <http://doi.acm.org/10.1145/2783258.2783339>.
- [41] William S. Zwicker. “Introduction to the Theory of Voting”. In: *Handbook of Computational Social Choice*. Ed. by Felix Brandt et al. 1st. New York, NY, USA: Cambridge University Press, 2016. Chap. 2.

Appendix A

Web-interface Screenshots

Truth discovery tool


Algorithm(s):

- AverageLog
- Investment
- PooledInvestment
- Sums
- TruthFinder
- MajorityVoting

Hold Ctrl/Cmd to select multiple algorithms

Dataset:

	Variable 1	Variable 2	Variable 3	Variable 4
Source 1	1	-	3	4
Source 2	2	2	-	-
Source 3	-	-	7	-
Source 4	1	2	5	-

[Add source](#) [Add Variable](#) [Load from CSV](#) 

OR

Choose a preset dataset:

[Load](#)

Compare against previous results

Show advanced options

[Run](#)

Figure A.1: Basic view of the input form in the web interface.

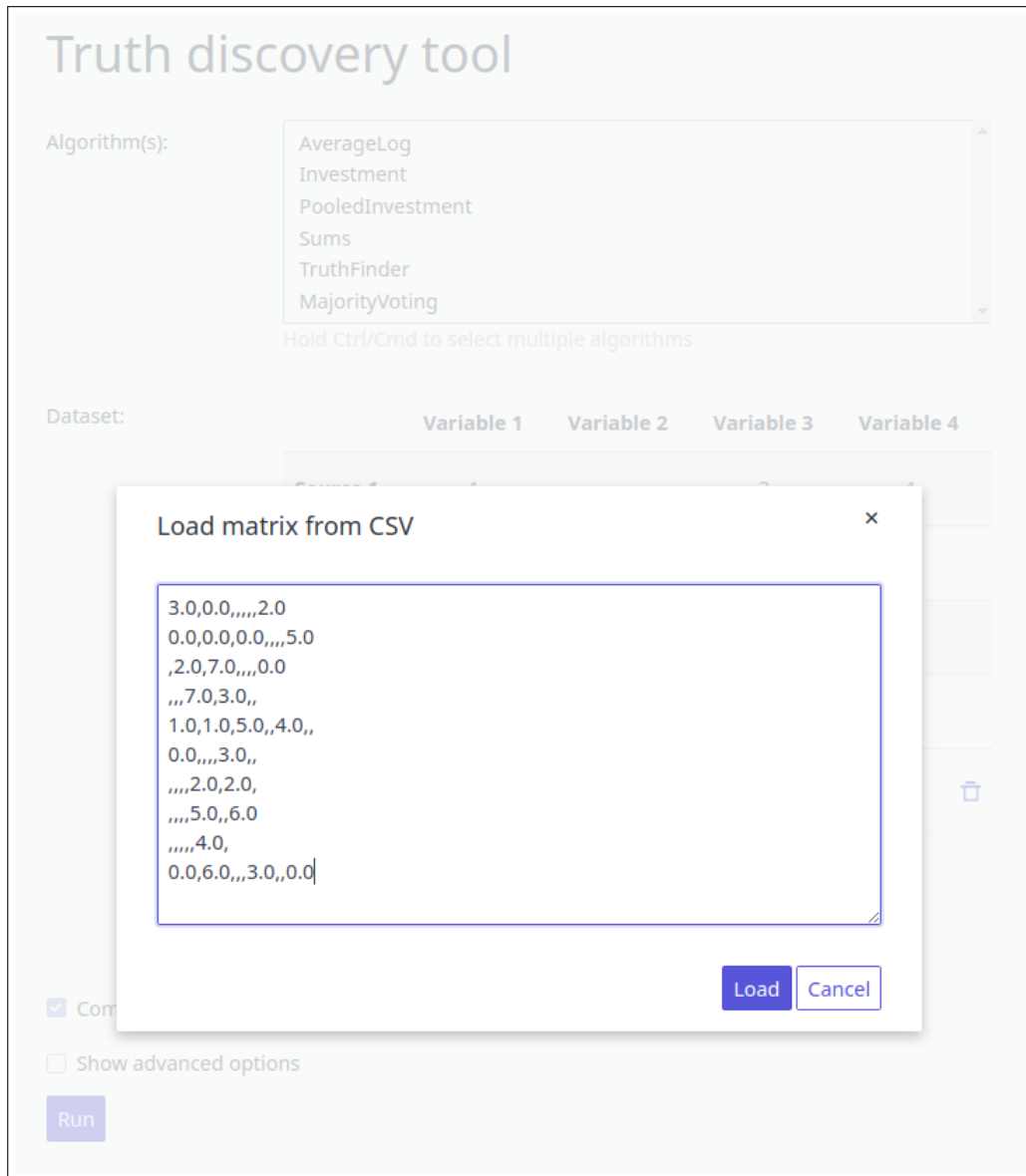


Figure A.2: CSV dataset entry in the web interface.

Show advanced options

Iteration method: Fixed number of iterations
 Until convergence
Iterate until the distance between successive trust scores becomes less than the given threshold

Distance measure: Threshold:

Algorithm parameters:

Algorithm-specific parameters in the form `key=value` (one per line)

Figure A.3: Advanced options for algorithm parameters in the web interface.

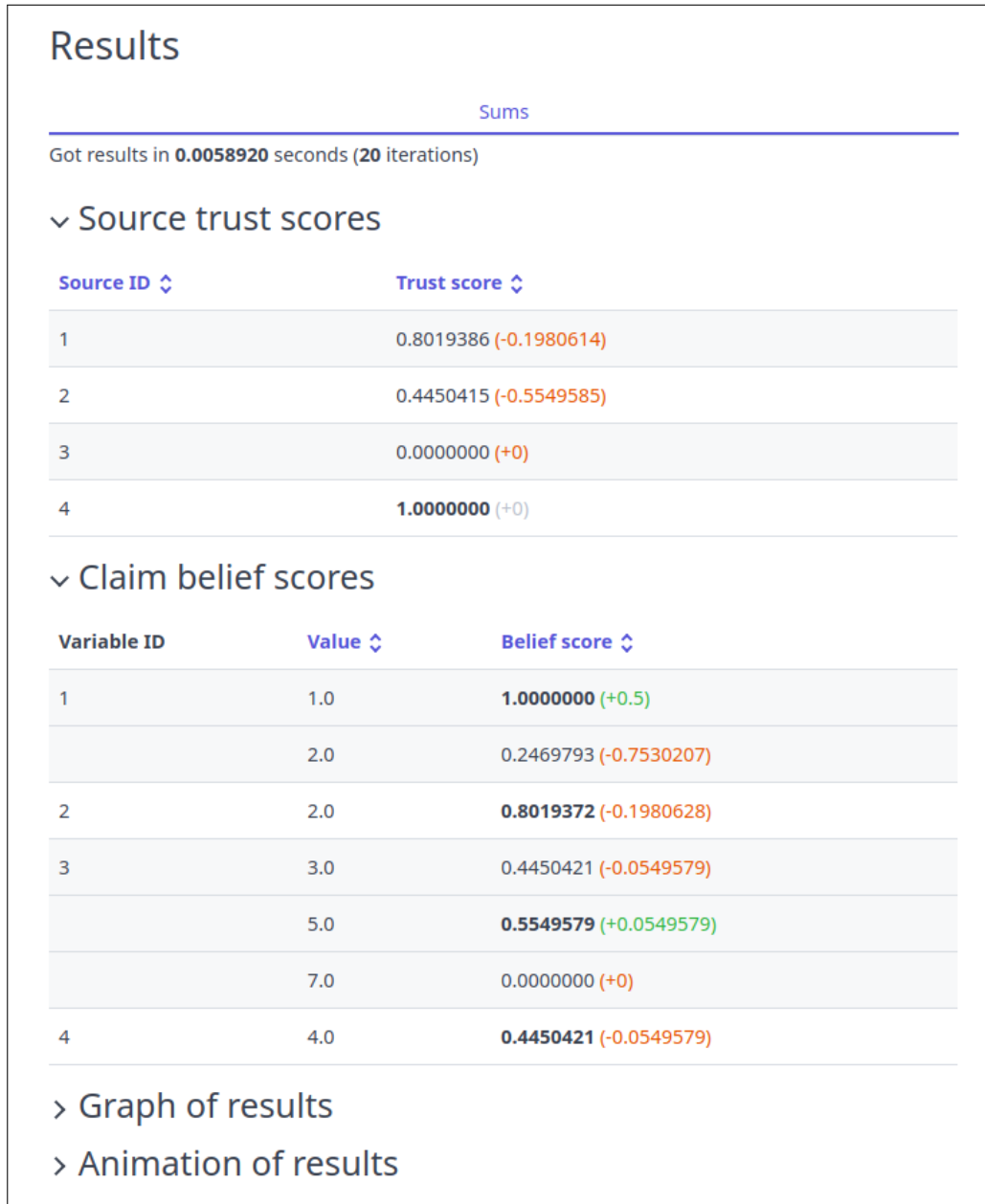


Figure A.4: Example results of a single algorithm run via the web interface.

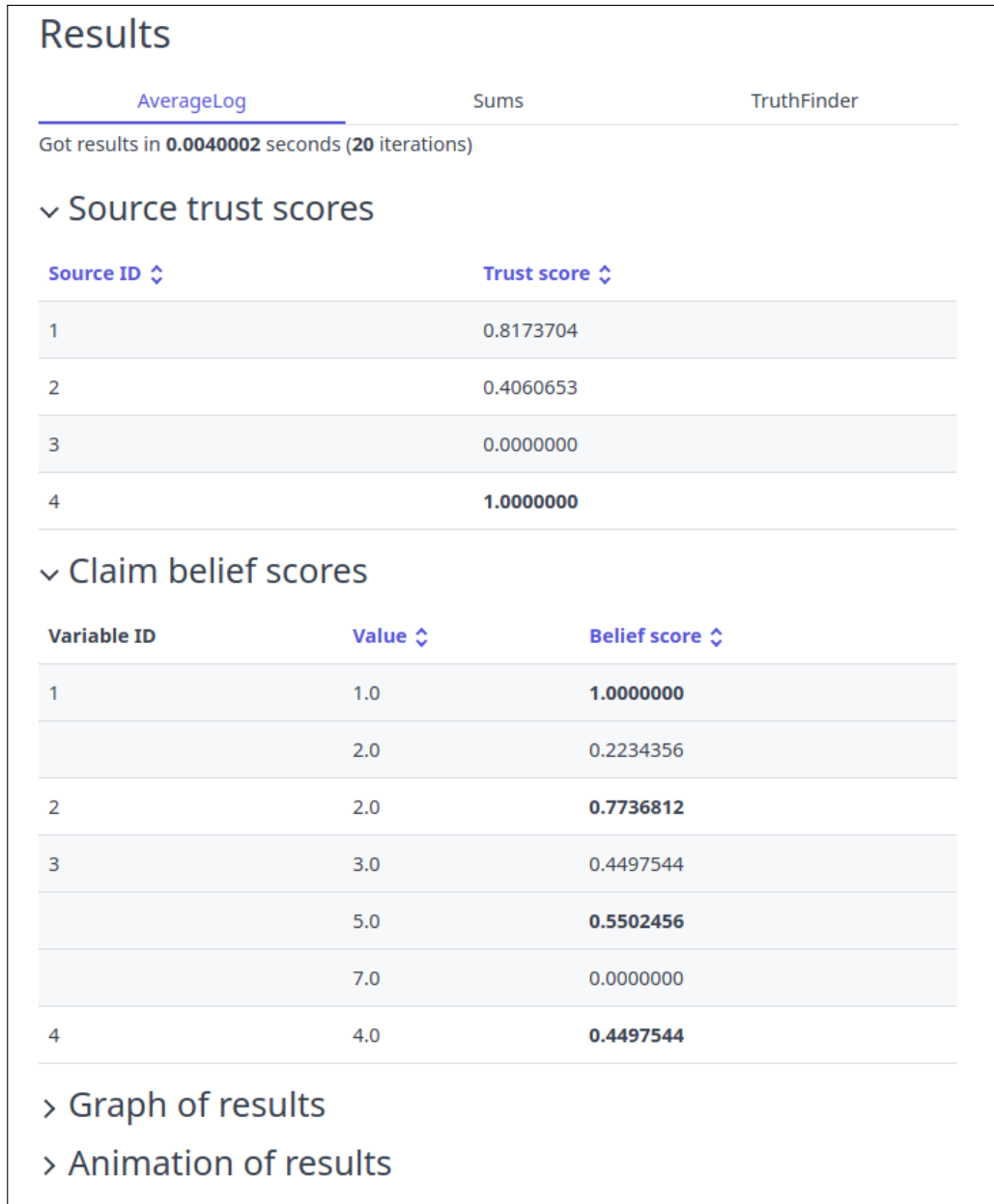


Figure A.5: Example results of several algorithms run simultaneously via the web interface.

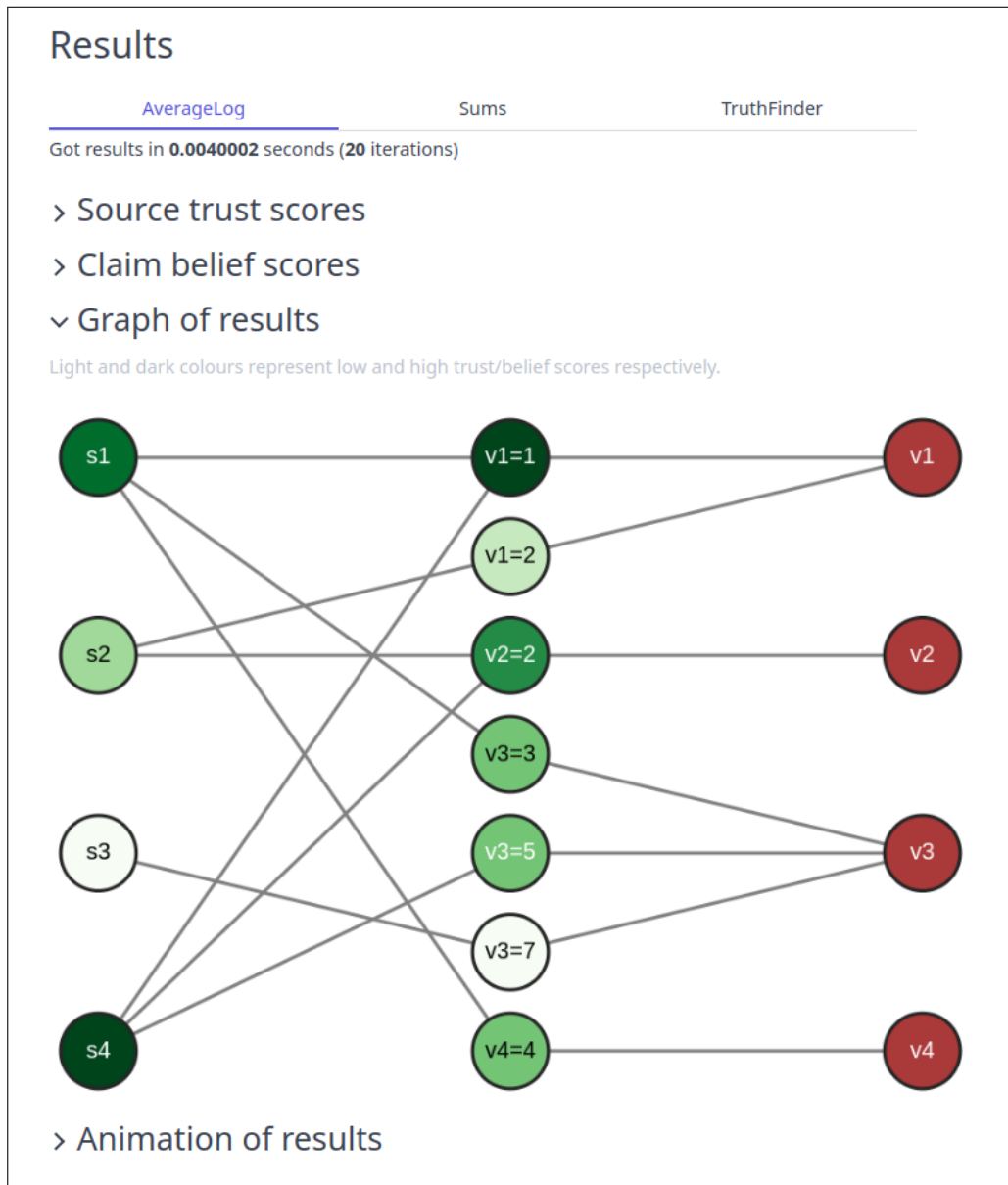


Figure A.6: Graph representation of results of a truth discovery algorithm, presented in the web interface.

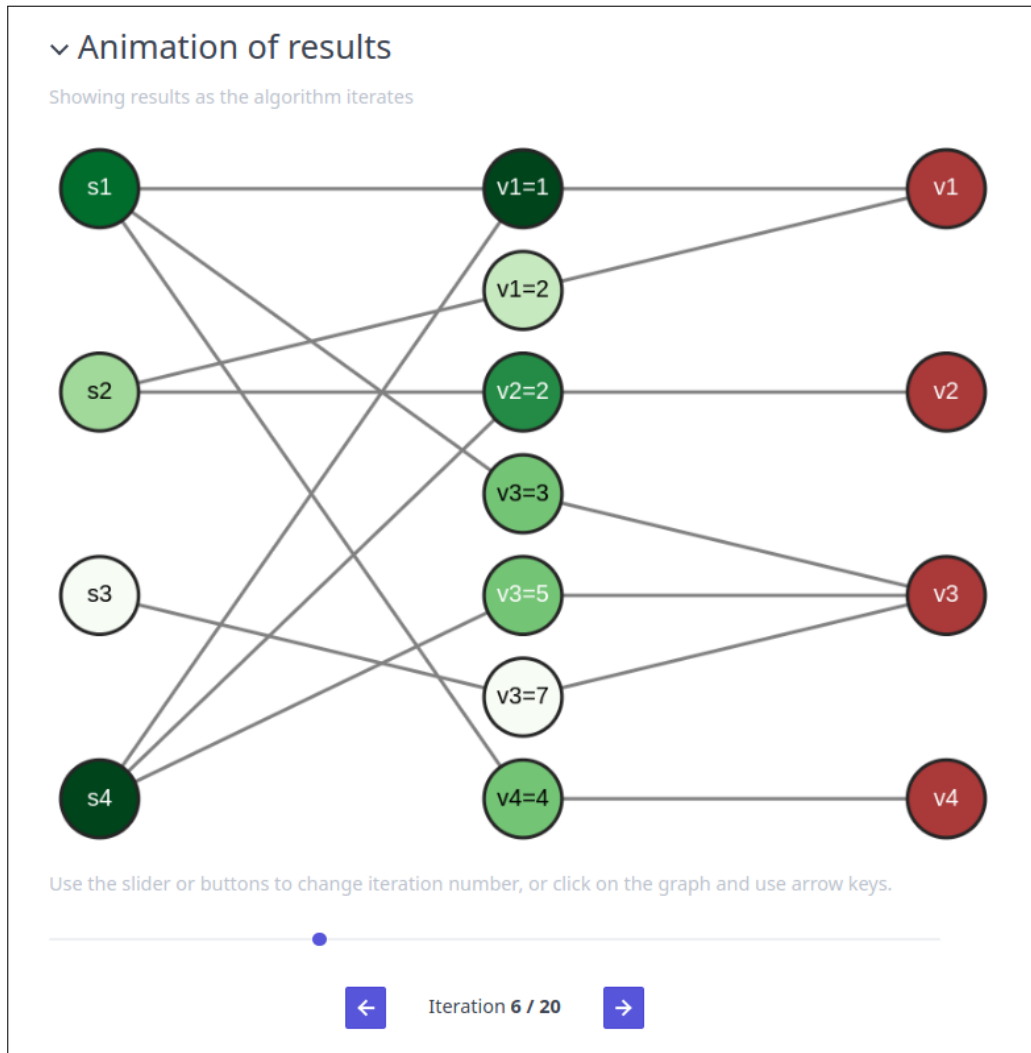


Figure A.7: Frame from an animation of the results of an algorithm as it iterates, presented in the web interface.

Appendix B

Test Results

```
===== test session starts
=====
platform linux -- Python 3.6.7, pytest-4.2.0, py-1.7.0, pluggy-0.8.1 -- /home/joe/
documents/uni/project/code/venv/bin/python3
cachedir: .pytest_cache
rootdir: /home/joe/documents/uni/project/code, inifile:
collecting ... collecting 31 items
collected 150 items

truthdiscovery/test/test_algorithms.py::TestBase::test_empty_dataset PASSED [ 0%]
truthdiscovery/test/test_algorithms.py::TestBase::test_get_parameter_names PASSED [ 1%]
truthdiscovery/test/test_algorithms.py::TestVoting::test_basic PASSED [ 2%]
truthdiscovery/test/test_algorithms.py::TestBaseIterative::test_run_fail PASSED [ 2%]
truthdiscovery/test/test_algorithms.py::TestBaseIterative::test_fixed_priors PASSED [ 3%]
truthdiscovery/test/test_algorithms.py::TestBaseIterative::test_voted_priors PASSED [ 4%]
truthdiscovery/test/test_algorithms.py::TestBaseIterative::test_uniform_priors PASSED [
 4%]
truthdiscovery/test/test_algorithms.py::TestBaseIterative::test_invalid_priors PASSED [
 5%]
truthdiscovery/test/test_algorithms.py::TestSums::test_basic PASSED [ 6%]
truthdiscovery/test/test_algorithms.py::TestAverageLog::test_basic PASSED [ 6%]
truthdiscovery/test/test_algorithms.py::TestInvestment::test_basic PASSED [ 7%]
truthdiscovery/test/test_algorithms.py::TestInvestment::test_converge_to_zero PASSED [ 8%]
truthdiscovery/test/test_algorithms.py::TestPooledInvestment::test_basic PASSED [ 8%]
truthdiscovery/test/test_algorithms.py::TestTruthFinder::test_basic PASSED [ 9%]
truthdiscovery/test/test_algorithms.py::TestTruthFinder::test_no_implications PASSED [
10%]
truthdiscovery/test/test_algorithms.py::TestTruthFinder::test_trust_invalid PASSED [ 10%]
truthdiscovery/test/test_algorithms.py::TestOnLargeData::test_sums PASSED [ 11%]
truthdiscovery/test/test_algorithms.py::TestOnLargeData::test_average_log PASSED [ 12%]
truthdiscovery/test/test_algorithms.py::TestOnLargeData::test_investment PASSED [ 12%]
```

```
truthdiscovery/test/test_algorithms.py::TestOnLargeData::test_pooled_investment PASSED [
 13%]
truthdiscovery/test/test_algorithms.py::TestOnLargeData::test_truthfinder PASSED [ 14%]
truthdiscovery/test/test_algorithms.py::TestOnLargeData::test_voting PASSED [ 14%]
truthdiscovery/test/test_algorithms.py::TestIteratorsForAlgorithms::
  test_default_iterator_types PASSED [ 15%]
truthdiscovery/test/test_algorithms.py::TestLoggingAlgorithm::test_final_result PASSED [
 16%]
truthdiscovery/test/test_algorithms.py::TestLoggingAlgorithm::test_no_logging PASSED [
 16%]
truthdiscovery/test/test_algorithms.py::TestLoggingAlgorithm::test_partial_results PASSED
 [ 17%]
truthdiscovery/test/test_algorithms.py::TestLoggingAlgorithm::test_sums_detailed PASSED [
 18%]
truthdiscovery/test/test_clients.py::TestBaseClient::test_get_iterator PASSED [ 18%]
truthdiscovery/test/test_clients.py::TestBaseClient::test_get_algorithm_parameter PASSED [
 19%]
truthdiscovery/test/test_clients.py::TestBaseClient::test_get_output_obj PASSED [ 20%]
truthdiscovery/test/test_clients.py::TestBaseClient::test_get_algorithm_params PASSED [
 20%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_no_commands PASSED [ 21%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_basic PASSED [ 22%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_results PASSED [ 22%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_multiple_parameters
 PASSED [ 23%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_multiple_algorithms
 PASSED [ 24%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_get_algorithm_instance
 PASSED [ 24%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_set_prior_belief PASSED [
 25%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_set_iterator PASSED [
 26%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::
  test_invalid_algorithm_parameter PASSED [ 26%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_invalid_algorithm PASSED
 [ 27%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_filter_sources_variables
 PASSED [ 28%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_default_output PASSED [
 28%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_custom_output PASSED [
 29%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_show_most_believed_values
 PASSED [ 30%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_belief_stats PASSED [
 30%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_synthetic_generation
 PASSED [ 31%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::
  test_synthetic_generation_claim_prob_1 PASSED [ 32%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::
  test_synthetic_generation_source_trust_1 PASSED [ 32%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::
  test_synthetic_generation_invalid_params PASSED [ 33%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::
  test_supervised_dataset_and_accuracy PASSED [ 34%]
```

```
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_accuracy_not_supervised
PASSED [ 34%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_accuracy_undefined PASSED
[ 35%]
truthdiscovery/test/test_clients.py::TestCommandLineClient::test_graph_generation PASSED [
36%]
truthdiscovery/test/test_clients.py::TestWebClient::test_routing PASSED [ 36%]
truthdiscovery/test/test_clients.py::TestWebClient::test_home PASSED [ 37%]
truthdiscovery/test/test_clients.py::TestWebClient::test_run_fail PASSED [ 38%]
truthdiscovery/test/test_clients.py::TestWebClient::test_run_success PASSED [ 38%]
truthdiscovery/test/test_clients.py::TestWebClient::test_run_multiple_algorithms PASSED [
39%]
truthdiscovery/test/test_clients.py::TestWebClient::test_results_diff PASSED [ 40%]
truthdiscovery/test/test_clients.py::TestWebClient::
test_results_diff_invalid_previous_results PASSED [ 40%]
truthdiscovery/test/test_clients.py::TestWebClient::test_get_json_graph PASSED [ 41%]
truthdiscovery/test/test_clients.py::TestWebClient::test_get_json_animated_gif PASSED [
42%]
truthdiscovery/test/test_clients.py::TestWebClient::test_voting_imagery PASSED [ 42%]
truthdiscovery/test/test_clients.py::TestWebClient::test_empty_dataset PASSED [ 43%]
truthdiscovery/test/test_clients.py::TestWebClient::test_non_convergence PASSED [ 44%]
truthdiscovery/test/test_codestyle.py::TestCodeStyle::test_pep8 PASSED [ 44%]
truthdiscovery/test/test_drawing.py::TestRendering::test_entity_positioning PASSED [ 45%]
truthdiscovery/test/test_drawing.py::TestRendering::test_node_ordering PASSED [ 46%]
truthdiscovery/test/test_drawing.py::TestRendering::test_single_source PASSED [ 46%]
truthdiscovery/test/test_drawing.py::TestRendering::test_invalid_node_size PASSED [ 47%]
truthdiscovery/test/test_drawing.py::TestRendering::test_no_horizontal_overlapping PASSED
[ 48%]
truthdiscovery/test/test_drawing.py::TestRendering::test_png_is_default PASSED [ 48%]
truthdiscovery/test/test_drawing.py::TestRendering::test_long_labels PASSED [ 49%]
truthdiscovery/test/test_drawing.py::TestRendering::test_matrix_renderer PASSED [ 50%]
truthdiscovery/test/test_drawing.py::TestRendering::test_image_size PASSED [ 50%]
truthdiscovery/test/test_drawing.py::TestRendering::test_progress_bar PASSED [ 51%]
truthdiscovery/test/test_drawing.py::TestBackends::test_base_backend PASSED [ 52%]
truthdiscovery/test/test_drawing.py::TestBackends::test_valid_png PASSED [ 52%]
truthdiscovery/test/test_drawing.py::TestBackends::test_results_based_valid_png PASSED [
53%]
truthdiscovery/test/test_drawing.py::TestBackends::test_json_backend PASSED [ 54%]
truthdiscovery/test/test_drawing.py::TestBackends::test_json_backend_entity_serialisation
PASSED [ 54%]
truthdiscovery/test/test_drawing.py::TestColourSchemes::test_get_gradient_colour PASSED [
55%]
truthdiscovery/test/test_drawing.py::TestColourSchemes::test_plain_colour_scheme PASSED [
56%]
truthdiscovery/test/test_drawing.py::TestColourSchemes::test_results_based_colours PASSED
[ 56%]
truthdiscovery/test/test_drawing.py::TestAnimations::test_base PASSED [ 57%]
truthdiscovery/test/test_drawing.py::TestAnimations::test_gif_animation PASSED [ 58%]
truthdiscovery/test/test_drawing.py::TestAnimations::test_json_animation PASSED [ 58%]
truthdiscovery/test/test_drawing.py::TestAnimations::test_renderer PASSED [ 59%]
truthdiscovery/test/test_drawing.py::TestAnimations::test_fps PASSED [ 60%]
truthdiscovery/test/test_drawing.py::TestAnimations::test_invalid_backend PASSED [ 60%]
truthdiscovery/test/test_drawing.py::TestAnimations::test_default_backend PASSED [ 61%]
truthdiscovery/test/test_drawing.py::TestAnimations::test_progress_bar PASSED [ 62%]
truthdiscovery/test/test_input.py::TestDataset::test_num_sources_variables_claims PASSED [
62%]
truthdiscovery/test/test_input.py::TestDataset::test_claims_matrix PASSED [ 63%]
```



```
truthdiscovery/test/test_input.py::TestDataset::test_mut_ex_matrix PASSED [ 64%]
truthdiscovery/test/test_input.py::TestDataset::
  test_source_multiple_claims_for_a_single_variable PASSED [ 64%]
truthdiscovery/test/test_input.py::TestIDMapping::test_insert PASSED [ 65%]
truthdiscovery/test/test_input.py::TestMatrixDataset::test_create PASSED [ 66%]
truthdiscovery/test/test_input.py::TestMatrixDataset::test_num_sources_variables_claims
  PASSED [ 66%]
truthdiscovery/test/test_input.py::TestMatrixDataset::test_dimension PASSED [ 67%]
truthdiscovery/test/test_input.py::TestMatrixDataset::test_from_csv PASSED [ 68%]
truthdiscovery/test/test_input.py::TestMatrixDataset::test_invalid_csv_shape PASSED [ 68%]
truthdiscovery/test/test_input.py::TestMatrixDataset::test_from_csv_empty_rows PASSED [
  69%]
truthdiscovery/test/test_input.py::TestMatrixDataset::test_from_csv_single_row_or_column
  PASSED [ 70%]
truthdiscovery/test/test_input.py::TestMatrixDataset::test_claims_matrix PASSED [ 70%]
truthdiscovery/test/test_input.py::TestMatrixDataset::test_mutual_exclusion_matrix PASSED
  [ 71%]
truthdiscovery/test/test_input.py::TestMatrixDataset::test_export_to_csv PASSED [ 72%]
truthdiscovery/test/test_input.py::TestSupervisedData::test_from_csv PASSED [ 72%]
truthdiscovery/test/test_input.py::TestSupervisedData::test_accuracy PASSED [ 73%]
truthdiscovery/test/test_input.py::TestSupervisedData::test_unknown_variable PASSED [ 74%]
truthdiscovery/test/test_input.py::TestSupervisedData::test_no_true_values_known PASSED [
  74%]
truthdiscovery/test/test_input.py::TestSupervisedData::test_not_enough_claimed_values
  PASSED [ 75%]
truthdiscovery/test/test_input.py::TestSyntheticData::test_trust_as_list PASSED [ 76%]
truthdiscovery/test/test_input.py::TestSyntheticData::test_invalid_trust_vector_shape
  PASSED [ 76%]
truthdiscovery/test/test_input.py::TestSyntheticData::test_trust_range_error PASSED [ 77%]
truthdiscovery/test/test_input.py::TestSyntheticData::test_valid_trusts PASSED [ 78%]
truthdiscovery/test/test_input.py::TestSyntheticData::test_claim_probability PASSED [ 78%]
truthdiscovery/test/test_input.py::TestSyntheticData::test_invalid_claim_probability
  PASSED [ 79%]
truthdiscovery/test/test_input.py::TestSyntheticData::test_domain_size PASSED [ 80%]
truthdiscovery/test/test_input.py::TestSyntheticData::test_export_to_csv PASSED [ 80%]
truthdiscovery/test/test_input.py::TestImplications::test_implications PASSED [ 81%]
truthdiscovery/test/test_input.py::TestImplications::test_invalid_implication_values
  PASSED [ 82%]
truthdiscovery/test/test_input.py::TestFileDataset::test_base PASSED [ 82%]
truthdiscovery/test/test_input.py::TestFileDataset::test_basic PASSED [ 83%]
truthdiscovery/test/test_input.py::TestFileDataset::test_implications PASSED [ 84%]
truthdiscovery/test/test_input.py::TestFileSupervisedData::test_base PASSED [ 84%]
truthdiscovery/test/test_input.py::TestFileSupervisedData::test_basic PASSED [ 85%]
truthdiscovery/test/test_iterators.py::TestBaseIterator::test_run_base_fail PASSED [ 86%]
truthdiscovery/test/test_iterators.py::TestBaseIterator::test_it_count PASSED [ 86%]
truthdiscovery/test/test_iterators.py::TestBaseIterator::test_reset PASSED [ 87%]
truthdiscovery/test/test_iterators.py::TestFixedIterator::test_invalid_limit PASSED [ 88%]
truthdiscovery/test/test_iterators.py::TestFixedIterator::test_finish_condition PASSED [
  88%]
truthdiscovery/test/test_iterators.py::TestConvergenceIterator::test_basic PASSED [ 89%]
truthdiscovery/test/test_iterators.py::TestConvergenceIterator::
  test_invalid_distance_measures PASSED [ 90%]
truthdiscovery/test/test_iterators.py::TestConvergenceIterator::test_did_not_converge
  PASSED [ 90%]
truthdiscovery/test/test_iterators.py::TestDistanceMeasures::test_l1 PASSED [ 91%]
truthdiscovery/test/test_iterators.py::TestDistanceMeasures::test_l2 PASSED [ 92%]
truthdiscovery/test/test_iterators.py::TestDistanceMeasures::test_l_inf PASSED [ 92%]
```

```
truthdiscovery/test/test_iterators.py::TestDistanceMeasures::test_cosine PASSED [ 93%]
truthdiscovery/test/test_result.py::TestResult::test_most_believed_values PASSED [ 94%]
truthdiscovery/test/test_result.py::TestResult::test_num_iterations PASSED [ 94%]
truthdiscovery/test/test_result.py::TestResult::test_time_taken PASSED [ 95%]
truthdiscovery/test/test_result.py::TestResult::test_filter_result PASSED [ 96%]
truthdiscovery/test/test_result.py::TestResult::test_stats PASSED [ 96%]
truthdiscovery/test/test_result.py::TestResultDiff::test_no_common_sources_or_vars PASSED
[ 97%]
truthdiscovery/test/test_result.py::TestResultDiff::test_common_sources PASSED [ 98%]
truthdiscovery/test/test_result.py::TestResultDiff::test_common_vars_but_no_common_values
PASSED [ 98%]
truthdiscovery/test/test_result.py::TestResultDiff::test_common_var_values PASSED [ 99%]
truthdiscovery/test/test_result.py::TestResultDiff::test_no_iteration_info PASSED [100%]

===== 150 passed in 9.57 seconds
=====
```

Appendix C

Proofs

C.1 Proof of proposition 1

Proof. The ‘only if’ direction is clear. For the converse, suppose T is source, fact and object symmetric, and let $N, N' = \pi(N)$ be equivalent networks. Define

$$\pi_{\mathcal{S}}(x) = \begin{cases} \pi(x) & \text{if } x \in \mathcal{S} \\ x & \text{if } x \in \mathcal{F} \cup \mathcal{O} \end{cases}$$

Define $\pi_{\mathcal{F}}, \pi_{\mathcal{O}}$ similarly, and set $\sigma = \pi_{\mathcal{S}} \circ \pi_{\mathcal{F}} \circ \pi_{\mathcal{O}}$. Then for $s \in \mathcal{S}$,

$$\begin{aligned} \sigma(s) &= \pi_{\mathcal{S}}(\pi_{\mathcal{F}}(\pi_{\mathcal{O}}(s))) \\ &= \pi_{\mathcal{S}}(\pi_{\mathcal{F}}(s)) \\ &= \pi_{\mathcal{S}}(s) \\ &= \pi(s) \end{aligned}$$

Similarly $\sigma(f) = \pi(f)$ and $\sigma(o) = \pi(o)$ for $f \in \mathcal{F}$ and $o \in \mathcal{O}$. Hence $\sigma = \pi$.

Let $s_1, s_2 \in \mathcal{S}$. Since $\pi_{\mathcal{O}}$ only permutes objects, we may apply object-symmetry to get

$$s_1 \sqsubseteq_N^T s_2 \iff \pi_{\mathcal{O}}(s_1) \sqsubseteq_{\pi_{\mathcal{O}}(N)}^T \pi_{\mathcal{O}}(s_2)$$

Then since $\pi_{\mathcal{F}}$ only permutes facts and $\pi_{\mathcal{S}}$ only permutes sources, we may successively apply fact and object symmetry to the right hand side to get

$$\begin{aligned} s_1 \sqsubseteq_N^T s_2 &\iff \sigma(s_1) \sqsubseteq_{\sigma(N)}^T \sigma(s_2) \\ &\iff \pi(s_1) \sqsubseteq_{\pi(N)}^T \pi(s_2) \\ &\iff \pi(s_1) \sqsubseteq_{N'}^T \pi(s_2) \end{aligned}$$

An identical argument for fact ranking gives $f_1 \preceq_N^T f_2 \iff \pi(f_1) \preceq_{N'}^T f_2$. Hence T is symmetric. \square

C.2 Proof of proposition 2

Proof. For the first claim, consider the permutation $\pi = (s_1, s_2)$. We claim that $\pi(N) = N$. Let E be the edges in N and $\pi(E)$ be the edges in $\pi(N)$. For any $(s, f) \in \mathcal{S} \times \mathcal{F}$ we have three cases:

1. $s \notin \{s_1, s_2\}$: in this case $(\pi(s), \pi(f)) = (s, f)$ and $(s, f) \in E$ iff $(\pi(s), \pi(f)) \in \pi(E)$ by definition, so $(s, f) \in E$ iff $(s, f) \in \pi(E)$.
2. $s = s_1$: Here we have $(s_1, f) \in E$ iff $(s_2, f) \in E$ by hypothesis. This is equivalent to $(\pi(s_2), \pi(f)) \in \pi(E)$ by definition of $\pi(E)$, which by definition of π is equivalent to $(s_1, f) \in \pi(E)$.
3. $s = s_2$: As above, $(s_2, f) \in E$ iff $(s_2, f) \in \pi(E)$

Also, it is clear that $(f, o) \in E$ iff $(f, o) \in \pi(E)$ for $f \in \mathcal{F}$, $o \in \mathcal{O}$. We have shown that $(x, y) \in E$ iff $(x, y) \in \pi(E)$, i.e. $\pi(E) = E$ and thus $\pi(N) = N$. Applying source-symmetry, this gives

$$\begin{aligned} s_1 \sqsubseteq_N^T s_2 &\iff \pi(s_1) \sqsubseteq_{\pi(N)}^T \pi(s_2) \\ &\iff s_2 \sqsubseteq_N^T s_1 \end{aligned}$$

i.e. $s_1 \simeq_N^T s_2$.

For the second claim, consider $\sigma = (f_1, f_2)$ and apply a similar argument to the above. Note that we require f_1 and f_2 to be associated with the same object here so that both facts have the same incoming and outgoing edges in N . \square

C.3 Proof of proposition 3

Proof. Suppose T is source-symmetric and a dictatorship with dictator s^* . Let $N = (V, E)$ be a truth discovery network.

1. Let $s_1, s_2 \in \mathcal{S}$. Without loss of generality $s_1 \sqsubseteq_N^T s_2$, since \sqsubseteq_N^T is complete. Consider the permutation $\pi = (s_1, s^*)$. We have $s_2 \sqsubseteq_{\pi(N)}^T s^*$ by dictatorship in $\pi(N)$, which by symmetry means $\pi^{-1}(s_2) \sqsubseteq_N^T \pi^{-1}(s^*)$, i.e. $s_2 \sqsubseteq_N^T s_1$. Hence $s_1 \simeq_N^T s_2$ as required.
2. Let $f_1, f_2 \in \mathcal{F}$ such that there is a source s with $(s, f_1) \in E$. Set $\sigma = (s, s^*)$. Then $\sigma(E)$ contains $(\sigma(s), \sigma(f_1)) = (s^*, f_1)$, so we have $\sigma(f_2) \preceq_{\sigma(N)}^T f_1 = \sigma(f_1)$ since the facts claimed by s^* rank above all others. Applying symmetry gives $f_2 \preceq_N^T f_1$ as required.

This implies there is no source-symmetric strict dictatorship. If T were such an operator then T is also a non-strict dictatorship, so symmetry implies all sources are ranked equally, but this contradicts $s \sqsubseteq_N^T s^*$ for all $s \neq s^*$. □

C.4 Proof of proposition 4

Proof. The operator T defined in example 2 is not a dictatorship but is a binary generalised dictatorship. Indeed, it is clearly a binary generalised dictatorship from the definition. To show it is not a dictatorship, it is sufficient by symmetry and proposition 3 to find a network in which T does not rank all sources equally. A suitable example is a network N where a source s claims two facts and all other sources claim only one fact. Then $Q_N = \{s\}$, so s is ranked strictly above all other sources.

For an operator that is a dictatorship but not a binary generalised dictatorship, fix two distinct sources $s^*, s^{**} \in \mathcal{S}$ and consider the numerical

operator T' defined by

$$t_N(s) = \begin{cases} 2 & \text{if } s = s^* \\ 1 & \text{if } s = s^{**} \\ 0 & \text{otherwise} \end{cases}$$

$$b_N(f) = \begin{cases} 1 & \text{if } s^* \in \text{src}(N, f) \\ 0 & \text{otherwise} \end{cases}$$

Clearly this is a dictatorship with dictator source s^* . Also, for any network N we have $s \sqsubset_N^{T'} s^{**} \sqsubset_N^{T'} s^*$; such a chain of strict inequalities cannot occur for a binary generalised dictatorship, so we are done. \square

C.5 Proof of proposition 5

Proof. An operator that satisfies unanimity but not groundedness is T that ranks facts according to the function

$$r_N(f) = \begin{cases} 1 & \text{if } \text{src}(N, f) \in \{\mathcal{S}, \emptyset\} \\ 0 & \text{otherwise} \end{cases} \quad (f \in \mathcal{F})$$

i.e. $f_1 \preceq_N^T f_2$ iff $r_N(f_1) \leq r_N(f_2)$ (note that T 's ranking of sources is irrelevant). Clearly T satisfies unanimity but not groundedness: consider any network N in which there is a fact f_1 with no corresponding sources, and a fact f_2 with at least one source, but whose sources are not the whole of \mathcal{S} . Then $f_1 \not\preceq_N^T f_2$ contrary to the requirements of groundedness.

Reversing the fact ordering of T gives an operator satisfying groundedness but not unanimity. \square

C.6 Proof of lemma 1

The following lemma will be used to prove the claim regarding weak independence.

Lemma 2. Let $(a_n)_{n \in \mathbb{N}}$, $(b_n)_{n \in \mathbb{N}}$ be convergent sequences of real numbers, and let $L = \lim_{n \rightarrow \infty} a_n$, $M = \lim_{n \rightarrow \infty} b_n$.

1. If $L < M$, then $a_n < b_n$ for sufficiently large n .
2. If $a_n \leq b_n$ for sufficiently large n , then $L \leq M$.

Proof. For the first claim, suppose $L < M$. Set $\epsilon = \frac{1}{2}(M - L) > 0$. By definition of the limit $a_n \rightarrow L$, there is $N_1 \in \mathbb{N}$ such that $|a_n - L| < \epsilon$ for all $n > N_1$. In particular, $a_n - L < \epsilon = \frac{1}{2}M - \frac{1}{2}L$ and so $a_n < \frac{1}{2}(M + L)$.

On the other hand, by definition of $b_n \rightarrow M$ there is $N_2 \in \mathbb{N}$ such that $|b_n - M| < \epsilon$ for $n > N_2$, and in particular $b_n - M > -\epsilon = \frac{1}{2}L - \frac{1}{2}M$ and so $b_n > \frac{1}{2}(M + L)$. Thus, for $n > N := \max\{N_1, N_2\}$ we have $a_n < \frac{1}{2}(M + L) < b_n$.

The second claim is now immediate; if $a_n \leq b_n$ for sufficiently large n and $L > M$ were false, then by the first claim we would have $b_n < a_n$ for sufficiently large n , which is a contradiction. \square

Proof of lemma 1. Let I be as in the statement of the lemma.

1. This is immediate since $t_N^*(s)$ is a limit of numbers in $[0, 1]$ which is a closed set, so $t_N^*(s) \in [0, 1]$ (and similar for $b_N^*(f)$).
2. Let N and $N' = \pi(N)$ be equivalent networks. For any $s_1, s_2 \in \mathcal{S}$ we have

$$\begin{aligned} s_1 \sqsubseteq_N^I s_2 &\iff \lim_{n \rightarrow \infty} t_N^n(s_1) \leq \lim_{n \rightarrow \infty} t_N^n(s_2) \\ &\iff \lim_{n \rightarrow \infty} t_{\pi(N)}^n(\pi(s_1)) \leq \lim_{n \rightarrow \infty} t_{\pi(N)}^n(\pi(s_2)) \\ &\iff \pi(s_1) \sqsubseteq_{\pi(N)}^I \pi(s_2) \\ &\iff \pi(s_1) \sqsubseteq_{N'}^I \pi(s_2) \end{aligned}$$

and for $f_1, f_2 \in \mathcal{F}$,

$$\begin{aligned} f_1 \preceq_N^I f_2 &\iff \lim_{n \rightarrow \infty} b_N^n(f_1) \leq \lim_{n \rightarrow \infty} b_N^n(f_2) \\ &\iff \lim_{n \rightarrow \infty} b_{\pi(n)}^n(\pi(f_1)) \leq \lim_{n \rightarrow \infty} b_{\pi(n)}^n(\pi(f_2)) \\ &\iff \pi(f_1) \preceq_{\pi(N)}^I \pi(f_2) \\ &\iff \pi(f_1) \preceq_{N'}^I \pi(f_2) \end{aligned}$$

Hence I is symmetric.

3. Let N be a truth discovery network and $f \in \mathcal{F}$ with $\text{src}(N, f) = \mathcal{S}$. By hypothesis there is $M \in \mathbb{N}$ such that $b_N^n(f) = 1$ for $n > M$, so $b_N^*(f) = \lim_{n \rightarrow \infty} 1 = 1 \geq b_N^*(f')$ for any f' . This means $f' \preceq_N^I f$, i.e. I satisfies unanimity.
4. Similarly, if $\text{src}(N, f) = \emptyset$ then by hypothesis $b_N^n(f) = 0$ for n sufficiently large, so $b_N^*(f) = 0 \leq b_N^*(f')$ for any f' , and $f \preceq_N^I f'$ as required for groundedness.
5. Let G, N_1, N_2 be as in the statement of the claim. For $s_1, s_2 \in G \cap \mathcal{S}$ we have

$$\begin{aligned}
s_1 \sqsubseteq_{N_1}^I s_2 &\iff \lim_{n \rightarrow \infty} t_{N_1}^n(s_1) \leq \lim_{n \rightarrow \infty} t_{N_1}^n(s_2) \\
&\iff \lim_{n \rightarrow \infty} t_{N_2}^n(s_1) \leq \lim_{n \rightarrow \infty} t_{N_2}^n(s_2) \\
&\iff s_1 \sqsubseteq_{N_2}^I s_2
\end{aligned}$$

and an identical argument proves the same for facts. Hence I satisfies independence of irrelevant stuff.

6. Let G, N_1, N_2 be as in the statement of the claim. Let $s_1, s_2 \in G \cap \mathcal{S}$ and suppose $s_1 \sqsubset_{N_1}^I s_2$, i.e. $\lim_{n \rightarrow \infty} t_{N_1}^n(s_1) < \lim_{n \rightarrow \infty} t_{N_1}^n(s_2)$. By lemma 2 part 1, there is $K \in \mathbb{N}$ such that $t_{N_1}^n(s_1) < t_{N_1}^n(s_2)$ for $n > K$. For any such n , $\alpha_n \geq 0$ means $t_{N_2}^n(s_1) = \alpha_n \cdot t_{N_1}^n(s_1) \leq \alpha_n \cdot t_{N_1}^n(s_2) = t_{N_2}^n(s_2)$. Lemma 2 part 2 then gives $\lim_{n \rightarrow \infty} t_{N_2}^n(s_1) \leq \lim_{n \rightarrow \infty} t_{N_2}^n(s_2)$, i.e. $s_1 \sqsubseteq_{N_2}^I s_2$.

An identical argument proves the result for fact ranking. Hence I satisfies weak independence.

□

C.7 Proof of theorem 1

Proof. Assume *Sums* is convergent. Since trust and belief scores for *Sums* are always in the range $[0, 1]$, lemma 1 can be applied.

1. **Symmetry:** We will use lemma 1 part 1. Let N and $N' = \pi(N)$ be equivalent networks. We show that $t_N^n(s) = t_{\pi(N)}^n(\pi(s))$ and $b_N^n(f) = b_{\pi(N)}^n(\pi(f))$ for all $n \in \mathbb{N}$ by induction.

The base case $n = 1$ is clear since $t_N^1, t_{\pi(N)}^1, b_N^1$ and $b_{\pi(N)}^1$ are constant with value $\frac{1}{2}$. Suppose $n \in \mathbb{N}$ is such that $t_N^n(s) = t_{\pi(N)}^n(\pi(s))$ and $b_N^n(f) = b_{\pi(N)}^n(\pi(f))$ for all $s \in \mathcal{S}$ and $f \in \mathcal{F}$.

Let $s \in \mathcal{S}$. Note that $f \in \text{facts}(N, s)$ iff $\pi(f) \in \text{facts}(\pi(N), \pi(s))$ by definition of $\pi(N)$. In particular, π restricted to $\text{facts}(N, s)$ is a bijection into $\text{facts}(\pi(N), \pi(s))$, so we may consider a ‘substitution’ $y = \pi(f)$ in the sum for $\hat{t}_N^{n+1}(s)$:

$$\begin{aligned}
\hat{t}_N^{n+1}(s) &= \sum_{f \in \text{facts}(N, s)} b_N^n(f) \\
&= \sum_{y \in \text{facts}(\pi(N), \pi(s))} b_N^n(\pi^{-1}(y)) \\
&= \sum_{y \in \text{facts}(\pi(N), \pi(s))} b_{\pi(N)}^n(\pi(\pi^{-1}(y))) \\
&= \sum_{y \in \text{facts}(\pi(N), \pi(s))} b_{\pi(N)}^n(y) \\
&= \hat{t}_{\pi(N)}^{n+1}(\pi(s))
\end{aligned}$$

Similarly, for $f \in \mathcal{F}$ note that π restricted to $\text{src}(N, f)$ is a bijection into $\text{src}(\pi(N), \pi(f))$: using the above and an identical argument we get $\hat{b}_N^{n+1}(f) = \hat{b}_{\pi(N)}^{n+1}(\pi(f))$.

Now we have

$$\begin{aligned}
t_N^{n+1}(s) &= \frac{\hat{t}_N^{n+1}(s)}{\max_{x \in \mathcal{S}} \hat{t}_N^{n+1}(x)} \\
&= \frac{\hat{t}_{\pi(N)}^{n+1}(\pi(s))}{\max_{x \in \mathcal{S}} \hat{t}_{\pi(N)}^{n+1}(\pi(x))}
\end{aligned}$$

Note that π restricted to \mathcal{S} is a bijection into \mathcal{S} itself, since by definition of equivalent networks each of \mathcal{S} , \mathcal{F} and \mathcal{O} are closed under π . Hence we may replace $\pi(x)$ in the maximum in the denominator with simply x by surjectivity of π , and so $t_N^{n+1}(s) = t_{\pi(N)}^{n+1}(\pi(s))$.

Similarly, $b_N^{n+1}(f) = b_{\pi(N)}^{n+1}(\pi(f))$. Hence, by lemma 1, *Sums* satisfies symmetry.

2. **Non-dictatorship:** We have shown that *Sums* satisfies symmetry. In particular *Sums* satisfies source-symmetry, so given proposition 3 it suffices to show that there is at least one truth discovery network N for which *Sums* does not rank all sources equally. The network shown in figure 4.4 is a suitable example. In this network there are three sources A , B and C , and two facts D and E , each relating to a different object¹.

We will show that *Sums* converges on this network, and that B ranks strictly beneath A .

For brevity write a_n for $t_N^n(A)$, \hat{a}_n for $\hat{t}_N^n(A)$ and similar for B, C, D, E . We claim that for all $n > 1$, $a_n = 1$, $b_n = c_n = \frac{1}{2}$ and $d_n = e_n = 1$. By definition, for all $n > 1$:

$$\hat{a}_n = d_{n-1} + e_{n-1}, \quad \hat{b}_n = d_{n-1}, \quad \hat{c}_n = e_{n-1}$$

$$\hat{d}_n = \hat{a}_n + \hat{b}_n, \quad \hat{e}_n = \hat{a}_n + \hat{c}_n$$

Recalling that $d_1 = e_1 = \frac{1}{2}$, taking $n = 2$ we get

$$\hat{a}_2 = 1, \quad \hat{b}_2 = \frac{1}{2}, \quad \hat{c}_2 = \frac{1}{2}$$

$$\hat{d}_2 = \frac{3}{2}, \quad \hat{e}_2 = \frac{3}{2}$$

and so $a_2 = 1$, $b_2 = c_2 = \frac{1}{2}$ and $d_2 = e_2 = 1$. For $n = 3$ we get

$$\hat{a}_3 = d_2 + e_2 = 2, \quad \hat{b}_3 = d_2 = 1, \quad \hat{c}_3 = e_2 = 1$$

$$\hat{d}_3 = \hat{a}_3 + \hat{b}_3 = 3, \quad \hat{e}_3 = \hat{a}_3 + \hat{c}_3 = 3$$

and so $a_3 = 1$, $b_3 = c_3 = \frac{1}{2}$ and $d_3 = e_3 = 1$, i.e. the trust/belief scores are unchanged. Repeating in this way we see that $a_n = 1$ and $b_n = \frac{1}{2}$ for all $n > 1$. Hence $t_N^*(A) = 1 > \frac{1}{2} = t_N^*(B)$: this means B ranks strictly below A , which completes the proof.

¹ If \mathcal{S} contains more than three sources we consider all other sources to have no outgoing edges (and similarly for if \mathcal{F} contains more than two facts). Note that the objects to not play any role in *Sums*.

3. **Unanimity:** Let N be a truth discovery network and suppose $f \in \mathcal{F}$ is such that $\text{src}(N, f) = \mathcal{S}$. Then, for $n > 1$ and any $f' \in \mathcal{F}$,

$$\hat{b}_n(f') = \sum_{s \in \text{src}(N, f')} \hat{t}_n(s) \leq \sum_{s \in \mathcal{S}} \hat{t}_n(s) = \hat{b}_n(f)$$

using the fact that $\hat{t}_n(s) \geq 0$. Therefore $\max_{y \in \mathcal{F}} \hat{b}_n(y) = \hat{b}_n(f)$, so $b_n(f) = 1$. By lemma 1 part 3, *Sums* satisfies unanimity.

4. **Groundedness:** Let N be a truth discovery network and suppose $f \in \mathcal{F}$ has $\text{src}(N, f) = \emptyset$. By definition $b_n(f) = 0$ for all $n > 1$, so by lemma 1, *Sums* satisfies groundedness.
5. **Weak independence:** To show weak independence we will use lemma 1 part 6. Let N_1, N_2 be truth discovery networks with a common connected component G . Suitable sequences $(\alpha_n)_{n \in \mathbb{N}}, (\beta_n)_{n \in \mathbb{N}}$ will be defined recursively. For $n = 1$, set $\alpha_1 = \beta_1 = 1$. We have, by definition of *Sums*,

$$t_{N_2}^1(s) = \frac{1}{2} = \alpha_1 \cdot \frac{1}{2} = \alpha_1 \cdot t_{N_1}^1(s)$$

$$b_{N_2}^1(f) = \frac{1}{2} = \beta_1 \cdot \frac{1}{2} = \beta_1 \cdot b_{N_1}^1(f)$$

for any $s \in G \cap \mathcal{S}$ and $f \in G \cap \mathcal{F}$. Now suppose $n \in \mathbb{N}$ is such that there are $\alpha_{n-1}, \beta_{n-1} \geq 0$ with

$$t_{N_2}^{n-1}(s) = \alpha_{n-1} \cdot t_{N_1}^{n-1}(s)$$

$$b_{N_2}^{n-1}(f) = \beta_{n-1} \cdot b_{N_1}^{n-1}(f)$$

for all $s \in G \cap \mathcal{S}$ and $f \in G \cap \mathcal{F}$. Fix a source $s \in G \cap \mathcal{S}$. Let E_1, E_2 and E_G denote the set of edges in N_1, N_2 and G respectively. Note that for any fact $f \in \mathcal{F}$,

$$\begin{aligned} f \in \text{facts}(N_1, s) &\iff (s, f) \in E_1 \\ &\iff (s, f) \in E_G \\ &\iff (s, f) \in E_2 \\ &\iff f \in \text{facts}(N_2, s) \end{aligned}$$

so $\text{facts}(N_1, s) = \text{facts}(N_2, s)$. Hence

$$\begin{aligned}\hat{t}_{N_2}^n(s) &= \sum_{f \in \text{facts}(N_2, s)} b_{N_2}^{n-1}(f) \\ &= \sum_{f \in \text{facts}(N_1, s)} \beta_{n-1} \cdot b_{N_1}^{n-1}(f) \\ &= \beta_{n-1} \sum_{f \in \text{facts}(N_1, s)} b_{N_1}^{n-1}(f) \\ &= \beta_{n-1} \hat{t}_{N_1}^n(s)\end{aligned}$$

Write

$$\gamma_1 = \frac{1}{\max_{x \in \mathcal{S}} \hat{t}_{N_1}^n(x)}, \quad \gamma_2 = \frac{1}{\max_{x \in \mathcal{S}} \hat{t}_{N_2}^n(x)}$$

so that

$$\begin{aligned}t_{N_1}^n(s) &= \gamma_1 \cdot \hat{t}_{N_1}^n(s) \\ t_{N_2}^n(s) &= \gamma_2 \cdot \hat{t}_{N_2}^n(s) \\ &= \gamma_2 \cdot \beta_{n-1} \cdot \hat{t}_{N_1}^n(s) \\ &= \frac{\gamma_2 \cdot \beta_{n-1}}{\gamma_1} \cdot \gamma_1 \cdot \hat{t}_{N_1}^n(s) \\ &= \frac{\gamma_2 \cdot \beta_{n-1}}{\gamma_1} \cdot t_{N_1}^n(s)\end{aligned}$$

Taking $\alpha_n = \frac{\gamma_2 \beta_{n-1}}{\gamma_1}$ we have the desired equality for trust scores. Note that $\gamma_1, \gamma_2 > 0$, so $\alpha_n \geq 0$.

The argument for belief scores is similar. Fix $f \in G \cap \mathcal{F}$. We have $\text{src}(N_1, f) = \text{src}(N_2, f)$, so

$$\begin{aligned}\hat{b}_{N_2}^n(f) &= \sum_{s \in \text{src}(N_2, f)} \hat{t}_{N_2}^n(s) \\ &= \sum_{s \in \text{src}(N_1, f)} \beta_{n-1} \cdot \hat{t}_{N_1}^n(s) \\ &= \beta_{n-1} \cdot \hat{b}_{N_1}^n(f)\end{aligned}$$

Write

$$\delta_1 = \frac{1}{\max_{y \in \mathcal{F}} \hat{b}_{N_1}^n(y)}, \quad \delta_2 = \frac{1}{\max_{y \in \mathcal{F}} \hat{b}_{N_2}^n(y)}$$

Then

$$\begin{aligned}
b_{N_1}^n(f) &= \delta_1 \cdot \hat{b}_{N_1}^n(f) \\
b_{N_2}^n(f) &= \delta_2 \cdot \hat{b}_{N_2}^n(f) \\
&= \delta_2 \cdot \beta_{n-1} \cdot \hat{b}_{N_1}^n(f) \\
&= \frac{\delta_2 \cdot \beta_{n-1}}{\delta_1} \cdot \delta_1 \cdot \hat{b}_{N_1}^n(f) \\
&= \frac{\delta_2 \cdot \beta_{n-1}}{\delta_1} \cdot b_{N_1}^n(f)
\end{aligned}$$

so we may take $\beta_n = \frac{\delta_2 \beta_{n-1}}{\delta_1}$.

By induction, there exist sequences $(\alpha_n)_{n \in \mathbb{N}}$, $(\beta_n)_{n \in \mathbb{N}}$, satisfying the hypothesis of lemma 1 part 6, so *Sums* satisfies weak independence.

□

C.8 Proof of theorem 2

Proof. Let N_1 be the network shown in figure 4.4, and N_2 be the network shown in figure 4.5. With $G = N_1$, G is a connected component of both networks. We have already shown in the proof of theorem 1 that $t_{N_1}^*(A) = 1$ and $t_{N_1}^*(B) = t_{N_1}^*(C) = \frac{1}{2}$, so in particular A is ranked strictly above B in N_1 . To prove *Sums* does not satisfy independence, we will show that A and B are in fact ranked *equally* in N_2 ; in particular, we shall have $A \sqsubseteq_{N_2}^{I_{sums}} B$ but $A \not\sqsubseteq_{N_1}^{I_{sums}} B$.

As before, write a_n for $t_{N_2}^n(A)$ and similar for the other nodes. We claim that, for $n > 1$:

$$\begin{aligned}
a_n &= \frac{2}{3^{n-1}}, & b_n = c_n &= \frac{1}{3^{n-1}}, & s_n = t_n = u_n &= 1 \\
d_n = e_n &= \frac{1}{3^{n-1}}, & v_n = w_n = x_n &= 1
\end{aligned}$$

The base case for induction is $n = 2$. We have

$$\hat{a}_2 = d_1 + e_1 = 1, \quad \hat{b}_2 = d_1 = \frac{1}{2}, \quad \hat{c}_2 = e_1 = \frac{1}{2},$$

$$\begin{aligned}\hat{s}_2 &= \hat{t}_2 = \hat{u}_2 = v_1 + w_1 + x_1 = \frac{3}{2} \\ \hat{d}_2 &= \hat{a}_2 + \hat{b}_2 = \frac{3}{2}, \quad \hat{e}_2 = \hat{a}_2 + \hat{c}_2 = \frac{3}{2} \\ \hat{v}_2 &= \hat{w}_2 = \hat{x}_2 = \hat{s}_2 + \hat{t}_2 + \hat{u}_2 = \frac{9}{2}\end{aligned}$$

The maximum trust score is $\frac{3}{2}$ and the maximum belief score is $\frac{9}{2}$, so

$$\begin{aligned}a_2 &= \frac{2}{3}, \quad b_2 = \frac{1}{3}, \quad c_2 = \frac{1}{3}, \quad s_2 = 1, \quad t_2 = 1, \quad u_2 = 1 \\ d_2 &= \frac{1}{3}, \quad e_2 = \frac{1}{3}, \quad v_2 = w_2 = x_2 = 1\end{aligned}$$

Thus the claim holds for $n = 2$. Now suppose that the claim holds for the $(n - 1)$ -th iteration. We have

$$\begin{aligned}\hat{a}_n &= d_{n-1} + e_{n-1} = \frac{2}{3^{n-2}}, \quad \hat{b}_n = d_{n-1} = \frac{1}{3^{n-2}}, \quad \hat{c}_n = e_{n-1} = \frac{1}{3^{n-2}} \\ \hat{s}_n &= \hat{t}_n = \hat{u}_n = v_{n-1} + w_{n-1} + x_{n-1} = 3 \\ \hat{d}_n &= \hat{a}_n + \hat{b}_n = \frac{3}{3^{n-2}}, \quad \hat{e}_n = \hat{a}_n + \hat{c}_n = \frac{3}{3^{n-2}} \\ \hat{v}_n &= \hat{w}_n = \hat{x}_n = 9\end{aligned}$$

The maximum trust and belief scores are 3 and 9 respectively, so we get

$$\begin{aligned}a_n &= \frac{2}{3^{n-2}} \cdot \frac{1}{3} = \frac{2}{3^{n-1}}, \quad b_n = \frac{1}{3^{n-2}} \cdot \frac{1}{3} = \frac{1}{3^{n-1}}, \quad c_n = \frac{1}{3^{n-2}} \cdot \frac{1}{3} = \frac{1}{3^{n-1}}, \\ s_n &= t_n = u_n = 1 \\ d_n &= \frac{3}{3^{n-2}} \cdot \frac{1}{9} = \frac{1}{3^{n-1}}, \quad e_n = \frac{3}{3^{n-2}} \cdot \frac{1}{9} = \frac{1}{3^{n-1}}, \\ v_n &= w_n = x_n = 1\end{aligned}$$

as required.

Finally, this means

$$t_{N_2}^*(A) = \lim_{n \rightarrow \infty} \frac{2}{3^{N-1}} = 0$$

$$t_{N_2}^*(B) = \lim_{n \rightarrow \infty} \frac{1}{3^{N-1}} = 0$$

and $A \simeq_{N_2}^{I_{sums}} B$, which completes the proof. □