Department of Computer Science
The University of Auckland
New Zealand

Fraud Detection in Online Auctions

Sidney Tsang

September 2014

Supervisors:
Gill Dobbie
Yun Sing Koh

A thesis submitted in partial fulfillment of the requirements of

Doctor of Philosophy in Science

Abstract

Online auctions are auctions held over the internet. Online auctions avoid some of the problems of traditional auctions, such as geographical and time restrictions, and limited audiences. However, this has also made them a target for fraud. The majority of previous approaches for reducing auction fraud make use of machine learning techniques to identify suspicious users and auctions. However, many of these approaches have generally encountered the same challenges during implementation and evaluation. These challenges include lack of good quality datasets, imbalance in the number of normal and fraudulent examples, heterogeneity of both normal and fraudulent behaviours, and behaviours which evolve over time. In this thesis, we explore methods of dealing with some of these challenges.

To address the lack of good datasets, we implement an agent-based simulation for online auctions as a means of generating synthetic auction data. The simulation models the behaviour of normal bidders and sellers, based on real online auction data. The synthetic data was evaluated using three methods, and results show that the synthetic data is similar to real data. We then applied the simulation to evaluate an existing fraud detection method to show that the simulation can be used as a source of data for evaluating fraud detection algorithms.

To address the difficulty in creating detection models for different fraud behaviours and strategies, we demonstrate using supervised learning methods with our simulation to easily create models for detecting arbitrary types of fraud. Models created using this approach were shown to have higher accuracy compared to an existing fraud detection method even after tuning. The caveat is the fraud type of interest must be explicitly defined.

To deal with the limitations of the previous approach and to avoid the need for model retraining when fraud behaviours change, we propose an unsupervised method based on anomaly detection. Since the method uses an anomaly detection approach, the model can adapt to changes in user and fraudulent behaviour: the method will identify users who behave differently to the majority of other users. The method makes use of additional network information to identify groups of users that appear suspicious. Extensive evaluation using synthetic data shows that it has higher accuracy than other related approaches. When applied to a real dataset, our method finds a reasonable number of potentially fraudulent users who exhibit unusual characteristics when compared to normal users.

List of publications

Parts of Chapter 3 have been published in

Sidney Tsang, Gillian Dobbie, and Yun Sing Koh. Evaluating fraud detection algorithms using an auction data generator. In *IEEE 12th International Conference on Data Mining Workshops (ICDMW), 2012*, pages 332–339.

Parts of Chapter 4 have been published in

Sidney Tsang, Gillian Dobbie, and Yun Sing Koh. *Generating Realistic Online Auction Data*, volume 7691 of *Lecture Notes in Computer Science*, chapter 11, pages 120–131. Springer Berlin Heidelberg, 2012.

Parts of Chapter 5 have been published in

Sidney Tsang, Yun Sing Koh, Gillian Dobbie, and Shafiq Alam. Detecting online auction shilling frauds using supervised learning. *Expert Systems with Applications*, 41(6):3027-3040, 2014.

Parts of Chapters 6 and 7 have been published in

Sidney Tsang, Yun Sing Koh, Gillian Dobbie, and Shafiq Alam. Span: Finding collaborative frauds in online auctions. *Knowledge-Based Systems*, 2014, http://dx.doi.org/10.1016/j.knosys.2014.08.016

Acknowledgements

I would like to thank my supervisor Dr. Gillian Dobbie and co-supervisor Dr. Yun Sing Koh for their continual guidance, encouragement, and feedback, and without whom this work would not be possible.

I would also like to thank members of the Knowledge Management Group for their helpful advice and comments.

My thanks also to my family, especially my sister, for their encouragement, and my friends, who were always willing to listen and offer support.

Contents

List of Figures

List of Tables

1 Introduction

Data mining is used to extract patterns which are reliable and previously unknown from different data sources. To apply data mining, the data must be relevant, adequate, and clean. Data mining has been applied to fraud detection in a wide range of industries including banking, insurance, healthcare, automotive, and retail. Fraud is defined as dishonest activity causing actual or potential financial loss to any person or entity[1].

Online auction sites such as eBay[2] and TradeMe[3] allow goods and services to be bought and sold online anonymously. The most common type of online auction is the English auction, where bids are placed in ascending order, publicly observable, and the winner is the final bidder with the highest bid [3]. In 2013, eBay had 128 million active

[1]https://www.sfo.govt.nz/what-is-serious-fraud
[2]http://www.ebay.com/
[3]http://www.trademe.co.nz/

users and an auction volume of more than $22 billion USD [4]. The volume of auctions, user anonymity, and low barriers to entry make online auctions a lucrative target for fraud. The anonymity and simplicity of creating multiple aliases allow unsuspecting users to be exploited by dishonest users. This exploitation can take many forms, including shilling, non-delivery, misrepresentation, or the sale of stolen goods [5]. Dishonest users will also disguise themselves to avoid detection by imitating normal behaviours [6], making fraudulent behaviours difficult to define. Previous work has noted that users often appear to behave irrationally [7], and previous attempts at clustering users into predefined types according to their bidding behaviour have failed to label the majority of users [8]. The range of potential fraudulent behaviour together with the number and range of legitimate behaviours makes it difficult to differentiate between fraudulent and legitimate users.

Frauds committed before or after an auction, such as misrepresentation or non-delivery fraud, are generally identified after-the-fact, for example, when the buyer never receives the item they paid for. Detection of these frauds generally relies on real-world evidence as opposed to online prevention and detection mechanisms [5]. This contrasts with in-auction fraud, such as reputation fraud or shilling, which occur while the auction is in progress. It can often be difficult to determine when and if in-auction fraud has occurred.

The majority of existing research use machine learning to detect or predict in-auction fraud. A range of methods have been used, including various regression models such as logistic regression and probit models; graphical models such as neural networks, Markov random fields and decision trees; and rule-based systems. Of the proposed methods, the majority are supervised, which has several drawbacks.

In this chapter, we first describe the challenges in developing methods for detecting and reducing fraud in Section 1.1. Then in Section 1.2 we propose methods for tackling each of the challenges. In Section 1.4 we list the contributions of the thesis. Section 1.5 gives an outline of the remaining chapters.

1.1 Problem definition

There are two main difficulties with producing effective online auction fraud detection algorithms: first, the lack of publicly available real data, and second, the nature of online auction fraud. Commercial companies who have access to this data are reluctant to publicly release it for many reasons, including privacy concerns, and the potential for loss in user confidence if a high number of fraudulent activities are identified. Evaluation in previous work has thus generally been performed on synthetic data, or using real data with limited size and scope.

The nature of online auction fraud also makes it difficult to develop effective fraud detection algorithms, supervised or unsupervised. Firstly, as in many other fraud detection domains, normal cases vastly outnumber fraudulent ones. eBay self reports an auction fraud rate of 0.01%, while others have estimated the fraud rate to be 0.211% [9]. This low number of fraudulent cases makes it difficult to apply some machine learning techniques, supervised learning in particular, when there is not enough information in the fraudulent examples to build an accurate model. Secondly, many different types of fraud exist in online auctions, each of which can be committed using different strategies. It is difficult to develop algorithms that can detect these diverse types of fraud, especially when normal behaviour of users is difficult to model as well [8]. Thirdly, as fraudulent accounts are identified and removed, remaining accounts and any new accounts may change their behaviour to avoid detection. This is especially a problem given the low barriers to entry and exit for online auctions [5]. For fraud detection methods to be effective, they must be able to adapt to evolving fraud behaviour, for example by generating new models or by using methods that do not require data where fraud cases have been previously identified, manually or otherwise.

1.2 Proposed solution

We propose methods to solve each of the problems described in Section 1.1. First, to tackle the problem of data availability, we design and implement an agent-based auction simulation. The simulation is modelled on a set of real online auction data, and can be used to generate auction data containing both fraudulent and non-fraudulent behaviours. The behaviour of both the bidders and sellers is modelled. This resulting synthetic data is shown to be similar to real data through multiple quantitative comparisons. We also demonstrate that the simulation can be used to evaluate the accuracy of an existing fraud detection method, and its variations, at detecting different types of online auction fraud.

Second, to reduce the effort required to develop classifiers for different types of fraud, we propose combining machine learning techniques with labelled synthetic data from the simulation. Classification models for arbitrary types of fraud can be created in this way. When fraud behaviour changes, a new model is trained using updated synthetic data.

Third, to avoid the need for training new models, we propose using an anomaly detection approach for fraud detection that does not require labelled synthetic data. The proposed method also aims to identify groups of users committing fraud collaboratively and to improve accuracy by making use of network information contained in the inter-user interactions.

1.3 Scope and objectives

The objective of this thesis is to explore methods for detecting multiple types of fraud in online auctions while addressing the challenges described previously. The work done to achieve this can be divided into two parts: implementing a method for generating labelled synthetic data, and exploring two fraud detection approaches.

The scope of the first part is limited to implementing an online auction simulation, and evaluating the synthetic data generated to ensure its similarity to real online auction data. The simulation is based on two sets of real online auction data from TradeMe over two different periods. We limit our source of real data to TradeMe, though we expect our work to extend to other similar online auction sites such as eBay. We also select a set of continuous features into which the raw data (a set of auctions and bids) is transformed. This is necessary to understand user behaviour in the real data, and subsequently for comparing the synthetic data to the real data. The set of features we choose are all per-user features, as opposed to per-auction (e.g., average bid value for a particular user instead of for a particular auction). Although per-auction features can also reveal bidding patterns, our aim is to model individual users, which makes per-user features more appropriate.

The scope of the second part is limited to developing approaches for fraud detection, and evaluating their effectiveness. We propose two approaches. The first proposed approach uses two supervised learning algorithms together with the labelled synthetic data to generate fraud detection models. The algorithms we use are neural networks and decision trees. Other supervised learning schemes are available, such as probit models, regression models or linear support vector machines, but the two algorithms we use are sufficient for demonstrating our first approach. The second proposed approach is based on anomaly detection. It requires a method for generating anomaly scores for each user. To demonstrate the overall approach, we use a relative density method that calculates a local outlier factor for each user. In the future, this can be replaced with methods such as k-means or EM-clustering [10] which may yield higher accuracy.

As part of the scope for the second part, we also evaluate both approaches. To evaluate these approaches, we focus on synthetic data containing two types of fraud, shilling fraud and reputation fraud, and their variations. Although there are many other types of fraud, these two fraud types are the most frequently described in previous literature, and are sufficient for demonstrating our approaches.

1.4 Contributions

The contributions of this thesis are:

1. We survey literature related to online auction fraud, and discuss and compare them in terms of the three main stages in developing fraud detection techniques: definition of the fraud of interest, feature set selection, and model construction.

2. We design and implement an agent-based simulation of online auctions based on real online auction data. The simulation can be used to generate synthetic auction data. The synthetic data is objectively compared against real auction data using statistical and clustering methods, and the results show that the synthetic data is similar to real auction data.

3. We demonstrate the simulation as a tool for evaluating fraud detection algorithms. Using the generated synthetic data and additional fraud agents, the accuracy of different fraud detection algorithms can be evaluated and compared.

4. We present a framework for creating classifiers using supervised machine learning methods by extending the simulation using additional agents. The method allows classifiers for arbitrary fraud types to be constructed, given that they are defined as agents in the simulation. Evaluation performed using synthetic data containing three different types of fraud shows that the classifiers are more accurate than previous methods, and evaluation using real data shows that they identify users who appear suspicious. We also propose a set of features that characterise general bidding behaviour and which produce classifiers with higher accuracy.

5. We propose an unsupervised fraud detection algorithm based on anomaly detection in graphs. The method identifies groups of closely collaborating users who exhibit unusual characteristics. Network information was used to improve accuracy by revising the initial anomaly score value. The method is evaluated using synthetic data under a range of conditions and compares favourably to other related methods. A case-study using real data shows the method identifies a reasonable number of users, who appear to be different from the majority of users, as suspicious.

1.5 Thesis outline

Chapter 2 surveys existing work in the area of online auction fraud. We discuss the approaches proposed for fraud detection and prediction, including the fraud types targeted, selected features, and their general limitations. Chapter 3 describes the agent-based simulation. It first provides background about agent-based simulation and modelling, and the use of simulations in online auctions in previous literature. It then defines the auction model, and describes the design and implementation of the simulation. Chapter 4 applies the simulation to evaluating variations of a fraud detection algorithm. Chapter 5 demonstrates the generation of classification models for fraud detection, using the simulation and supervised learning techniques. Multiple types of fraud are defined, and models to detect them are generated for each. Chapter 6 presents our unsupervised approach for fraud detection based on anomaly detection in graphs, called SPAN. It defines multiple types of collaborative fraud and presents results comparing SPAN to other related methods. Chapter 7 presents a case study applying SPAN to a real online auction dataset. Chapter 8 concludes the thesis.

2 Related work

In this chapter, we describe previous work in the area of auction fraud. Section 2.1 describes different types of online auction fraud and the strategies used to commit them. Section 2.2 describes methods for preventing or detecting auction fraud in terms of the targeted fraud type, selected features, and the technique used. Some of the limitations of previous work are also discussed.

2.1 Fraud types

There are several types of fraud that are common in online auctions. These fraud types can be classified according to when they occur in an auction: before, during or after. Frauds that occur before the auction begins include misrepresentation, for example by giving false or misleading information about the item, and sale of stolen goods. Frauds that occur after the auction has concluded include non-delivery, where the item is not delivered after payment, and fee-stacking, where additional fees are charged after the auction completes. These fraud types are described in more detail in [5].

The main types of fraud previously studied in online auctions are shilling and reputation inflation, which occur during auctions. Other types of fraud, described in [5] cannot occur in English auctions, since they do not allow bid retractions. Even though there are only two main types of fraud that occur during auctions, there are subtypes that have different goals, and each have multiple ways of achieving those goals.

For example, Kauffman et al. [11] described a type of shilling called *reserve-price shilling*, where the goal is to avoid paying auction house fees. This type of fraud is possible only under particular auction fee structures, where fees increase as the starting price increases. As a result, it is beneficial for the seller to set the starting price low, and use a collaborating bidder to make a bid after the auction begins to have a high starting auction price without paying the corresponding fee. In another type of shilling, called *buy-back shilling*, the seller prevents an item from being sold below its value by using an account the seller controls to win the item rather than let the item be sold at a low price [5]. The loss incurred from selling the item below its value outweighs the auction fees incurred as a result. A third type of shilling, *competitive shilling* has the goal of increasing the final price of items sold to legitimate sellers. This is done using one or more other accounts to submit fraudulent bids in the auction to increase the price. This fraud type has been investigated by Trevathan and Read [12] where they implemented a simple shill agent, and a collection of “zero-intelligence” agents that model normal users to generate synthetic data for evaluation. This is similar to our work in Chapters 3 and 4. However there are two key differences: first, we focus on simulating user behaviour over time and over multiple auctions, rather than individual auctions. Secondly, our normal agents are modelled on real data, and the resulting synthetic data is validated. Competitive shilling is described in greater detail in Section 4.2.2 of Chapter 4.

Reputation manipulation has also been identified [13, 9, 14], where accounts coordinate to give each other positive feedback to inflate their positive reputation and appear trustworthy. The accounts can then be used to commit non-delivery fraud. One way this is achieved is by using two sets of accounts, accomplice accounts and fraud accounts. The purpose of the accomplice accounts are to inflate the reputation of fraud accounts, and all frauds are committed using the fraud accounts. When the fraud accounts are identified and removed, the accomplice accounts may remain undetected since they did not directly commit fraud, and can continue to inflate the reputation of other fraud accounts. The benefits of a positive reputation for sellers has been shown in [15]. Synthetic data containing reputation fraud is used in Chapter 6.

There is another type of fraud called credit-card phantom transactions [16], observed in online auctions in Korea. The goal is to provide illegal loans to the bidder using a collaborating seller. The seller creates an auction for an item valued at the loan requested, which the bidder will pay for using a credit card. The seller receives payment, and the bidder is given the loan by the seller, minus the advanced interest (typically 15-25% of loan). The bidder may then fail to repay the credit card company.

For both competitive shilling and reputation inflation, the goal can be accomplished by accounts working together in pairs, or in large networks. Chapters 4 and 5 focus on pairs of collaborating fraud agents. Chapters 6 and 7 investigates networks of collaborating fraud agents.

2.2 Methods for predicting or detecting fraud

The approaches used in previous work to create fraud detection methods for online auctions generally follow three main steps: (1) define the behaviour of interest, (2) identify features that differentiate normal and fraudulent users or auctions, and (3) develop a fraud detection algorithm based on the selected set of features. Below, we describe previous work in terms of these three steps from Sections 2.2.1 to 2.2.3.

2.2.1 Fraudulent behaviours

We concentrate on three types of fraud most commonly investigated in previous literature: shilling fraud, reputation manipulation, and non-delivery fraud. Other types of online auction fraud have been described in detail in [5].

Shilling fraud has been investigated in [5, 11, 17, 18], and described in detail in Chapter 4. Reputation manipulation has been investigated in [13, 9, 19, 14]. Non-delivery fraud has been investigated in [20, 6].

2.2.2 Feature selection

Features used to differentiate between fraudulent and normal users can be divided into user level and network level features. User level features describe the behaviour of individual users, such as the number of auctions they participate in, or the average value and frequency of their bids. Chang and Chang [6] list a comprehensive set of user-level features along with a brief description for each. Network level features describe the relationships between users. Network features are discussed in greater detail in Chapter 6.

The vast majority of previous work uses only user-level features [20, 6, 5, 9, 11, 17, 18, 14], and does not make use of network-level features to detect fraudulent users or auctions. Since users are considered individually, those proposed methods are not very effective for identifying groups of users committing fraud collaboratively. The only exception is the work by Chau et al. [13], where the network is used to identify groups of fraudulent users, and to a limited extent the work by Lin et al. [19], which uses the network to derive a feature as part of the set of inputs to a neural network.

2.2.3 Proposed methods

The methods that have been proposed to reduce fraud can be divided into broad categories of detection and prediction. Methods in fraud prediction include work by [11, 18, 6, 20]. Kauffman et al. [11] constructed a probit model to predict the auctions in which reserve-price shilling is likely to occur. The dataset is a set of coin auctions from eBay, with an auction classified as fraudulent if a “questionable bid” occurred in the auction, and legitimate otherwise.

Xu et al. [18] proposed a real-time model checking method for detecting suspicious bidding behaviours. The method consists of three main parts: a dynamic auction model, which is updated as the auction state changes with new bids and time; a set of linear temporal logic (LTL) formulas which define shilling behaviour, and a model checker (SPIN) which recognises when the LTL formulas are violated. Action can then be taken according to the number and types of suspicious behaviours identified. However, the set of LTL rules used in the model and the model detection performance were not reported. Chang et al. [6] proposed a partitioning technique for identifying fraudulent users. User histories are partitioned into two parts according to time. For fraudulent users, the later part will contain the fraudulent transactions. Since the goal is to identify fraudulent users in their latent stage, before they commit fraud, only the transactions contained in the first part are used as training data in an instance-based learner. A recall of 93% was reported over a commercial dataset with 1,467 instances.

Almendra and Enachescu [20] proposed the use of boosted trees for identifying non-delivery frauds before they are completed. Auctions are assigned rankings that reflect how likely they are to be fraudulent. Detection performance is improved using score propagation, whereby if an auction is found to be fraudulent, other auctions by the same seller will have their rankings increased as they are more likely to be fraudulent. Evaluation results were reported over a commercial dataset, classified in a previous work by [21] using textual comments indicating non-delivery fraud.

Fraud detection methods include work proposed by [13, 19, 17, 22]. Trevathan et al. [17] combined five heuristic features into one final rating for each user, and used it to rank users in order of degree of suspicion. Users above a user-defined threshold were classified as fraudulent.

Chau et al. [13] describe an algorithm, called 2LFS, specifically for detecting collusive fraud. In 2LFS, users and transactions are modelled as a Markov random field (MRF), where each user is a node in the graph, and an edge between two nodes represents an interaction between the two users. ‘User’ refers to a node in the graph: multiple users may be controlled by the same person. 2LFS attempts to identify groups of users that engage in reputation inflation, and subsequently engage in non-delivery fraud when they have a sufficiently high reputation. The authors hypothesise that users will be split into two groups with different roles. The first will only perform legitimate transactions and be used to increase the reputation of a second group of users, which will be used to commit non-delivery fraud. In the graph, these two groups will form a bipartite core, where only the accomplices will interact with users outside the core. Thus each node can exist in one of three states: ‘accomplice’ and ‘fraud’, which correspond to the two user groups, and a third state ‘honest’, which represents legitimate users. Using a specifically tailored compatibility matrix, and a loopy belief propagation algorithm, 2LFS can identify fraud-accomplice bipartite cores.

Lin et al. [19] use a fuzzy neural network, ANFIS, to identify collusive reputation inflation using a dataset with some identified fraudulent users. The three features used as input to ANFIS are *Date_Difference*, *Credit*, and *AFR*. AFR (Auction Fraud Rank) is a modified page rank score, where each users “page rank” is modified using two additional terms that should be higher if they are committing reputation fraud. Thus AFR should be higher for groups that are committing reputation fraud together.

Zhang et al. [22] proposed a online probit model framework, using features selected by experts. As new auctions are submitted, a subset of them are labelled as either fraudulent or legitimate by experts. The subset is not random, but is selected using an existing expert system for detecting fraud. This partially labelled data is used to train and revise the model, using the assumption that the unchecked auctions are legitimate. As fraud behaviours change over time, the model adapts as well by revising the importance of each feature for detecting fraud using the labelled instances. The authors did not give the feature set used, citing company policy.

Table 2.1 shows a summary of approaches taken by previous work to generate classification models. The table lists the method used to generate the classification model, the types of features used, the evaluation dataset used, and the fraud type targeted. The last two rows of the table shows how our work presented in Chapters 5 and 6 compare with the others.

Table 2.1
Summary of previous work proposing fraud classification models.

First author	Year	Classification model			Features used			Fraud Type
		Unsupervised	Supervised	Other	Transaction	User	Network	
Kauffman	2005		✓		✓	✓		Shilling
Chau	2006	✓			✓	✓	✓	Reputation
Chae	2007		✓		✓			Credit-card
Trevathan	2007			✓	✓	✓		Shilling
Xu	2009			✓	✓			Shilling
Lin	2012		✓		✓	✓		Shilling
Zhang	2012		✓		?	?	?	General
Almendra	2013		✓		✓	✓		Non-delivery
Chang	2012		✓		✓	✓		Non-delivery
Chapter 5			✓		✓	✓		Shilling
Chapter 6		✓			✓	✓	✓	Collaborative

✓indicates the model or feature type used
? indicates the feature types used are unknown

2.2.4 Limitations of previous work

The work described above illustrates some of the problems mentioned earlier, and which our approach is able to address. Firstly, there are difficulties with using real datasets for evaluation where ground truth is unknown. In [23], where an unsupervised approach was used, it is difficult to measure the false negative rate, since it is infeasible to manually examine all cases classified as legitimate for all but the smallest datasets. In [20, 16, 24, 22], which used supervised approaches, it is difficult to assess the accuracy of classification labels. In cases where instances are manually labelled, the datasets are small [24]. In [20, 25], where the authors specify rules to label instances as either legitimate or fraudulent, it is necessary to show that the rules used are independent to the proposed detection method, otherwise evaluation results are unreliable.

Secondly, each labelled dataset can only be used to build classifiers for one purpose. For example, given a set of labels for non-delivery fraud and normal users, as in [20], classifiers can only be trained to detect non-delivery fraud. Different sets of labels can be allocated to the same dataset denoting different behaviours, however this must be done carefully to ensure labels are accurate.

Thirdly, some of the features selected are only useful in detecting specific types of fraud, such as those in [16, 25, 2]. More general attributes such as those used in [20], or the use of a larger set of attributes in [24] will increase the likelihood that those same features will be useful in the detection of multiple types of fraud.

Our work in Chapter 5 alleviates all three problems by: (1) using a validated synthetic data generator in which ground truth is known; (2) allowing new datasets to be created by implementing a corresponding fraud agent; and (3) using the proposed set of 10 features that capture general bidder behaviour to detect multiple types of fraud.

2.2.5 Other methods against fraud

Here we describe additional work on reducing the impact of fraud in auctions. Instead of directly identifying fraud, these methods aim to combat fraud indirectly by making it unprofitable.

Wang et al. [26] proposed the shill proof fee schedule to make reserve-price shilling unprofitable. The idea is to calculate commission fee as a function of, not the final price, but as the difference between the final price and the seller's reserve price.

Bhargava et al. [27] proposed a bidding strategy SCBS (shill-counteracting bidding strategy) that minimises the amount an honest bidder will pay given the presence of a competitive shill bidder. Evaluation of SCBS against five other bidding strategies showed that bidders using SCBS paid the least in longer auctions, where a shiller has the most time to act. Intuitively, SCBS encourages bidders to stop bidding in an auction instead of being cheated. This reduces the benefit of shilling.

Li et al. [28] proposed a method of determining whether a user is trustworthy in a reputation system where feedback is sparse, and contains potential feedback manipulation. The authors claim the method is resilient against feedback manipulation.

3 Auction simulation

3.1 Introduction

In order to develop and evaluate fraud detection methods, it is necessary to have auction data to operate on. Past work in online auction fraud detection is based on either real data from commercial sites, or synthetic data. For synthetic data, the fraudulent bids, auctions, or users are known. For real data, instances may be labelled as fraudulent using prior knowledge, or using certain criteria that make them seem suspicious.

Both synthetic and real datasets have drawbacks. Synthetic data that is dissimilar to real data will produce models and results that do not extend to the real world. Although it is possible to modify the synthetic data to simulate the real world, this is non-trivial due to the range of possible bidding and selling behaviours and their interactions. Shah et al. [8]

have previously attempted to classify bidders into categories depending on their bidding strategies: evaluate, skeptic, snipe, late bidder and unmask, and found that 88% of users do not fall into any of these categories. However, synthetic data is appealing because the instances and types of fraud can be controlled.

Evaluation using auction data collected from commercial sites may give results that are more representative of the real world than synthetic data. In some cases, a fraction of instances are labelled, perhaps as a result of user complaints. For small datasets, it is possible to manually examine each user to classify them as either fraudulent or non-fraudulent. For larger datasets, manual classification is infeasible. In addition, it is often time-consuming to gather and process data collected from auction sites, as seen in [23] which used an elaborate system to collect auction data from eBay. Also, data may be selectively missing or removed by administrators of the auction site, such as profiles of users that have been identified as fraudsters, or auctions that have been identified afterwards as fraudulent. This further complicates evaluation using real data.

In this chapter, we present an approach which is a middle ground between synthetic and real data, and hope to capture the benefits of both types, while avoiding the drawbacks. We implement an agent-based simulation where agents are modelled on users in real auction data. The simulation can be used to quickly generate realistic auction data. We validate our simulation by performing quantitative evaluations comparing the generated data to commercial TradeMe auction data, and show that the generated and commercial data show a high degree of similarity.

The simulation can be easily extended with additional agents to inject fraudulent behaviour. Since the simulation models users (as opposed to auctions), the synthetic data with injected fraud will be valuable for effective evaluation of fraud detection algorithms which often focus on per-user features.

3.2 Background and related work

Agent-based simulation and modelling (ABM) has been used in many different domains, including traffic flow modelling [29], ecological systems [30], economic systems [31, 32], and social phenomena [33]. Macal and North [34] propose four possible reasons for the widespread use of ABM. First, systems of interest are increasingly complex in terms of their dependencies, so that traditional modelling tools may no longer be sufficient. Second, some systems have always required certain assumptions to be computationally tractable, for example in economic models: perfect information, rationality and long-run equilibrium [34], which are not necessary in ABM. Third, the increase in granularity of data collected provides sufficient information for individual-based simulation. Fourthly, increase in computing power allows simulations to be computed that were previously infeasible.

However, these reasons only state why ABMs are feasible among other modelling methods. The benefits of ABM over other modelling techniques is due to three reasons [35]. First, they are able to capture emergent phenomena. Emergent phenomena are a result of interactions of individual entities in the system, and are most naturally modelled by ABMs, where entities (agents) in the system interact to produce the emergent behaviour. Second, ABM is often the most natural way of describing a system of entities. For example, in traffic flow models, it is more natural to describe individual vehicles, than to develop equations that govern the speed and density of traffic. Third, ABM is flexible, where the number of agents, their behaviour, and the exchange of information can be controlled.

An agent in ABM individually assesses its situation based on available information, and acts based on a set of rules. Different agents may have different roles and behaviours, and interact with other agents based on the rules of interaction. Some agents can also adapt, where their behaviour changes to better achieve certain objectives. Thus, agents can be arbitrarily complex, using neural networks or evolutionary algorithms to learn and adapt, or simple, using only a static set of rules to govern their behaviour.

From the description of ABM and agents, we can see that ABM can be naturally applied to the domain of online auctions. Bidders and sellers are naturally modelled by agents, and online auction sites contain sufficient individual user data for agents to be developed. Assumptions previously used in formal auction analysis such as perfect rationality, perfect market and homogeneous agents are not necessary in ABM. In addition, though the agents themselves may be complex, their observable actions and rules of communication between agents are defined.

One of the earliest works in agent-based simulation in auctions was done by Mizuta and Steiglitz [7], who modelled agents of two types: early bidders and snipers, and investigated the effects of the strategies on the probability of winning, auction price increment, and final auction price. More recently, agent-based modelling of bidders has been used to perform evaluations of fraud related algorithms [27, 17]. However, the models were not based on real auction data, and the model was not validated.

Bhargava et al. [27] developed a bidding strategy to counteract shill bidding in online auctions. To test this strategy, they used an ABM with five agent types based on bidding strategies described by Shah et al. [8], a shill agent, and an agent using the bidding strategy they developed. The performance of each type of agent was then calculated over multiple iterations, where performance is the expected difference between the winning agent's perceived value of the item and that agent's final bid.

Bapna et al. [36] constructed a simulation model of Yankee auctions, a type of multi-item auction. They identified three broad types of bidders according to their bidding strategies: evaluators, participators and opportunists, and constructed a model containing agents that used each strategy. The simulation model is validated by comparing the simulated revenue with the observed revenue from data collected from 21 real auctions. The model was used to investigate the effect of bid increment size on seller revenue, and to find the bid increment that maximises revenue.

Yue et al. [37] applied genetic network programming to allow agents to evolve their bidding behaviour over multiple auction iterations. These agents were used in two auction models, each of which have some similarities with the English auction model. The goal was to observe whether agents would improve their performance over time under different auction conditions, where performance depends on the number of auctions won and the amount of money spent.

We know of no previous work that attempts to generate accurate synthetic online auction data using agent-based modelling.

Resampling methods such as bootstrapping [38] provide an alternative to ABMs for generating datasets from a single real dataset. Variation can be introduced by adding zero centred noise or by changing resampling weights. Resampling would likely be done at the auction level since sampling at the bid level makes it difficult to take dependencies between bids into account. However, it is not obvious how to incorporate different fraudulent behaviours into the resulting samples, making it difficult to apply this approach.

3.3 Auction Model

In this section we present a formal model for English auctions, which consists of a set of users $\mathcal{U}$, a set of auctions $\mathcal{A}$, a set of bids $\mathcal{B}$, and a set of rules that govern the relationship between them. The notation used in this section is also used to describe the features defined in Section 3.5.4. Figure 3.1 gives an overview of the entities and their relationships.

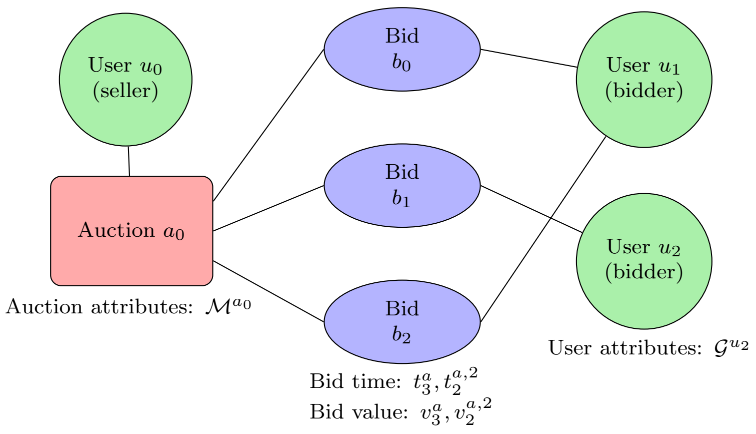

User u_0 (seller)
Bid b_0
User u_1 (bidder)
Bid b_1
Auction a_0
User u_2 (bidder)
Auction attributes: $\mathcal{M}^{a_0}$
Bid b_2
User attributes: $\mathcal{G}^{u_2}$
Bid time: $t^a_3, t^{a,2}_2$
Bid value: $v^a_3, v^{a,2}_2$

Fig. 3.1. Relationship between bids, auctions and users

3.3.1 User Model

A user can list auctions, and participate in auctions by bidding. For user $u \in \mathcal{U}$, let $\mathcal{S}^u = \{m \,|\, m \in \mathcal{A}, m \text{ is listed by } u\}$, where $\mathcal{S}^u$ is the set of auctions listed by u; let $\mathcal{P}^u = \{m \,|\, m \in \mathcal{A},\ u \text{ bids in } m\}$, where $\mathcal{P}^u$ is the set of auctions u participated in; and let $\mathcal{G}^u$ represent a set of attributes associated with user u, such as account age and reputation score.

3.3.2 Auction and Bid Model

An auction can have no bids, one bid, or multiple bids. For auction $a \in \mathcal{A}$, let $\mathcal{B}^a$ be the complete set of bids submitted to a, and let b^a_c, where $1 \leq c \leq |\mathcal{B}^a|$, be the c^{th} bid submitted to a.

Each auction has a set of attributes. The set of attributes includes reserve price (RS), start time (ST), end time (ET) and duration (D). For $a \in \mathcal{A}$, let $\mathcal{M}^a = \{m^a_{RS},\ m^a_{ST},\ m^a_{ET},\ m^a_D\}$ be the set of auction attributes for auction a.

For auction $a \in \mathcal{A}$ and user $u \in \mathcal{U}$, let $\mathcal{B}^{a,u}$ be the complete set of bids submitted to a by u, and let $b^{a,u}_c$, $1 \leq c \leq |\mathcal{B}^a|$ be the c^{th} bid submitted to a by u, and $\mathcal{G}^q_{PosRep}$ and $\mathcal{G}^q_{NegRep}$ be the positive and negative feedback received by q respectively.

3.3.3 Model of Auction Rules

In online auctions, bid values and times increase monotonically. Given b^a_c (the c^{th} bid in auction a), we define t^a_c as the time since the auction start time (w^a_{ST}) that b^a_c was submitted, and let $\mathcal{T}^a = \{t^a_c \,|\, 1 \leq c \leq |\mathcal{B}^a|\}$ be the set of bid submission times for auction a. Correspondingly, given $b^{a,u}_c$ (the c^{th} bid by user u in auction a), let $t^{a,u}_c$ be the time since the auction's start that $b^{a,u}_c$ was submitted. Let $\mathcal{T}^{a,u} = \{t^{a,u}_c \,|\, 1 \leq c \leq |\mathcal{B}^a|\}$ be the set of submission times for bids by user u in auction a.

Given $b^{a,u}_c$, let $v^{a,u}_c$ be the value of $b^{a,u}_c$, and $\mathcal{V}^{a,u} = \{v^{a,u}_c \,|\, 1 \leq c \leq |\mathcal{B}^a|\}$ be the set of values of bids submitted by user u in auction a.

According to the mechanics of online auctions:

a) The time and value of each bid submitted must be greater than the previous bid in the auction, if one exists. That is, $\forall a \in \mathcal{A},\, x \in \mathbb{N}, (t_x^a,\, t_{x-1}^a) \in \mathcal{T}^a \rightarrow t_x^a > t_{x-1}^a, (v_x^a,\, v_{x-1}^a) \in \mathcal{V}^a \rightarrow v_x^a > v_{x-1}^a$.

b) An auction accepts no new bids after the auction has ended. That is, $\forall a \in \mathcal{A}, t_c^a \in \mathcal{T}^a \rightarrow t_c^a \leq w_{ET}^a$.

c) The winner of an auction is the bidder who made the last bid in the auction, given that the reserve price is met. That is, for $a \in \mathcal{A}$, let $\mathcal{H}^a = (u \in \mathcal{U} \mid b_{|\mathcal{B}^a|}^a \in \mathcal{B}^{a,u},\, v_{|\mathcal{B}^a|}^a > a_{RP})$ be the winner of auction a.

Finally, let the set of auctions won by user $u \in \mathcal{U}$ be $\mathcal{W}^u = \{a \mid a \in \mathcal{P}^u, \mathcal{H}^a = u\}$ and let the set of auctions lost by user u be $\mathcal{L}^u = \{a \mid a \in \mathcal{P}^u, \mathcal{H}^a \neq u\}$.

3.4 Auction simulation overview

Our data generator is an agent-based simulation of online English auctions, the auction type used in online auction sites such as eBay[1] and TradeMe[2]. For more details on the English auction and other auction types, see [3]. Auction data is generated by running the simulation and recording the actions of all of the agents, such as auctions listed and bids made. By carefully specifying the behaviour of the agents, we can generate synthetic auction data that closely approximates real auction data.

We choose to use per-user features instead of per-auction features because the simulation is used to evaluate fraud related algorithms which tend to use per-user features instead of per-auction features [27, 23, 17].

There are three main stages in the implementation of the simulation:

Stage 1. The framework for an agent-based auction simulation is written in Java with the following parts:

[1]http://www.ebay.com/
[2]http://www.trademe.co.nz/

1. An auctioneer agent that enforces the auction rules. The rules are based on the auction mechanism of TradeMe, however, they can be easily changed to model other auction sites, such as eBay.
2. A collection of agents, each of which behave independently according to a set of rules. Each agent represents a bidder or seller. Agents communicate with an Auctioneer entity to submit bids or list auctions.

Stage 2. Auction and user data is collected from TradeMe. The data is cleaned, and the sequence of auctions and bids transformed into a set of features for each user.

Stage 3. We iteratively modify the rules governing agents according to patterns identified in the TradeMe data. Examples of rules are given in Section 3.5.3. Iteration continues until the synthetic data is similar to the collected data. The two measures of similarity used is described in Section 3.6.

Each of these stages is described further in the Sections 3.4.2 to 3.5.3.

3.4.1 Generated auction example

Table 3.1 gives an example of an auction generated by the simulation. Table 3.1a lists the bids made for that auction. Table 3.1b gives information about the auction. *Bid Time*, *Start Time*, and *End Time* represent time units since the beginning of the simulator execution, with each unit representing five minutes. *Duration* is the number of time units the auction runs for.

There are 8 bids made for this auction, and User 2285 wins the item for $40.61 at time 2026. Note that *Duration* is less than the difference between the start and end times because the auction was extended due to a bid submission in the last 3 time units.

3.4.2 Simulation execution over one time unit

Figure 3.2 shows events over one time unit in the simulation. Each time unit is split into two stages: an Auctioneer phase (A), and an Agent phase (B). During the Auctioneer

Table 3.1
Example of a generated auction

(a) Auction bids

BidID	UserID	BidTime	BidAmount (cents)
1	801	871	2.50
2	1820	1710	23.61
3	801	1735	26.61
4	2910	1963	27.61
5	1820	1998	37.61
6	2285	2014	38.61
7	801	2025	39.61
8	2285	2026	40.61

(b) Auction information

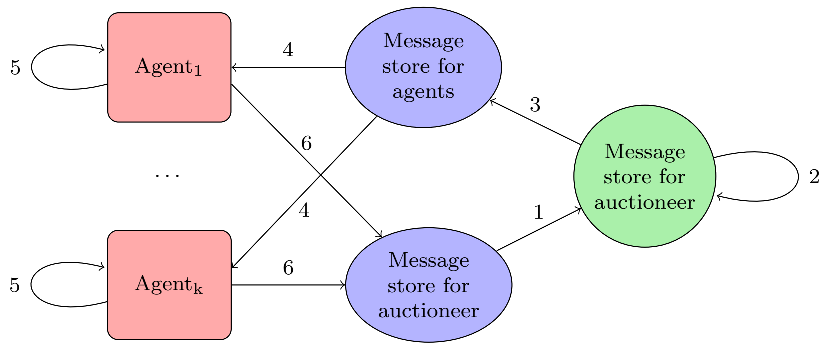
Attribute	Value
SellerId	3000
ListingId	0
StartTime	11
EndTime	2029
Duration	2016
ItemValuation (cents)	39.35
itemTypeId	1
startPrice (cents)	1.00

phase, the Auctioneer sends and receives messages, and enforces auction rules. All other agents are inactive. The messages sent by the auctioneer are announcements for agents. The messages may be for new auctions, auctions updated by new bids from other agents, or for the results of auctions that have terminated. During the agent phase only agents act, and the auctioneer is inactive. Agents retrieve the messages from the auctioneer, decide on what actions to take, and send any new messages to the auctioneer, such as to list new auctions or to submit bids to existing auctions. The pseudocode for simulation execution is given in Section A.1 in Appendix A.

3.4.3 Controlling agent behaviours

The behaviour of agents is governed by a collection of rules. These rules are applied at different moments, such as at the beginning of each time unit to decide what actions to perform, and when. This is shown in Table 3.2. Each agent acts on the information available locally, such as the current auction state, current time, and its previous actions. Agents do not have a global view of the system. To craft the agent rules, we began initially with very simple rules, then added complexity to the rules until the resulting synthetic data has an empirical distribution similar to that of the real data from TradeMe, based on 10 user features defined below in Section 3.5.4.

The parameter for a specific rule for a group of agents can be parameterised using a probability distribution. Different values drawn from the distribution contributes to the variation in agent behaviour. By controlling the statistical variance of the distributions, we



A. Beginning of Auctioneer phase

1. Retrieve messages from Agents
2. Process messages
   a. update auction states, terminate expired auctions
   b. ensure auctions and bids have valid states before and after update
3. Send messages (current time, new auction, end of auction, price change messages)

B. Beginning of Agent phase

4. Retrieve and process messages from Auctioneer
5. Decide on what actions to take, if any
6. Send messages (new auctions or bids)

Fig. 3.2. Events during the execution of the simulator in one time unit

can also control the variation seen in agent behaviour. For example, an agent generates a perceived value v for all items it bids on. This value is drawn from a normal distribution. The probability of an agent submitting a bid to that auction correlates with the ratio between the current price and the item's perceived value, so that as price increases, probability of bidding decreases. Changing how this perceived value is distributed for a group of agents results in different synthetic data, where agents with a high perceived value are more likely to win. The pseudocode for the bidder agent implementation is given in Section A.2 in Appendix A.

Table 3.2
Bidding agent decisions

Moment	Decision	Factors considered
Beginning of time unit	Whether to begin participating in an existing auction	The agent's auction participation frequency; agent's interest in the item category; time remaining for auction; number of previous auctions won
Received outbid notifications	Whether to rebid in running auctions	Current price vs. perceived value of the item,

3.5 Commercial dataset

Commercial data is used to construct the rules for simulation agents and to select their parameters. This section describes the method for gathering and storing the auction data, the cleaning procedure, and the limitations of the dataset.

3.5.1 Data collection and storage

Data is collected from the TradeMe online auction site using a web crawler. There are two types of web pages of interest: the auction history page, and the completed auction page. Each auction history page contains information about a particular user, and a list of links to previous auctions the user participated in. Each of these links lead to an auction history page, which contains information about a completed auction; and a table of bids made by different users in that auction. Each row of the table contains a link to the completed auction page of the user who submitted the bid. The page also contains a link to the auction history page for the seller. This is shown in Figure 3.3.

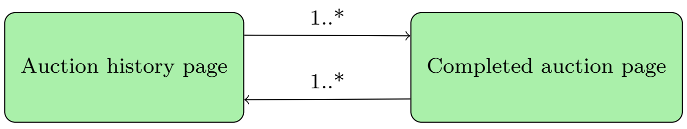


Fig. 3.3. Links between the auction history page and the completed auction page

To collect information about users and their bidding history, we simply select a random auction history page, and follow all links to either page types. This is done in a breadth first order, so that all of the available information for a user is collected before we move on to another user. From auction history pages, we extracted the number of positive, neutral and negative feedback each user received, and the number of auctions they participated in.

From completed auction pages, we extracted seller name, reputation, bid times and bid values, and the item name. This information is stored in a database, with the database schema shown in Figure 3.4.

There are several limits to the information available for collection. Firstly, only the most recent 20 bids are retrievable for a completed auction. Secondly, bid histories for auctions are available only for 60 days after it has completed. Thirdly, auctions that were not sold successfully are not displayed in a user's history. Fourthly, the user pages for removed accounts cannot be viewed. The reason for removal is not given, but a possibility is that those accounts belong to identified fraudsters. This may be an advantage since we only wish to model legitimate users; and removal of suspicious users will improve data quality.

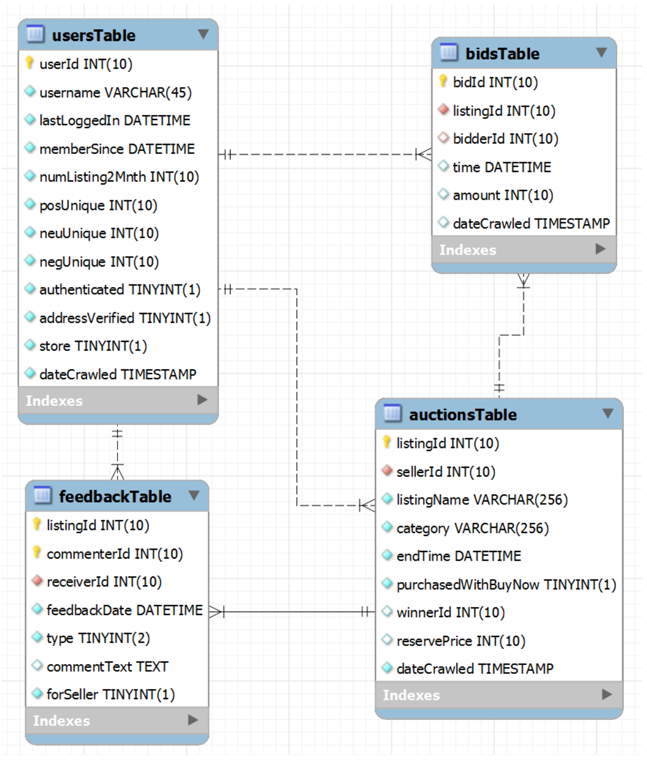


Fig. 3.4. Tables used to store auction data

Dataset cleaning

During the cleaning process, we identified auctions and users who were not useful for modelling user bidding behaviour. Items that were sold using "buy now" were removed, since the auction process did not occur. Users who did not have an accessible auction history page were not included during the analysis of users, though their bids were retained during the calculation of user features for other users. We excluded users for whom we did not collect all available history. We also excluded certain categories of auctions, such as vehicle and real estate, most of which did not go through an auction process. UserIds were also anonymised at this stage.

3.5.2 Dataset statistics

Two datasets were collected from TradeMe over two different time periods. The first, smaller dataset was collected during October 2011. The second was collected during August 2012. The Oct 2011 dataset is used to implement the simulation agent models in this section, while the Aug 2012 dataset is used for evaluation in Section 3.6. Table 3.3 shows the number of users, bids and auctions before and after cleaning for each of the datasets. Table 3.4 shows the number of users who acted as bidders only, who won one or more auctions, or acted as sellers only, or combinations thereof. Table 3.5 shows the number of users with net positive or negative feedback scores, and the total number of positive and negative feedback received by users. As shown, the amount of negative feedback given is extremely low. This may be partially due to the removal or abandonment of accounts that have high numbers of negative feedback.

Table 3.3
Dataset statistics

Dataset date		User count	Bid count	Auction count
Oct 2011	Before cleaning	93197	351878	161895
	After cleaning	66862	298938	54630
Aug 2012	Before cleaning	256889	1040745	668017
	After cleaning	113411	703645	153239

Table 3.4
User behaviour counts

Dataset date	Bid only	Win only	Sell only	Bid and sell	Win and sell
Oct 2011	49104	19067	12911	4847	3304
Aug 2012	80290	53212	21796	11325	8965

Table 3.5
Reputation statistics

(a) Net reputation counts

Dataset Date	Net positive	Neutral	Net negative
Oct 2011	66856	6	0
Aug 2012	113194	194	23

(b) Total feedback counts

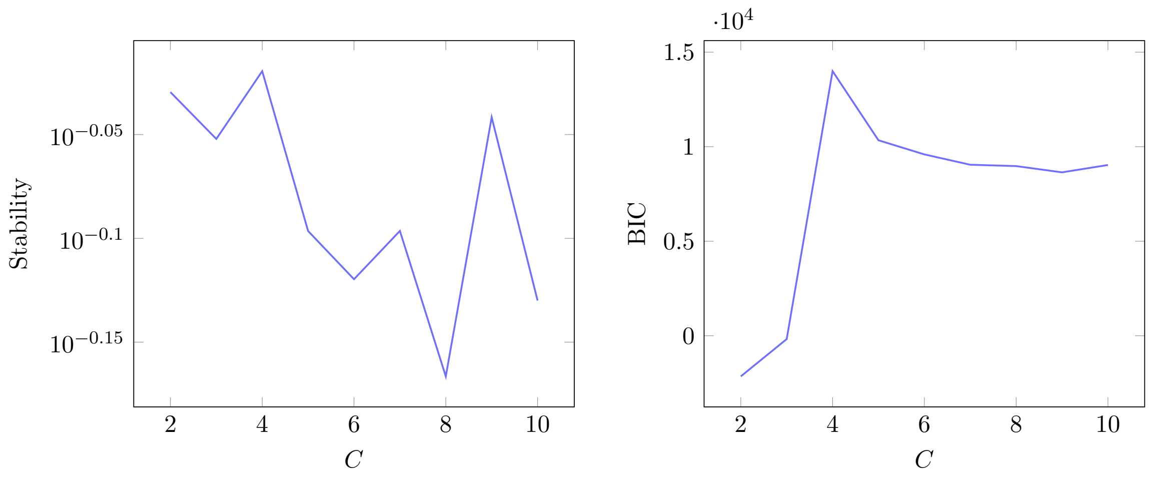
Dataset Date	Total positive	Total negative
Oct 2011	20486893	94608
Aug 2012	35338601	167904

3.5.3 Modelling bidder-agent behaviour

To understand the general patterns in user behaviour, the sequence of bids and auctions of each user in the Oct 2011 dataset was transformed into a set of 10 continuous attributes defined in Section 3.5.4. Each feature summarises an aspect of the users' bidding pattern. To identify patterns, the 10-feature vectors were clustered. Two methods were used to find the appropriate number of clusters C. The first is a stability based measure proposed by Lange et al. [39] to measure the appropriateness of C by calculating the stability of cluster labels over multiple runs of a clustering algorithm. The clustering algorithm we used was simple k-means, testing values of C between 2 and 10. The highest stability was observed when C was 4, as shown in Figure 3.5(a).

The second method used was X-means [40], which is an extension of the k-means algorithm where the Bayesian information criterion measure (BIC) is used to select the best model given a range of C. The BIC measures the probability of observing the model given the data, and includes a penalty for the number of parameters (centroids) in the model. Since X-means relies on k-means, which is sensitive to initialisation conditions, BIC was calculated for 30 iterations of X-means. Figure 3.5(b) shows the median BIC values over 30 runs of the X-means algorithm at each C. As before, the best model found by X-means had 4 clusters.

We then identified patterns from the cluster model. For example: a significant number of users only bid close to the end of the auction and rebid quickly after they have been outbid; users who bid close to the end win more often; and users who bid early and win are often the only bidder in an auction. Bidder agents are given behaviours that allow the synthetic data to contain these same patterns. For example, for the last pattern, the rule may be to increase the likelihood of entering a bid as the number of existing bids in



(a) Median stability value **(b)** Median BIC value for X-means

Fig. 3.5. Quality of clustering for C between 2 and 10

the auction increases, which leaves a fraction of auctions to have only one participating bidder. These identified patterns are used to design the agent rules and to select their parameters.

However, it should be noted that the user clusters cannot be directly translated into agent types or agent rules. This is because some of the patterns emerge from agent interactions, and cannot be directly controlled using rules. For example, the proportion of auctions won by a particular agent depends on the behaviour of other agents. Therefore, the clusters can only provide guidance to what the rules should be in order to observe similar feature-value distributions to that of the collected dataset.

3.5.4 User features

It is possible to view the sequence of bids and auctions in different ways to calculate features, such as per-auction, and per-user. Below, we defined per-user features, though it is also possible to define per-auction features. However, it is more difficult to understand user behaviour in order to implement the agent-based simulation using per-auction features, since they cannot capture the dependency between user behaviour across different auctions.

Below, we describe the ten user features into which the collected data is transformed.

Many of the features described here have been used in previous literature described in Section 3.2 for modelling users, identifying bidder behaviour ([36, 8]) and for detecting different types of auction fraud, ([16, 25, 2]). The 10 features, labelled ϕ_1 to ϕ_{10}, are defined for the user $q \in \mathcal{U}$, where $|\mathcal{P}^q| > 0$:

Definition 1 ϕ_1, or bid amount, gives the average value of bids user q made in an auction, averaged over all auctions participated in, $\mathcal{P}^q$.

$$\phi_1 = ln\left(1 + \frac{1}{|\mathcal{P}^q|}\sum_{a\in\mathcal{P}^q}\frac{1}{|\mathcal{B}^{a,q}|}\sum_{b\in\mathcal{B}^{a,q}} b\right) \tag{3.1}$$

For example, if the user made bids with value {\$4.00, \$10.00}, and {\$5.00} in only 2 auctions, then $\phi_1 = ln(1+6)$.

Definition 2 ϕ_2, or excess bid increment, gives the average increment of the user's bid over the preceding bid, after subtracting the required minimum increment.

Let $minVal : \mathbb{N} \to \mathbb{N}$ be a function mapping a bid value to the minimum allowed increment. For example, $minInc(\$20.00) = \$0.80$.

Let $prevVal(v_c^{a,q}) = v_d^a$, where $v_c^{a,q} \in \mathcal{V}^a$, $v_d^a \in \mathcal{V}^a, a \in \mathcal{A}$, $2 \leq c \leq |\mathcal{B}^a|$, and $v_c^{a,q} = v_{d+1}^a$ be a function that maps from a bid's value $(v_c^{a,q})$ to the value of the bid immediately preceding it (v_d^a) in the same auction a. Then,

$$\begin{aligned}\phi_2 = ln\Bigg(1 + \frac{1}{|\mathcal{P}^q|}\sum_{a\in\mathcal{P}^q}\frac{1}{|\Delta V^{a,q}|}\sum_{\Delta v_c^{a,q}\in\Delta V^{a,q}} \\ \Delta v_c^{a,q} - minVal(prevVal(\Delta v_c^{a,q}))\Bigg)\end{aligned} \tag{3.2}$$

For example, if the average increments were {\$4, \$8} and {\$5} in two auctions, then $\phi_2 = ln(1 + 5.5)$.

Definition 3 ϕ_3, or win proportion, is the number of auctions a user won divided by the number of auctions the user participated in.

$$\phi_3 = \frac{|\mathcal{W}^q|}{|\mathcal{P}^q|} \tag{3.3}$$

For example, a user winning 2 out of 5 auctions will have $\phi_3 = 0.4$.

Definition 4 ϕ_4, or bids per auction, is the average number of bids made per auction by the user.

$$\phi_4 = ln\left(\frac{1}{|\mathcal{P}^q|}\sum_{a\in\mathcal{P}^q}|\mathcal{B}^{a,q}|\right) \tag{3.4}$$

For example, if a bidder made 2 and 6 bids in 2 auctions respectively, then $\phi_4 = ln(4)$.

Definition 5 ϕ_5, or bid time, gives the user's average bid time as a fraction of auction time elapsed, averaged over auctions $\mathcal{P}^q$.

Given an auction $a \in \mathcal{A}$, let $timeFrac(t_c^{a,q}) = (t_c^{a,q} - m_{ST}^a)/m_D^a$, where $t_c^{a,q} \in \mathcal{T}^{a,q}$ is a function $\mathbb{N} \mapsto [0-1]$ that returns the proportion of auction time elapsed for a given bid-submission time. For example, if the auction beginning at time 1,000 has duration 2000, then a bid submitted at time 1500 would have $timeFrac = 0.25$.

$$\phi_5 = \frac{1}{|\mathcal{P}^q|}\sum_{a\in\mathcal{P}^q}\frac{1}{|\mathcal{T}^{a,q}|}\sum_{s\in\mathcal{T}^{a,q}} timeFrac(s) \tag{3.5}$$

For example, $\phi_5 = 0.71$ for user who made only one bid 5 days after the start of a 7-day auction.

Definition 6 ϕ_6, or bid amount proportion, gives the user q's bid values as a proportion of the maximum (final) bid in that auction, averaged over auctions $\mathcal{P}^q$.

$$\phi_6 = \frac{1}{|\mathcal{P}^q|}\sum_{a\in\mathcal{P}^q}\frac{1}{|\mathcal{V}^{a,q}|}\sum_{r\in\mathcal{V}^{a,q}}\frac{r}{v^a_{|\mathcal{B}^a|}} \tag{3.6}$$

For example, if a user made one bid in one auction with a bid valued at \$6, and the highest bid was \$10, then $\phi_6 = 0.6$.

Definition 7 ϕ_7, or bid proportion, gives the number of bids made by the user divided by the total number of bids submitted in an auction, averaged over auctions $\mathcal{P}^q$.

$$\phi_7 = \frac{1}{|\mathcal{P}^q|} \sum_{a \in \mathcal{P}^q} \frac{|\mathcal{B}^{a,q}|}{|\mathcal{B}^a|} \tag{3.7}$$

For example, if a user made 3 bids out of a total of 5 bids in a single auction, then $\phi_7 = 0.6$.

Definition 8 ϕ_8, or auction count, is the number of auctions the user participated in.

$$\phi_8 = ln\left(\mathcal{P}^q\right) \tag{3.8}$$

Definition 9 ϕ_9, or net reputation, is the difference between the number of positive and negative feedback.

$$\phi_9 = \begin{cases} ln\left(\mathcal{G}^q_{PosRep} - \mathcal{G}^q_{NegRep} + 1\right) & \text{if } \mathcal{G}^q_{PosRep} > \mathcal{G}^q_{NegRep} \\ 0 & \text{if } \mathcal{G}^q_{PosRep} = \mathcal{G}^q_{NegRep} \\ ln\left(\mathcal{G}^q_{NegRep} - \mathcal{G}^q_{PosRep} + 1\right) & \text{if } \mathcal{G}^q_{PosRep} < \mathcal{G}^q_{NegRep} \end{cases} \tag{3.9}$$

Definition 10 ϕ_{10}, or first bid time, is the average time the first bid was made by the user in each auction.

$$\phi_{10} = ln\left(\frac{1}{\mathcal{P}^q} \sum_{a \in \mathcal{P}^q} t_0^{a,q}\right) \tag{3.10}$$

Some features are log-transformed so that values are spread out more evenly during clustering, since many values are very similar for some features.

Some features, such as ϕ_8 (auction count), ϕ_4 (bids per auction), and ϕ_{10} (first bid time), can be directly used in rules for bidder-agents. Other features, such as ϕ_6 (bid amount proportion), ϕ_3 (win proportion), and ϕ_7 (bid proportion), cannot, since their values depend on the behaviour of other agents. However, those features can be used to determine whether synthetic data from the generator is sufficiently similar to the TradeMe data.

Other features were also considered. These features include the standard deviations of the features selected, and other features based on user bidding times relative to other bids. They are defined in Appendix B, and the correlation matrix for all features is shown in Table B.1. The features which were not used have a high correlation with one or more other features. For each set of highly correlated features, the simplest is chosen. For the set of chosen features, the highest correlation is 0.68 between ϕ_3 and ϕ_7, the lowest is 0.01, and the average pairwise correlation is 0.058. The correlation matrix for the 10 chosen features is shown in Table 3.6. The Pearson's correlation measure is used.

Table 3.6
Feature correlation matrix

	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6	ϕ_7	ϕ_8	ϕ_9	ϕ_{10}
ϕ_1	1	0.26	0.1	0.34	0.14	0.42	0.01	-0.01	-0.06	-0.21
ϕ_2	0.26	1	-0.04	0.15	-0.02	-0.03	0.05	0.1	0.02	0.07
ϕ_3	0.1	-0.04	1	-0.09	-0.09	**0.54**	**0.68**	0.21	0.12	-0.16
ϕ_4	0.34	0.15	-0.09	1	0.29	-0.06	-0.03	0.01	-0.03	-0.1
ϕ_5	0.14	-0.02	-0.09	0.29	1	0.09	-0.39	0.04	0.02	-0.42
ϕ_6	0.42	-0.03	**0.54**	-0.06	0.09	1	**0.54**	0.03	0.07	-0.38
ϕ_7	0.01	0.05	**0.68**	-0.03	-0.39	**0.54**	1	0.16	0.12	-0.04
ϕ_8	-0.01	0.1	0.21	0.01	0.04	0.03	0.16	1	0.25	-0.01
ϕ_9	-0.06	0.02	0.12	-0.03	0.02	0.07	0.12	0.25	1	-0.03
ϕ_{10}	-0.21	0.07	-0.16	-0.1	-0.42	-0.38	-0.04	-0.01	-0.03	1

3.5.5 Modelling network features

The simulation also models network features of the collected auction data. Specifically, we modelled the agents so that the distribution of the numbers of total and unique user interactions that each user has with another user matches users in the collected data. This is done by modifying bidder agent behaviour so that they are more likely to participate in the auctions of sellers they have interacted with previously.

3.6 Evaluation

We validate the simulation by comparing the resulting synthetic data to the Aug 2012 dataset. To compare the datasets, both are transformed into features previously defined in Section 3.5.4. Three approaches are used to evaluate the similarity of the transformed datasets. First, we compare the value distributions for each feature separately in Section 3.6.1. Second, we use a statistical method called multi-response permutation procedure (MRPP) for comparing multivariate distributions in Section 3.6.2. Third, we compare the distributions of multiple features at once using a clustering-based method in Section 3.6.3. We also compare the number of total and unique user interactions in Section 3.6.4, and report the execution time of the simulation in Section 3.6.5.

It should be noted other tests could be used in Sections 3.6.1 to 3.6.3. For single feature distribution tests, there are the parametric unpaired Student's t-test [41], and the non-parametric Kolmogorov-Smirnov test [42]. For comparing multivariate distributions, there exists the cross-match test [43]. For the clustering based comparison, different clusterer/classifier pairs can be used as noted in [39]. Below, we use k-means with k-nearest neighbour. The chosen classifier should 'mimic' the clustering algorithm to produce reliable results [39]. The relative merit of different methods is beyond the scope of this thesis, and the choice of, for example, MRPP over cross-match is simply because MRPP has been more commonly used in the literature.

3.6.1 Single feature distribution comparison

Two correlation measures are used to calculate whether the distributions of the feature values from the synthetic data match the real data. A high correlation would indicate that the synthetic data is similar, and that the simulation model, is accurate. The two correlation measures used are Pearson's correlation coefficient and Spearman's rank correlation coefficient [44]. Pearson's correlation is a parametric test, while Spearman's rank correlation is a non-parametric test.

For Pearson's correlation coefficient, a value of $+1$ or -1 indicates that a linear equation describes the relationship between the two variables perfectly. For Spearman's rank correlation coefficient, $+1$ or -1 occurs when each of the variables is a perfect monotone function of the other. For both, a value close to $+1$ shows that the TradeMe and synthetic data are similar.

Table 3.7
Correlation results for all features

Feature	Pearson's correlation		Spearman's rank correlation
	95% CI	Correlation	ρ
Auction Count	0.9468 - 0.9485	0.9476	0.9625
Reputation	0.9379 - 0.9399	0.9389	0.99997
Bid Amount	0.5207 - 0.5329	**0.5269**	0.9999
Excess Bid Increment	0.8545 - 0.8590	**0.8567**	**0.8497**
Bids Per Auction	0.9702 - 0.9711	0.9706	0.9930
First Bid Time	0.9889 - 0.9892	0.9890	0.99998
Bid Time	0.9701 - 0.9710	0.9706	0.9967
Win Proportion	0.9752 - 0.9760	0.9756	0.9515
Bid Amount Proportion	0.9873 - 0.9877	0.9875	0.9987
Bid Proportion	0.9841 - 0.9846	0.9844	0.9981

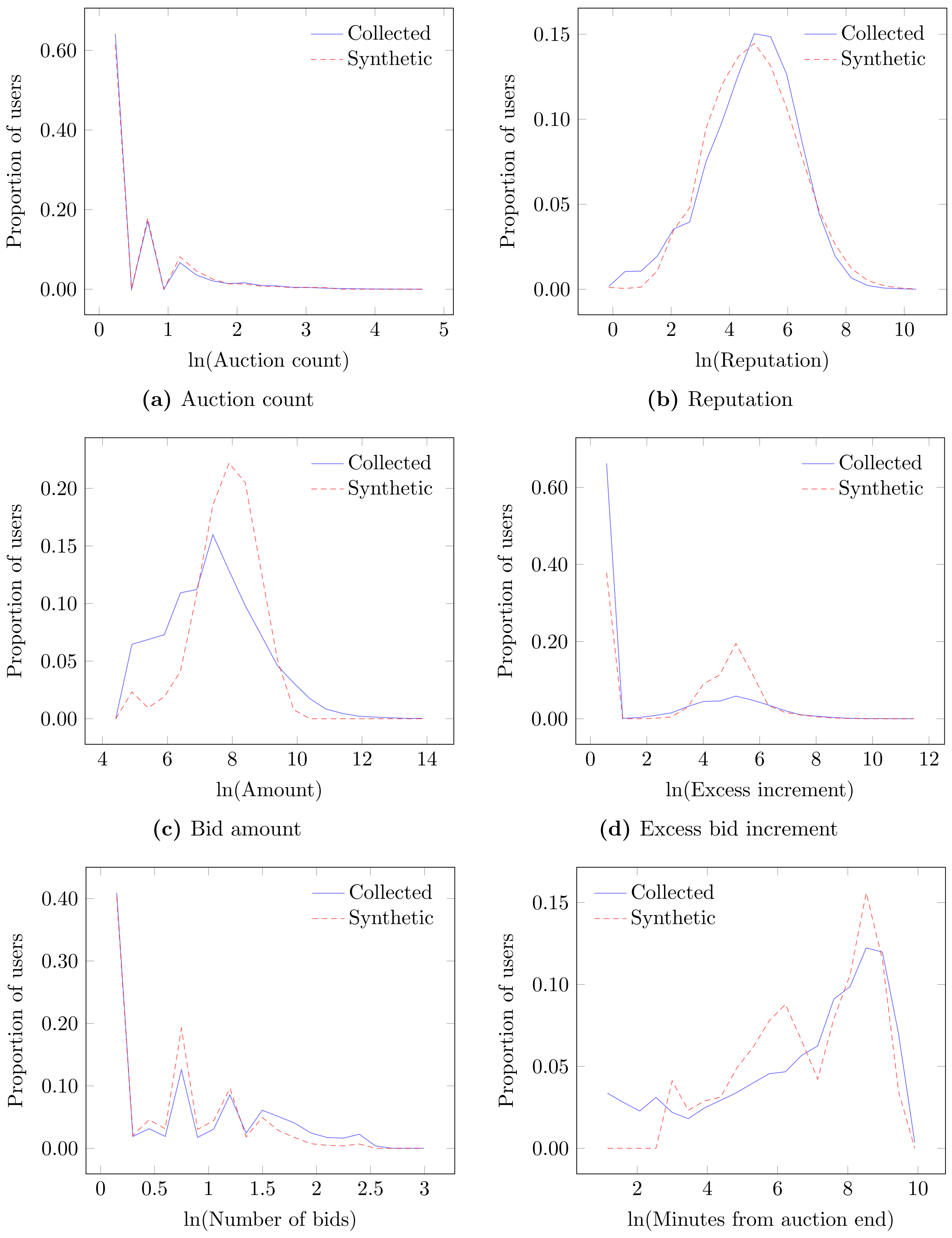
Evaluation results using individual features

Figure 3.6 shows the distribution of values of the 10 user features for both the synthetic and TradeMe datasets. To construct the figures, all values from the 30 synthetic datasets were used, giving 111677 values for each feature. Values are split into 20 equidistant bins using the maximum and minimum values for that feature. The plots reflect the proportion of users that have a value that falls into a particular bin. We see that distribution of values for the 30 synthetic datasets generally match those from the TradeMe dataset.

Table 3.7 shows the correlation between synthetic and TradeMe data for each feature as measured by Pearson's correlation coefficient and Spearman's rank correlation coefficient. The correlation values for both measures are greater than 0.9 for all features, with the exception of Bid Amount, with a Pearson's correlation of 0.53 and Excess Bid Increment with 0.86. The low similarity for Bid Amounts and Excess Bid Increment is due to the lack of information on the perceived and actual value of items, making it more difficult to accurately model bid values. As a whole, the high correlation values indicates that the value distribution of features from our generated data is similar to TradeMe data.

3.6.2 Multivariate dataset comparison

The multiple response permutation procedure (MRPP) is a non-parametric procedure for testing the null hypothesis of no difference between groups [45]. Given a labelled dataset and a distance measure, the outputs of MRPP are an A-value, which gives the effect size, and a p-value, estimated using Monte-Carlo simulations. The A-value reflects the



(a) Auction count

(b) Reputation

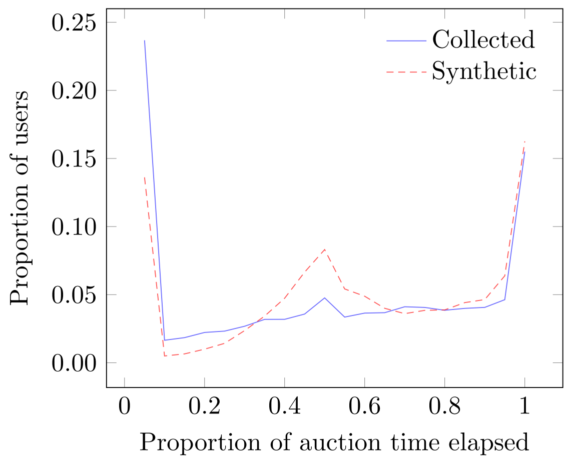
(c) Bid amount

(d) Excess bid increment

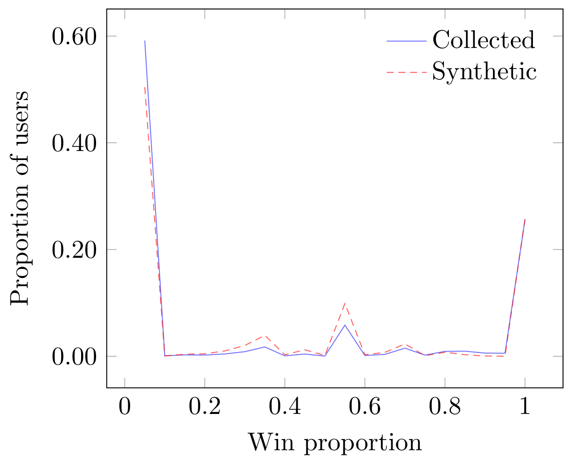
(e) Bids per auction

(f) First bid time

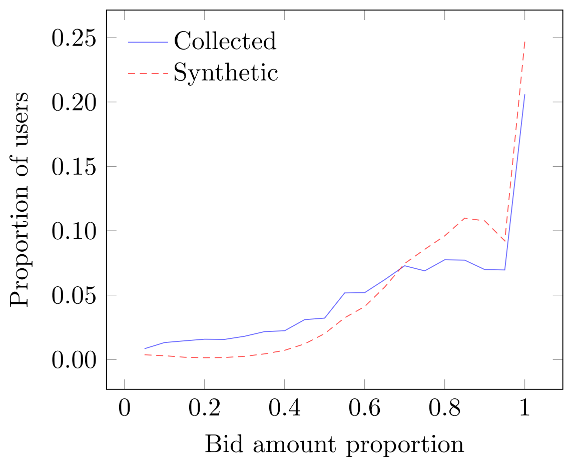
Fig. 3.6. Value distributions of features



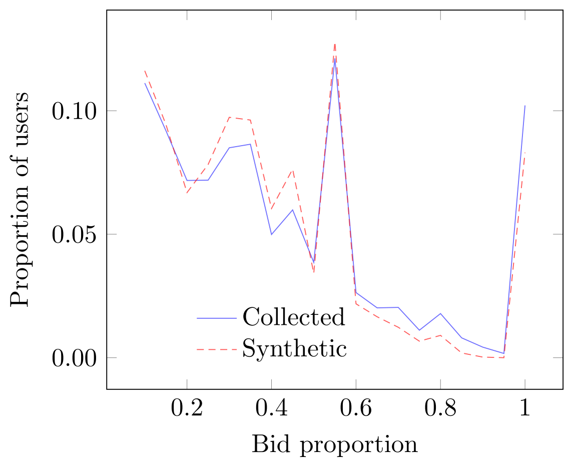
(g) Bid time



(h) Win proportion



(i) Bid amount proportion



(j) Bid proportion

Fig. 3.6. Value distributions of features (cont.)

difference between the observed and the expected (that is, if instance labels were ignored) average distance between instances. Intuitively, an A-value of 0 occurs when there is no difference between each group, while a value of 1 occurs when all instances are identical within each group, and at least two groups do not overlap. For more details on MRPP refer to [45]. The distance metric used here is the range-normalised euclidean distance, shown in Equation 3.11,

$$d[j,k] = \frac{1}{M}\sqrt{\sum_{i=1}^{M} \frac{(x[i,j] - x[i,k])^2}{(max(x[i]) - min(x[i]))^2}} \tag{3.11}$$

where M is the number of attributes, N is the number of instances, x is a N by M matrix, i is the attribute index, j and k are two different instance indices. $x[i, j]$ is the value of the ith attribute of instance j, and $x[i]$ is the ith column of x.

The synthetic data generated by the simulation is compared to the collected data. MRPP was used to compare each of 20 sets of synthetic data to the collected data. Using 999 permutations, an average p-value of 0.001 indicates there is a statistically significant difference between the two groups. However, the effect size is very small, as shown by the A-values in Table 3.8, thus practically, difference in the distributions is negligible. Two sets of comparisons were performed, using all 10 features described previously, and using the 4 features used in Section 3.6.3. Thus we are confident in concluding that the synthetic and collected data are sufficiently similar.

Table 3.8
MRPP A-values for dataset comparison

	μ	σ
4 features	0.0098	0.0014
10 features	0.0067	0.0009

3.6.3 Clustering based dataset comparison

Lange et al. [39] proposed a method to solve the model order selection problem in cluster analysis; that is, to find a suitable number of clusters C. They state that a cluster solution should be robust: that the clusters should be reproducible using other datasets drawn from the distribution, and not depend on the particular sample [39]. The degree to which the cluster solution is reproducible indicates whether a given C is appropriate for a dataset. However, since only one dataset is usually available, the dataset is split into two, and treated as two samples.

Conversely, if the number of clusters for a dataset is given, the degree of reproducibility gives an indication of the similarity of the second dataset to the first. We adapt the method proposed by Lange et al. to evaluate the similarity between the synthetic and real auction data. This requires two small changes to the original method. First, there is no need to normalise results according to the number of clusters since the number remains unchanged. Second, two different datasets are used (synthetic and real) instead of partitioning a single dataset into two distinct subsets.

For evaluating the similarity between datasets, the equation for measuring the normalised Hamming distance between two labelling vectors used in [39] can be slightly modified to

$$d(\gamma(X'), Y') = \frac{1}{n}\sum_{i=1}^{n} I\{\gamma(X'_i) = Y'_i\} \tag{3.12}$$

where $I\{\gamma(X'_i = Y'_i)\} = 1$, if $\gamma(X'_i) = Y'_i$. The equation then gives the proportion of instances that are given the same labelling via the classifier trained on X (which in our case is the TradeMe dataset) and the clustering solution Y' for X' (the synthetic dataset). Since the corresponding clusters may be assigned different labels in two different clustering solutions even if the datasets are the same, agreement is maximised using the Hungarian method as in [39].

We used the simple k-means algorithm for clustering the TradeMe and synthetic data into four clusters. Clustering the TradeMe dataset X, gives a set of labels Y, one for each instance. Together, (X, Y) is used to construct a closest neighbour classifier γ. Clustering the synthetic dataset X' gives a set of labels Y'. The classifier γ is then used to classify instances in X', giving the labels $\gamma(X')$. With Y' and $\gamma(X')$ we can calculate the accuracy using Equation 3.12.

Features used

We used four features for evaluating the similarity between the synthetic and TradeMe data: Proportion of Max Bid, Win Proportion, Bids Per Auction, Bid Proportion. These four features were chosen because the others:

1. Have almost the same values for all users, e.g. excess bid increment as shown in Figure 3.6(d), and are less useful for clustering.

2. Are identically distributed regardless of their cluster assignments, e.g. reputation, and bid amount are not useful for clustering.

3. Are known not to closely approximate real data (features i and j), as discussed in Section 3.7.1.

Evaluation results using multiple features

We evaluated our simulator using 30 synthetic datasets, each with roughly 3700 users. We report the results for evaluations using random initial centroids for simple k-means, and using manually defined centroids.

Using 50 sets of random initial centroids for evaluating each synthetic dataset, the average accuracy is 0.745 with a standard deviation of 0.0957. Using the first set of centroids shown in Table 3.9, the average accuracy is 0.878, an increase of 0.143, with a standard deviation of 0.0282.

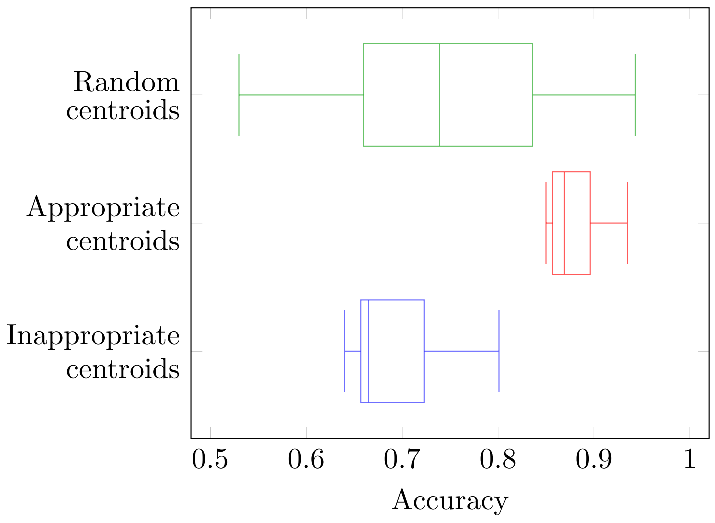


Fig. 3.7. Accuracy using different centroids for clustering using four features: Bid Amount Proportion, Win Proportion, Bids Per Auction, Bid Proportion

Table 3.9
Selected centroid centers

	Centroid ID	Bid amount proportion	Win proportion	Bids per auction	Bid proportion
Appropriate centroids	1	0.8	0.0	0.0	0.2
	2	0.8	0.0	0.0	0.2
	3	1.0	**1.0**	0.0	0.5
	4	1.0	1.0	0.0	1.0
Inappropriate centroids	1	0.8	0.0	0.0	0.2
	2	0.8	0.0	0.0	0.2
	3	1.0	**0.0**	0.0	0.5
	4	1.0	1.0	0.0	1.0

Table 3.10
Accuracy using different centroids

	Random centroids	Appropriate centroids	Inappropraite centroids
Max	0.943	0.935	0.801
Upper Quartile	0.836	0.896	0.723
Median	0.739	0.869	0.665
Lower Quartile	0.660	0.857	0.657
Min	0.530	0.850	0.640

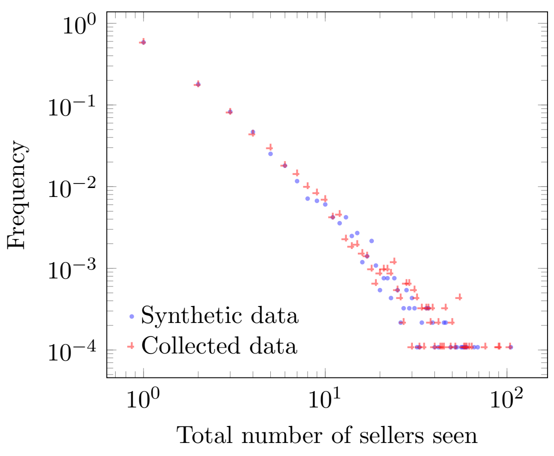
Centroid Selection

The k-means algorithm is sensitive to the centroids chosen during initialisation [46], which in turn heavily influences the result given by Equation 3.12. In our case, this problem may be worsened by characteristics of our dataset. First, clusters are not well defined in the TradeMe data, which means clusters will tend to grow from their initial centroid position. Secondly, the Win Proportion feature is almost binary, where the majority of users have a value of zero or one. Given four initial centroids, there are five configurations the centroids could take for Win Proportion: with zero to five centroids beginning with a random value of 0. If the configuration of centroids were different for two datasets, the accuracy value will be very low even though the datasets may be very similar. Therefore, it is necessary to manually define the initial centroids so that two will have a value of one, and two with zero. This is similar to the method used by Fang et al. [46] to select centroids that maximise inter-centroid distance.

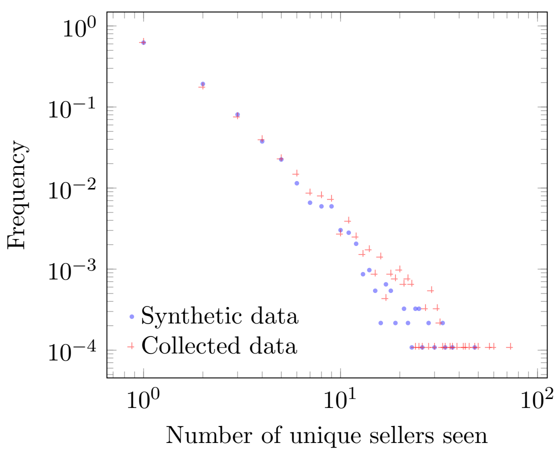
We give evidence of this in Table 3.9 and Figure 3.7. Table 3.9 shows two sets of centroids. The set named "appropriate centroids" have two centroids with the value of 0, and two centroids with the value of 1 for Win Proportion. For the set named "inappropriate centroids", the difference is that the third centroid has value of 0 instead of 1 as Win Proportion. Figure 3.7 shows that this change reduces median accuracy by 0.203 and mean accuracy by 0.192.

3.6.4 Network feature comparison

Figures 3.8 and 3.9 compare the distributions of the total and unique user interactions for users in synthetic data and collected data, for both bidders and sellers respectively. The frequency values of the y-axis are normalised. Figure 3.8 shows that the number of interactions by bidders in both the synthetic and collected data are similar. Figure 3.9 shows the number of interactions are generally similar, though the sellers in the synthetic data interact with slightly more users than expected. The discrepancy at the far right of both charts is due to the normalisation process, where the minimum frequency of 1, after normalisation by different constants, results in two different minimum values of 1.08×10^{-4} and 3.01×10^{-4}. All datasets generated using this method for evaluation in Section 6.7 contain approximately 23100 users, with 20000 bidders and 3100 sellers.



(a)

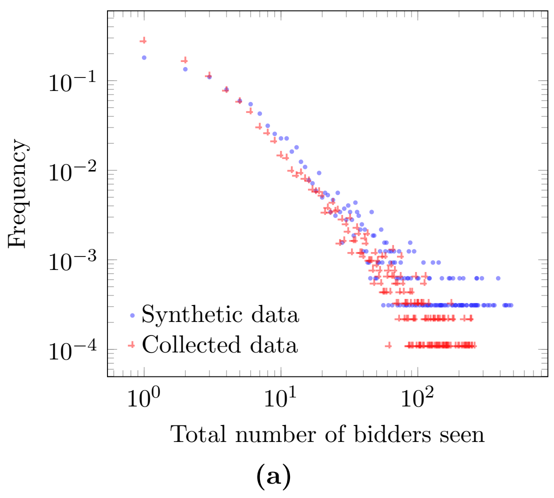


(b)

Fig. 3.8. Distribution of number of total and unique sellers encountered by bidders

3.6.5 Execution time

The execution time for simulations containing 20000 agents and over 100 days is measured and reported below. In all other experiments, one time unit is 5 minutes. Here, we show the effect of selecting different time units on execution time. Figure 3.10 shows the average execution time over 20 trials, and 1 standard deviation error bars, for time units from length 10 minutes to 1 minute, corresponding to resolutions of 0.1 to 1. The



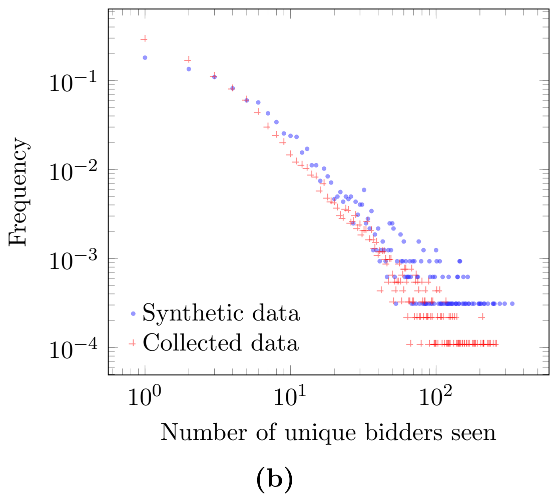


Fig. 3.9. Distribution of the number of total and unique bidders encountered by sellers

results show increasing resolution (or reducing length of each time unit), increases execution time proportionally. The line in Figure 3.10 is the linear regression with equation $y = 2014.5x + 175.37$. To faithfully represent user reaction times in auctions, which is of the order of seconds, resolution must be increased to approximately 20 (corresponding to 3 seconds per time unit). Using linear extrapolation, a resolution of 20 would result in an average execution time of 11.24 hours. Execution time measurements were done on a machine with an Intel i7-4700MQ CPU and 8GB RAM.

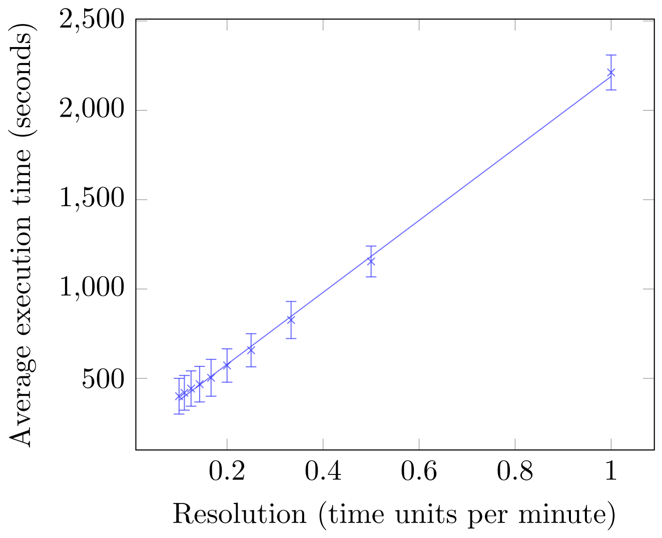


Fig. 3.10. Simulation execution time at different time unit resolutions

3.7 Conclusion

Online auction data was collected from a commercial online auction website. This data was used to construct an agent-based simulation of online auctions, with the agents based on users observed in the collected data. To understand the behaviours of users, the dataset was transformed into a set of 10 features, and clustering was applied to find groups of users with similar behaviours. The resulting simulation generated synthetic data that were shown to be similar to the original collected data, in terms of distribution for each of the 10 features, a clustering based similarity measure, and a multivariate statistical comparison. Using this simulation, we are able to generate realistic synthetic auction data.

3.7.1 Limitations

A known limitation of the simulation is that it cannot accurately model bidding times close to the end of the auction. This is due to our choice of resolution of five minutes, which is the minimum time an agent can respond to another bid. In the auction data collected, numerous bids may occur within a single minute. Thus, to accurately model bidding times near the end of auctions, resolution must be increased by roughly two orders of magnitude, to the order of seconds. Results in Section 3.6.5 suggest that running the simulation using a resolution of 3 instead of 300 seconds would increase the computation time 100 fold.

The choice of the five minute resolution was a calculated trade-off between performance on available hardware and precision: it allowed us to generate data in a reasonable time while retaining sufficient simulation accuracy over most of the auction. As a result, bidding wars close to auction termination which occur over multiple time units in the simulation may correspond to 30 minutes or more, instead of within minutes as seen in the collected data. Thus, while the simulation can accurately model user bidding time patterns earlier in the auction, it cannot do so close to the end of the auction and distribution of values for features i and j in Section 3.5.4 from the simulator will not closely match those from the TradeMe data.

4 Application of simulation

4.1 Introduction

In this chapter, we demonstrate the use of the auction simulation from Chapter 3 for evaluating fraud detection algorithms. This requires adding agents with fraudulent behaviours into the simulation to generate synthetic data that contains both normal and fraudulent transactions. The synthetic data is then used to evaluate different fraud detection algorithms.

In this chapter, we implement three types of fraudulent agents to generate three synthetic datasets, used to evaluate three different fraud detection algorithms. The first type of fraud agent is described by Trevathan and Read [2]. The agent exhibits shilling behaviour, where its goal is to increase the final selling price of auctions for a collaborating seller. The second and third fraud agents are variations of the first which attempt to hide some of their fraudulent characteristics to avoid detection. Synthetic datasets containing these shilling behaviours are used to test a shill detection algorithm, also described in [2], and two of its variations.

This chapter is organised as follows. Section 4.2 describes background for shilling fraud. Sections 4.3 and 4.4 describe the implementation of three fraud agents and fraud detection algorithms. Section 4.5 and 4.6 compare the detection results between different fraud behaviours and detection strategies. Section 4.7 concludes the chapter.

4.2 Background

This section provides information that may be helpful in understanding subsequent sections. Section 4.2.1 describes the generation and quality of synthetic data used in this chapter. Section 4.2.2 describes the characteristics of shilling fraud.

4.2.1 Generating synthetic data that contains fraud

As noted previously, this chapter builds on the validated agent-based simulation from Chapter 3. The agent-based simulation is extended here by implementing additional agents, described in Section 4.3. By adding these agents to the simulation, the resulting synthetic data will contain fraudulent bids and auctions. Table 4.1 shows the bids and auction information for three example auctions, containing the type of information that is generated by the simulation. *User A* and *B* in Table 4.1 are examples of these fraudulent agents.

Though the agents implemented only commit competitive shilling, it demonstrates the usefulness of the simulation in generating synthetic data that contains fraudulent behaviour. In Chapter 6, we will see that other types of fraud, including collusive fraud, can be simulated.

Table 4.1
Example auctions

(a) Bid sequences for three auctions.

Auction 1 listed by User 8			
BidID	UserID	BidTime	BidAmount
1	A	1	1.00
2	1	1750	20.00
3	A	1751	21.00
4	2	2020	27.00
5	1	2054	38.00
6	3	2066	39.00

Auction 2 listed by User 9			
BidID	UserID	BidTime	BidAmount
7	5	2100	2.00
8	A	2101	3.00
9	4	2216	5.00
10	A	2217	6.00
11	4	2400	7.00

Auction 3 listed by User 10			
BidID	UserID	BidTime	BidAmount
12	6	5700	50.00
13	B	5800	55.00
14	6	6000	60.00
15	5	6220	60.80
16	B	6290	62.00
17	6	6350	67.00
18	7	6516	68.00

(b) Auction attributes for auction 1.

Auction Attribute	Value
SellerId	8
ListingId	0
StartTime	40
EndTime	2058
Duration	2016
ItemValuation	40.00
ItemTypeId	1
Starting Price	1.00

4.2.2 Shilling Behaviour

Shilling is defined as any activity where a seller, or an associate, places bids in auctions held by the seller. Competitive shilling occurs when a user submits bids to a collaborating seller's auction to elevate the final auction price, without the intention of winning. The legitimate bidder is cheated by paying more than they otherwise would when winning the item. The goal is to increase the price the legitimate winner would otherwise pay, to increase seller profit [11]. For example, suppose there were only two bidders in an auction: one legitimate (L), and one fraudulent (F), with bidding proceeding like so: L:\$10, F:\$11, L:\$12, F:\$13; L:\$14. If there are no additional bids, L, the legitimate bidder, pays an additional \$4 due to bids by F.

Users engaging in competitive shilling often display a set of behaviours that distinguish them from normal users [5, 17]. These behaviours allow them to function effectively as shills. These behaviours are:

i) Shills only attempt to elevate prices of auctions held by collaborating sellers, since only those auctions affect the collaborating sellers' profit.

ii) Shills have a higher bid frequency since they must bid after legitimate bids in order to stimulate bidding to maximise the final price.

iii) Shills will win a low number of auctions, since winning will mean the seller bought their own or their collaborator's item, and they will be forced to pay the auction fee.

iv) Shills tend to bid soon after a legitimate bid to maximise the time legitimate users have to submit a new bid, to both maximise final price and reduce the chance of winning.

v) Shills bid the minimum amount possible to not discourage bidding by legitimate bidders.

vi) Shills tend to have bids early in the auction to reduce the chances of accidentally winning.

The prevalence of shilling behaviour in commercial auction sites is unknown. Commercial auction sites do not publicly disclose information about identified fraudulent accounts. Even with complete transaction histories, it is difficult to prove that an account is engaging in shilling. Shilling behaviours, and fraudulent behaviours in general, are not binary, but are continuous; each account may display different degrees of each type of suspicious behaviour. The shill agents described in Section 4.3 below represent three possible instances of shilling behaviour: the simple shilling agent in Section 4.3.1, and two agents that attempt to hide some of their fraudulent characteristics in Sections 4.3.2 and 4.3.3. Other agent variations are possible. For example, they may attempt to hide different aspects of their behaviour, such as bidding after two legitimate bids; or incorporating different legitimate actions, such as intentionally losing in other auctions to hide their true collaborator.

4.3 Fraud agents

In this section, we describe three shilling fraud agents and their implementation. The first is an implementation of the simple shill agent described by Trevathan et al. [12]. The second and third agent types build on the first, where the agents attempt to hide part of their fraudulent behaviour, as one would expect to see in real life.

4.3.1 Simple shill agent

The simple shill agent has five directives to maximise its potential as a shill. The directives are consistent with the shill characteristics listed in Section 4.2.2. Informally, they are:

1) Bid the minimum amount required.

2) Bid quickly after a rival bid.

3) Do not bid too close to the auction's end.

4) Bid until the target price has been reached.

5) Only bid when the current bidding volume is high.

The behaviour of the shill is tuned by three parameters θ, α, and μ, which govern the thresholds at which the simple shill agent ceases bidding, and in turn controls the amount of risk the agent takes while inflating the price of an auction. The agent will stop bidding if the auction is greater than $\theta\%$ complete, or if $\mathit{current price} > \alpha$. The μ parameter affects when the bidding frequency is considered "high" by controlling how far into the auction history that bids should be included during counting. Increasing the value of these parameters tends to increase the final price, but also the risk of accidentally winning the auction.

In the simulation, each shill bidding agent is paired with a seller agent, and the shill bidding agent only bids in auctions held by the corresponding seller. Normal agents, modelling legitimate users, may participate in these fraudulent auctions according to their behaviour.

Table 4.1 gives an example auction where a simple shill agent, *User A*, is working with *User 8*. In the first auction where *User 8* is the seller, *User A* bids immediately after a legitimate bid, with the minimum required bid: $1.00 over the previous bid. Section A.3 in Appendix A shows the pseudocode for the simple shill bidding agent.

4.3.2 Late-start shill agent

The behaviour of the simple shill bidding agent is to bid immediately after the auction has been submitted by their counterpart. However, in TradeMe, users do not tend to bid immediately after the auctions begin. 43.0% of auctions in TradeMe have the first bid occur when there is less than 24 hours before termination, and 78% of auctions have the first bid occur when there are fewer than 5 days remaining. This can be seen in Figure 4.1, which shows the distribution of times at which the first bid appears. Even though bidding immediately after the auction begins may give other users more opportunities to bid, agents that bid this way are suspicious. Therefore, agent behaviour is changed so that they do not begin bidding unless there are already one or more bids in an auction, or if the auction is more than two days from termination. *User B* in Table 4.1 is an example of this shill agent, where the agent waits for a legitimate bid to appear before bidding.

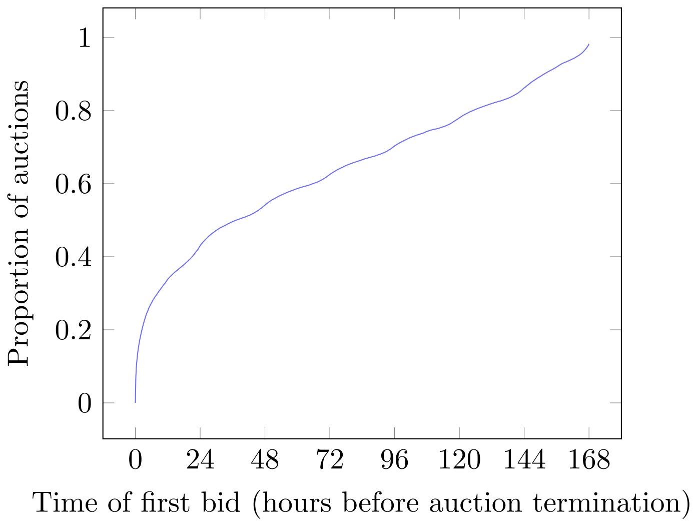


Fig. 4.1. Distribution of times at which the first bid in the auction is made

4.3.3 Legitimate-bidding shill agent

To raise the final selling price, the shill agent only needs to bid in auctions held by the collaborating seller. However, this makes the agent easy to identify since the agent only bids in auctions held by a single seller, in addition to losing most of the auctions it participates in. This agent attempts to conceal this by bidding in low priced legitimate auctions held by other sellers. For every shill auction that the agent participates in, it also participates in a legitimate auction.

4.3.4 Agent shilling performance

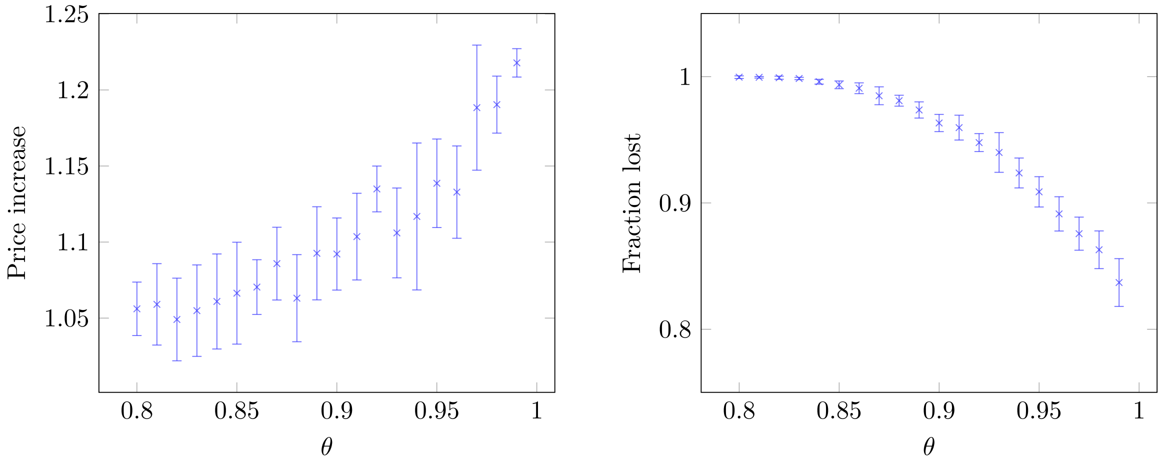
We evaluated the effectiveness of each agent type as shills using two criteria: ability to raise the final selling price of an auction; and the probability of losing the auction, as intended, to a legitimate bidder. An effective shill will have high values for both. Table 4.2 shows the performance of the simple shill agent within the simulation, using the parameters $(\theta, \alpha, \mu) = (0.95, 0.85, 0.85)$. On average, the simple shill agent successfully loses 75.3% of auctions, and raises the final price by 14.1%. The performance of the other agent types, late-start shill, and legitimate-bidding shill, is similar to that of the simple shill.

We also report the influence of changing the parameters θ, α and μ on performance. Results are calculated from 20 synthetic datasets each containing 100 shill bidding agents, and make up roughly 1% of the bidding agents in the simulation. Figures 4.2 to 4.4 shows the average performance of 1000 agents as the values of θ, α and μ range over 0.80 to 0.99 at 0.01 intervals. The error bars represent one standard deviation. The θ parameter has a strong influence on the observed win rate and price increase. As expected, as θ increases, the effect on the final price increases, at the cost of reducing the probability of losing the auction. α has a similar effect, though weaker. For μ, there is no obvious effect. Table 4.3 shows the correlation between the choice of parameters and the price increase, and the fraction of auctions lost, respectively. The correlation values are consistent with that observed from Figures 4.2 to 4.4.

The performance of the shill bidding agents in our simulation is significantly different to that reported in [12], where the agent using similar parameters increased final price by 15% and successfully lost in only 30% of auctions. The difference in performance is most likely due to the difference in the behaviour of the non-fraud agents in our simulation.

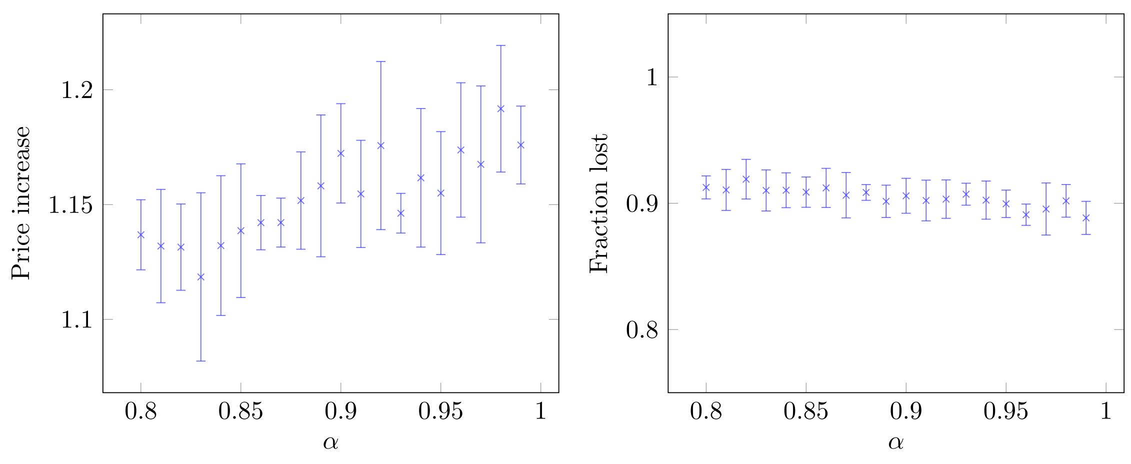
Table 4.2
Shill bidding agent performance.

	Simple shill		Late-start shill		Legitimate-bidding shill	
	μ	σ	μ	σ	μ	σ
Likelihood to lose	0.763	0.0373	0.783	0.0353	0.746	0.0364
Price Increase	1.14	0.0260	1.17	0.0267	1.16	0.0397



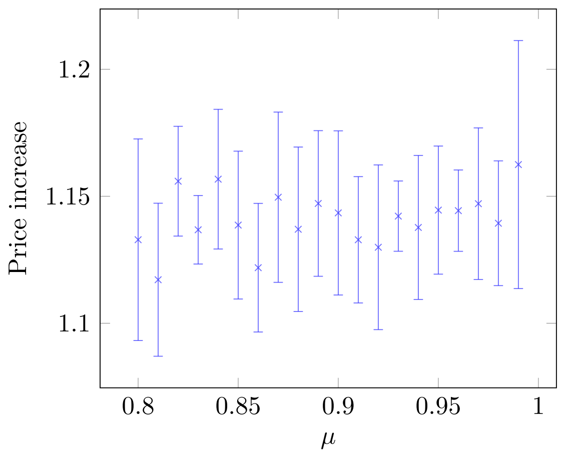
(a) Change in price increase **(b)** Change in fraction lost

Fig. 4.2. Observed agent behaviour over different values of θ

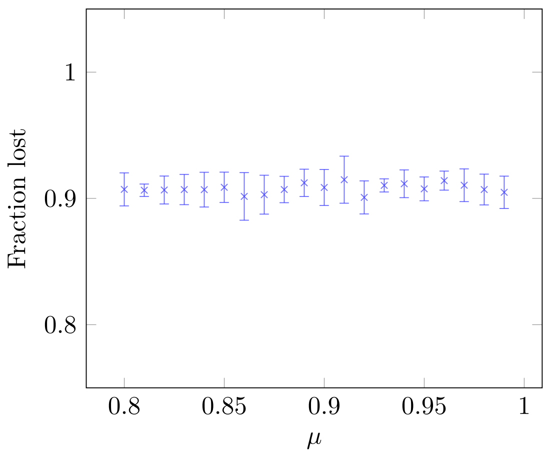


(a) Change in price increase **(b)** Change in fraction lost

Fig. 4.3. Observed agent behaviour over different values of α



(a) Change in price increase



(b) Change in fraction lost

Fig. 4.4. Observed agent behaviour over different values of μ

Table 4.3
Correlation between parameter value and observed behaviour

	θ	α	μ
Price increase	0.937	0.934	0.565
Likelihood of loss	-0.949	-0.864	0.257

4.4 Fraud detection algorithm

In this section we describe the shill score algorithm by Trevathan et al. [2]. We also describe and explain the two changes that we propose to improve detection performance.

4.4.1 Shill score

Trevathan et al. proposed the shill score algorithm (SSA) [2] to detect competitive shilling behaviour. SSA uses the bidding history of users to calculate a score which reflects the likelihood that a user is acting as a shill in a given auction. SSA works by identifying users that exhibit the six shill behaviours listed in Section 4.3.1. Trevathan defines six ratings, α through ζ, that each target one shill behaviour. For definitions of the six ratings, refer to [2].

The six ratings are combined in a weighted average according to Equation 4.1. The values of (θ_1, ..., θ_6) were defined to be $(9, 2, 5, 2, 2, 2)$ by the authors.

$$\text{Shill Score} = \frac{\theta_1\alpha + \theta_2\beta + \theta_3\gamma + \theta_4\delta + \theta_5\epsilon + \theta_6\zeta}{\theta_1 + \theta_2 + \theta_3 + \theta_4 + \theta_5 + \theta_6} \tag{4.1}$$

Table 4.4 gives the six ratings and the shill score, calculated using the weights above, for two normal agents and two fraud agents from synthetic data. Normal users tend not to participate in auctions of only one seller, and hence User 1 and 2 have low α-ratings. User 1, the normal late bidder, has behaviour inconsistent with shills since it bids aggressively near the end of auctions, giving it a low shill score. User 2, the normal early bidder, has some similarities to a shill: it made bids early in the auction, as reflected by its ζ-rating, and made the minimum bids allowed, giving a high ϵ-rating. Thus User 2 has a higher shill score. User 3, the simple shill, acts as an obvious shill, including bidding in auctions by a single seller, bidding early and often. Thus most of the ratings are high, and has a fairly high shill score. User 4, the delayed-start shill, waits until a legitimate user has made a bid, and so has lower β and ζ-ratings, and a shill score that is only slightly higher than User 2.

Table 4.4
Example of shill score ratings using weights in [2]

User Id	Agent Type	Wins	Losses	α	β	γ	δ	ϵ	ζ	Score
1	Normal late bidder	1	3	0.167	0.700	0.000	0.331	0.764	5.89E-04	2.310
2	Normal early bidder	0	1	0.333	0.125	0.000	0.800	0.800	0.800	4.205
3	Simple shill	2	8	0.800	0.690	0.000	0.800	0.800	0.800	6.628
4	Late-start shill	1	5	0.833	0.443	0.000	0.453	0.800	0.631	4.581
5	Legitimate-bidding shill	8	6	0.714	0.600	0.000	0.750	0.800	0.580	4.581

Since the algorithm was designed and tested on small samples of synthetic and commercial auction data that may not represent the range of behaviours by users in TradeMe, modifications to the algorithm may improve detection results.

4.4.2 Shill score with Bayesian average

Users that have participated in only one auction may receive very high scores from SSA and be incorrectly labelled as shills. SSA fails to take into account that bidding patterns observed from a small number of auctions is not sufficient evidence to classify an agent as a shill. This increases the number of false-positives. To avoid this problem, we take into account the amount of evidence that exists for an agent's score by recalculating SSA using

$$x' = \frac{C\mu + nx}{C + n},$$

where μ is the average score from SSA for legitimate users, C is a user-defined constant, and n is the number of auctions, x is the score from SSA, and x' is the new score.

The value of C is defined to be the average number of auctions each legitimate user participates in. In our real dataset, $C = 1.8$. The effect is that if an agent participates in very few auctions, the resulting score x' will be much closer to the group average. If the number of auctions the agent participates in is large, the resulting score will be similar to the original x from SSA.

4.4.3 Shill score with alternative weights

The weights for SSA were designed under different assumptions that do not hold here. For example, shill agents are assumed in SSA to have low accidental win rates, and α is given a very high weight of 9 out of a total of 22. This assumption is observed to be untrue in the real dataset and in the synthetic data, as shown in Table 4.2. Therefore, using a different set of weights to combine the six ratings may produce better detection performance.

We test whether combining ratings with equal weights will give better performance. Using equal weights is unlikely to be optimal, but allows us to test whether altering weights is worth further investigation. Below, we will see that the weights have a large effect on performance, and in Chapter 5, we use supervised techniques to search for an optimal set of weights for SSA.

4.5 Detection accuracy for each agent type

In this section we compare the ability of SSA to detect three different agent fraud types. For each agent type, we used 20 simulation runs each with 100 fraud agents and 10000 legitimate agents. The detection algorithm used was SSA.

4.5.1 Metric definitions

Some commonly used metrics for measuring classification accuracy are defined below. True positives (TP) is the number of instances correctly labelled as fraudulent or suspicious. False positives (FP) is the number of instances incorrectly labelled as fraudulent or suspicious. True negatives (TN) is the number of instances correctly labelled as normal. False negatives (FN) is the number of instances incorrectly labelled as normal.

$$TPR = \frac{TP}{TP + FN} \tag{4.2}$$

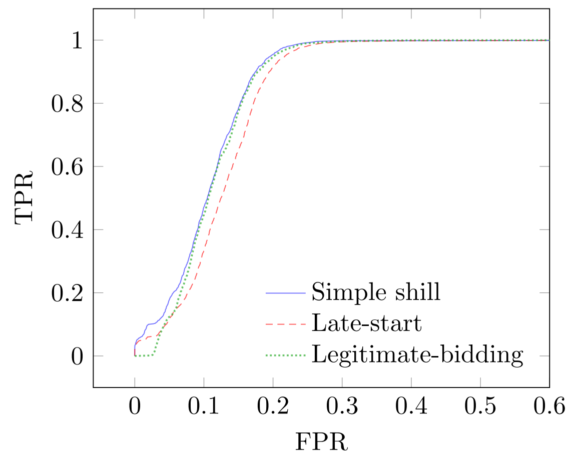
$$FPR = \frac{FP}{FP + TN} \tag{4.3}$$

TPR is also known as recall. The receiver operating characteristic (ROC) curve is a method for illustrating the performance of a binary classifier, and shows the trade-off between FPR and TPR. They are constructed by varying the discrimination threshold and plotting the corresponding TPR and FPR values.

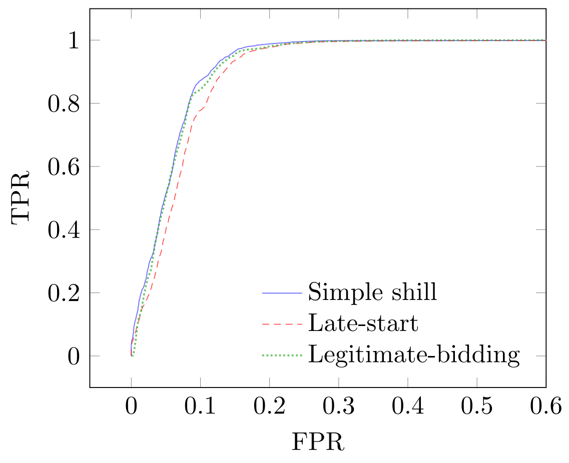
4.5.2 Accuracy

We examine the effect of choosing different score thresholds on the detection of three different shill agent types. Agents are identified as shills if they have a score that is above a defined threshold. Figure 4.5 shows the trade-off that exists: as we decrease the score threshold, both the number of shill agents we correctly identify (true positive), and the number of legitimate agents we incorrectly identify as shills (false positive) increase.

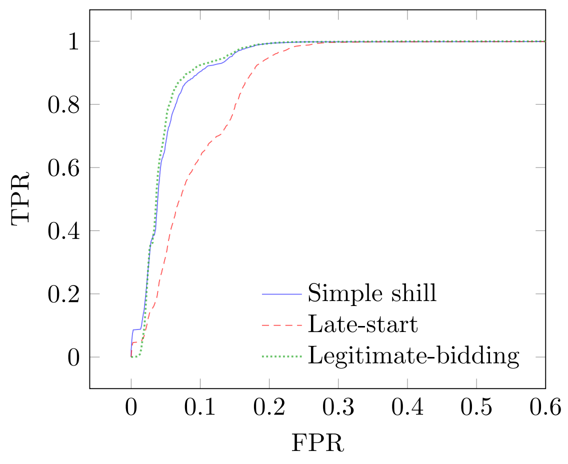
Late-start shills are consistently more difficult to detect, compared to the simple shill agent and agents with legitimate bids, as shown in Figure 4.5. At low values of FPR, the agents with legitimate bids are detected with the lowest accuracy, when there are no true positives found at all. This can be seen in Figure 4.5(a) and (c). The similarity in detection accuracy for the simple shill agent and legitimate-bidding shill is likely due to the agent winning the majority of the legitimate auctions it participates in. Since auctions that the agent wins do not contribute to the ratings, the agent fails to hide its behaviour.



(a) Shill score



(b) Bayesian average



(c) Equal weights

Fig. 4.5. ROC curve for detection of shill agents

4.5.3 Recall

We evaluated the ability of SSA to correctly identify the shilling agent in auctions that are known to contain shill bids. A shilling agent is identified in an auction if the shill agent had the highest score among all agents that participated in that auction. Table 4.5 shows that for all three shill agent types, SSA was able to accurately identify the shill agent 90% of the time.

Table 4.5
Detection performance of SSA for three shill agent types

	μ	σ
Simple shill	0.904	0.0216
Late-start shill	0.893	0.0428
Legitimate-bidding shill	0.903	0.0240

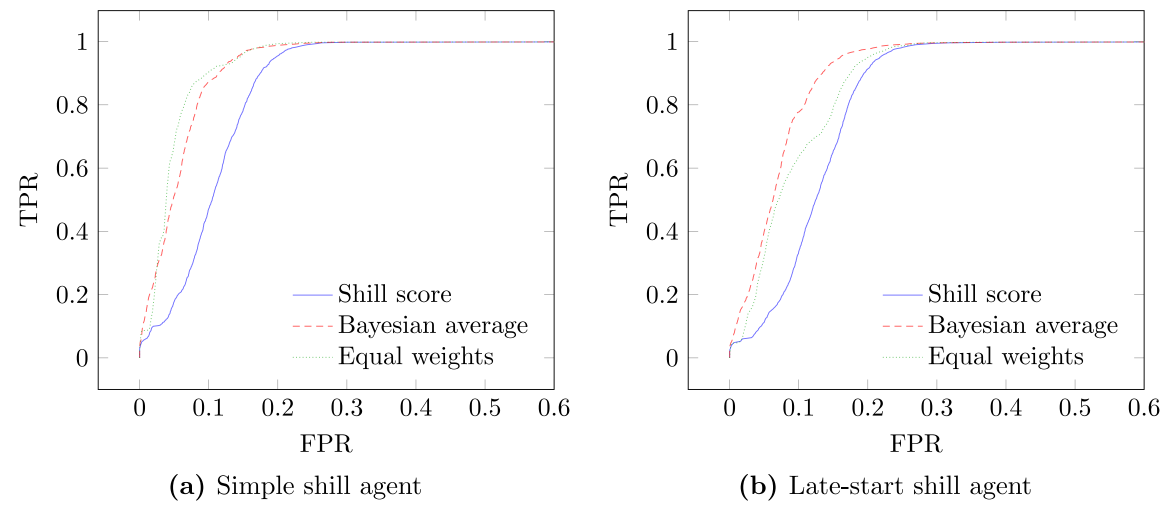
4.6 Performance comparison of three fraud detection algorithms

In this section we compare the ability of three fraud detection algorithms to identify simple shill agents. Once again, we used 20 simulation runs each with 100 simple shill agents and 10000 normal bidding agents.

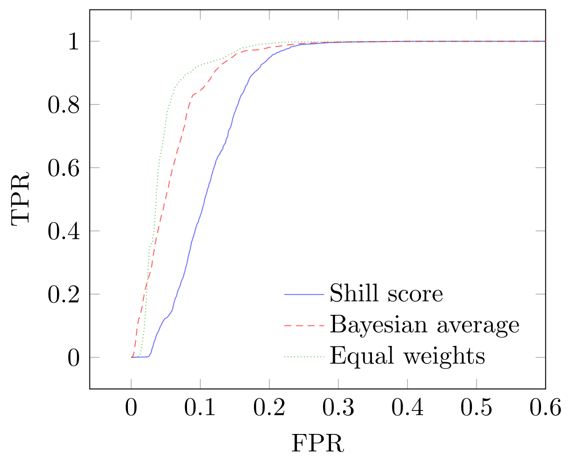
4.6.1 Accuracy

Plots in Figure 4.6 are constructed using the same data as plots in Figure 4.5, but it is presented differently to show the relative accuracy for each shill detection algorithm variant. Figure 4.6(a) shows that the original SSA has the poorest performance, while SSA with Bayesian average and with equal weights has similar accuracy. This is also seen in the other two fraud types, in Figure 4.6(b) and (c). This is consistent with AUC values shown in Table 4.7.

The poor performance of the original SSA is likely due to the selection of weights. The SSA algorithm was originally evaluated on a small set of auctions and for a specific item, and it is possible the authors tuned their algorithm for that specific dataset, which has a limited range of bidder behaviour. As a result, when SSA is evaluated on the synthetic data, where there is a greater range of bidder behaviour, the accuracy of SSA is reduced.



(a) Simple shill agent **(b)** Late-start shill agent



(c) Legitimate-bidding shill agent

Fig. 4.6. ROC curve for detection of shill agents

Table 4.6
AUC values for detection accuracy against shill agents

	Simple shill	Late-start	Legitimate-bidding
Shill sore	0.895	0.875	0.888
Bayesian averaged	0.945	0.931	0.941
Equal weights	0.901	0.909	0.954

4.6.2 Recall

There is no significant difference in the ability of the three fraud algorithms to correctly identify the shill agent in a known shill auction, as shown in Table 4.7.

Table 4.7
Detection performance of three detection methods for simple shill agents

	μ	σ
SSA	0.904	0.0216
Bayesian average	0.904	0.0212
With equal weights	0.909	0.0310

4.7 Conclusion

We have demonstrated that the simulation is able to generate synthetic data containing different types of fraud, and be used to evaluate different fraud detection algorithms. The three types of agents exhibiting variations of competitive shilling behaviour were implemented, along with three variations of a detection algorithm. Using the simulation, we can evaluate the accuracy of each of the three detection methods for detecting the different fraudulent behaviours in the synthetic data. Evaluation results using synthetic data show that late-start shill agents were the most difficult to detect, and that both SSA variations performed significantly better than the original SSA algorithm.

In the next chapter, we will use supervised learning methods to improve the SSA algorithm in a systematic way, and to generate other classification models trained using synthetic data. We will see that results obtained are significantly better than the ad hoc modifications to the SSA algorithm presented here.

5 Supervised fraud detection

5.1 Introduction

Past research in online auction fraud has focused on detecting specific fraudulent behaviours using a range of techniques, including decision trees [20, 24], clustering [47], regression models [16, 25], model checking [18], and graph mining methods [23]. The general approach taken involves identifying the type of behaviour or fraud of interest, then selecting a set of features that are hypothesised to be able to differentiate between users with normal and suspicious behaviour. A fraud detection algorithm is then developed using the selected feature set. The algorithm is evaluated using commercial auction datasets without knowledge of ground truth, or using synthetic datasets without a guarantee of its similarity with real data. Both types of datasets have weaknesses that reduce the reliability of any conclusions drawn about method accuracy or effectiveness, as discussed in Chapter 3.

In this chapter, validated synthetic data is used together with supervised learning methods to develop models for fraud detection. Classifiers can be constructed using supervised learning methods to detect different types of fraud, given an appropriate training set. Depending on the types of agents added to the simulation, different types of fraud can be added to the resulting synthetic data. This in turn determines the types of fraud that can be detected by the classifier. An appropriate training set for supervised learning methods is created in three steps: first, define an agent-type that represents the fraud type of interest; second, generate synthetic data using the defined agent; and third, transform the generated synthetic data, which is a sequence of auctions and bids, into values for a set of user-defined features. This transformed synthetic dataset is then used as a training set for the selected supervised learning technique. This approach allows models for detecting specific types of fraud to be constructed more easily than in previous work, and with improved accuracy, as shown by the experimental results in Section 5.3. To our knowledge, no previous work has combined the use of a data generator and supervised learning methods to develop fraud detection methods.

5.1.1 Contributions

Our contributions in this chapter are as follows:

- We propose an approach for generating classification models for detecting suspicious behaviours in commercial auction data. The approach uses a synthetic data generator with supervised learning techniques. To demonstrate this approach, we define two types of fraudulent behaviours, and develop two corresponding classification models for detecting those behaviours. We show that these models, created using synthetic data, can also be applied to commercial online auction datasets to identify suspicious users. We also use supervised learning techniques to improve the performance of an existing fraud detection algorithm.

- We describe and define a set of user features for fraud detection. The set of features captures many aspects of bidding behaviour, and may be useful for developing models for the detection of different types of auction fraud.
- We present experimental results on both synthetic and real datasets. Results on synthetic datasets show that our supervised approach produces classification models of greater accuracy than existing algorithms, and that accuracy is further increased using our proposed feature set. Evaluation results on a real dataset using these same models show that they are able to identify users that exhibit suspicious behaviour.

The remainder of the chapter is organised as follows: Section 5.2 describes the methods used to improve and develop classifiers for identifying shilling fraud. Section 5.3 presents the evaluation procedure and results on synthetic data; and Section 5.4 is a case study applying the classifiers to commercial auction data.

5.2 Improving detection performance using supervised learning methods

Supervised learning may provide a method for quickly developing classifiers that are capable of detecting different types of fraud or suspicious behaviour. However, effectiveness of the resulting classifier depends on four things: (1) the fidelity of the synthetic data to real world data, so that the models trained will be useful in commercial data; (2) the ability to encapsulate the behaviour of interest with an agent; (3) the ability of the set of selected features, into which the synthetic data is transformed, to capture the differences between normal and fraudulent users; and (4) the choice of the supervised learning technique.

Both (1) and (2) have been addressed in Section 4.2 in Chapter 4. For (3), a set of 10 features to describe bidder behaviour is defined in Section 5.2.4, which also discusses their differences to the set of six ratings defined in Section 4.4.1. This set of 10 features, and the set of 6 ratings, are used to construct two decision trees, as described in Section 5.2.3. For (4), there are many different techniques available. For weight selection, possibilities other than neural networks include: probit models, a type of regression where the dependent variable can take 2 values (normal or fraudulent), as seen in [25], or linear support vector machines (linear SVM) which give weights to each feature. For supervised learning methods, methods other than decision trees include Naive Bayes, SVM, nearest neighbour classifiers or various ensemble methods such as bagging and boosting.

The use of neural networks and decision trees may not yield the highest accuracy, but they have some advantages over other algorithms, described later, and are sufficient for demonstrating our approach.

This section describes the use of neural networks for weight tuning in Section 5.2.2, the use of decision trees in Section 5.2.3, and finally the definition of a set of features to construct an alternative decision tree model in Section 5.2.4.

5.2.1 Shill agent variation - delayed-start shill

Two fraud types are used in this chapter. The first is the simple shill agent, described in Chapter 4. The second is the *delayed-start* shill agent which is based on the *late-start* shill agent, also from Chapter 4. The third type of agent described in Chapter 4, the *legitimate-bidding* shill, is not used here since it is ineffective in hiding its behaviour.

User B in Table 4.1 is working with *User 10* as a delayed-start shill: it begins bidding only after another user, *User 6*, makes the first bid, then submits a bid after waiting a variable amount of time, for an amount that may be above the minimum increment of $1.00.

In the simulation, the wait-time used by the delayed-start shill is drawn from a uniform distribution of [50-99] time units (which corresponds to 250-495 minutes). Bids exceed the minimum required 20% of the time, with an amount that is proportional to the difference between the current price and the value of the item, as shown in Equation 5.1, with $k = 0.1$. The value of 20% was chosen because in the Oct 2011 dataset, 19.5% of bids (204458 out of 244308) have bid increments that are greater than the minimum required. However, since we do not have information about the value of the items, we use the simple

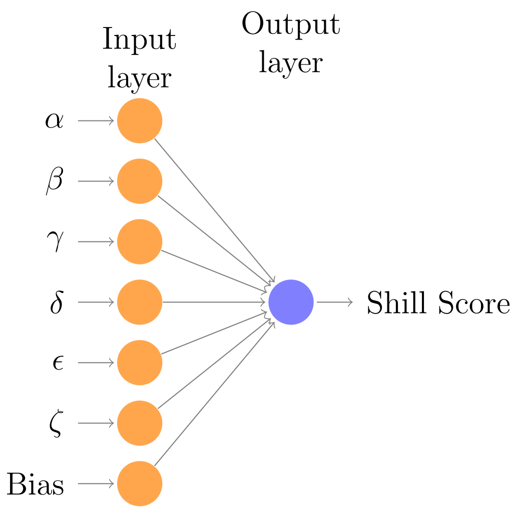


Fig. 5.1. Neural network structure

Equation 5.1 to generate bid amounts that appear reasonable. For example, if an item was valued at \$100.00, and the current bid was at \$20.00, after deciding that it should bid over the minimum, the bid made would be $((\$100.00 - \$20.00) * 0.1 + \$21.00) = \$29.00$.

$$\begin{aligned} \mathit{Bid\ Amount} = (\mathit{Item\ Value} - \mathit{Current\ Price}) * k \\ + \mathit{Next\ Minimum\ Bid} \end{aligned} \tag{5.1}$$

5.2.2 Using neural networks

The shill score algorithm described in Section 4.4.1, requires a set of weights to combine six ratings into a single value. In the original algorithm, the weights are selected by the authors. A neural network can be used to systematically search for a different set of weights to increase the performance of SSA. Figure 5.1 shows the neural network structure used: a 1-layer back-propagation neural network with six input nodes, one output node, and no hidden layer. Each input node receives one of the six ratings defined by SSA. The value of the output node is given by the linear combination of the six inputs. During training, as the neural network reduces error by adjusting the set of weights, it is, in effect, searching for the optimum set of weights for SSA. After training, the weights from the network are used in the SSA algorithm.

5.2.3 Using decision trees

Even though the neural network improves SSA by identifying an improved set of weights, the method by which SSA combines the weights is very simple: it is a linear combination of the ratings, and may be unable to identify less obvious shilling behaviour. A decision tree, trained using the same dataset as used in Section 5.2.2, may have better performance, since it is able to generate a more complex model. Decision trees were chosen over other methods such as Naive Bayes and nearest neighbour classifiers, since decision trees do not assume independence of variables (for example ϕ_3 and ϕ_4 in Section 5.2.4 below), and can produce a model that is easily understandable and explainable, which is important when justifying why a particular account is deemed suspicious. Also, the C4.5 algorithm used for generating the decision trees allows pruning to remove redundant branches to mitigate overfitting. For details regarding the training and testing sets used, and the parameters used for training the decision tree, refer to Sections 5.3.1 and 5.3.3 respectively.

5.2.4 Using user features as attributes

In addition to the 6 ratings, another feature set was used for training a decision tree. This feature set is slightly modified from the one described in Section 3.5.4, where features ϕ_8 to ϕ_9 are replaced with ϕ_{14} to ϕ_{15}. Those features were originally chosen to capture as many aspects of the user's behaviour as possible, and may capture other fraudulent behaviours, not just those related to shilling, unlike the six ratings. ϕ_9 (net reputation) was replaced because we have no information about reputation of shills from previous work. In addition, reputation scores can be easily manipulated [9], making it an unreliable feature for fraud detection. ϕ_8 (auction count) was replaced because auction count is not well modelled by shilling agents. They tend to participate in far more auctions than other users. This can be solved by increasing the simulation length and reducing the frequency of auction participation by the shill agents, however, this would increase the simulation execution time. If ϕ_8 is used, all agents will be detected very easily.

As mentioned previously, the 10 features were selected from a larger set that included features related to user reputation, time intervals between bids, and features that represent feature value variances over multiple auctions, some of which have been used in previous work [16, 13, 11]. The features were selected based on the results from correlation analysis and principle component analysis (PCA) to discover which attributes were redundant, or whose values were similar across all users. Using a subset of the features inproves the performance of the resulting models, as shown by the accuracy measures in Appendix C.

The use of a more comprehensive set of features produces a model that can identify shills using a different, broader, set of characteristics, and may improve performance by capturing additional shill behaviours overlooked by the six ratings.

5.3 Synthetic data evaluation: methods and results

This section describes the evaluation methods and results of each model described in Section 5.2. Each model is tested on two types of fraudulent agents: the simple shill, and the delayed-start shill, and the accuracy of each model is measured and compared.

5.3.1 Dataset setup

Evaluation was performed using two sets of synthetic data, each containing normal users and only one type of fraudulent agent. In each synthetic dataset, the ratio between normal and fraudulent users is roughly 180:1. The severe class imbalance reduces the performance of the resulting neural networks and decision trees. In all experiments, random undersampling was used to adjust the class balance to 1:1, giving a labelled dataset with 40,000 examples. The effect of class balance on model accuracy is given in Section 5.3.5.

Sections 5.3.2 and 5.3.3 describe the parameters and procedure used for evaluating SSA and the decision tree models respectively. Section 5.3.4 reports the detection results of each method.

5.3.2 SSA using optimised weights

10-fold cross validation was used to train neural networks that would provide the optimised weights for SSA. The dataset was partitioned into two sets in a 9:1 ratio: the smaller set was used as a testing set to evaluate the performance of the neural network, and the larger set was further partitioned into a training and validation set in a 7:1 ratio. The validation set was used to identify over-fitting and to set a stopping threshold during training. Each of the 10 folds produces a neural network and a corresponding set of weights. This was repeated 10 times, producing 100 neural networks. The weights from the neural network with the highest F-measure according to the testing set were normalised to 1 and replaced those in the original SSA. Table 5.1 shows the weights of the two neural networks that had the best performance for detecting simple and delayed-start shills in the testing set.

Performance of the optimised SSA was evaluated using another synthetic dataset with 40,000 instances, which was not previously used for training or testing. Since both SSA and optimised SSA produces a score for each user instead of a class label, a threshold is used to classify all users who fall below the threshold as legitimate, and all users above it as fraudulent. For example, for the detection of simple shills, if we set the threshold at the $99.7th$ percentile of scores for legitimate users, optimised SSA will find 95.5% of fraudulent users, and $1-99.7\% = 0.3\%$ of legitimate users will be incorrectly identified as fraudulent, as shown in Table 5.5. The precise threshold that minimises the costs of misclassification depends on several factors: the expected class imbalance, the misclassification cost for normal and fraudulent users, and the trade-off between TPR and FPR for the algorithm. In this case, assuming a class imbalance of 12:1, equal misclassification costs, and the ROC curve for SSA optimised, the optimal threshold should be set at the $96.4th$ percentile.

The same evaluation procedure was used for the datasets containing simple shill agents and the dataset containing delayed-start shill agents.

Table 5.1
Normalised sets of weights used for weighted average in SSA

Weight Source	Weights					
	α	β	γ	δ	ϵ	ζ
Original SSA	0.4091	0.0909	0.2273	0.0909	0.0909	0.0909
Data with simple shills	0.1615	0.0014	-0.1043	0.5237	0.0451	0.3725
Data with delayed-start shill	0.6591	0.0700	0.1077	0.0515	-0.2213	0.3331

5.3.3 Decision trees

10-fold cross validation was used to construct and evaluate decision trees. The datasets used were the same as in Section 5.3.2: 40,000 labelled instances each with 6 attributes. The C4.5 algorithm in WEKA was used to construct the decision trees. The parameters were: use subtree-raising, with a minimum leaf size of 50, and use pruning. Other parameters were left as defaults. Decision trees trained using these parameters had on average: 26.8 nodes and 13.9 leaves for the simple shill dataset; and 67.6 nodes, and 34.3 leaves for the delayed-start shill dataset. Figure 5.2 shows the first four levels of one of the trained decision trees trained on the latter. Table 5.3 shows the features that were part of the decision trees produced.

To construct the decision trees with 10 features, the same procedure was repeated using a dataset where each instance had 10 attributes. The decision trees trained had on average: 44.4 nodes and 21.7 leaves for the simple shill dataset; and 63.2 nodes, and 32.1 leaves for the delayed-start shill dataset. Figure 5.3 shows the first four levels of one of the trained decision trees trained on the latter. Table 5.4 shows the features that were part of the decision trees produced. It indicates that ϕ_1 (average bid amount), ϕ_{15} (average final bid value), and ϕ_{16} (final bid amount proportion), are not important for the identification of shills. On the other hand, ϕ_4 (bid proportion) seems to be an important feature, given its position close to the root in the decision tree.

For convenience, in the results we refer to each of the models trained using datasets

Table 5.2
Time for model construction and instance classification (ms)

	SSA with optimised weights	DT-r	DT-f
Training	5916 (2114)	351 (87)	927 (56)
Testing	1.76 (0.43)	7.86 (0.42)	10.94 (0.79)

containing 6 ratings for attributes or 10 features for attributes as DT-r and DT-f respectively.

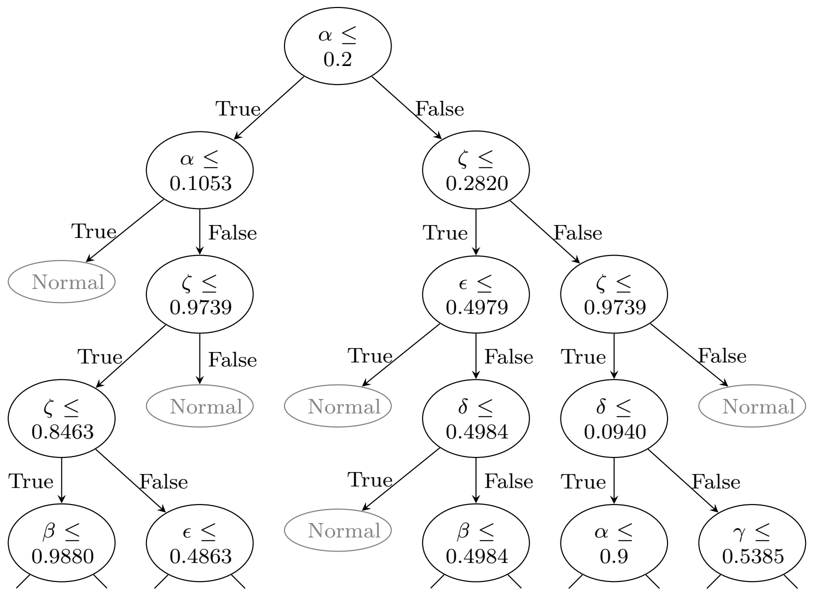


Fig. 5.2. Decision tree with ratings (DT-r) for delayed-start shills, shown to a depth of four

Table 5.3
Features used in DT-r

	α	β	γ	δ	ϵ	ζ
Simple shill	✓	✓	–	✓	–	✓
Delayed-start shill	✓	–	✓	✓	✓	✓

Table 5.4
Features used in DT-f

	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6	ϕ_7	ϕ_{14}	ϕ_{15}	ϕ_{10}
Simple shill	–	✓	✓	✓	✓	✓	–	✓	–	–
Delayed-start shill	–	✓	✓	✓	✓	✓	✓	✓	–	–

5.3.4 Detection performance

This subsection reports the detection performance of the models on synthetic data. First the metrics used to measure performance are defined, then the detection accuracy of each algorithm at detecting simple shills and delayed-start shills is reported.

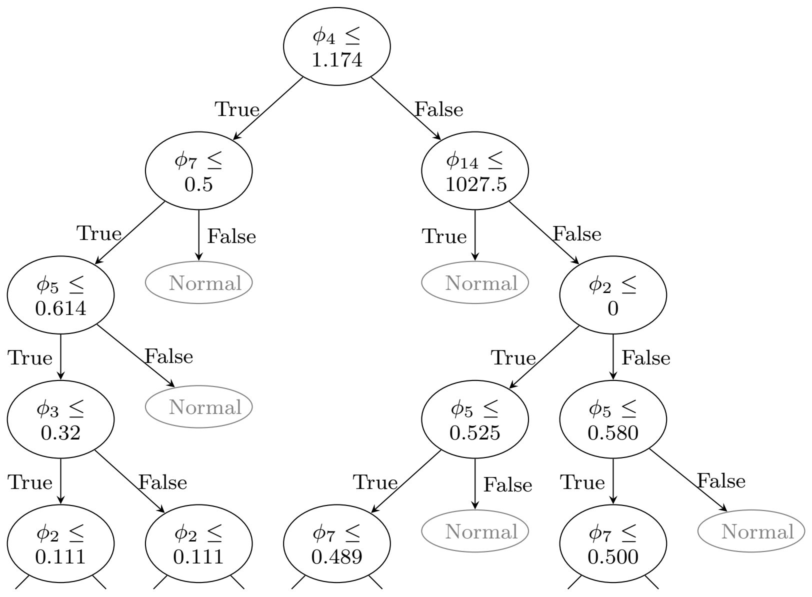


Fig. 5.3. Decision tree with features for delayed-start shills (DT-f) shown to a depth of four

Metric definitions

Some commonly used metrics for measuring classification accuracy are defined below.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

$$F - measure = \frac{2 \times TPR \times FPR}{TPR + FPR} \tag{5.3}$$

The area under the ROC curve (AUC) is a summary statistic of ROC where the value of 1 is optimal, and a random classifier gives a value of 0.5. The other measures used here have been defined in Section 4.5.1 in Chapter 4.

Simple shill

Figure 5.4 shows, for the simple shill agent, the ROC curve of each algorithm. For all values of FPR, SSA (optimised), DT-r and DT-f have higher TPR than original SSA. DT-r and DT-f have very similar TPR for all FPR, performing only slightly better than SSA using optimised weights. Table 5.5 shows the average values, where applicable, for five measures of performance for each algorithm. Values in brackets are the standard deviations. For SSA, the values shown are calculated when using a FPR threshold of 0.003. The values are consistent with the performance of the algorithm seen in Figure 5.4.

The results from Figure 5.4 and Table 5.5 show that the performance of the models were quite similar, especially between SSA (optimised), DT-r, and DT-f, where they have close to perfect classification accuracy. For example, the AUC for the 3 models are 0.986, 0.997 and 0.998 respectively. This reflects the ease with which the simple shill can be detected due to its obvious shilling behaviour.

ROC curves can be constructed using a decision tree by varying the class labels for the leaves. Usually, the class for a leaf node is the majority class. For binary classification as we are doing here, all instances in a node is normal if more than 50% of instances are normal. Increasing or decreasing this threshold for all leaves allows an ROC curve to be constructed from a single decision tree.

Delayed-start shill

Figure 5.5 shows, for the delayed-start shill agent, the ROC curve of each algorithm. As was the case in Section 5.3.4, the two decision tree models and SSA (optimised) dominate the two models using SSA. The difference in performance between DT-f and DT-r is the greatest at low values of FPR of about 0.01, where DT-f detects 93.2% of frauds, while DT-r detects 73.0%. Optimised SSA also dominates the original SSA, but performs significantly worse than both decision trees.

Once again, values for the various measures in Table 5.6 reflect the relative performance of each algorithm seen in Figure 5.5, where DT-f slightly outperforms DT-r, which in turn significantly outperforms both SSA (optimised) and SSA (original).

Evaluation results also indicate that the delayed-start shill is more difficult to detect than the simple shill. This can be seen in the ROC curves, where for every FPR, each model has a lower TPR for the delayed-start shill than the simple shill. That is, fewer delayed-start shill agents are detected given the same false positive rate. This trend is seen across all models for all performance metrics used.

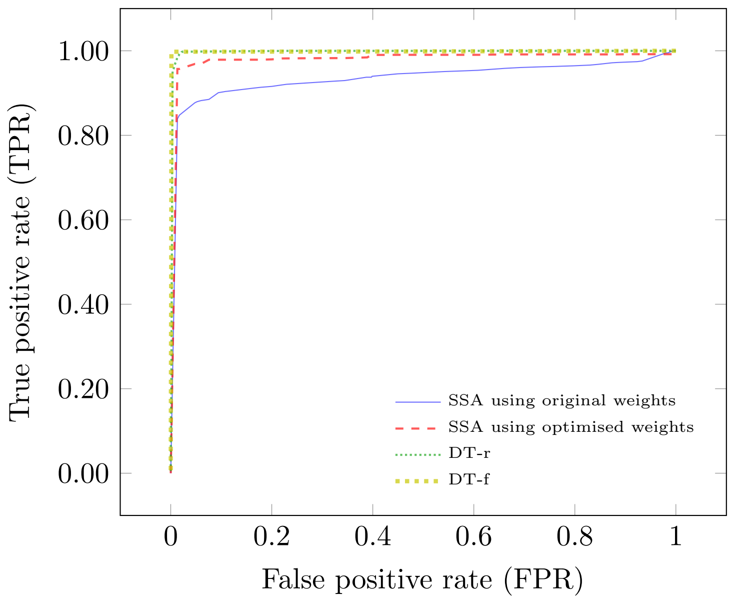


Fig. 5.4. ROC for detecting simple shill agents

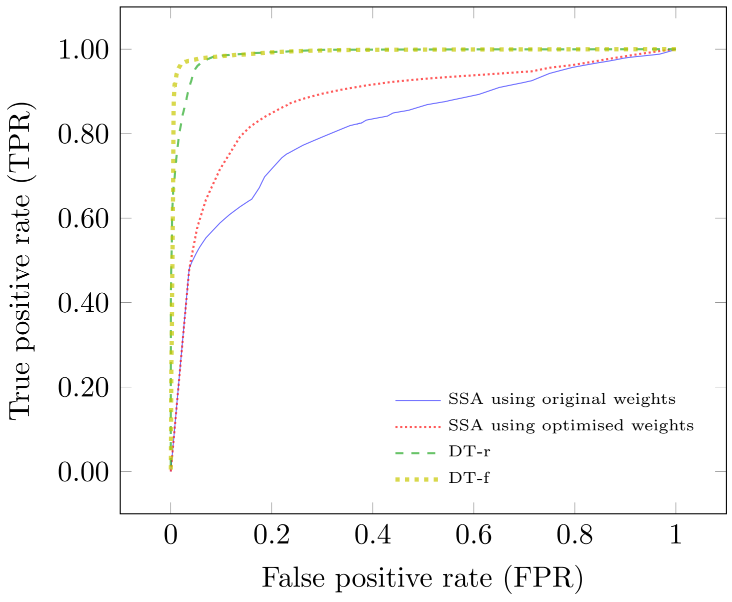


Fig. 5.5. ROC for detecting delayed-start shill agents

Table 5.5
Mean detection accuracy of simple shills

Model	TPR (SD)	FPR (SD)	AUC (SD)	Precision (SD)	F-measure (SD)
SSA original	0.790	0.003	0.940	0.955	0.881
SSA optimised	0.955	0.003	0.986	0.997	0.975
DT-r	0.996 (1.72E-04)	0.019 (1.17E-04)	0.997 (3.78E-05)	0.981 (1.11E-04)	0.988 (8.18E-05)
DT-f	0.998 (4.64E-05)	0.001 (4.67E-05)	0.998 (5.67E-05)	0.999 (4.68E-05)	0.998 (4.02E-05)

Table 5.6
Mean detection accuracy of delayed-start shills

Model	TPR (SD)	FPR (SD)	AUC (SD)	Precision (SD)	F-measure (SD)
SSA original	0.714	0.196	0.824	0.785	0.748
SSA optimised	0.847	0.196	0.879	0.812	0.830
DT-r	0.963 (9.75E-04)	0.054 (5.45E-04)	0.988 (1.47E-04)	0.947 (1.47E-04)	0.955 (5.58E-04)
DT-f	0.967 (1.37E-04)	0.023 (1.58E-04)	0.992 (0.81E-04)	0.977 (1.55E-04)	0.972 (9.39E-05)

Algorithm run time

Table 5.2 shows the mean time required, in ms, and standard deviation for training and testing one instance of each classifier. Values were obtained over 30 trials. The size of the dataset was 36000 for training, and 4000 for testing. As expected, training the neural network took the most time. Training DT-f took longer than DT-r due to the additional attributes. The testing time for optimised SSA is the lowest due to its simplicity.

5.3.5 Effect of class imbalance on model accuracy

Figures 5.6 and 5.7 show the effects of changing the ratios of normal and fraudulent instances in the training dataset from 1 to 33. In general, increasing the proportion of normal instances reduces the FPR, and increases the TPR, since the resulting decision tree favours the majority class. The change in TPR and FPR is small as the class ratio increases, as seen in Figure 5.6, since the simple shill is easy to detect [48]. The increase in TPR for DT-f is likely due to the increase in training set size outweighing the effects of the class imbalance. For the delayed-start shill, the class imbalance has a greater effect on DT-r than DT-f, which reduced TPR by 10.8% and 3.7% respectively as the ratio increased to 33.

5.4 Applying models to commercial data

The two decision tree models DT-r and DT-f were applied to a commercial auction dataset containing 113411 users. This section reports the results obtained, and compares the groups of users labelled as normal or fraudulent by each model.

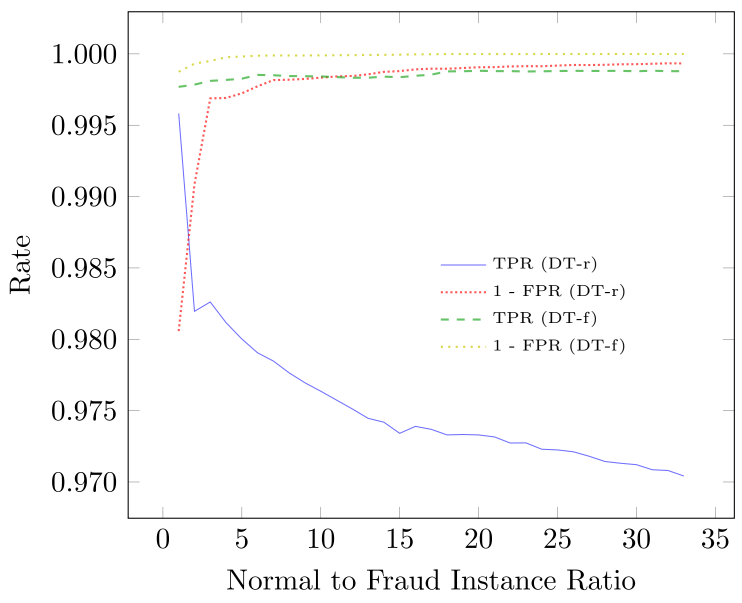


Fig. 5.6. TPR and FPR for different dataset class balance ratios for simple shill agents

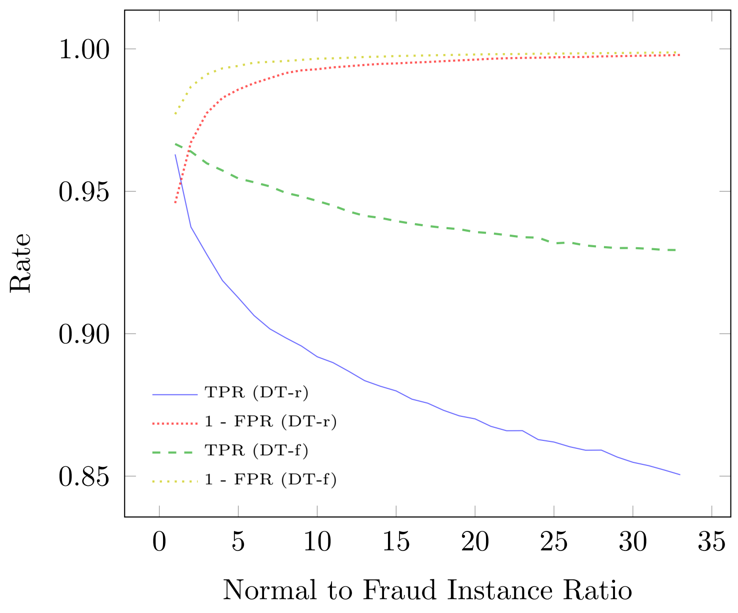


Fig. 5.7. TPR and FPR for different dataset class balance ratios for delayed shill agents

5.4.1 Analysis of classification results

DT-r and DT-f classify 7.8% and 6.0% of instances as fraudulent respectively, as shown in Table 5.7. The degree of overlap between the set of users identified as fraudulent by each model is shown in Table 5.8. The results show they have a relatively low overlap: 15.5% for the decision tree using ratings, and 20.0% for the decision tree using features, compared with the overlap for synthetic data of 97.2% and 97.6% respectively.

The low degree of overlap may be due to the existence of other groups of suspicious users in the commercial data which display different subsets of shilling characteristics. Since the two models detect shill agents using different features, each model would selectively identify the additional groups resulting in low overlap. Another explanation for the low overlap is that the synthetic data from the simulation is different to the collected data. However, the results from MRPP in the following section show that this is unlikely.

5.4.2 Comparing group similarities

MRPP is used to compare the normal and fraudulent groups of users that were found by each of the two models in the synthetic and commercial datasets. MRPP was described in Section 3.6.2 in Chapter 3. The distance matrix used by MRPP was calculated in two ways: using ratings, and using features. The MRPP values are shown in Table 5.9. For example, the A-value of 0.0035 is obtained by comparing two groups of users identified as normal (in the synthetic and commercial data) by *DT-f*, with the distances between instances measured using the 6-ratings attributes of those instances. The p-value was 0.001 for all tests, showing that it is unlikely that the A-values obtained occured by chance. The average A-values are very close to zero for both models, using both methods of distance calculation.

The A-values indicate that there is no practical difference between the groups of users in the synthetic and commercial datasets, despite $p < 0.05$. This is because although the p-value shows that the differences are *statistically* significant, the magnitude of this difference is so small that, in practice, the groups of users can be treated as if they were identical.

This result indicates two things. First, since the normal users between the synthetic and commercial data are similar, the simulation modelling normal users and the resulting synthetic data are valid. Second, the models found the groups of users they were trained to find, given the set of attributes used in the training data.

Table 5.7
Proportion of instances in each class

	DT-r	DT-f
Legitimate	0.9220	0.9395
Fraudulent	0.0780	0.0605

Table 5.8
Model agreement over instance classifications

		DT-f	
		Legitimate	Fraudulent
DT-r	Legitimate	0.8721	0.0655
	Fraudulent	0.0503	0.0121

Table 5.9
Mean MRPP A-values

Model		Using distances calculated with ratings		Using distances calculated with features	
		Normal	Fraud	Normal	Fraud
DT-r	Mean	0.0069	0.0494	0.0041	0.0284
	SD	0.0012	0.0035	0.0006	0.0012
DT-f	Mean	0.0035	0.0257	0.0079	0.0270
	SD	0.0007	0.0018	0.0010	0.0014

5.4.3 Comparing other features

The descriptive statistics for other features of the groups classified as fraudulent are compared to those classified as legitimate. Difference in these values would indicate whether the groups identified as fraudulent actually exhibit characteristics of shills. The four features are: auctions per seller, bids per auction, number of other bidders, and number of auctions, the mean, standard deviation and median which are shown in Table 5.10. *Auctions per seller* is the number of auction participations divided by the number of unique sellers those auctions belong to. *Bids per auction* is described in Section 5.2.4. *Number* *of other bidders* is the average number of unique bidders in auctions the user participated in. *Number of auctions* is the number of bids the user submits on average in an auction.

The results show that the mean and median values were higher for all four features for users classified by either decision tree as fraudulent. This indicates that at least some users in each of the groups identified by both DT-r and DT-f exhibit shilling characteristics.

Table 5.10
Mean, standard deviation and median

(a) Auctions per seller

Group	μ	σ	Median
Identified by neither as fraud	0.811	0.733	1.00
Identified by DT-r as fraud	1.375	0.831	1.00
Identified by DT-f as fraud	1.488	1.718	1.00
Identified by both as fraud	1.697	2.546	1.167

(b) Bids per auction

Group	μ	σ	Median
Identified by neither as fraud	2.791	2.126	2.00
Identified by DT-r as fraud	2.643	1.494	2.48
Identified by DT-f as fraud	3.459	1.736	3.00
Identified by both as fraud	3.012	1.377	2.75

(c) Number of other bidders

Group	μ	σ	Median
Identified by neither as fraud	2.878	1.673	2.67
Identified by DT-r as fraud	2.920	1.350	2.78
Identified by DT-f as fraud	3.466	0.804	3.25
Identified by both as fraud	3.418	1.110	3.25

(d) Number of auctions

Group	μ	σ	Median
Identified by neither as fraud	3.403	5.907	2.00
Identified by DT-r as fraud	7.617	10.018	4.00
Identified by DT-f as fraud	6.534	10.874	4.00
Identified by both as fraud	8.418	10.640	5.00

5.5 Conclusion

Development and evaluation of fraud detection methods have been difficult in the past for several reasons: (1) the variety of possible fraudulent behaviours, (2) changes in fraudulent behaviours to avoid detection, and (3) the difficulties with using synthetic and commercial datasets. Our proposed approach mitigates these problems by allowing classification models to be created for arbitrary types of fraud by defining a corresponding fraud agent and generating a set of synthetic data. This approach has advantages over previous work using real data, since in synthetic data, ground truth is known, and multiple datasets can be generated for different types of fraud. In addition, the feature set proposed allows the created models to work across multiple fraud types, and not just shilling fraud.

Using our approach, an existing shill detection algorithm was improved, and two new decision tree classifiers were created. Evaluation results over synthetic and commercial data validates our approach. Results over synthetic data show that both decision tree models perform significantly better than the original and optimised SSA algorithms in identifying two types of shilling behaviours: simple and delayed-start shilling. DT-f, which used our proposed feature set, performed significantly better than DT-r over low values of FPR. Results over a commercial dataset show the decision tree models, which were trained on synthetic data only, can identify users exhibiting characteristics consistent with shilling.

6 Unsupervised fraud detection

6.1 Introduction

There are two weaknesses in the approach proposed in Chapter 5 common to other supervised approaches. First, the dataset used for training determines the type of fraud that can be identified using the resulting model. The model will be less accurate when attempting to identify other types of fraud, or variations of the same type of fraud. In addition, strategies to commit fraud change over time as users who employ them are found and removed. To maintain model accuracy, it is then necessary to retrain models using updated labelled datasets. Second, our previous approach is not very good at detecting collaborative frauds since it does not make use of the available network information. While some previous methods, such as [19], make use of features derived from modelling the auction network as a graph, each user is still considered individually when determining whether they are fraudulent. The only exception is the 2-Level Fraud Spotting algorithm (2LFS) by Chau et al. [13], described in Chapter 2, which we compare against in Section 6.7.

In this chapter, we propose a novel approach which avoids the weaknesses listed above. Our approach consists of an anomaly detection phase and a belief propagation phase. Firstly, by using an anomaly detection method, our approach identifies all users who behave sufficiently differently from the majority of potentially fraudulent users, and can detect users with different fraud strategies, including previously unknown strategies, with the assumption that the majority of users are legitimate/normal. Secondly, groups of suspicious users can be found during the belief propagation phase. This is based on the assumption that users who interact with other suspicious users are also likely to be suspicious. Thirdly, we use validated synthetic data as previously done in Chapter 5 to avoid the problems associated with unknown ground truth in real data, and the problem of limited generalisability when using synthetic data. However, we also evaluate our method using real data in Chapter 7 to ensure that our approach does in fact identify anomalous groups of users in the real world.

6.1.1 Contributions

Our contributions in this chapter are as follows:

- We propose a novel approach for detecting collaborative fraud in online auctions. The approach, which we name SPAN, for Score Propagation over an Auction Network, contains two phases. In Phase 1, the anomaly scoring phase, the anomaly score of each user is calculated using a set of features describing that user. Specific to our approach is that outlier detection is performed, not in the whole feature space, but in carefully selected two-dimensional subspaces; this has several advantages, as described in Section 6.4. In Phase 2, the score propagation phase, the anomaly score for each user from Phase 1 are revised depending on their interactions with other users. This additional phase improves the overall accuracy of SPAN and allows groups of collaborating fraudulent users to be identified.

- We implement three types of collusive frauds described in previous literature. Combined with the auction simulation, we create multiple sets of synthetic data containing each fraud type, which is used to evaluate our proposed algorithm.

Section 6.2 gives background information important for understanding the chapter. Sections 6.3 to 6.5 describe the two phases of our proposed approach, and its time complexity. Section 6.6 defines the fraud types contained in the generated synthetic data used in evaluation. Section 6.7 presents evaluation results for SPAN and 2LFS under different conditions. Section 6.8 concludes the chapter.

6.2 Background

In this section, we give background information. Section 6.2.1 describes Oddball, an approach for finding unusual nodes in graphs. The idea of using feature pairs is used in Phase 1 of SPAN. Section 6.2.2 describes LOF, a density based outlier detection algorithm also used in Phase 1 of SPAN. Section 6.2.3 describes Markov random fields which is used to model the users and interactions in the online auction network.

6.2.1 Oddball

Oddball [49] is an algorithm for identifying anomalies in weighted graphs. The authors investigate the patterns present in 1-step neighbourhoods of nodes, which they call the *egonet*, and select four pairs of features whose values follow a power law relationship across different nodes: that is, $A \propto B^k$ for features A and B. If a node has an egonet that does not follow the power law relationship followed by the other nodes, then that node is likely to be an anomaly. The anomaly score of an instance is based on the distance of the node from the power law fitting the data.

For each anomaly identified using each of the four feature pairs, the anomalies will correspond to particular graph structures. They are: *CliqueStars*, where anomalies are more clique-like or star-like than non-anomalous nodes; *HeavyVicinity*, where anomalies have unusually high, or low, total edge weight compared to the number of edges; and *DominantPair*, where anomalies have a single dominant heavy edge in the egonet.

Phase 1 of our algorithm is based on the same idea that values of specific pairs of features will have relationships that are followed by the majority of nodes; which in our case, are users in an online auction. By carefully selecting the multiple of feature pairs, anomalies identified in these 2-dimensional feature subspaces can similarly be associated with specific structures in the graph network, which in turn correspond to specific bidding and selling behaviours. It should be noted that Oddball fits a power-law model to the data to identify outliers. However, Phase 1 of SPAN uses LOF, which we describe in Section 6.2.2 to identify outliers. Though any anomaly detection method that generates outlier scores can be used in place of LOF.

6.2.2 Local outlier factor

Local outlier factor (LOF) by Breunig et al. [50], is a density-based anomaly detection technique. As with other density-based techniques, LOF operates on unlabelled data. LOF has advantages over other density and distance-based anomaly detection techniques when there are regions of varying densities. The anomaly score of a given instance, called LOF, is the ratio of the average density of the k-closest neighbours and the local density of the instance itself. An instance will have a high anomaly score if the local density of the instance is significantly lower than the local average density. The parameter k is named *MinPts*. In [50], the authors propose calculating the LOF between two bounds: the lower bound *MinPtsLB*, and the upper bound *MaxPtsLB*, and taking the minimum value to avoid unwanted fluctuations at specific values of *MinPts*. Chandola et al. surveys other outlier detection methods, and describes proposed variations to the LOF algorithm [51].

6.2.3 Markov random field and belief propagation

Here, we give only a brief description of Markov random fields (MRF). For more details, refer to [1] which gives a comprehensive explanation of MRFs and belief propagation.

MRFs are a type of graphical model for solving inference problems given noisy observed data. MRFs have been used in computer vision problems, including noise-removal, inferring missing high-resolution details or other high-level features. An MRF consists of an undirected graph, containing nodes which can be in any number of states. For every hidden node about which we want to infer some quantity (x_i), there may be a fixed, possibly noisy, observation providing information about that node (y_i). In MRFs, the assumption is that there is some statistical dependency between x_i and y_i, represented using a *local evidence matrix* (ϕ), and between a hidden node (x_i) and each of its neighbours (x_j), represented using a *compatibility matrix* (ψ). Figure 6.1, taken from [1], shows an example of a square lattice MRF, where the empty circles represent hidden nodes, each of which are connected to an observed node, represented by filled circles.

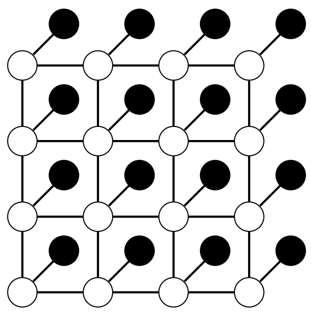


Fig. 6.1. Markov random field of a lattice [1]

For a particular set of states assigned to the hidden nodes in the MRF, the likelihood of that particular set of states occurring can be calculated using ϕ, ψ and the values of the observed nodes. To find the set of states with the maximum likelihood of occurring, one can calculate the likelihood of every combination of states, which increase exponentially with the number of nodes. Belief propagation can be used to efficiently find the set of states with the maximum likelihood, and takes time proportional to the number of nodes. Belief propagation is an iterative message passing algorithm, where the messages sent are opinions about what another node's state (belief) should be. The message m_{ij} going from node i to j is the product over all messages from neighbours of i, except for j:

$$m_{ij}(x) \leftarrow \sum_{x_i} \phi_i(x_i)\psi_{ij}(x_i, x_j) \prod_{k \in N(i \backslash j)} m_{ki}(x_i) \tag{6.1}$$

Each node then updates its belief according to the messages sent to it by its neighbours. The belief at a node i is the product of the evidence at that node $\phi_i(x_i)$, and all messages from neighbours of i:

$$b_i(x_i) = k\phi_i(x_i) \prod_{j \in N(i)} m_{ji}(x_i) \tag{6.2}$$

where x_i and $m_{ji}(x_i)$ are vectors with a length equal to the number of possible states, k is the normalisation constant, and $N(i)$ denotes the set of the nodes neighbouring i.

6.3 SPAN algorithm

SPAN (anomaly Score Propagation over an Auction Network) is divided into two phases, described in Sections 6.4 and 6.5 respectively. Phase 1 calculates an anomaly score for each user, and Phase 2 uses belief propagation over the MRF to use network features to improve the anomaly score accuracy. Section 6.5.1 discusses the theoretical complexity for each step of SPAN, and its overall complexity.

6.4 Phase 1: Calculating anomaly scores

Calculating the anomaly scores involves three steps. First, select an appropriate set of feature pairs; second, calculate LOF scores for each feature pair; and third, combine the set of LOFs for each user into one value.

Calculating LOFs over multiple feature pairs then combining them provides several advantages over using all features at once. Firstly, in higher dimensions, instances are spread in a larger volume, become less dense, and boundaries become more difficult to discern [52], making it more difficult to detect anomalies. Secondly, due to the variability of normal behaviours, normal instances may be misinterpreted as anomalies. By using manually selected feature pairs, any outliers identified are more likely to be significant, and can be more easily understood and explained. The quality of anomaly scores found using feature pairs and all features is compared in Section D.1 in Appendix D. Third, the computational complexity of nearest neighbour-based techniques, including LOF, do not scale well if the number of attributes is high [52]. By calculating LOF over only two dimensions, our approach scales well as the dataset size increases (see Section 6.5.1). A potential disadvantage of using feature pairs is that some anomalies may only be found from higher dimensions. However, we feel that the additional complexity of selecting and justifying feature triplets (or more) outweighs the benefits.

Section 6.4.1 describes how features are derived in the auction network, and modelled as a graph. Sections 6.4.2 and 6.4.3 explain the feature pairs chosen.

These features are combined into pairs, and an LOF calculated for each feature pair, for each instance. The choice of features and pairs are described and explained in Sections 6.4.2 and 6.4.3. The set of LOFs for each user can be combined in many ways, including using different types of means, or by taking into account the variance of each LOF score before averaging. In this work, we chose to simply use the maximum LOF value as the final anomaly score. Though this increases the number of normal users that will be identified as anomalous, Phase 2 of SPAN can remove these false positives.

6.4.1 Graphs and features

As mentioned previously, the interactions between users in the auction network can be represented using a graph. Each node of the graph represents a user. Depending on the definition of an edge, different weighted graphs can be created. Here we describe the edge definitions and node attributes used to create features in Sections 6.4.2 and 6.4.3. The notation used below has been used in describing the auction model in Section 3.3 of Chapter 3.

Edge definitions: Let the directed graph constructed using each edge definition be $\mathcal{D}$, where $\mathcal{D} = (V, A)$, where V is the set of vertices, and A is the set of directed edges. Given

$x \in V, y \in V$, where $u \in V \rightarrow u \in \mathcal{U}$, let $W(\{x, y\})$ be the weight of directed edge {x, y}. There is a directed edge from x to y if $W(\{x, y\}) > 0$.

- *Participation*: $W(\{x, y\}) = |\{a | a \in \mathcal{P}^x \wedge a \in \mathcal{S}^y\}|$, where $\mathcal{P}^x$ is the set of auctions x has submitted one or more bids to, and $\mathcal{S}^y$ is the set of auctions listed by y.

  There is a directed edge from x to y if x has bid in an auction listed by y. The weight of the edge corresponds to the number of auctions x has bid on that were listed by y.

- *Win*: $W(\{x, y\}) = |\{a | a \in \mathcal{W}^x \wedge a \in \mathcal{S}^y\}|$, where $\mathcal{W}^x$ is the set of auctions won by x.

  There is a directed edge from node x to y if x has won in an auction listed by y. The weight of the edge corresponds to the number of auctions won by x, which were listed by y.

- *Loss*: $W(\{x, y\}) = |\{a | a \in \mathcal{L}^x \wedge a \in \mathcal{S}^y\}|$, where $\mathcal{L}^x$ is the set of auctions lost by x. There is a directed edge from node x to y if x has lost in an auction listed by y. The weight of the edge corresponds to the number of auctions lost by x, which were listed by y.

- *($\mathcal{D}^T$)*: The transpose of a directed graph $\mathcal{D}$.

- *Seen*: $W(\{x, y\}) = W(\{y, x\}) = |\{a | a \in \mathcal{P}^x \wedge a \in \mathcal{S}^y\}|$

  There is a directed edge from node x to y, and from y to x, if x has bid in an auction listed by y, or vice versa. This is simply the *Participation* graph with undirected edges. If the graph is bipartite, bidders will have exactly the same number of edges and edge weights. Thus comparing node attributes in this graph to the *Participation* graph is effective in identifying users who are behaving as both sellers and bidders.

- *Connected seller*: $W(\{x, y\}) = W(\{y, x\}) = |\{z | z \in \mathcal{U} \wedge a \in \mathcal{P}^z \wedge b \in \mathcal{P}^z \wedge a \in \mathcal{S}^x \wedge b \in \mathcal{P}^y\}|$

  There is a directed edge from x to y and from y to x if another user z has participated in at least one auction listed by users x and y respectively. The weight of the edge corresponds to the number of users z for which this is true.

Node Attributes: Each node has a value for each of the following node attributes. We also give the value for each node attribute for the seller u_0 in Figure 6.2.

- *Edge count*: number of edges leaving this node. Here *edge count*$(u_0) = 4$.
- *Weight count*: sum of the weights of edges leaving this node. Here *weight count*$(u_0) = 5$.
- *Egonet edge count*: sum of all node degrees within the 1-step neighbourhood subgraph of the node. If the graph is bipartite, this value equals *edge count*. Here *egonet edge count*$(u_0) = 6$.
- *Egonet weight count*: sum of all edge weights within the 1-step neighbourhood subgraph of the node. If the graph is bipartite, this value equals *weight count*. Here *egonet weight count*$(u_0) = 8$.
- *First eigenvalue*: first eigenvalue of the 1-step neighbourhood subgraph of the node, for undirected graphs only (*Seen*). The value depends on the edge weight distributions of the subgraph. If Figure 6.2 is treated as an undirected graph, *first eigenvalue*$(u_0) = 3.837$.
- *Repeat count*: same as *edge count* −*weight count*. Represents the number of interactions with previously known users. Here *repeat count*$(u_0) = 1$.

6.4.2 Bidder features selected

Table 6.1 lists the feature pairs selected for bidders. Each feature consists of the graph edge definition used, and a node attribute. Each feature pair in Table 6.1 is explained below:

Table 6.1
Feature pairs for bidders

	Feature 1		Feature 2	
	Graph edge	Node attribute	Graph edge	Node attribute
1	Participation	Weight count	Seen	Egonet edge count
2	Win	Edge count	Seen	Repeat count
3	Loss	Weight count	Seen	Egonet edge count
4	Loss	Edge count	Seen	First eigenvalue

1. The number of auctions per unique seller each user participates in. Identifies users who participate in the same set of sellers' auctions. In bipartite graphs, *Seen* and *Egonet edge count* in Feature 2 corresponds to *Particpation* and *Weight count*, but are used here in order to identify users who do not form bipartite cores: who act as both bidders and sellers. This feature pair will identify users who interact with the same sellers more frequently than normal.

2. The number of auctions won per repeated seller. The number of auctions won, and the number of participations in auctions from previously known sellers. Identifies users who tend to win more auctions from the same sellers compared to normal users, such as those committing reputation fraud.

3. The number of auctions lost per seller. Identifies users who lose repeatedly in auctions from the same sellers, such as those committing competitive shilling.

4. The number of auctions lost to first eigenvalue of the 1-step neighbourhood. Identifies users who repeatedly interact with the same set of sellers, even if they attempt to mask this by interacting with other users, due to the difference in the distribution of edge weight compared to normal users.

6.4.3 Seller features selected

Table 6.2 lists the feature pairs selected for bidders. Each feature pair in Table 6.1 is explained below:

1. The number of times a seller has a bidder in common with other sellers. Identifies sellers who have bidders who bid in auctions by the same set of sellers repeatedly.

Table 6.2
Feature pairs for sellers

	Feature 1		Feature 2	
	Graph edge	Node attribute	Graph edge	Node attribute
1	Connected seller	Edge count	Connected seller	Weight count
2	Loss^{T}	Repeat count	Seen	Repeat count
3	Participation^{T}	Edge count	Seen	Egonet weight count
4	Participation^{T}	Weight count	Seen	First eigenvalue

2. Compares the number of bidders who lost to this user's auction, to the total number of bidders seen. Identifies users who have bidders losing to them repeatedly.

3. Compares the number of bidders seen, and the number of unique bidders. Users who are in bipartite graphs, *Seen* and *Egonet weight count* corresponds to *Participate^{T}* and *Weight count*, but are used here to identify users who act as both bidders and sellers.

4. Compares the number of bidders seen to the first eigenvalue of the 1-step neighbourhood. Identifies users who repeatedly interact with the same set of bidders which produces several edges with heavy weights.

6.5 Phase 2: Propagating anomaly scores

The auction network is modelled using an MRF. Each user is represented by a hidden node in the graph, and each user (hidden node) is connected to a corresponding observed node representing what is observed about the user. The graph formed by the hidden nodes is the graph constructed using the *Seen* edge definition described in the previous section. Each node i, can have two states: fraudulent (i_f) or normal (i_n). The beliefs at each node sum to 1: $i_f + i_n = 1$. The fraudulent state subsumes the additional accomplice state used in [13]. Using only two states allows SPAN to be more general, as discussed in Section 6.7.1.

The anomaly scores found for each user in the first phase becomes the observed state of the node in the graph. Since the anomaly scores can be any positive value, they are first normalised to between 0 and 1 using Equation 6.3.

$$o_f = max\left\{0, 1 - \frac{1}{a \times (anomaly\ score - b)}\right\} \tag{6.3}$$

The normalised value o_f is the fraudulent belief of the observed node, and the normal belief of the observed node is $o_n = 1 - o_f$. The observed states for each node comprise the *Observation Matrix* input to Algorithm 1. The matrices underlying the inputs *Local evidence function* (ϕ) and *Compatibility function* (ψ) are shown in Tables 6.3 and 6.4.

Algorithm 1 follows the message propagation and belief update rules in Equations 6.1 and 6.2. However, to take into account the strength of the interaction between users, we modify the belief update rule by adding an extra term, which gives additional weight to messages from users who are connected with a high edge weight, as shown in Equation 6.4. $W(\{j,i\})$ is the weight of the edge between j and i.

$$b_i(x_i) = k\phi_i(x_i) \prod_{j \in N(i)} m_{ji}(x_i)^{W(\{j,i\})} \tag{6.4}$$

After the algorithm converges, the nodes are ranked according to their belief for the fraudulent state. A threshold is then used to divide the set of users into fraudulent and normal. Choosing an appropriate threshold depends on two factors: the relative misclassification cost for normal and fraudulent users, and the ratio of normal and fraudulent users in the data. Without knowledge of these factors, it is difficult to select an appropriate threshold. Previous work to estimate the prevalence of fraud ([9, 15]) estimates that around 0.2% of auctions are fraudulent, based on user feedback. However, this is likely an underestimation since frauds such as shilling may not be recognised by legitimate bidders when they occur. In any case, the value of 0.2% does not directly translate to the fraction of users that are fraudulent. Without more knowledge, we suggest a conservative threshold and classify the most suspicious 1% of users as fraudulent.

Table 6.3
Local evidence matrix (ϕ): entry (p, q) gives the probability that the node is in state q given the observed state p.

Observed state	Node belief	
	Fraud	Normal
Fraud	$1-l$	l
Normal	l	$1-l$

Table 6.4
Compatibility matrix (ψ): entry (p, q) gives the probability that the node is in state q given the neighbour state p.

Neighbour state	Node belief	
	Fraud	Normal
Fraud	$1-j$	j
Normal	k	$1-k$

Algorithm 1 Pseudocode for belief propagation

```
Input: Number of States: $S = 2$, Number of Nodes: $N$, Observation Matrix: $O[N][S]$,
Local evidence function: $\psi$, Compatibility function: $\phi$
Output: Belief Matrix: $B[N][S]$
1: $B$ = newMatrix$[N][S]$                              ▷ Matrix containing beliefs for hidden nodes
2: $M$ = newMatrix$[N][S]$                              ▷ Matrix containing messages for neighbours
3: for $i \in 1..N$ do
4:     $B[i] = \psi(\mathrm{O}(k))$                     ▷ Initialise node beliefs with observations
5: end for
6: while not converged do
7:     for $i \in \{1..N\}$ do
8:         for $k \in$ neighbours($i$) do
9:             $M[i]$ = multiply($m$, $\psi(\mathrm{B}(k))$  ▷ Gather propagated message from neighbour
10:        end for
11:    end for
12:    for $i \in \{1..N\}$ do
13:        for $k \in$ neighbours ($i$) do
14:            $m_k$ = newArray$[S]$                     ▷ Array storing new message
15:            $m_k = \psi(M[k])$                        ▷ Retrieve the message from the neighbour
16:            $m_k$ = divide($m_k$, $\psi(M[i])$)       ▷ Remove the contribution by this node
17:            $w_k$ = edge_weight($i$,$k$)
18:            $m_k = {m_k}^w$                           ▷ Effect of edge weight connecting the two nodes
19:            $m$ = multiply($m$, $m_k$)
20:        end for
21:        $m$ = multiply($m$, $\phi(O[i])$)             ▷ Effect of node observation
22:        $m$ = normalise($m$)                          ▷ Normalise beliefs to sum to 1
23:        $B[i] = m$                                    ▷ Update belief matrix
24:    end for
25: end while
```

Parameter selection

Here we suggest values for parameters required for SPAN. The parameter b in Equation 6.3 determines how strongly changes in LOF changes fraudulent belief for the corresponding observed node (o_f). A value of $b = 0.4$ is suggested. At this value, when $anomalyscore = 3.5$, $o_f = 0.5$. The parameter l reflects uncertainty of the observed beliefs. We set $l = 0.2$, a default suggested by [13].

The values j and k reflect the likelihood a user in either state will interact with another user. A range of values for j and k were tested, and the pair with the highest difference in average fraudulent belief between the fraudulent instances and normal instances was chosen. The optimum values found are: $j = 0.12$, $k = 0.36$. Intuitively, these values seem reasonable: a fraudulent user is likely to interact mostly with other fraudulent users, while a normal user does not differentiate between normal and fraudulent users when participating in auctions; thus $j < k$. The same procedure was used to find the optimum value of $\epsilon_p = 0.12$ for 2LFS, which we evaluate SPAN against in Section 6.7. The parameter ϵ_o (equivalent to parameter l in SPAN) was set at a default of 0.2. The synthetic dataset used for tuning is separate from that used later in Section 6.7 for evaluation.

6.5.1 Theoretical time complexity

Here we describe the theoretical complexity for each step of SPAN. Let w be the number of bids in the auction network, n be the number of nodes in the graph, e be the number of edges in the graph, and $m = e/n$ be the average number of edges per node.

Complexity of each step in Phase 1

The graph construction step has time complexity $\mathcal{O}(w)$, where all bids must be looked at once to construct each graph. The node attribute calculation step has time complexity $\mathcal{O}(e)$ for all features, except for the *first eigenvalue* feature which is found in $\mathcal{O}(nm^2)$ using the power iteration approximation method. This can be simplified to $\mathcal{O}(\frac{e^2}{n})$, then approximated to $\mathcal{O}(e)$, assuming that e and n grows proportionally to each other. For the LOF calculation step, the time complexity is $\mathcal{O}(n)$ using the grid method for low dimensional data [50]. Since $w > e > n$, the time complexity for Phase 1 is $\mathcal{O}(w)$.

Complexity of each step in Phase 2 The initialisation step has time complexity $\mathcal{O}(n)$, where the anomaly score for each node is normalised and data structures are allocated. The time complexity of belief propagation is $\mathcal{O}(e)$ per iteration, where the number of messages passed is proportional to the number of edges. In the average case, the number of iterations during the belief propagation step can be treated as a constant, which gives the belief propagation step a time complexity of $\mathcal{O}(e)$. Thus the time complexity for Phase 2 is $\mathcal{O}(e)$, and the overall complexity of SPAN is $\mathcal{O}(w)$.

6.6 Fraudulent behaviours

Additional agents are inserted into the simulation to generate synthetic data containing fraudulent users and transactions. The three types of collaborative frauds implemented are described below. Figures 6.3, 6.4 and 6.5 show the interactions between users committing each type of fraud. In these figures, an arrow from node A to B shows that user A made a bid on an auction held by user B. Arrows which have a free end represent interactions with legitimate users which are not shown.

6.6.1 Reputation fraud

As stated previously, reputation fraud is committed by two groups of users: one group which wants their reputation to be increased, and another who provides the positive feedback and comments. As shown in Figure 6.3, the users generally form a bipartite core. To mask their suspicious behaviour, the bidders also participate in normal auctions, while the sellers simply conduct legitimate auctions with the benefit of the fraudulently obtained positive reputation scores. This strategy is the one investigated in [9] and [14]. A variation of this has been identified in real data in [13].

In the simulation, reputation fraud is modelled as 50 sellers and 400 bidders who attempt to inflate the sellers' reputation. Each seller attempts to inflate their reputation by between 30 and 100 points over the duration of the simulation, where each fraudulent auction increases reputation by one. Each bidder has an equal chance of being chosen to win a particular auction. In the simulation, a single agent is responsible for coordinating the behaviour of the 50 sellers and 400 bidders to commit reputation fraud. The pseudocode for the controlling agent is shown in Section A.4 in Appendix A.

Collusive star

In collusive shilling fraud, multiple users work to bid on auctions by a single seller. The fraudulent bidders are scheduled so that a random subset bids on a particular auction. In addition, the bidders will participate in normal auctions to mask their suspicious behaviour, while normal users participate in auctions held by the seller. Since all the fraudulent bidders only collaborate with one seller, the fraudulent group forms a star.

In the simulation, collusive stars are modelled as 1 seller working with 6 bidders, who submit between 20 to 50 fraudulent auctions on average over the simulation duration. In each fraudulent auction, a random set of 3 bidders are selected to shill in the auction. In each dataset, there are a total of 50 groups working independently. The pseudocode for the controlling agent is shown in Section A.5 in Appendix A.

Collusive clique

This is a variation of collusive shilling fraud. Here, multiple users act as both bidders and sellers, and bid on auctions listed by other users in the group. Once again, the users mask their behaviour by participating in legitimate auctions, while normal users participate in fraudulent auctions held by the sellers. As the number of interactions between fraudulent users increases, the group forms a clique.

In the simulation, collusive cliques are modelled as groups of between 10 and 20 users. Each user in the clique submits six fraudulent auctions which are bid on by 3 users randomly selected from within the same clique, excluding the user itself. In each synthetic dataset, there are 30 groups working independently. The pseudocode for the controlling agent is shown in Section A.6 in Appendix A.

6.7 Evaluation results

In this section, we present evaluation results for SPAN. Evaluation results are split into 5 parts, and each part reports the accuracy of SPAN under different conditions.

- Section 6.7.1 reports results under normal conditions. SPAN is evaluated using synthetic datasets containing three types of fraud. We also report the accuracy of 2LFS, and the accuracy of Phase 1 of SPAN. It is expected that the performance of 2LFS will be lower than SPAN. However, 2LFS is the only previous work that is comparable to ours. We also report results obtained by searching for k-core and clique graph structures to identify groups of users with close trading relationships.

- Section 6.7.2 shows the average change in beliefs between iterations during belief propagation in Phase 2 of SPAN, to show that beliefs converge within a reasonable number of iterations.

- Section 6.7.3 shows the effect of the weight exponent in Equation 6.4 on accuracy.

- Section 6.7.4 shows the change in accuracy observed when the network structure, which Phase 2 of SPAN relies on, is altered. Specifically, we report accuracy changes when we remove connections between fraudulent users, so that the collusive groups appear less connected; and when we remove connections between fraudulent and normal users, so that fraudulent groups become more isolated.

- Section 6.7.5 shows the change in accuracy when we reduce the quality of the anomaly scores that are given to Phase 2 as input. Specifically, we randomly replace different proportions of values of the observed states (normalised anomaly scores) of fraudulent users using the observed states of normal users, and measure the effect this has on detection accuracy. Similarly, we replace observed states of normal users using the states of fraudulent users as well.

To generate the results for 2LFS, the inputs used were: the auction graph, the optimised propagation and observation matrices, and the same set of observed states that are input to Phase 2 of SPAN. All evaluations were done on an Intel Core i5 machine with 8GB RAM running Windows 7. All of the results reported in each section, and for each fraud type, are average values from 20 datasets.

6.7.1 Performance

In addition to comparing SPAN and 2LFS, we also report results for Phase 1 of SPAN alone, and results for traditional graph algorithms. Results for Phase 1 only shows the effect Phase 2 has on accuracy. Results for traditional graph algorithms serves as the baseline against which SPAN and 2LFS are compared.

Three sets of data were used, each of which contains one of the three fraud types described in Section 3.5.5. Since the output of the SPAN algorithm is a ranked list of scores, a higher true positive rate (TPR) can be obtained at a cost of a higher false positive rate (FPR). Thus we report a range of TPR/FPR value pairs, ROC curves and the AUC. Tables 6.5 to 6.7 show, for each algorithm, the area under ROC (AUC), and the true positive rate (TPR) when the false positive rate (FPR) is 0.005, 0.01 and 0.05, while Figures 6.6 to 6.8 show the ROCs.

The ROCs show that SPAN has much higher detection accuracy than 2LFS, especially over low values of FPR. For collusive cliques, 2LFS has slightly better detection accuracy when FPR is greater than 0.05, though at lower values, it performs worse than SPAN, with and without the propagation phase. The results also show the value of the second phase of SPAN, where accuracy is increased for all fraud types over all FPRs. Wilcoxon's signed rank test was used to ensure that the difference in performance from SPAN and 2LFS was significant. Using a threshold of 0.01, each instance with the correct classification is given a value of 1, otherwise 0. Using the alternative hypothesis that SPAN has more correctly classified instances, the p-values were all less than 0.05 for each of 20 datasets for each of the three fraud types, indicating that the accuracy improvement of SPAN over 2LFS is statistically significant.

Comparison with traditional graph algorithms

The search for connected groups of users in Phase 2 of SPAN has some similarities with classical graph mining algorithms for finding maximal cliques and k-cores. To show that SPAN outperforms these algorithms, we attempt to detect groups of fraudulent users using their k-core and clique memberships. Briefly, a k-core of a graph $\mathcal{G}$ is the maximal connected subgraph of $\mathcal{G}$ in which every vertex has degree of at least k [53]. A clique is a set of vertices where every pair of vertices is connected by an edge. For each node, we can find the value j, which is the highest k-core the node is a member of. All nodes can be ranked using j. Given a threshold, we can treat nodes with j above the threshold as anomalous, and calculate the proportion of legitimate and fraudulent users that are found. Using every value of j as the threshold, we obtain a set of (TPR, FPR) pairs. These pairs are used to construct the ROC curves in Figures 6.6 to 6.8, and calculate the AUC. To obtain the values in Tables 6.5 to 6.7 for FPR values of 0.005, 0.01 and 0.05, we simply use linear interpolation to find the corresponding TPR.

Similarly, we can find the largest clique each node belongs to, and apply the same series of steps as for k-core. The results show that for Reputation Fraud and Collusive Cliques, using cliques performs no better than a random-classifier. However, for collusive stars it achieves almost perfect accuracy, since legitimate users almost never form cliques. That is a limitation of the fraud agent programming, and it is possible to modify the agents so that cliques do not form. In fact, if edges between fraudulent users were removed as for experiments in Section 6.7.4, accuracy using clique detection would fall to zero.

Discussion

2LFS performs better than Phase 1 of SPAN when detecting reputation fraud, but worse for collusive stars and collusive cliques; i.e. 2LFS actually produced a set of beliefs that were less accurate than those given as input. There are several reasons for the poor performance of 2LFS. First, 2LFS assumes that normal users do not form bipartite cores and do not behave as both bidders and sellers. However, this is false in the Aug 2012 dataset (described in Chapter 3) used in the case study in Chapter 7, where 90.01% of users either bid or submit auctions, but not both. Given the states and structure of the compatibility matrix chosen by 2LFS which tends to identify bipartite cores as suspicious, this results in a high number of false positives. Second, 2LFS is tailored to a specific fraud type: reputation fraud where the sellers are very unlikely to interact with normal users except when they commit non-delivery fraud. As a result, for the collusive star and collusive clique fraudulent behaviours, the accuracy of 2LFS is lower than Phase 1 of SPAN.

Using k-cores and cliques to identify users with unusually close trading relationships yields limited success. For all three fraud types, using k-cores gives significantly lower accuracy. Searching for cliques yields near perfect accuracy for Collusive Clique frauds only. The difficulty with using this approach is that one must have prior knowledge of the graph structures that the fraudulent agents will form, this is difficult especially in real settings where users adapt their behaviour to avoid detection.

Table 6.5
Detection accuracy for reputation fraud

Method	FPR=0.005		FPR=0.01		FPR=0.05		AUC	StdDev
	TPR	StdDev	TPR	StdDev	TPR	StdDev		
SPAN	0.985	1.91E-03	0.989	5.40E-04	1.000	0	1.000	8.58E-07
2LFS	0.071	1.42E-03	0.133	2.59E-02	0.761	0.292	0.959	8.14E-04
Phase 1 only	0.016	5.76E-04	0.091	1.68E-02	0.357	0.308	0.844	0.153
k-core	0.544	0.067	0.582	0.059	0.753	0.051	0.954	9.84E-03

Table 6.6
Detection accuracy for collusive stars

Method	FPR=0.005		FPR=0.01		FPR=0.05		AUC	StdDev
	TPR	StdDev	TPR	StdDev	TPR	StdDev		
SPAN	0.223	5.74E-03	0.476	0.650	0.977	5.00E-03	0.985	3.54E-04
2LFS	0.141	1.35E-04	0.144	1.55E-04	0.314	0.141	0.756	0.014
Phase 1 only	0.152	3.01E-03	0.204	2.48E-02	0.809	0.305	0.924	0.050
k-core	0.043	0.012	0.086	0.024	0.197	0.010	0.783	7.18E-03

Table 6.7
Detection accuracy for collusive cliques

Method	FPR=0.005		FPR=0.01		FPR=0.05		AUC	StdDev
	TPR	StdDev	TPR	StdDev	TPR	StdDev		
SPAN	0.910	0.492	0.944	0.204	0.983	1.25E-02	0.992	3.00E-03
2LFS	0.036	2.99E-02	0.312	0.544	0.982	7.39E-03	0.980	1.36E-03
Phase 1 only	0.586	0.717	0.758	0.715	0.950	0.456	0.971	0.125
k-core	0.685	0.067	0.755	0.054	0.949	0.028	0.980	0.013

6.7.2 Convergence

Results show, empirically, that the node beliefs during belief propagation in Phase 2 of SPAN converges in 30 to 80 iterations. Convergence is generally reached within 30-40 iterations for reputation fraud and collusive stars, and within 80-90 iterations for collusive cliques. Figure 6.9 shows the average change in belief between each iteration of belief propagation, averaged over 20 datasets.

6.7.3 Weight exponent

The effect of the weight exponent $W\{(j,i)\}$ in Equation 6.4 on accuracy is reported here. We compare the accuracy of the SPAN algorithm with and without the W term during Phase 2. The accuracy results for all three fraud types are shown in Table 6.8. The results show that accuracy is similar with and without the W term for reputation fraud and collusive cliques. However, for collusive stars, omitting W greatly reduces accuracy. The ROC curve in Figure 6.10 shows that accuracy is reduced to similar to that of 2LFS.

This result is reasonable if we consider the graph structure of each fraud type. For both reputation fraud and collusive cliques, there are multiple edges that connect the agents in the group together. The number of edges connecting the users allows the groups to be identified even without the weight term. However, for collusive stars, each node is only connected to another in the group by one edge, save the node at the star's centre. Thus, the weight term in Equation 6.4 is required for collusive stars to be detected, since it takes into account the strength of the edge connecting the group.

Table 6.8
Detection accuracy of SPAN with and without weight exponent W

	Method	FPR=0.005		FPR=0.01		FPR=0.05		AUC	StdDev
		TPR	StdDev	TPR	StdDev	TPR	StdDev		
Reputation fraud	SPAN	0.985	1.90E-03	0.989	5.70E-04	1.000	0	1.000	8.58E-07
	No weight	0.989	6.40E-04	0.993	3.21E-04	1.000	0	1.000	1.14E-07
Collusive star	SPAN	0.223	0.283	0.476	0.650	0.977	5.00E-03	0.985	3.54E-04
	No weight	0.101	2.90E-02	0.115	0.029	0.230	0.119	0.663	0.361
Collusive clique	SPAN	0.910	0.492	0.944	0.204	0.983	0.013	0.992	0.003
	No weight	0.898	0.458	0.934	0.181	0.977	0.045	0.984	0.044

6.7.4 Edge removal

In this section, we evaluate the behaviour of the SPAN algorithm when different proportions of fraud-fraud edges and fraud-normal edges are randomly removed. For example, if 20% of fraud-fraud edges were to be removed, then each edge connecting two fraudulent users will have 20% chance of being removed. Below, we show how the number of detected fraudulent users changes for the SPAN and 2LFS algorithms as the proportion of removed edges changes from 0% to 100%. For all figures shown, the TPR values correspond to an FPR value of 0.01.

Fraud-Fraud edge removal

As edges between fraudulent users are removed, they become more difficult to identify as a group since the degree of their collaboration is reduced. As the proportion of all fraud-fraud edges are removed, the accuracy of SPAN is expected to decrease. The accuracy of SPAN, shown in Figure 6.12(a) decreases as expected for the fraud types collusive star and collusive clique, and levels off when all edges are removed. This level of remaining accuracy is likely due to the influence the observed nodes have on the final beliefs of the hidden nodes for the fraudulent users.

However, for reputation fraud, the accuracy reduces to around 0.01 when roughly 30% of edges are removed, then increases again. We suspect the extreme changes in accuracy may be due to the fraudulent accounts connected as a single bipartite core (Figure 6.11(a)). When sufficient fraud-fraud edges are removed, the belief of the fraud nodes decreases since most interactions are now with normal users (Figure 6.11(b)), which in turn affects all other nodes in the bipartite core, causing all beliefs in the network to change. As even more edges are removed, fraudulent users become isloated and no longer affect each others' beliefs (Figure 6.11(c)), at which point the observed nodes have greater influence over the final belief of the fraudulent users, causing the accuracy to increase.

The change in accuracy for 2LFS is generally as predicted. The consistent accuracy for collusive star is due to all ROCs having the same TPR when FPR = 0.01 when different proportions of fraud-fraud edges are removed. This is explained in more detail in Appendix E.

Fraud-Normal edge removal

As edges between normal and fraudulent users are removed, the collaborative frauds become more isolated. Accuracy is expected to increase as the proportion of fraud-normal edges are removed. Figure 6.13(a) shows that the accuracy changes as expected for all three fraud types for SPAN. There are much more significant increases in accuracy for collusive star since accuracy is lower to begin with, so that there is more room for improvement when fraud-normal edges are removed.

The unpredictability in the accuracy for 2LFS is likely due to the various assumptions about bipartite cores and user networks being met or violated when different proportions of edges are removed. The relatively consistent accuracy for reputation fraud is explained in Appendix E.

6.7.5 Randomized states for observed nodes

This section evaluates the effect on final accuracy when we add noise to the states of observed nodes (normalised anomaly scores from Phase 1) used as input to Phase 2 of SPAN. This is done by reassigning the states of a proportion of the observed nodes. If we were to change 20% of the states for fraudulent users, each observed node would have a 20% probability of having their states replaced with that of a normal user's state. The normal user is chosen at random, with replacement. In Section 6.7.5, we report the effect on detection accuracy as the proportion of states changed increases from 0% to 100% for the observed nodes of fraudulent users. In Section 6.7.5, we report the results for the reverse by changing the observed states for normal users. In Figures 6.14 and 6.15, the TPR values correspond to an FPR value of 0.01.

Randomised states of observed nodes for fraudulent users

As we increase the proportion of observed nodes' states that are reassigned for fraudulent users, detection accuracy is expected to decrease. Figure 6.14 shows that the accuracy for all three fraud types changes as expected. The accuracy for detecting collusive stars and collusive cliques changes gradually, while accuracy for reputation fraud decreases much more abruptly. Once again, this is due to the single bipartite core formed by the fraudulent users, so that the beliefs of all users in the core heavily influence each other. It should be noted that when less than 15% of states are changed, there is very little change in the accuracy for SPAN. This value is higher for collusive stars and collusive cliques. The relative resilience of 2LFS for reputation fraud is due to the ability of 2LFS to specifically identify bipartite cores due to the propagation matrix selected, despite inaccurate initial beliefs for fraudulent nodes, as noted previously in [13]. The change in accuracy for the other two fraud types for 2LFS is as expected. The constant accuracy for collusive star is explained in Section E.

Randomised states of observed nodes for normal users

As we increase the proportion of observed nodes' states that are reassigned for normal users, detection accuracy is expected to decrease. Figure 6.15 shows that the direction of change is again as expected. Once again, the change in accuracy when detecting reputation fraud is different from collusive star and collusive clique due to the size of the bipartite core. Unlike results in Section 6.7.5, accuracy begins falling immediately when score states are changed. This is particularly marked for collusive star. This is likely due to the lower quality of scores generated by SPAN in Phase 1 for collusive star, compared to the other two fraud types (compare Figures 6.6 to 6.8). The reason behind 2LFS's resilience for Reputation Fraud is the same as before: due to the ability of 2LFS to focus on bipartite cores. The change in accuracy for the other two fraud types for 2LFS is as expected. Once again, the constant accuracy for collusive star is explained in Section E.

6.8 Conclusion

Existing fraud detecting approaches have several limitations that make them less effective when used in a real world setting. First, each method is designed for one type of fraud, and will gradually lose accuracy as fraud behaviour changes over time. They will also be ineffective for detecting other types of fraud. Second, the majority of existing methods do not make use of the user interaction graph to detect collaborative frauds. Third, datasets used in previous work are limited: no previous work has validated their synthetic data before use, and ground truth cannot be known for real data.

We propose a method called SPAN for identifying groups of users that commit fraud collaboratively. SPAN is able to accurately identify groups of fraudulent users committing a range of collaborative frauds. The advantage of SPAN over previous methods is due to two factors. Firstly, the use of an anomaly detection approach to label all users who behave differently from the majority as suspicious. This allows different types of unusual behaviours to be detected without the need to define them explicitly. Secondly, SPAN assumes that fraudulent users are more likely to interact with each other than with normal users, without assuming patterns in those interactions. SPAN's weaker assumptions are less likely to be violated, which allows SPAN to perform well under a wider range of conditions than previous methods such as 2LFS.

For evaluation, we generated multiple sets of validated synthetic data which contain three different types of fraud. The extensive evaluations performed show that SPAN is able to perform well across a range of fraud types, and that both phases of SPAN are valuable in accurately identifying groups of fraudulent users. We have also evaluated SPAN's behaviour under different conditions, and found that the behaviour of SPAN is generally consistent with intuition.

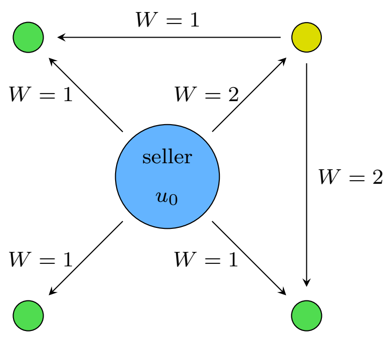


Fig. 6.2. Example 1-step neighbourhood of seller u_0

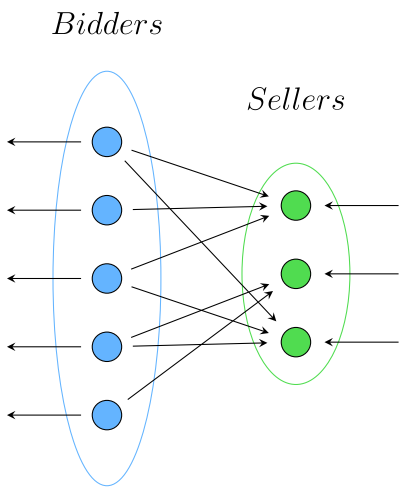


Fig. 6.3. Reputation Fraud

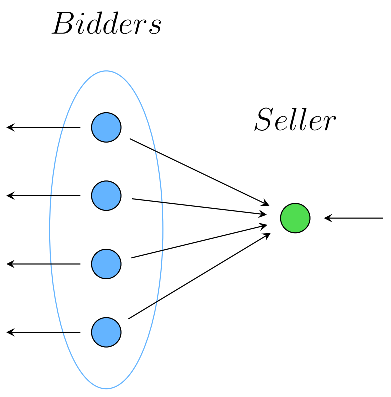


Fig. 6.4. Collusive Star

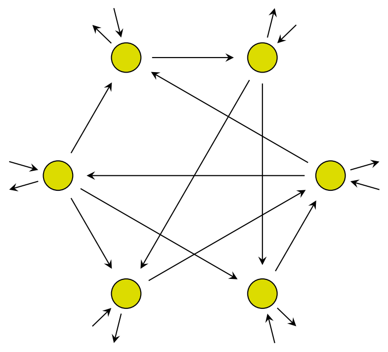
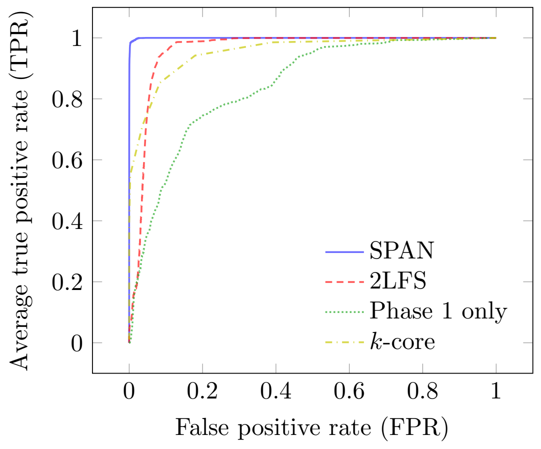
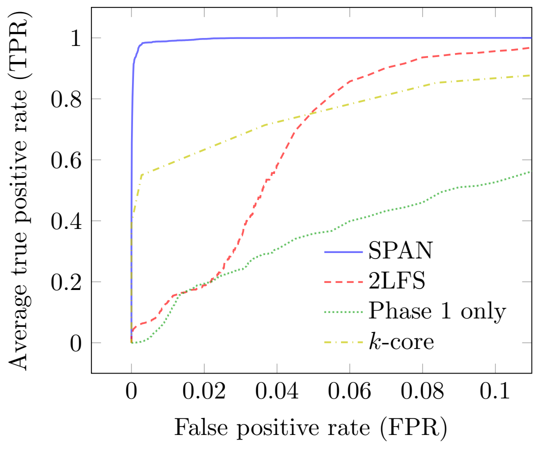


Fig. 6.5. Collusive Clique

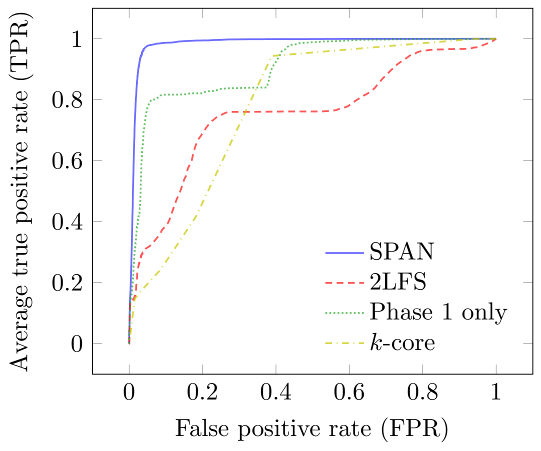


(a) All values of FPR

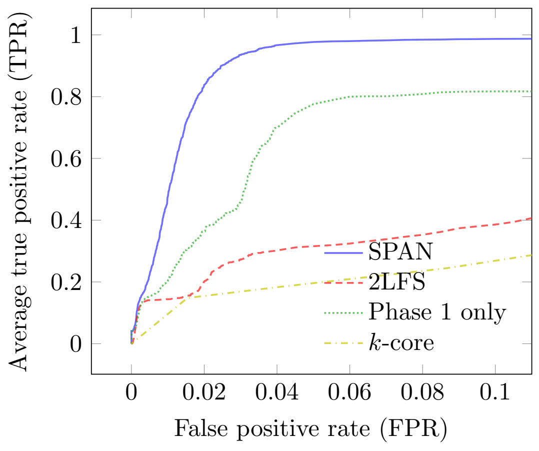


(b) FPR between 0 and 0.1

Fig. 6.6. Average ROCs for SPAN, 2LFS and without score propagation for reputation fraud

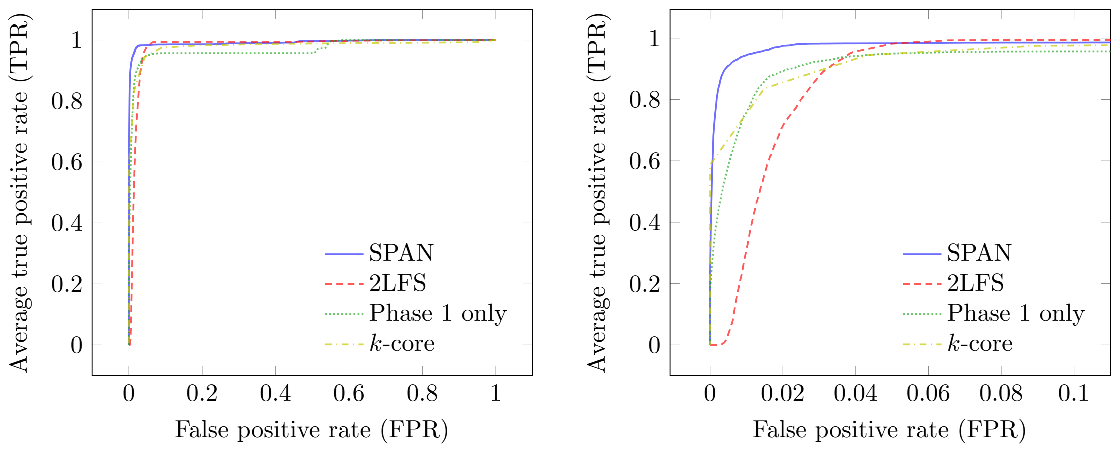


(a) All values of FPR



(b) FPR between 0 and 0.1

Fig. 6.7. Average ROCs for SPAN, 2LFS and scores without propagation for collusive star



(a) All values of FPR **(b)** FPR between 0 and 0.1

Fig. 6.8. Average ROCs for SPAN, 2LFS and scores without propagation for collusive clique

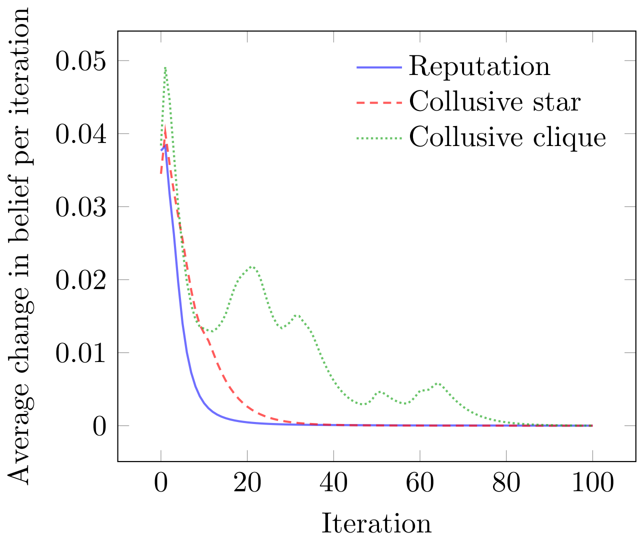


Fig. 6.9. Average change in belief between each iteration

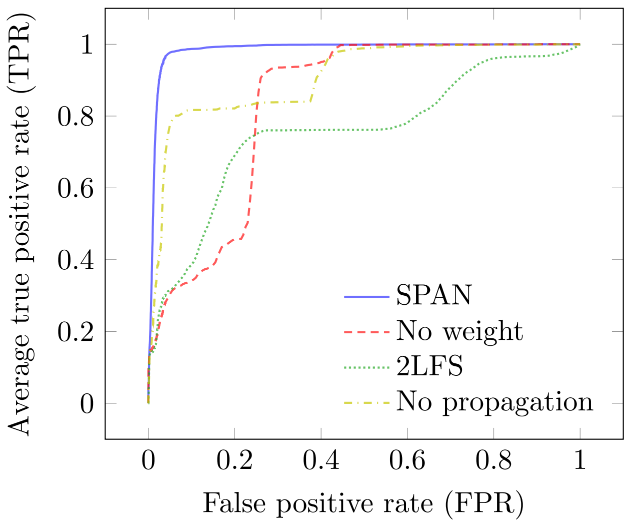
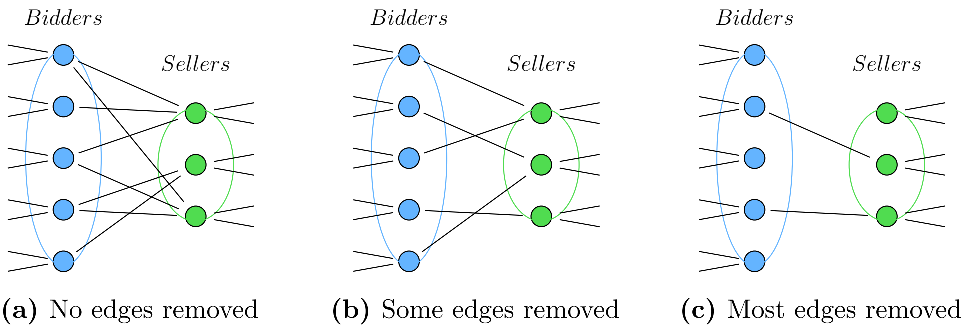
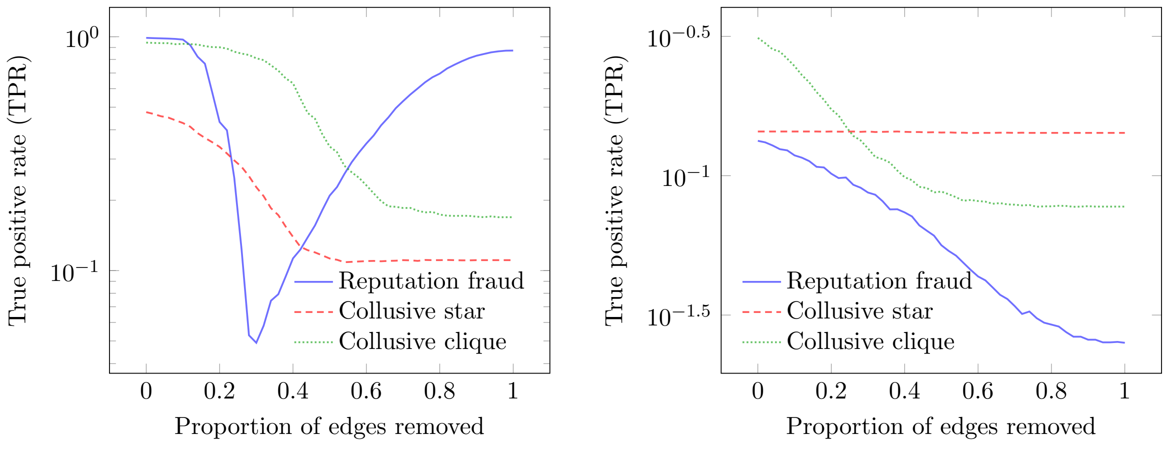


Fig. 6.10. Accuracy for detecting collusive stars



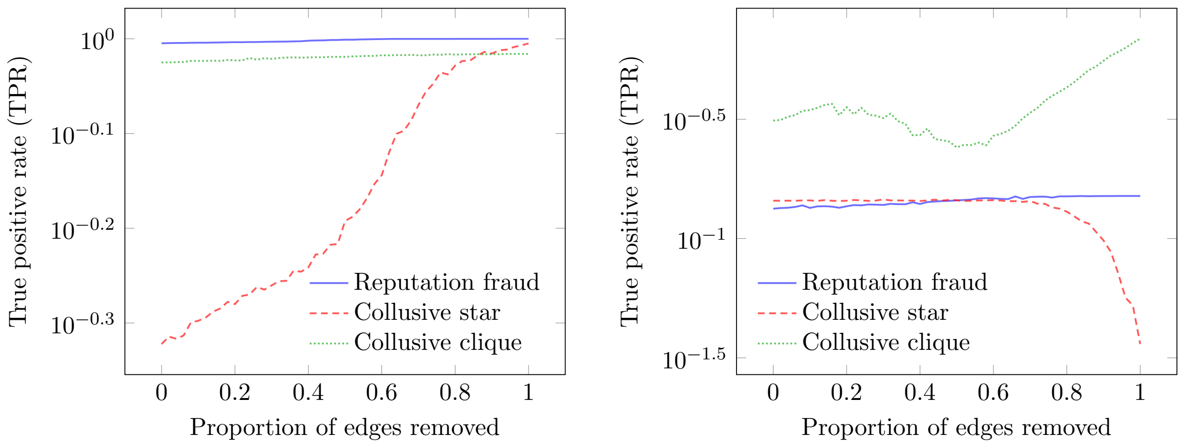
(a) No edges removed **(b)** Some edges removed **(c)** Most edges removed

Fig. 6.11. Graph structure for reputation fraud as edges are removed



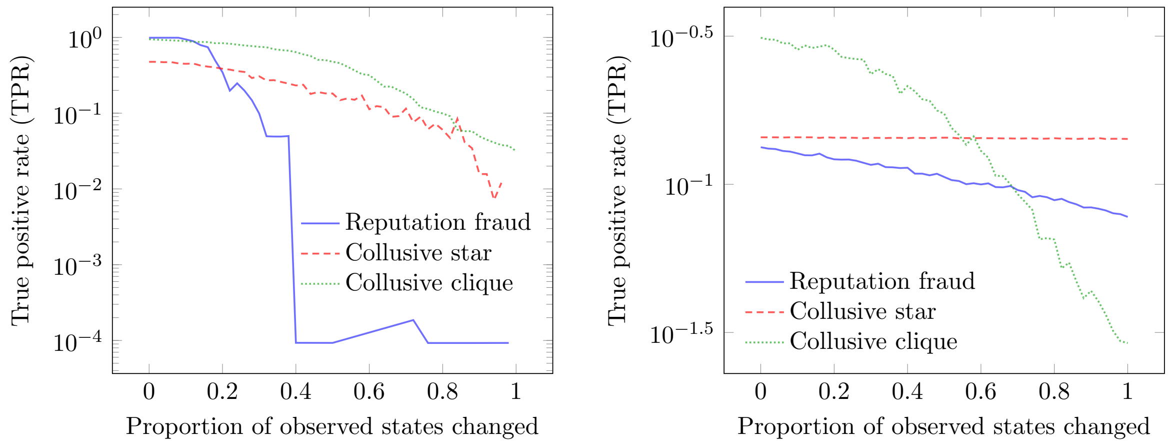
(a) SPAN **(b)** 2LFS

Fig. 6.12. TPR with different proportions of edges between fraud nodes removed



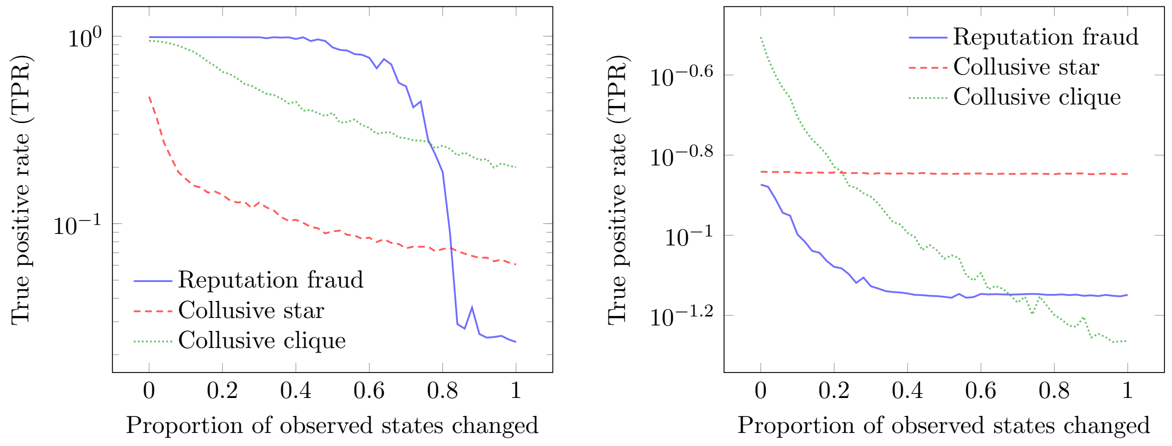
(a) SPAN **(b)** 2LFS

Fig. 6.13. TPR of 2LFS with different proportions of edges between fraud and normal nodes removed



(a) SPAN **(b)** 2LFS

Fig. 6.14. TPR with different proportions of fraud nodes' observed states changed



(a) SPAN **(b)** 2LFS

Fig. 6.15. TPR with different proportions of normal nodes' observed states changed

7 Case study: TradeMe

7.1 Introduction

SPAN is applied to the Aug 2012 dataset. Since the dataset is unlabelled, we will present results for the numbers of potentially fraudulent users identified by SPAN and 2LFS, the characteristics of those users, and present some examples of users identified as fraudulent. The results indicate that the behaviour of SPAN is reasonable when applied to real data, and when compared to 2LFS. The parameters for SPAN and 2LFS are the same as those described in Section 6.5.

7.2 Results

Section 7.2.1 describes the number of fraudulent users found by SPAN and 2LFS. Sections 7.2.2 and 7.2.3 compare the characteristics of the fraudulent and normal groups. Sections 7.2.4 gives some examples of some potentially fraudulent groups found.

To understand the types of users found by each algorithm, we also describe the characteristics of the group as a whole, and give several examples of the users identified as suspicious.

7.2.1 Number of users found as fraudulent

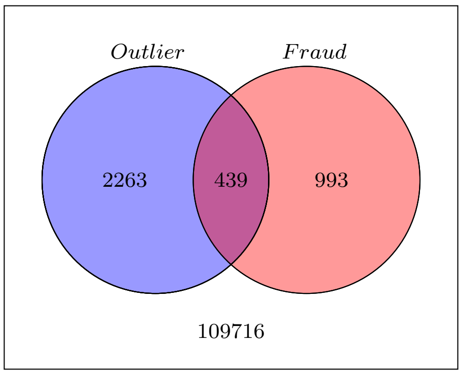
Each phase of SPAN and 2LFS produces values between 0 and 1 for each user. We define any user with a score above 0.5 as an outlier or as potentially fraudulent. Using this definition, we can divide the set of users into groups: as legitimate users and outliers in Phase 1, and as legitimate and fraudulent users in Phase 2 and 2LFS. A threshold is used instead of selecting a fraction of users with the highest scores, as done in the synthetic data evaluation in Section 6.7 in Chapter 6, due to the difficulty in selecting an appropriate threshold without information about the frequency of suspicious users in the data.

Phase 1 SPAN identifies 2702 users as outliers, and Phase 2 identifies 1432 users as potentially fraudulent, as seen in Figure 7.1. Of the 2702 outliers identified in Phase 1, 2263 were classified by Phase 2 as normal, and the remaining 439 as fraudulent. An additional 993 non-outliers were classified as potentially fraudulent in Phase 2, totalling 1432 users (1.26%) which are potentially fraudulent.

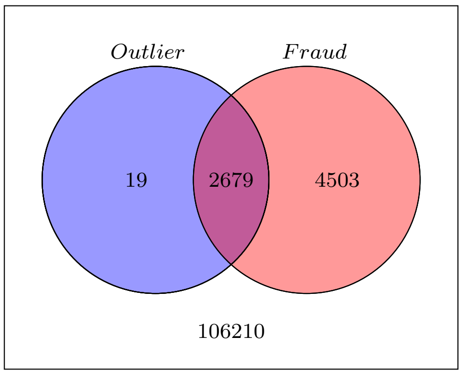
The anomalies identified in Phase 1 that were classified as normal in Phase 2 may be due to the removal of false positives as discussed in Section 6.4. The additional users identified as fraudulent in Phase 2 are due to the use of network information. That is, Phase 2 identifies groups of users who may, individually, have low anomaly scores, but are identified because of their interaction with other suspicious users. The vast majority of users (109716 or 96.7%) were identified as neither outliers nor fraudulent by SPAN.

This is shown in Figure 7.1(a). This compares to 2LFS, where 7182 users (6.33%) were identified as fraudulent, which includes all but 19 of the users identified as outliers, in addition to 4503 other users (Figure 7.1(b)).

The degree of overlap in the users classified as potentially fraudulent by SPAN and 2LFS is shown in Figure 7.2. 679 users were classified as fraudulent by both SPAN and 2LFS, which corresponds to 47.4% of users identified by SPAN, and 9.45% of users identified by 2LFS.



(a) SPAN



(b) Phase 1 + 2LFS

Fig. 7.1. The number of users labelled as outliers in Phase 1, and classified as fraudulent by SPAN and 2LFS

7.2.2 Descriptive statistics of user groups

The set of users can be divided into four groups: those identified by both SPAN and 2LFS as potentially fraudulent (Both); those identified as fraudulent by SPAN only (SPAN only); those identified as fraudulent by 2LFS only (2LFS only); and those identified

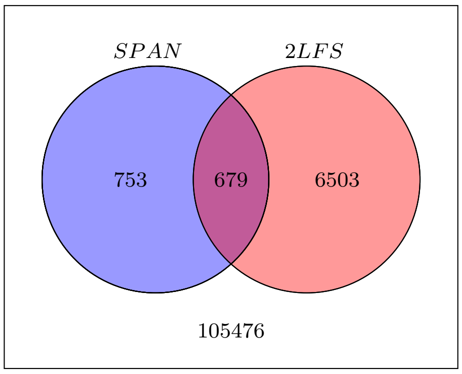


Fig. 7.2. The number of users classified as fraudulent by SPAN and 2LFS

by neither SPAN nor 2LFS as fraudulent (Normal). We report the average values for user-features and graph-features for each of these groups, in addition to the outlier/non-outlier groups found by Phase 1 SPAN to illustrate characteristics of the users identified as fraudulent by SPAN and 2LFS. Tables 7.2 to 7.9 reports these average values, as well as the percentage difference of each group compared to the Normal group.

As seen in Table 7.2, there are some differences in the average values for each user-feature between outlier/non-outlier groups. For SPAN, the fraudulent users identified have very similar average values compared to the normal users, where no percentage difference is greater than 10%. This result is expected since SPAN identifies fraudulent users based on the interaction between users, instead of individual behaviour. However, for 2LFS, there is a significant difference in the values of both ϕ_2 and ϕ_5. It should be noted that users that behave solely as sellers are not included in this table, as they do not have values for these user features.

The average graph feature values for users identified as outliers are significantly different to non-outliers, as shown in Tables 7.5 and 7.8, with 7 features having a percentage difference of 100 or greater. This is expected, since the outliers are identified based on graph feature pairs. However, for SPAN, the magnitude of these differences decreased after Phase 2, while for 2LFS, they increased, as shown in Tables 7.6 and 7.9. This is due to SPAN and 2LFS focussing on different types of users. We describe and examine the types of users identified by each algorithm in the next subsection.

Each phase of SPAN and 2LFS produces values between 0 and 1 for each user. We define any user with a score above 0.5 as an outlier or as potentially fraudulent. Using this definition, we can divide the set of users into groups: as normal users and outliers in Phase 1, and as normal and fraudulent users in Phase 2 and 2LFS.

From Phase 1 of SPAN, 2544 users had an anomaly score of above 0.5. The remaining 104263 users had a score equal or below 0.5. After Phase 2, 1348 users (1.26%) had a final score above 0.5, as shown in Figure 7.1(a). This compares to the 6771 users (6.34%) identified as fraudulent by 2LFS, as shown in Figure 7.1(b). The figures also show that some of the users identified as anomalous had their final scores lowered during Phase 2 to be labelled as normal, while most of the outliers were classified as fraudulent by 2LFS. Even though we lack data on the rates of user fraud in the dataset, a fraud rate of 1.26% seems reasonable, while the rate of 6.34% is higher than expected, potentially a result of false positives. Appendix F gives some examples of potential false positives found by 2LFS compared to SPAN.

Table 7.1
Degree of overlap in users identified by SPAN and 2LFS.

	SPAN	2LFS	Both
All	1892	6771	639
Egonet edge count > 1	1161	6059	516

Table 7.2
Average value of user features for bidders identified as outliers

User feature	Normal	Outlier	% diff
ϕ_1	7.535	7.358	-2.35
ϕ_2	1.629	2.360	44.83
ϕ_3	0.515	0.399	-22.58
ϕ_4	1.156	1.225	6.01
ϕ_5	0.467	0.517	10.75
ϕ_6	0.781	0.709	-9.17
ϕ_7	0.486	0.424	-12.86
ϕ_8	1859.454	1881.955	1.21
ϕ_9	7.606	7.418	-2.48
ϕ_{10}	0.866	0.811	-6.31

Table 7.3
Average value of user features for normal and potentially fraudulent bidders

User Feature	Classifications by SPAN			Classifications by 2LFS		
	Normal	Fraud	% change	Normal	Fraud	% change
ϕ_1	7.53	7.73	2.7	7.53	7.50	-0.5
ϕ_2	1.64	1.75	6.8	1.60	2.50	56.3
ϕ_3	0.51	0.52	1.7	0.51	0.50	-3.1
ϕ_4	1.16	1.15	-1.0	1.15	1.23	6.4
ϕ_5	0.47	0.46	-1.1	0.46	0.52	12.5
ϕ_6	0.78	0.80	2.7	0.78	0.74	-5.2
ϕ_7	0.48	0.51	4.5	0.49	0.46	-5.0
ϕ_8	1859.72	1870.81	0.6	1867.07	1718.39	-8.0
ϕ_9	7.60	7.78	2.4	7.61	7.52	-1.1
ϕ_{10}	0.86	0.88	2.0	0.87	0.85	-2.4

7.2.3 Types of users found

SPAN and 2LFS identify different types of users as fraudulent. SPAN identifies users with a range of auction participation frequencies while 2LFS tends to identify users that

Table 7.4
Average value of user features for normal bidders and bidders identified as potentially fraudulent by SPAN and 2LFS

User feature	Normal	SPAN only		2LFS only		Both	
		Fraud	% change	Fraud	% change	Fraud	% change
ϕ_1	7.53	7.78	3.4	7.50	-0.4	7.51	-0.2
ϕ_2	1.60	1.63	1.7	2.52	57.3	2.28	42.7
ϕ_3	0.51	0.52	0.6	0.49	-3.7	0.54	5.7
ϕ_4	1.15	1.14	-0.9	1.23	6.8	1.16	0.4
ϕ_5	0.46	0.46	-0.9	0.53	13.3	0.47	1.3
ϕ_6	0.78	0.81	3.5	0.74	-5.4	0.77	-1.9
ϕ_7	0.49	0.51	4.2	0.46	-5.6	0.51	4.4
ϕ_8	1866.96	1875.23	0.4	1708.96	-8.5	1852.14	-0.8
ϕ_9	7.60	7.84	3.1	7.52	-1.1	7.54	-0.9
ϕ_{10}	0.87	0.89	2.4	0.84	-2.5	0.86	-0.4

Table 7.5
Average value of graph features for bidders identified as outliers

Graph edge	Node attribute	Normal	Outlier	% diff
Loss	Edge count	1.10	2.12	92.93
Loss	Weight count	1.30	3.53	171.34
Participation	Weight count	2.95	6.39	116.70
Seen	Egonet edge count	6.91	13.80	99.77
Seen	First eigenvalue	2.02	6.54	223.49
Seen	Repeat count	0.81	2.78	244.46
Win	Edge count	1.37	2.05	49.55

Table 7.6
Average value of graph features for bidders identified as potentially fraudulent

Graph edge	Node attribute	Classifications by SPAN			Classifications by 2LFS		
		Normal	Fraud	% diff	Normal	Fraud	% diff
Loss	Weight count	1.34	1.94	45.4	1.17	4.90	320.0
Loss	Edge count	1.12	1.19	6.6	1.03	2.97	189.1
Participate	Weight count	3.00	4.40	46.9	2.66	10.13	281.0
Seen	Egonet edge count	7.04	7.32	4.0	5.36	40.22	649.8
Seen	Repeat count	0.82	2.58	214.6	0.48	8.08	1578.4
Seen	First eigenvalue	2.03	3.30	62.2	1.78	7.55	325.3
Win	Edge count	1.39	1.48	6.9	1.28	3.55	177.5

Table 7.7
Average value of graph features for normal bidders and bidders identified as potentially fraudulent by SPAN and 2LFS

Graph edge	Node attribute	Normal	SPAN only		2LFS only		Both	
			Fraud	% change	Fraud	% change	Fraud	% change
Loss	Weight count	1.16	1.25	6.6	4.90	292.9	4.88	291.4
Loss	Edge count	1.03	0.97	-6.0	3.03	212.3	2.14	120.6
Participate	Weight count	2.65	2.91	8.8	10.09	246.7	10.69	267.2
Seen	Egonet edge count	5.38	4.49	-19.7	41.70	828.3	19.26	328.9
Seen	Repeat count	0.48	0.89	46.7	7.97	792.9	9.68	984.6
Seen	First eigenvalue	1.77	2.05	13.8	7.48	264.2	8.55	316.2
Win	Edge count	1.28	1.18	-8.5	3.60	205.4	2.76	133.6

Table 7.8
Average value of graph features for sellers identified as outliers

Graph edge	Node attribute	Normal	Outlier	% diff
Connected seller	Edge count	42.75	37.97	-11.18
Connected seller	Weight count	47.07	46.65	-0.89
Loss^T	Repeat count	0.72	1.66	131.49
Participate^T	Edge count	7.73	8.58	10.94
Participate^T	Weight count	9.71	11.63	19.67
Seen	Egonet weight	24.22	80.88	234.00
Seen	First eigenvalue	3.30	7.11	115.34
Seen	Repeat count	2.46	3.84	56.17

Table 7.9
Average value of graph features for sellers identified as potentially fraudulent

Graph edge	Node attribute	Classifications by SPAN			Classifications by 2LFS		
		Normal	Fraud	% diff	Normal	Fraud	% diff
Connected Seller	Weight count	47.56	27.94	-41.3	22.88	158.05	590.8
Connected Seller	Edge count	43.03	22.16	-48.5	22.57	133.98	493.6
Seen	Repeat count	2.46	5.09	106.6	0.76	10.69	1315.5
Seen	First eigenvalue	3.35	4.40	31.5	2.43	7.73	218.0
Participate^T	Edge count	7.87	4.58	-41.7	3.69	26.58	620.9
Participate^T	Weight count	9.85	8.63	-12.4	4.12	36.00	774.2
Loss^T	Repeat count	0.75	1.55	107.5	0.07	3.98	5910.8
Seen	Egonet weight count	25.35	25.87	2.1	11.81	87.60	641.9

participate in a high number of auctions. This is seen in Figure 7.5, which shows the cumulative distribution of edge counts of the identified fraudulent users. The figure shows that, for example, 28.9% of users identified by 2LFS as potentially fraudulent have 3 edges or fewer, while 77.8% of users identified by SPAN have 3 edges or fewer, as shown in Figure 7.5(a). In addition, most of the potentially fraudulent users identified by both SPAN and 2LFS tend to have a high number of transactions as well, as shown in Figure 7.5(b), and the users identified by SPAN only have a similar edge count distribution to normal users.

We also observe that users identified as fraudulent by SPAN tend to be more closely

Table 7.10
Average value of graph features for sellers identified as potentially fraudulent by SPAN and 2LFS

Graph edge	Node attribute	Normal	SPAN only		2LFS only		Both	
			Average	% change	Average	% change	Average	% change
Connected Seller	Weight count	22.97	12.47	-84.2	172.54	1283.5	33.96	172.3
Connected Seller	Edge count	22.66	12.32	-83.9	146.60	1089.8	25.99	111.0
Seen	Repeat count	0.75	1.49	49.7	11.18	651.7	6.49	336.4
Seen	First eigenvalue	2.43	2.68	9.3	8.04	200.3	5.08	89.6
Participate^T	Edge count	3.70	2.03	-82.3	29.03	1329.5	5.58	174.7
Participate^T	Weight count	4.13	2.65	-56.0	38.92	1370.1	10.95	313.7
Loss^T	Repeat count	0.07	0.07	1.6	4.20	6136.1	2.12	3051.8
Seen	Egonet weight count	11.82	10.16	-16.4	94.10	826.6	31.98	214.9

connected to each other compared to those found by 2LFS. This can be seen in Figure 7.7 where potential frauds found by SPAN have a higher proportion of interactions with other potential frauds compared to 2LFS. For example, for 56.8% of frauds identified by SPAN, 75% of interactions are with other potentially fraudulent users, compared to 30.9% for 2LFS.

The behaviour of SPAN may be more desirable than 2LFS, since SPAN is more likely to identify colluding groups of users that have a wide range of auction participation frequencies, instead of focussing on high-activity users as 2LFS does.

7.2.4 Potentially fraudulent groups found

Out of the 993 potentially fraudulent users identified, 410 had a two-node egonet. The majority of the remaining users with 2 or more neighbours had star egonets, with 511 stars and 14 with 1 or more edges between neighbours. Figure 7.3 gives an example of a star egonet, while Figure 7.4 gives an example of a non-star egonet.

Figure 7.10 shows the egonets of 3 suspected frauds. Figure 7.10(a) was identified as fraudulent by SPAN but not 2LFS, (b) was identified by 2LFS but not SPAN, while (c) was identified by both SPAN and 2LFS as fraudulent. Due to the nature of the dataset, the egonets of most users are stars, where the neighbours of a user are unlikely to have interacted with each other. Thus, the users identified by both SPAN and 2LFS tend to be star-like. However, on average, the fraudulent users identified by SPAN and 2LFS tend to be less star-like than normal users. This is shown in Table 7.11 and in Figure 7.10. This indicates that SPAN is able to identify anomalous groups of users that form tighter networks than the vast majority of users.

7.2.5 Convergence

Node beliefs stabilise within around 20 iterations in Phase 2. The number of iterations for convergence is in the same order as that observed for synthetic data in Section 6.7.2.

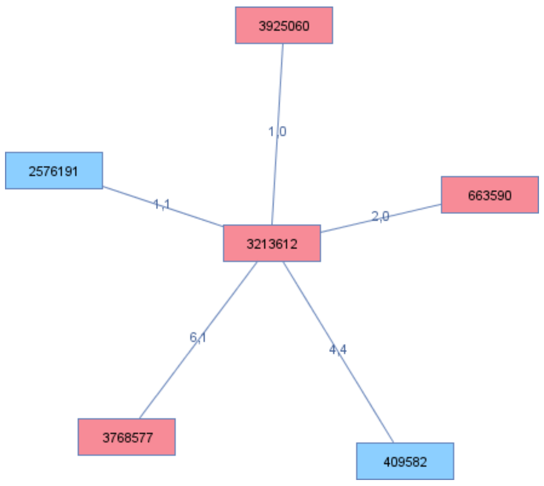


Fig. 7.3. Example of a star egonet of suspected fraud 3213612

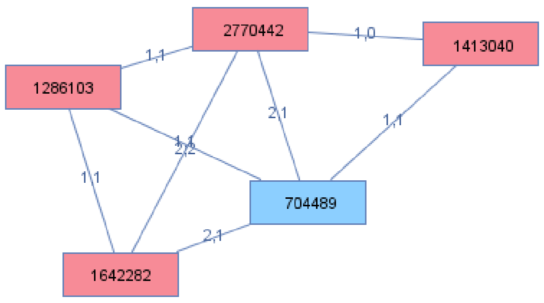
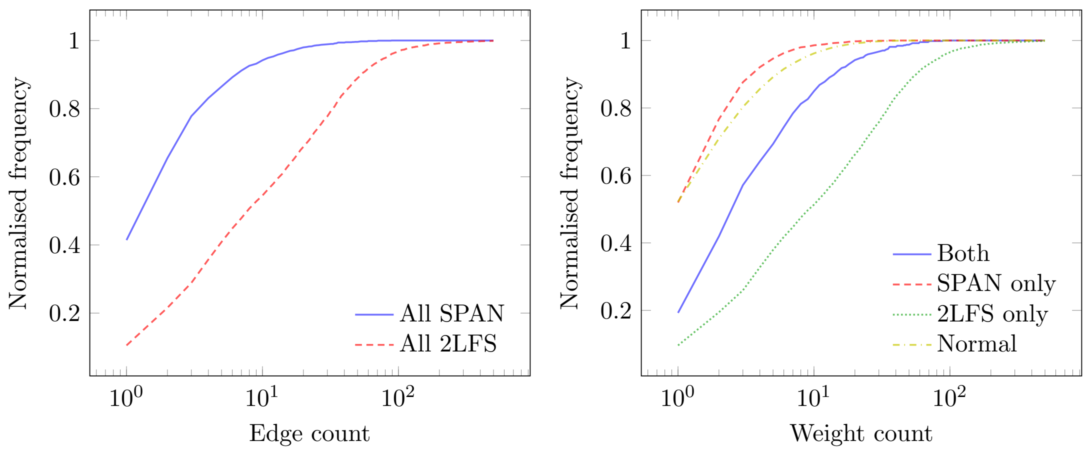


Fig. 7.4. Example of a non-star egonet of suspected fraud 2770442

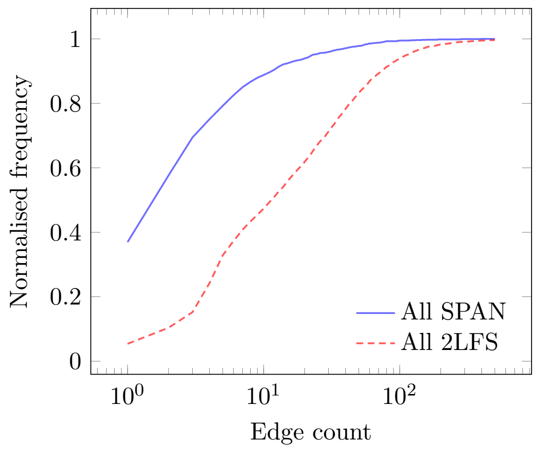


(a) Cumulative distribution of edge counts **(b)** Cumulative distribution of edge counts

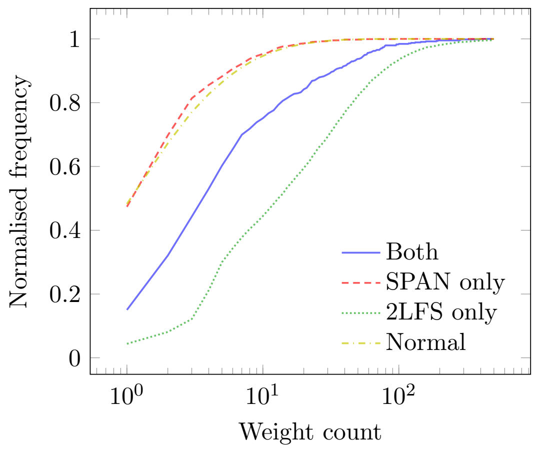
Fig. 7.5. Cumulative distributions of edge counts for suspected users

Table 7.11
Average number of connections between neighbours, normalised by node edge degree

	Normal	Span	Span only	Both	2LFS only	2LFS
Average	0.0034	0.0255	0.0085	0.0611	0.0487	0.0499



(a) Cumulative distribution of weight counts



(b) Cumulative distribution of weight counts

Fig. 7.6. Cumulative distributions of weight counts for suspected users

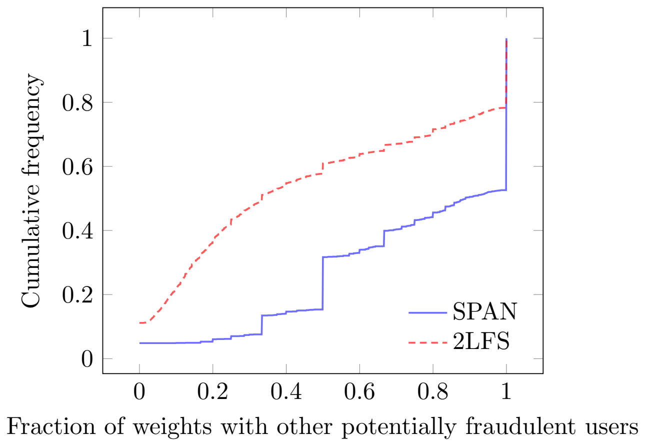


Fig. 7.7. Fraction of weights that are fraud-fraud

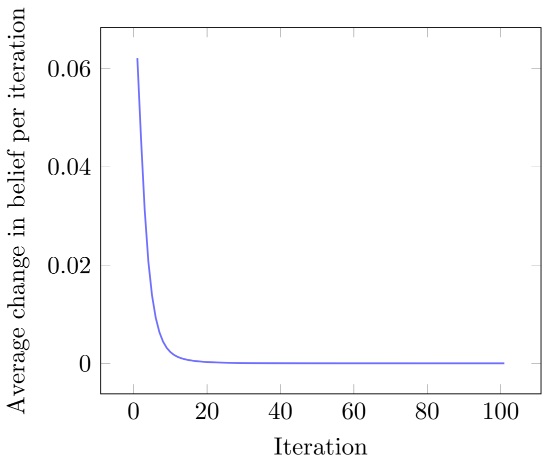


Fig. 7.8. Average change in belief between each iteration

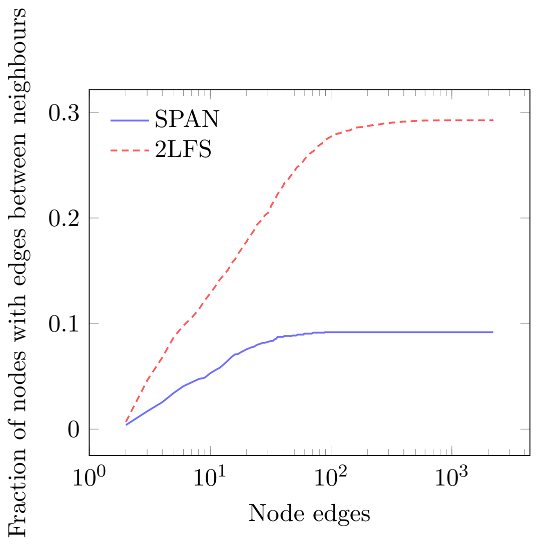
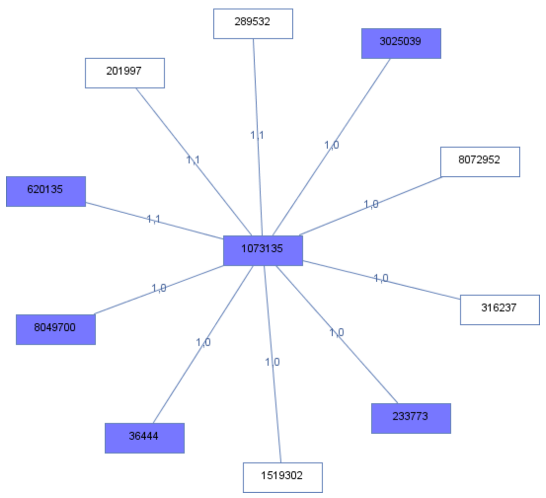


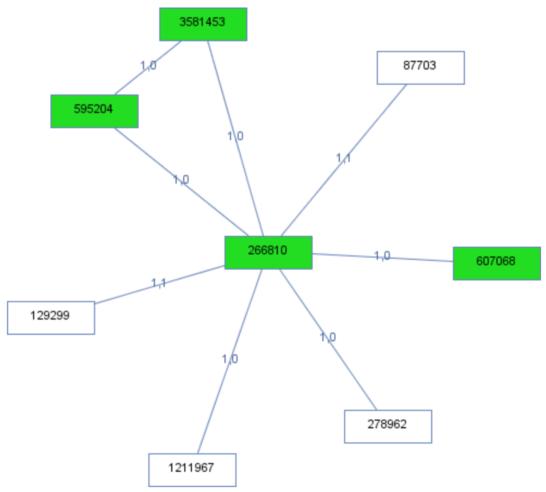
Fig. 7.9. Cumulative fraction of nodes with edges between neighbours as edge count increases

7.3 Conclusion

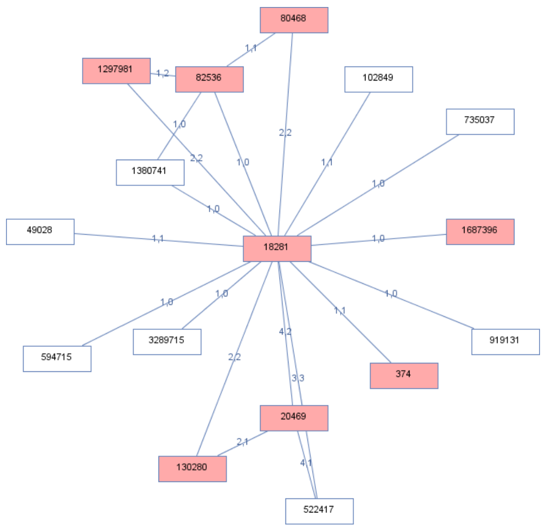
We applied SPAN to a real online auction dataset. The results show that SPAN behaves reasonably with real data. SPAN identified 1.26% of users as potentially fraudulent, which is within expectations, and the users identified have characteristics that are consistent with fraud. SPAN also compares favourably against 2LFS.



(a) Egonet of 1073135



(b) Egonet of 266810



(c) Egonet of 18281

Fig. 7.10. Example egonets of suspected frauds identified by (a) SPAN only, (b) 2LFS only and (c) both SPAN and 2LFS. Coloured nodes denote potentially fraudulent users.

8 Conclusion and future work

Past research in reducing fraud in online auctions has typically focussed on three fraud types: shilling, reputation fraud and non-delivery fraud. Developing effective methods to predict or detect instances of fraud has been difficult for several reasons, including: lack of fraud cases, lack of appropriate datasets, evolving fraudulent and normal behaviour, and heterogeneity in normal behaviour. Our research focussed on addressing some of these challenges. This chapter presents a summary of our research, its contribution, limitations and possible future work.

8.1 Summary

To solve the problem of a lack of datasets, we designed and implemented an agent based model (ABM) in Chapter 3. The simulation is based on real auction data, and is capable of generating realistic synthetic data. An ABM is used because it naturally describes a system of entities, including online auctions, it is flexibile, and it can capture emergent behaviour that is otherwise difficult to model. The implementation of the model, and validation of the synthetic data is given in Chapter 3. We also propose a set of features to describe general bidding behaviour of users. Transforming the set of auctions and bids into these features allows user behaviour to be understood more easily, and simplifies synthetic data validation.

In Chapter 4, the simulation is extended with additional agents with fraudulent behaviour. The simulation is able to generate synthetic data containing different types of fraud, and is used to evaluate three fraud detection algorithms.

We then propose, in Chapter 5 the use of the simulation together with supervised machine learning techniques to create models for detecting arbitrary types of fraud. We demonstrate this with two types of fraud agents, and use the resulting synthetic data to train multiple decision trees. Results show that this method outperforms another manually crafted fraud detection algorithm.

In Chapter 6, we proposed an unsupervised anomaly detection method, SPAN, which avoids some additional challenges of the previous approach, specifically the need for re-training of models when the behaviour of normal or fraudulent users changes. SPAN also improved on the previous approach by making use of network-level features to identify groups of fraudulent users and to increase accuracy. SPAN is applied to a real dataset in Chapter 7.

8.2 Contributions

The contributions of this thesis are:

1. We surveyed literature related to online auction fraud, and discuss and compare them in terms of the three main stages in developing fraud detection techniques: definition of the fraud of interest, feature set selection, and model construction. The challenges encountered by previous methods is also discussed.

2. We designed and implemented an agent-based simulation of online auctions for generating synthetic auction data. Agents in the simulation are modelled on real online auction data by transforming the original data into a set of 10 features to describe user behaviour, and applying clustering to identify patterns in user behaviour. The synthetic data is objectively compared against real auction data using statistical and clustering methods, and the results show that the synthetic data is similar to real auction data.

3. The simulation is demonstrated as a tool for evaluating fraud detection algorithms. Using the generated synthetic data and three shilling fraud agents, the accuracy of three different fraud detection algorithms was evaluated and compared.

4. We presented a framework for creating classifiers using supervised machine learning methods and synthetic data. The method allows classifiers for arbitrary fraud types to be constructed, provided that they are defined as agents in the simulation. Evaluations performed using synthetic data containing three different types of fraud show that the models trained this way are more accurate than previously proposed methods, and evaluations using real data show they identify users who appear suspicious. We also proposed a set of 10 features that characterise general bidding behaviour and which produce classifiers with higher accuracy.

5. We proposed an unsupervised fraud detection algorithm based on anomaly detection in graphs. The method identifies groups of closely collaborating users who exhibit unusual characteristics. Network information was used to improve accuracy by calculating an initial anomaly score value using LOF, then revising that score using belief propagation so that neighbours in the network tend to have similar scores.

The method was evaluated using synthetic data under a range of conditions and compares favourably to other related methods. A case-study using real data shows the method identified a reasonable number of users as suspicious, who appear to be different from the majority of users.

8.3 Limitations

The first limitation is the dataset collected from TradeMe is limited: it is incomplete and contains only information that is publicly accessible. If additional information about users is available, such as log-in frequency, log-in times, web page access information, or associated IP-addresses, a greater set of features can be built, and different approaches can be used to detect fraud. For example, identifying multiple accounts that are likely controlled by the same person by comparing their page visit orders and duration to detect suspicious behaviour.

The second limitation is that for each of the approaches we proposed for fraud detection, supervised or unsupervised, there is a trade-off between false positives and true positives: additional true positives come at the cost of additional false positives. The optimal trade-off depends on the relative cost and frequency of false positives and true positives. In addition, in a real setting, cases may be manually examined before users are declared fraudulent or normal. Since these costs are unknown, our work does not take these factors into account, and simply reports performance over the entire range using the ROC curves.

8.4 Future work

Below, we list possible future work based on three chapters:

Chapter 3: Auction simulation

- As noted previously, currently the simulation does not model user behaviour well near the end of auctions, since the length of the simulation time unit is constrained by computational time. One possible solution would be to reduce the length of time units for users participating in auctions that are close to ending, e.g. the last 2% of a 7-day auction, or 3.36 hours. This would increase computational cost slightly due to additional overhead and agent execution frequency, but may increase model accuracy significantly.

Chapter 5: Supervised fraud detection

- Our proposed feature set for describing general bidder behaviour has been shown to produce models of higher accuracy. However, this claim should be tested using additional agent types.

- Other methods of selecting relevant features are available. In our work, the features used are selected based on previous work and domain knowledge. Instead, another strategy may be to select a large number of features, then apply feature selection or dimensionality reduction techniques to discard irrelevant features. This may reduce the need for domain knowledge for selecting a small set of relevant features.

- Two supervised learning methods were used. Other methods such as Naive Bayes, k-nearest neighbour algorithms, or support vector machines may achieve higher accuracy.

Chapter 6: Unsupervised fraud detection

- One of the difficulties when applying SPAN is parameter selection. While the parameters may be tuned by creating an additional partition, the parameter space is large since Phase 1 and 2 of SPAN is coupled, so that search must be over parameters of both phases at once. This requires large amounts of additional time and data. In addition, this tuning can only be done with labelled data, which is unavailable in a real application setting.

- Related to this is the definition of possible states for users. In SPAN, nodes had two states: fraudulent and normal. While the fraudulent state includes accomplices who work with fraudulent users without directly committing fraud, it may be valuable to extend SPAN to identify accomplices explicitly by redefining the ϕ and ψ matrices. The difficulty is selecting appropriate matrix values.

- There are many methods of calculating anomaly scores in Phase 1. The method currently used, LOF, performs adequately, but has some known weaknesses. For example, if density changes gradually across the input space, LOF will assign anomalies low anomaly scores. It may be worthwhile to investigate alternative measures, or to augment LOF, which is a relative density measure, with an absolute density measure such as simple k-means.

- Currently SPAN is an offline algorithm. However it is possible to extend SPAN to be an online algorithm to make it more useful in a real setting. To do so, Phase 1 can be extended to incrementally update scores in the online setting according to user actions (e.g., auction participations, auction wins, and so on). For Phase 2, additional users and interactions correspond to new nodes and edges in the MRF model. The beliefs of the existing nodes are simply retained, and the belief propagation algorithm simply iterates using the new observed node values and propagation values until convergence.

Appendices

A

Pseudocode

A.1 Simulation execution

Algorithm 2 shows the pseudocode for the agent-based simulation. Agents representing bidders and sellers are instantiated in lines 2-7. The agents and the auction house communicate via buffers P and Q, instantiated in lines 9-10. Execution then alternates between the auction house and the agents in lines 12-20. All auctions and bids submitted by agents are stored in S, instantiated in line 11.

Algorithm 2 Pseudocode for simulation execution

Input: array of agent types: T, array of agent counts: N, simulation length: L
Output: Synthetic data: S

```
1: $U$ = NEWLIST                                  ▷ List containing agent instances
2: for $i \in [1..T.length]$ do        ▷ For each agent type, instantiate the number wanted
3:     for $n \in [1..N[i]]$ do
4:         $u_{i,n}$ = instantiateAgentType($T[i]$)
5:         $U = U$.add($u_{i,n}$)
6:     end for
7: end for
8: $H$ = instantiateAuctionHouse()
9: $P$ = NEWLIST                        ▷ List of messages generated by the agents
10: $Q$ = NEWLIST                    ▷ List of messages generated by the auctioneer
11: $S$ = NEWLIST                     ▷ All messages from agents; synthetic data
12: for $i \in [1..L]$ do
13:     $Q = H$.run(i, $P$)
14:     $P$ = NEWLIST
15:     for all $u \in \mathcal{U}$ do
16:         $P_u = a$.run(i, $Q(u)$)
17:         $P = P$.add($P_u$)
18:     end for
19:     $S = S$.add($P$)
20: end for
```

A.2 Normal agent

Pseudocode in Algorithms 3 and 4 show the basic behaviours for the normal agents implemented in the simulation. The variables defined in Algorithm 3 for each agent persist over the duration of the simulation. There are two types of normal agents in the simulation, based on what is observed in the collected data: *early bidders*, and *snipers*, who have different bidding strategies. Early bidders tend to begin bidding in auctions early, while snipers wait until just before auction termination before beginning to bid. The difference in these bidding behaviours are encapsulated in the function *participationTime()*, which generates the time an agent should begin bidding for an auction, or revisit it for possible rebidding. Lines 1-13 are for handling different message types sent from the auctioneer agent. Each time the agent decides to participate in an auction, it also decides on the next time it will participate in another new auction (lines 3-6). Participation does not immediately take place; instead the agent decides on a time unit in the future at which to consider submitting a bid. Lines 14-22 show that at each time unit, the agent will also monitor the auctions it is participating in, and submit a bid if conditions are met. It is worth noting that the methods *privateValuation(), nextAuctionTime(), participationTime(), shouldBid(), and amountToBid()* use pseudorandom numbers to add variability to agent behaviours.

Algorithm 3 Pseudocode for agent instantiation

```
1: $v$ = privateValuation()
2: $L$ = NEWLIST                                  ▷ List of auctions to check on
3: $n$ = nextAuctionTime()
```

Algorithm 4 Pseudocode for the execution of a normal bidder agent over 1 time unit

Input: current time unit: *time*, messages for agent u: $\mathcal{M}_u$
Output: messages for auctioneer: $\mathcal{N}_u$

```
    ▷ Process messages sent by the auctioneer
 1: for all $m \in \mathcal{M}_u$ do          ▷ Each message has an associated type and auction
 2:     if $m.type$ == NEW then               ▷ Message type is about a new auction
 3:         if $time > n$ then
 4:             $f$ = participationTime()
 5:             $L = L$.add($f, m.auction$)   ▷ Assign a future time $f$ to bid in auction
 6:             $n$ = nextAuctionTime()       ▷ Assign a time for the next auction participation
 7:         end if
 8:     else if type == PRICE_CHANGE then     ▷ Message type for a new auction bid
 9:         continue
10:     else if type == EXPIRING then         ▷ Message about an auction close to expiring
11:         continue
12:     end if
13: end for
    ▷ Actions performed during every time unit
14: for all $\{l \mid l \in L \wedge l.f \leq time\}$ do     ▷ Set of auctions to be checked now
15:     $L = L$.remove($l$)                   ▷ Remove auction once it has been checked
16:     if shouldBid($t, l.auction, v$) then
17:         $amount$ = amountToBid($l.auction, v$)   ▷ Based on price and private valuation
18:         $bid$ = NEWBID($l.auction, amount$)      ▷ Instantiate a new bid
19:         $\mathcal{N}_u = \mathcal{N}_u$.add($bid$)   ▷ Add the bid to the set of messages for the auctioneer
20:         $f'$ = participationTime()
21:         $L = L$.add($f', l.auction$)     ▷ Assign a future time for rebidding in auction
22:     end if
23: end for
```

A.3 Shill agent

The pseudocode shown in Algorithms 5 and 6 are for the simple shill agent. The variables defined in Algorithm 5 for each agent persist over the duration of the simulation. Algorithm 6 shows that the actions of the shill is divided into two parts, similar to normal agents, as shown in Figure 3.2. Lines 1-7 are for processing messages from the auctioneer agent, where new fraudulent auctions submitted by the selling agent is recorded. Then, in lines 8-13, the agent will iterate through each current shill auction, and submit a bid to the auctioneer if all directives are satisfied.

The pseudocode for the late-start shill agent is similar. An additional condition is added at line 9, testing whether sufficient time has elapsed since the start of the auction. For the legitimate-bidding agent, an additional test is added between lines 5 and 6 to identify messages of type EXPIRING, and bid in those auctions which are legitimate and have a low price.

Algorithm 5 Pseudocode for shill bidding agent instantiation

```
Input: parameters: $\alpha$, $\beta$, $\gamma$
1: $v$ = privateValuation()
2: $L$ = NEWLIST                                ▷ List of auctions to check on
3: $n$ = nextAuctionTime()
4: $S$ = NEWLIST                                ▷ List of fraudulent auctions
```

A.4 Collaborative reputation fraud agents

The reputation fraud agent controls a set of bidding and selling agents. Both agent types have the behaviours of normal agents, similar to that shown in Algorithm 4, the only difference being that the agent can submit bids and auctions in their name for the purpose of reputation inflation. Algorithm 7 instantiates the lists of selling (R_S) and bidding (R_B) agents, and stores the average target for reputation inflation for each agent ($T[i]$). Algorithm 8 shows the execution of the agent, where it submits auctions in the name of the selling agents it controls (lines 9-14). The function $\exp(1/\mu)$ returns a value drawn from the exponential distribution with mean μ. The agent also processes messages

Algorithm 6 Pseudocode for the execution of a simple shill bidding agent over 1 time unit

```
Input: current time unit: $time$, messages for agent $u$: $\mathcal{M}_u$
Output: messages for auctioneer: $\mathcal{N}_u$
    ▷ Process messages received
 1: for all $\{m \mid m \in \mathcal{M}_u \wedge m.type == \textsc{New} \wedge m.auction \in S\}$ do
 2:     $S = S.\text{add}(m.auction)$
 3: end for
    ▷ Actions performed during every time unit
 4: for all $auction \in S$ do                              ▷ Check on all shill auctions
 5:     if isTerminated($auction$) then
 6:         $S = S.\text{remove}(auction)$
 7:         continue
 8:     end if
 9:     if (directive3($\theta, auction, time$) & directive4($\alpha, auction$) &
10:             directive5($\mu, auction, time$)) then
11:         $amount$ = nextMinimumBid($auction$) ▷ Minimum allowed bid per directive 1
12:         $bid$ = newBid($auction$, $amount$)
13:         $\mathcal{N}_u = \mathcal{N}_u.\text{add}(bid)$
14:     end if
15: end for
```

from the auctioneer notifying it when the same auctions are about to expire, and submits bids using the bidders it controls (lines 1-8).

Algorithm 7 Pseudocode for collaborate reputation fraud agent instantiation

```
 1: $R_S$ = newArray[50]                       ▷ Agents wanting reputation inflation
 2: $T$ = newArray[50]                                  ▷ Stores the inflation target
 3: $N$ = newArray[50]          ▷ Stores the next time to submit a fraudulent auction
 4: $S$ = newList                                      ▷ List of fraudulent auctions
 5: for $i \in [1..50]$ do
 6:     $R_S[i]$ = instantiateAgentType($T_{RepSeller}$)
 7:     $T[i] = U(30, 100)$                  ▷ Inflation target drawn from a uniform dist.
 8:     $N[i] = \exp(1/T[i])$              ▷ Draw from exponential dist. with mean $T[i]$
 9: end for
10: $R_B$ = newArray[400]
11: for $i \in [1..400]$ do
12:     $R_B[i]$ = instantiateAgentType($T_{RepBidder}$)
13: end for
```

Algorithm 8 Pseudocode for the execution of a collaborate reputation fraud agent over 1 time unit

Input: current time unit: *time*, messages for agent u: $\mathcal{M}_u$
Output: messages for auctioneer: $\mathcal{N}_u$

```
1: for all $\{m \mid m \in \mathcal{M}_u \wedge m.type == \textsc{Expiring} \wedge m.auction \in S\}$ do
2:     $rdmBidder = \text{random}(R_B, 1)$                ▷ Randomly select a bidder
       ▷ Submit a bid in the name of $rdmBidder$
3:     $\mathcal{N}_u = \mathcal{N}_u.\text{add}(\textsc{newBid}(rdmBidder, m.auction, \text{nextMinimumBid}(m.auction)))$
4:     $S = S.\text{remove}(m.auction)$
5: end for
6: for $i \in [1..50]$ do
7:     if $N[i] \leq time$ then
           ▷ Submit an auction in the name of $R_S[i]$
8:         $fraudAuction = \textsc{newAuction}(R_S[i])$
9:         $\mathcal{N}_u = \mathcal{N}_u.\text{add}(fraudAuction)$
10:        $S = S.\text{add}(fraudAuction)$
           ▷ Generate the next time for this agent to submit an auction
11:        N[i] = $\exp(1/T[i])$    ▷ Draw the next auction time from the exponential dist.
12:    end if
13: end for
```

Algorithm 9 Pseudocode for collusive star shilling fraud agent instantiation

```
1: $R_S = \text{instantiateAgentType}(T_{CollusiveShillSeller})$
2: $n = \exp(1/40)$                  ▷ Draw from exponential dist. with mean 40
3: $Q = \textsc{newList}$            ▷ Stores the set of shills allocated to each auction
4: $R_B = \textsc{newArray}[6]$
5: for $i \in 1..6$ do
6:     $R_B[i] = \text{instantiateAgentType}(T_{CollusiveShillBidder})$
7: end for
```

A.5 Pseudocode for collusive star shilling fraud agent

The collusive star composed of one seller and six bidders is controlled by one agent. The variables needed for the agent are shown in Algorithm 9, where Q is a list storing tuples of auctions and a set of bidders that has been allocated to shill in that auction. Once again, R_S and R_B contain agents that also exhibit normal behaviour. In lines 9-12, the agent submits auctions in S_B's name. Once the auction is acknowledged by the auctioneer agent and the message notifying the start of the auction is received, the three random bidders are allocated to the auction (lines 4-5). During each time unit, the agent checks each auction in Q and submits a bid if the directives are satisfied.

Algorithm 10 Pseudocode for the execution of a collusive star shilling fraud shill agent over 1 time unit

Input: current time unit: *time*, messages for agent u: $\mathcal{M}_u$

Output: messages for auctioneer: $\mathcal{N}_u$

```
1: for all $\{m \mid m \in \mathcal{M}_u \wedge m.type ==$ New $\wedge$ isShillAuction$(m.auction)\}$ do
2:     $randomBidders$ = random$(R_B, 3)$
       ▷ Store the auction and the 3 bidders allocated to it
3:     $Q = Q$.add$(m.auction, randomBidders)$
4: end for
5: if $n \leq time$ then
6:     $fraudAuction$ = newAuction$(R_S)$
7:     $\mathcal{N}_u = \mathcal{N}_u$.add$(fraudAuction)$
8:     nextAuction = exp$(1/T[i])$
9: end if
10: for all $q \in Q$ do
11:     if isTerminated$(q.auction)$ then
12:         $Q = Q$.remove$(q)$
13:         continue
14:     end if
15:     if (directive3$(\theta, q.auction, time)$ & directive4$(\alpha, q.auction)$ &
16:             directive5$(\mu, q.auction, time)$) then
17:         $amount$ = nextMinimumBid$(q.auction)$
18:         $bid$ = newBid(getNext$(bidders, 1)$, $q.auction$, $amount$)
19:         $\mathcal{N}_u = \mathcal{N}_u$.add$(bid)$
20:     end if
21: end for
```

A.6 Pseudocode for collusive clique shilling fraud agent

The collusive clique is similar to the collusive star. The main differences are that all agents in the clique submit auctions in which the others in the clique act as shills. On average, each agent submits six auctions in the clique. Once again, the agents act as normal bidders in addition to being coordinated by the controller agent. Pseudocode is shown in Algorithms 11 and 12.

Algorithm 11 Pseudocode for collusive clique shilling fraud agent instantiation

Input: collusive group size: g

```
 1: $N = \exp(1/40)$                              ▷ Draw from exponential dist. with mean 40
 2: $Q = $ NEWLIST                               ▷ Stores the set of shills allocated to each auction
 3: $R = $ NEWARRAY[g]
 4: for $i \in [1..50]$ do
 5:     $R_S[i] = $ instantiateAgentType($T_{CollusiveShillBidder}$)
 6:     $N[i] = \exp(1/6)$                         ▷ Draw from exponential dist. with mean 6
 7: end for
```

Algorithm 12 Pseudocode for the execution of a collusive clique shilling fraud shill agent over 1 time unit

Input: current time unit: *time*, messages for agent u: $\mathcal{M}_u$

Output: messages for auctioneer: $\mathcal{N}_u$

```
 1: for all $\{m \mid m \in \mathcal{M}_u \wedge m.type ==$NEW $\wedge$ isShillAuction($m.auction$)$\}$ do
 2:     $randomBidders = $ random($R_B, 3$)
 3:     $Q = Q$.add($m.auction, randomBidders$)
 4: end for
 5: for i $\in$ 1..6 do
 6:     if $N[i] \leq time$ then
 7:         $fraudAuction = $ NEWAUCTION($R_S[i]$)
 8:         $\mathcal{N}_u = \mathcal{N}_u$.add($fraudAuction$)
 9:         nextAuction = $\exp(1/N[i])$
10:     end if
11: end for
12: for all $q \in Q$ do
13:     if isTerminated($q.auction$) then
14:         $Q = Q$.remove($q$)
15:         continue
16:     end if
17:     if (directive3($\theta, q.auction, time$) & directive4($\alpha, q.auction$) &
18:             directive5($\mu, q.auction, time$)) then
19:         $amount = $ nextMinimumBid($q.auction$)
20:         $bid = $ NEWBID(getNext($q.bidders$, 1), $q.auction$, $amount$)
21:         $\mathcal{N}_u = \mathcal{N}_u$.add($bid$)
22:     end if
23: end for
```

B

Alternative features

Listed below are the other potential features that were investigated. The additional 13 features are defined for the user $q \in \mathcal{U}$, where $|\mathcal{P}^q| > 0$.

Definition 11 ϕ_{1SD} is related to ϕ_1. It gives the standard deviation of the set of bid values that user q made in all auctions.

$$\mathcal{N}_1 = \left\{ \frac{1}{|\mathcal{P}^q||\mathcal{B}^{a,q}|} \sum_{b \in \mathcal{B}^{a,q}} b, \text{ where } a \in \mathcal{P}^q \right\} \tag{B.1}$$

$$\phi_{1SD} = \sigma(\mathcal{N}_1) \tag{B.2}$$

Definition 12 ϕ_{2SD} is related to ϕ_2. It gives the standard deviation of the set of bid increments made by user q in all auctions. The definitions of $minVal$ and $prevVal$ are the same as in Definition 2.

$$\mathcal{N}_2 = \left\{ \frac{1}{|\mathcal{P}^q||\Delta V^{a,q}|} \sum_{\Delta v_c^{a,q} \in \Delta V^{a,q}} \Delta v_c^{a,q} - minVal(prevVal(\Delta v_c^{a,q}))\text{ , where } a \in \mathcal{P}^q \right\} \tag{B.3}$$

$$\phi_{2SD} = \sigma(\mathcal{N}_2) \tag{B.4}$$

Definition 13 ϕ_{4SD} is related to ϕ_4. It gives the standard deviation of the bid counts of user q per auction.

$$\mathcal{N}_4 = \left\{ \frac{1}{|\mathcal{P}^q|}|\mathcal{B}^{a,q}|, \text{ where } a \in \mathcal{P}^q \right\} \tag{B.5}$$

$$\phi_{4SD} = \sigma(\mathcal{N}_4) \tag{B.6}$$

Definition 14 ϕ_{5SD} is related to ϕ_5. It gives the standard deviation of the bid times as a fraction of auction time elapsed by user q. The definition of *timeFrac* is the same as in Definition 5.

$$\mathcal{N}_5 = \left\{ \frac{1}{|\mathcal{P}^q||\mathcal{T}^{a,q}|} \sum_{s \in \mathcal{T}^{a,q}} timeFrac(s), \text{ where } a \in \mathcal{P}^q \right\} \tag{B.7}$$

$$\phi_5 SD = \sigma(\mathcal{N}_5) \tag{B.8}$$

Definition 15 ϕ_{6SD} is related to ϕ_6. It gives the standard deviation of all bid values as a proportion of the maximum (final) bid by user q.

$$\mathcal{N}_6 = \left\{ \frac{1}{|\mathcal{P}^q||\mathcal{V}^{a,q}|} \sum_{r \in \mathcal{V}^{a,q}} \frac{r}{v^a_{|\mathcal{B}^a|}}, \text{ where } a \in \mathcal{P}^q \right\} \tag{B.9}$$

$$\phi_{6SD} = \sigma(\mathcal{N}_6) \tag{B.10}$$

Definition 16 ϕ_{7SD} is related to ϕ_7. It gives the standard deviation of the proportion of bids made by user q in each auction.

$$\mathcal{N}_7 = \left\{ \frac{|\mathcal{B}^{a,q}|}{|\mathcal{P}^q||\mathcal{B}^a|}, \text{ where } a \in \mathcal{P}^q \right\} \tag{B.11}$$

$$\phi_{7SD} = \sigma(\mathcal{N}_7) \tag{B.12}$$

Definition 17 ϕ_{10SD} is related to ϕ_{10}. It gives the standard deviation of times of first bids made by user q.

$$\mathcal{N}_{10} = \left\{ \frac{b_1^{a,q}}{|\mathcal{P}^q|}, \text{ where } a \in \mathcal{P}^q \right\} \tag{B.13}$$

$$\phi_{10SD} = \sigma(\mathcal{N}_{10}) \tag{B.14}$$

Definition 18 ϕ_{12}, or self bid interval, gives the average difference in time between bids made by user q in the same auction. If user q makes fewer than two bids in a given auction a (this is, if $|B^{a,q}| < 2$), the auction is ignored. If all auctions in $\mathcal{P}^q$ are ignored in this way, then user q has no value for ϕ_{12}.

$$\phi_{12} = ln \left(\frac{1}{|\mathcal{P}^q|} \sum_{a \in \mathcal{P}^q} \frac{1}{|\mathcal{B}^{a,q}| - 1} \sum_{c=2}^{|B^{a,q}|} t_c^{a,q} - t_{c-1}^{a,q} \right), \text{ where } |B^{a,q}| > 1 \tag{B.15}$$

Definition 19 ϕ_{12SD} is related to ϕ_{12}. It gives the standard deviation of time differences between bids made by user q. As in Definition 18, if $|B^{a,q}| < 2$, the auction a is ignored.

$$\mathcal{N}_{12} = \left\{ \frac{1}{|\mathcal{P}^q|(|\mathcal{B}^{a,q}| - 1)} \sum_{c=2}^{|B^{a,q}|} t_c^{a,q} - t_{c-1}^{a,q}, \text{ where } a \in \mathcal{P}^q \right\} \tag{B.16}$$

$$\phi_{12SD} = \sigma(\mathcal{N}_{12}) \tag{B.17}$$

Definition 20 ϕ_{13}, or any bid interval, gives the difference in time between bids made by user q and any preceding bid. If there are fewer than two bids in a given auction a (that is, if $|B^a| < 2$), the auction is ignored. If all auctions in $\mathcal{P}^q$ are ignored in this way, then user q has no value for ϕ_{13}.

$$\phi_{13} = ln \left(\frac{1}{|\mathcal{P}^q|} \sum_{a \in \mathcal{P}^q} \frac{1}{|\mathcal{B}^{a,q}| - 1} \sum_{c=2}^{|B^{a,q}|} t_c^a - t_{c-1}^a \right), \text{ where } |B^a| > 1 \tag{B.18}$$

Definition 21 ϕ_{13SD} is related to ϕ_{13}. It gives the standard deviation of the time differences between bids made by user q and the preceding bid. As in Definition 20, if $|B^a| < 2$, then the auction is ignored.

$$\mathcal{N}_{13} = \left\{ \frac{1}{|\mathcal{P}^q|(|\mathcal{B}^{a,q}| - 1)} \sum_{c=2}^{|B^{a,q}|} t_c^a - t_{c-1}^a, \text{ where } a \in \mathcal{P}^q \right\} \tag{B.19}$$

$$\phi_{13SD} = \sigma(\mathcal{N}_{13}) \tag{B.20}$$

Definition 22 ϕ_{14} is the average number of minutes before the end of the auction that the user made a bid.

$$\phi_{14} = ln \left(1 + \frac{1}{|\mathcal{P}^q|} \sum_{a \in \mathcal{P}^q} \frac{1}{|\mathcal{T}^{a,q}|} \sum_{s \in \mathcal{T}^{a,q}} (m_{ET}^a - s) \right) \tag{B.21}$$

For example, if the auction ends at time 2500, and the user made bids at times {1800, 2100}, then $\phi_{1008} = ln(550)$.

Definition 23 ϕ_{14SD} is related to ϕ_{14}. It gives the standard deviation of bid times, as the number of minutes before the end of the auction, that user q made.

$$\mathcal{N}_{14} = \left\{ \frac{1}{|\mathcal{P}^q||\mathcal{T}^{a,q}|} \sum_{s \in \mathcal{T}^{a,q}} (m_{ET}^a - s), \text{ where } a \in \mathcal{P}^q \right\} \tag{B.22}$$

$$\phi_{14SD} = \sigma(\mathcal{N}_{14}) \tag{B.23}$$

For example, if the auction ends at time 2500, and the user made bids at times {1800, 2100}, then $\phi_8 = ln(5500)$.

Definition 24 ϕ_{15} is the value of the last bid by the user in each auction, averaged over all auctions.

$$\phi_{15} = ln\left(1 + \frac{1}{|\mathcal{P}^q|}\sum_{a\in\mathcal{P}^q} v^{a,q}_{|\mathcal{B}^{a,q}|}\right) \tag{B.24}$$

For example, if the last bids in two auctions were {\$60, \$40}, then $\phi_{15} = 50$.

Definition 25 ϕ_{15SD} is related to ϕ_{15}. It gives the standard deviation of the value of the last bid by user q in each auction.

$$\mathcal{N}_{15} = \left\{\frac{1}{|\mathcal{P}^q|} v^{a,q}_{|\mathcal{B}^{a,q}|}, \text{ where } a \in \mathcal{P}^q\right\} \tag{B.25}$$

$$\phi_{15SD} = \sigma(\mathcal{N}_{15}) \tag{B.26}$$

Definition 26 ϕ_{16}, or final bid amount proportion, gives the average final bid amount as a proportion of maximum: amount of the last bid by the user as a proportion of the amount of the final bid, averaged over all auctions the user participated in.

$$\phi_{16} = \frac{1}{|\mathcal{P}^q|}\sum_{a\in\mathcal{P}^q} \frac{v^{a,q}_{|\mathcal{B}^{a,q}|}}{v^{a}_{|B^{a}|}} \tag{B.27}$$

For example, if the user made bids {\$60, \$40} in 2 auctions with the highest bids as {\$80, \$40} respectively, then $\phi_{16} = 0.875$.

Definition 27 ϕ_{16SD} is related to ϕ_{10}. It gives the standard deviation of final bid amounts, as a proportion of the maximum bid in the auction, by user q in each auction.

$$\mathcal{N}_{16} = \left\{\frac{1}{|\mathcal{P}^q|} \frac{v^{a,q}_{|\mathcal{B}^{a,q}|}}{v^{a}_{|B^{a}|}}, \text{ where } a \in \mathcal{P}^q\right\} \tag{B.28}$$

$$\phi_{16SD} = \sigma(\mathcal{N}_1 6) \tag{B.29}$$

B.1 Correlation between user features

The pairwise feature Pearson correlation shows that several of the features related to bid amounts are strongly correlated to each other, such as between ϕ_6 and ϕ_{16}, and between ϕ_1 and ϕ_{15}. Similarly for features related to bid times: ϕ_{10} and ϕ_{14}. However, surprisingly, ϕ_3 and ϕ_4, which are average bid amount and average bid increment, shows only a weak correlation.

Table B.1
Correlation matrix for all user features

	ϕ_1	ϕ_{1SD}	ϕ_2	ϕ_{2SD}	ϕ_3	ϕ_4	ϕ_{4SD}	ϕ_5	ϕ_{5SD}	ϕ_6	ϕ_{6SD}	ϕ_7	ϕ_{7SD}	ϕ_8	ϕ_9	ϕ_{10}	ϕ_{10SD}	ϕ_{12}	ϕ_{12SD}	ϕ_{13}	ϕ_{13SD}	ϕ_{14}	ϕ_{14SD}	ϕ_{15}	ϕ_{15SD}	ϕ_{16}	ϕ_{16SD}
ϕ_1	1	0.24	0.26	0.18	0.1	0.34	0.19	0.14	-0.05	0.42	-0.26	0.01	0.03	-0.01	-0.06	-0.21	-0.01	-0.15	-0.12	-0.19	-0.15	-0.25	-0.14	0.99	0.2	0.42	-0.1
ϕ_{1SD}	0.24	1	0.11	0.12	0.05	0.02	0.07	-0.02	0.06	0.05	0	0.03	0.02	0.07	0.01	0	0.05	-0.02	0	-0.01	-0.01	-0.01	0.05	0.23	0.94	0.04	-0.03
ϕ_2	0.26	0.11	1	0.83	-0.04	0.15	0.09	-0.02	-0.01	-0.03	0.11	0.05	0	0.1	0.02	0.07	0.04	-0.02	0.06	-0.09	0.03	0.03	0.02	0.29	0.07	0.08	-0.02
ϕ_{2sd}	0.18	0.12	0.83	1	-0.02	0.03	0.08	-0.02	0.06	-0.05	0.14	0.02	0.02	0.1	0.02	0.07	0.05	0	0.08	0	0.04	0.05	0.07	0.19	0.07	0.02	-0.02
ϕ_3	0.1	0.05	-0.04	-0.02	1	-0.09	0.03	-0.09	0.13	0.54	0	0.68	0.17	0.21	0.12	-0.16	-0.05	0	0.02	0	0.03	-0.17	0.03	0.08	0.02	0.53	-0.24
ϕ_4	0.34	0.02	0.15	0.03	-0.09	1	0.38	0.29	-0.09	-0.06	-0.05	-0.03	-0.05	0.01	-0.03	-0.1	-0.01	-0.17	-0.11	-0.34	-0.22	-0.25	-0.2	0.41	0.05	0.26	-0.06
ϕ_{4sd}	0.19	0.07	0.09	0.08	0.03	0.38	1	0.11	0.05	-0.01	0.04	0	0.06	0.06	0	-0.04	0.01	-0.13	-0.06	-0.19	-0.12	-0.12	-0.09	0.19	0.11	0.1	-0.08
ϕ_5	0.14	-0.02	-0.02	-0.02	-0.09	0.29	0.11	1	-0.09	0.09	0.01	-0.39	-0.04	0.04	0.02	-0.42	-0.1	-0.1	-0.04	0.07	0.08	-0.52	-0.18	0.16	0	0.21	-0.05
ϕ_{5sd}	-0.05	0.06	-0.01	0.06	0.13	-0.09	0.05	-0.09	1	0.01	0.27	0.16	0.23	0.34	0.1	0.16	0.12	0.4	0.34	0.12	0.09	0.03	0.48	-0.07	0.01	0.02	0.04
ϕ_6	0.42	0.05	-0.03	-0.05	0.54	-0.06	-0.01	0.09	0.01	1	-0.24	0.54	0.1	0.03	0.07	-0.38	-0.09	-0.09	-0.09	-0.02	-0.04	-0.37	-0.11	0.38	0.04	0.89	-0.23
ϕ_{6sd}	-0.26	0	0.11	0.14	0	-0.05	0.04	0.01	0.27	-0.24	1	0.02	0.15	0.21	0.08	0.19	0.18	0.18	0.22	0.07	0.16	0.12	0.34	-0.22	-0.03	-0.05	0.51
ϕ_7	0.01	0.03	0.05	0.02	0.68	-0.03	0	-0.39	0.16	0.54	0.02	1	0.18	0.16	0.12	-0.04	0	0.04	0.06	0	0.04	-0.05	0.09	0.01	0.01	0.56	-0.2
ϕ_{7sd}	0.03	0.02	0	0.02	0.17	-0.05	0.06	-0.04	0.23	0.1	0.15	0.18	1	0.12	0.06	-0.02	0.05	0.01	0.01	0.01	0	-0.02	0.04	0.03	0.02	0.1	0.12
ϕ_8	-0.01	0.07	0.1	0.1	0.21	0.01	0.06	0.04	0.34	0.03	0.21	0.16	0.12	1	0.25	-0.01	0.07	0.03	0.16	-0.01	0.21	-0.05	0.27	-0.04	-0.01	0.08	-0.01
ϕ_9	-0.06	0.01	0.02	0.02	0.12	-0.03	0	0.02	0.1	0.07	0.08	0.12	0.06	0.25	1	-0.03	0.02	0.01	0.05	0	0.07	-0.04	0.09	-0.07	-0.01	0.07	0
ϕ_{10}	-0.21	0	0.07	0.07	-0.16	-0.1	-0.04	-0.42	0.16	-0.38	0.19	-0.04	-0.02	-0.01	-0.03	1	0.28	0.42	0.41	0.05	0.07	0.93	0.53	-0.2	-0.01	-0.36	0.14
ϕ_{10SD}	-0.01	0.05	0.04	0.05	-0.05	-0.01	0.01	-0.1	0.12	-0.09	0.18	0	0.05	0.07	0.02	0.28	1	0.17	0.28	0.01	0.03	0.23	0.68	-0.01	0.05	-0.09	0.26
ϕ_{12}	-0.15	-0.02	-0.02	0	0	-0.17	-0.13	-0.1	0.4	-0.09	0.18	0.04	0.01	0.03	0.01	0.42	0.17	1	0.48	0.2	0.17	0.26	0.61	-0.14	-0.02	-0.07	0.06
ϕ_{12sd}	-0.12	0	0.06	0.08	0.02	-0.11	-0.06	-0.04	0.34	-0.09	0.22	0.06	0.01	0.16	0.05	0.41	0.28	0.48	1	0.08	0.17	0.21	0.55	-0.11	-0.01	-0.04	0.06
ϕ_{13}	-0.19	-0.01	-0.09	0	0	-0.34	-0.19	0.07	0.12	-0.02	0.07	0	0.01	-0.01	0	0.05	0.01	0.2	0.08	1	0.43	0.08	0.15	-0.21	-0.03	-0.12	0.05
ϕ_{13sd}	-0.15	-0.01	0.03	0.04	0.03	-0.22	-0.12	0.08	0.09	-0.04	0.16	0.04	0	0.21	0.07	0.07	0.03	0.17	0.17	0.43	1	0.06	0.16	-0.16	-0.03	-0.06	0.05
ϕ_{14}	-0.25	-0.01	0.03	0.05	-0.17	-0.25	-0.12	-0.52	0.03	-0.37	0.12	-0.05	-0.02	-0.05	-0.04	0.93	0.23	0.26	0.21	0.08	0.06	1	0.37	-0.25	-0.02	-0.42	0.13
ϕ_{14sd}	-0.14	0.05	0.02	0.07	0.03	-0.2	-0.09	-0.18	0.48	-0.11	0.34	0.09	0.04	0.27	0.09	0.53	0.68	0.61	0.55	0.15	0.16	0.37	1	-0.15	0	-0.12	0.22
ϕ_{15}	0.99	0.23	0.29	0.19	0.08	0.41	0.19	0.16	-0.07	0.38	-0.22	0.01	0.03	-0.04	-0.07	-0.2	-0.01	-0.14	-0.11	-0.21	-0.16	-0.25	-0.15	1	0.2	0.44	-0.11
ϕ_{15sd}	0.2	0.94	0.07	0.07	0.02	0.05	0.11	0	0.01	0.04	-0.03	0.01	0.02	-0.01	-0.01	-0.01	0.05	-0.02	-0.01	-0.03	-0.03	-0.02	0	0.2	1	0.03	-0.03
ϕ_{16}	0.42	0.04	0.08	0.02	0.53	0.26	0.1	0.21	0.02	0.89	-0.05	0.56	0.1	0.08	0.07	-0.36	-0.09	-0.07	-0.04	-0.12	-0.06	-0.42	-0.12	0.44	0.03	1	-0.31
ϕ_{16sd}	-0.1	-0.03	-0.02	-0.02	-0.24	-0.06	-0.08	-0.05	0.04	-0.23	0.51	-0.2	0.12	-0.01	0	0.14	0.26	0.06	0.06	0.05	0.05	0.13	0.22	-0.11	-0.03	-0.31	1

C

Model performance when including alternative features

Table C.1 and C.2 compares the performance results of decision tree models trained using all features listed in Appendix B and Section 5.2.4, and DT (Features). Performance of DT (Features) is better for all accuracy measures.

Table C.1
Detection performance of simple shills

Model	TPR	FPR	AUC	Precision	F-measure
DT (features)	0.998 (4.64E-05)	0.001 (4.67E-05)	0.998 (5.67E-05)	0.999 (4.68E-05)	0.998 (4.02E-05)
DT (All Features)	0.980 (0.010)	0.040 (7.44E-03)	0.971 (4.65E-03)	0.961 (7.52E-03)	0.971 (8.89E-03)

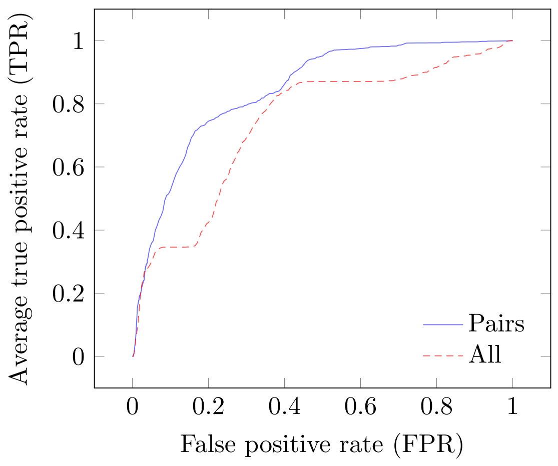
Table C.2
Detection performance of delayed-start shills

Model	TPR	FPR	AUC	Precision	F-measure
DT (Features)	0.967 (1.37E-04)	0.023 (1.58E-04)	0.992 (0.81E-04)	0.977 (1.55E-04)	0.972 (9.39E-05)
DT (All Features)	0.927 (0.013)	0.084 (9.67E-03)	0.951 (5.2E-03)	0.917 (8.3E-03)	0.922 (5.7E-03)

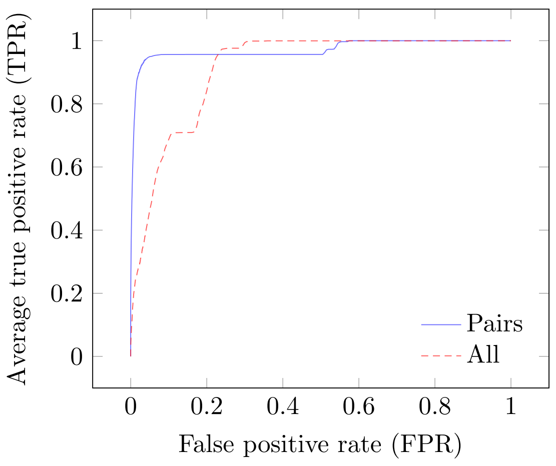
D Behaviour of SPAN

D.1 Quality of anomaly scores

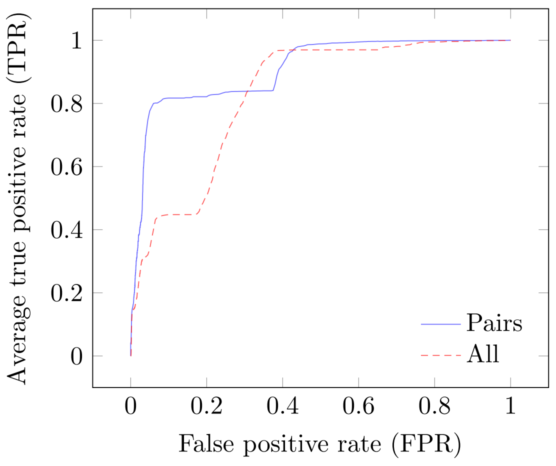
For all three fraud types, LOFs calculated using feature pairs produces scores that more accurately identify fraudulent users (by giving them higher scores) than using all features simultaneously. To compare the quality of the anomaly scores, we rank users according to their anomaly score, then partition the users into fraudulent and normal groups over a range of thresholds. For each threshold, a pair of TPR and FPR values can be calculated. These have been plotted in Figure D.1 for each fraud type. Figure D.1 shows that for most FPR values, anomaly scores calculated using feature pairs give a higher TPR value, indicating that using feature pairs produces more accurate anomaly scores.



(a) Reputation fraud



(b) Collusive star



(c) Collusive clique

Fig. D.1. Average ROCs for anomaly scores calculated using feature pairs, and using all features at once

E

Effect of removing fraud-fraud edges for collusive star for 2LFS

Figure E.1 shows the effect of removing fraud-fraud edges on ROC for collusive star for 2LFS. The values in the legend for each line correspond to the fraction of fraud-fraud edges that were removed from the graph. There is virtually no difference in the TPR for the different ROC curves when FPR is low, which results in the horizontal line seen in Figure 6.12(b), where TPR remains the same when different proportions of fraud-fraud lines are removed. This is also the reason for the horizontal lines seen in Figures 6.14(b) and 6.15(b).

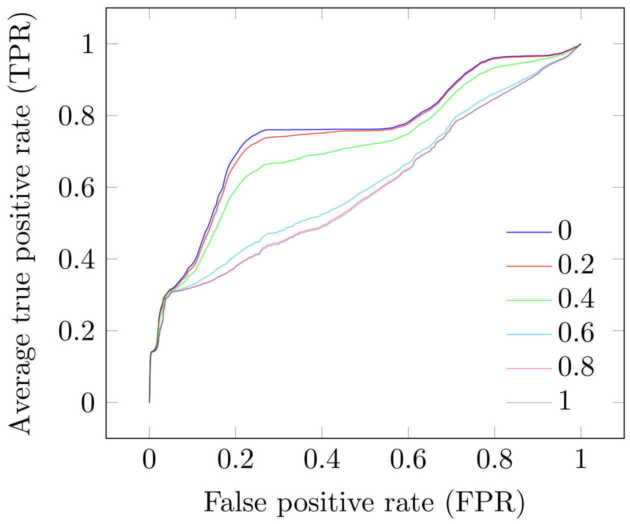
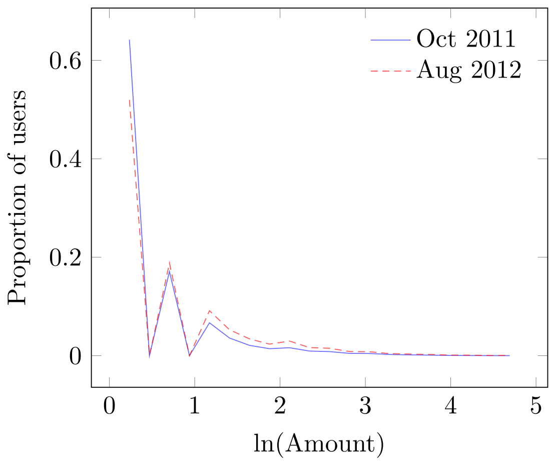
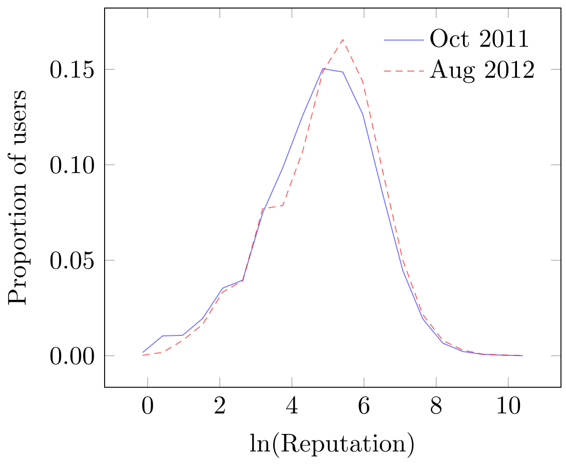


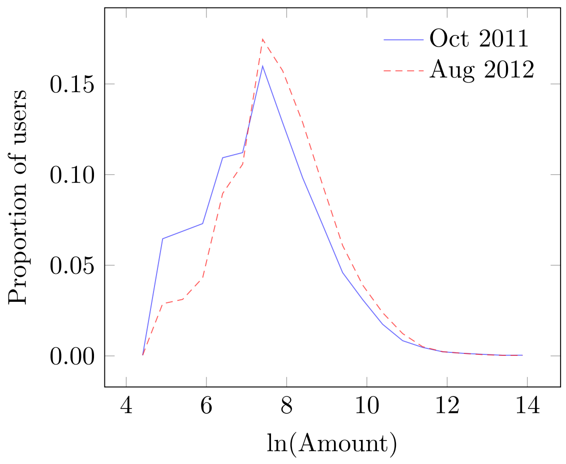
Fig. E.1. Shift in ROC for collusive star fraud for 2LFS as fraud-fraud edges are removed



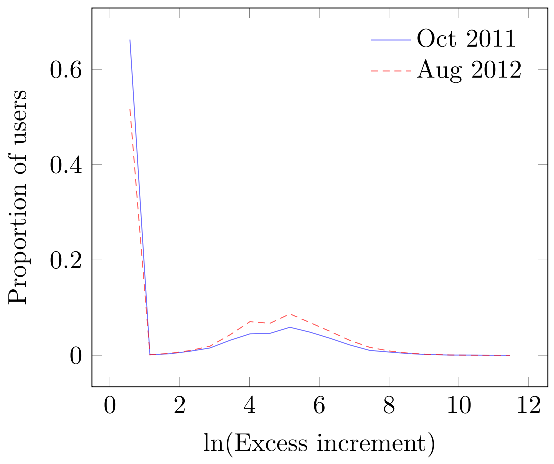
(a) Auction count



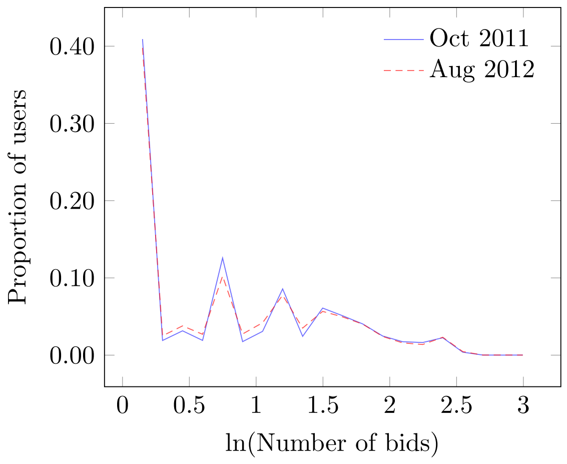
(b) Reputation



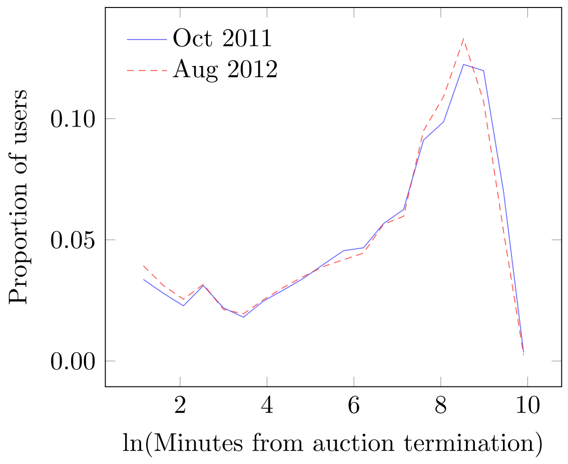
(c) Bid amount



(d) Excess bid increment

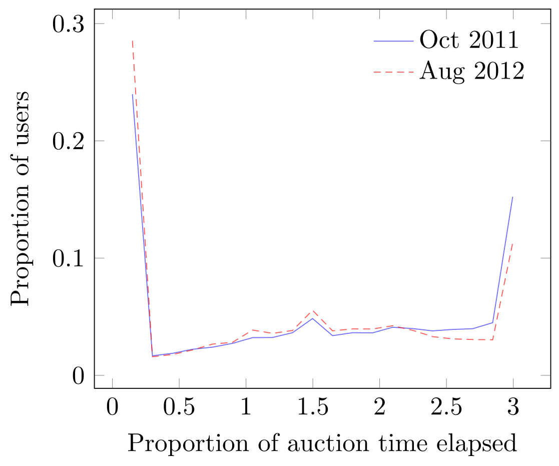


(e) Bids per auction

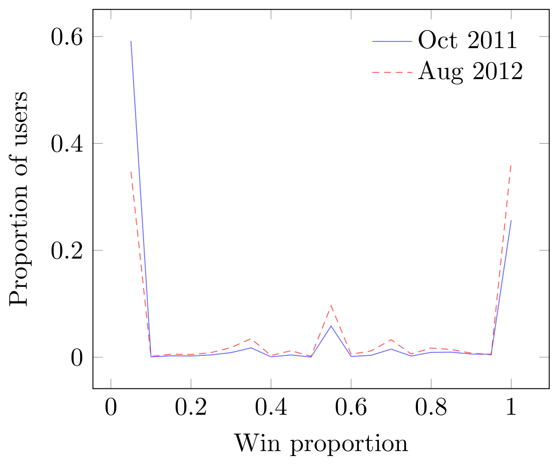


(f) First bid time

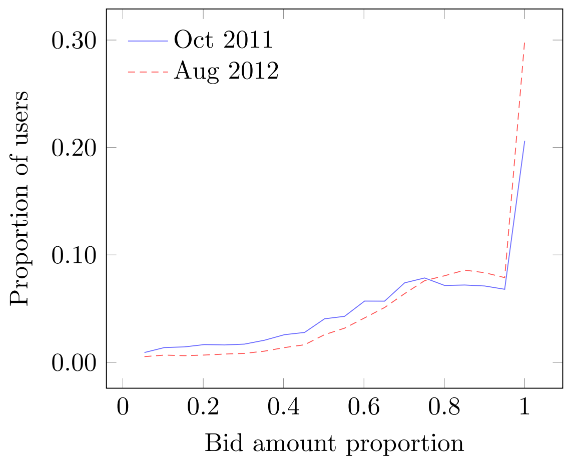
Fig. E.2. Value distributions of two collected datasets



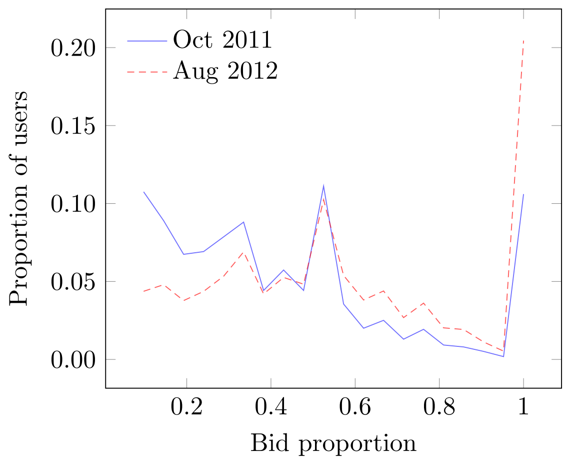
(g) Bid time



(h) Proportion won



(i) Bid amount proportion

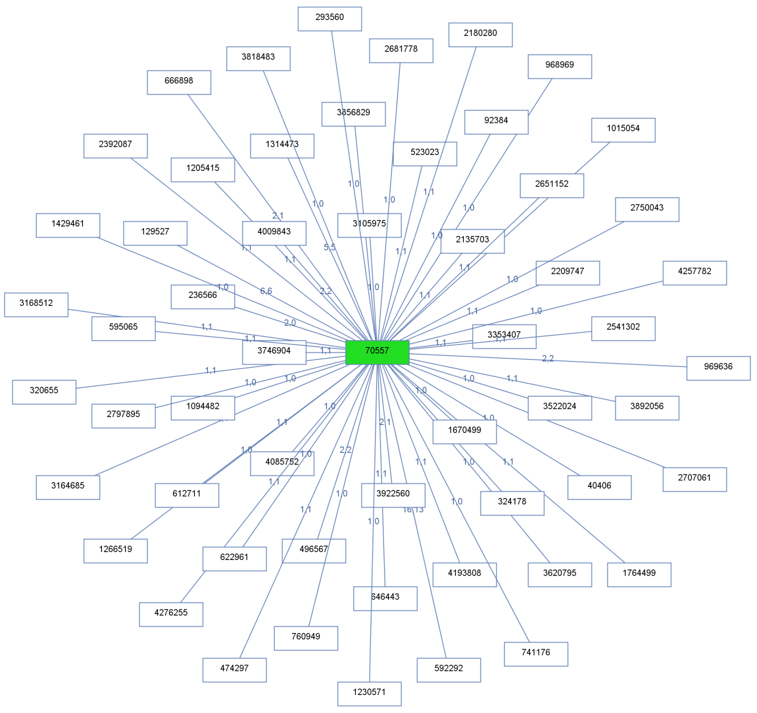


(j) Bid proportion

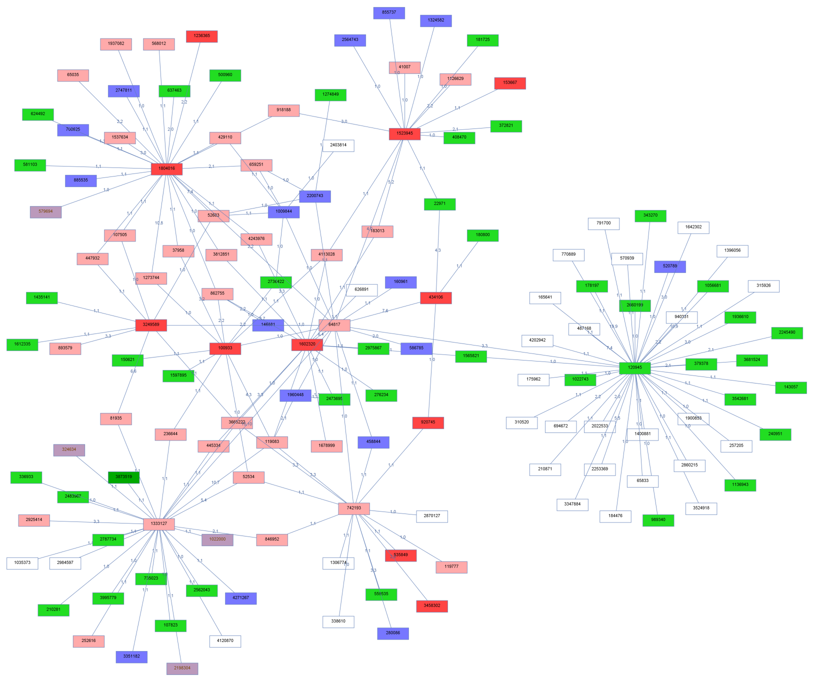
Fig. E.2. Value distributions of two collected datasets (cont.)

F
False positive examples

Figure F.1 shows some examples of possible false positives found by 2LFS. In Figure F.1(a), 2LFS classifies 378087 as fraudulent with all of its neighbours classified as normal. In Figure F.1(b), both 2LFS and SPAN identify a group of nodes as non-normal. However, 2LFS identifies a large number of additional nodes as fraudulent compared to SPAN. This is likely due to 2LFS's incorrect assumptions about bipartite cores, as discussed earlier.



(a) Egonet of 70557



(b) 2-step neighbourhood of 64817

Fig. F.1. Example of user classifications by 2LFS and SPAN. Uncoloured nodes are normal; light green nodes are accomplices found by 2LFS; dark green nodes are frauds found by 2LFS; blue nodes are frauds found by SPAN; red nodes are frauds found by both 2LFS and SPAN.

9
Bibliography

[1] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Understanding belief propagation and its generalizations," *Exploring Artificial Intelligence in the New Millennium*, vol. 8, pp. 236–239, 2003.

[2] J. Trevathan and W. Read, "Detecting shill bidding in online English auctions," *ACM Transactions on Computational Logic*, vol. 5, 2006.

[3] F. M. Menezes and P. K. Monteiro, *An Introduction to Auction Theory*, ch. 2.3.2 Auction Types, pp. 10–11. Oxford University Press, USA, 2005.

[4] J. Hainline, "Last years seller successes, and top 2014 tips for you." `http://for-business.ebay.com/last-year%E2%80%`

`99s-seller-successes-and-top-2014-tips-you`, 2014. [Online, accessed 7-July-2014].

[5] F. Dong, S. M. Shatz, and H. Xu, "Combating online in-auction fraud: Clues, techniques and challenges," *Computer Science Review*, vol. 3, no. 4, pp. 245–258, 2009.

[6] W. Chang and J. Chang, "An effective early fraud detection method for online auctions," *Electronic Commerce Research and Applications*, vol. 11, no. 4, pp. 346–360, 2012.

[7] H. Mizuta and K. Steiglitz, "Agent-based simulation of dynamic online auctions," *Simulation Conference, 2000. Proceedings. Winter*, vol. 2, pp. 1772–1777, 2000.

[8] H. Shah, N. Joshi, A. Sureka, and P. Wurman, *Mining eBay: Bidding Strategies and Shill Detection*, vol. 2703 of *Lecture Notes in Computer Science*, pp. 17–34. Springer Berlin / Heidelberg, 2003.

[9] D. G. Gregg and J. E. Scott, "The role of reputation systems in reducing on-line auction fraud," *Int. J. Electron. Commerce*, vol. 10, no. 3, pp. 95–120, 2006.

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

[11] R. J. Kauffman and C. A. Wood, "The effects of shilling on final bid prices in online auctions," *Electron. Commer. Rec. Appl.*, vol. 4, no. 1, pp. 21–34, 2005.

[12] J. Trevathan and W. Read, "A simple shill bidding agent," *Fourth International Conference on Information Technology, 2007. ITNG '07.*, pp. 766–771.

[13] D. Chau, S. Pandit, and C. Faloutsos, *Detecting Fraudulent Personalities in Networks of Online Auctioneers*, vol. 4213 of *Lecture Notes in Computer Science*, pp. 103–114. Springer Berlin / Heidelberg, 2006.

[14] W. You, L. Liu, M. Xia, and C. Lv, "Reputation inflation detection in a Chinese C2C market," *Electronic Commerce Research and Applications*, vol. 10, no. 5, pp. 510–519, 2011.

[15] P. Resnick and R. Zeckhauser, *Trust Among Strangers in Internet Transactions: Empirical Analysis of eBay's Reputation System*, vol. 11, pp. 127–157. Elsevier Science, 2002.

[16] M. Chae, S. Shim, H. Cho, and B. Lee, "An empirical analysis of fraud detection in online auctions: Credit card phantom transaction," *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, p. 155a, 2007.

[17] J. Trevathan and W. Read, "Detecting collusive shill bidding," *Fourth International Conference on Information Technology, 2007. ITNG '07.*, pp. 799–808, 2007.

[18] H. Xu, C. K. Bates, and S. M. Shatz, "Real-time model checking for shill detection in live online auctions," *Proceedings of the International Conference on Software Engineering Research and Practice*, pp. 134–140, 2009.

[19] S. Lin, Y. Jheng, and C. Yu, "Combining ranking concept and social network analysis to detect collusive groups in online auctions," *Expert Systems with Applications*, vol. 39, no. 10, pp. 9079–9086, 2012.

[20] V. Almendra, "Finding the needle: A risk-based ranking of product listings at online auction sites for non-delivery fraud prediction," *Expert Systems with Applications*, vol. 40, no. 12, pp. 4805–4811, 2013.

[21] V. Almendra and D. Enachescu, "A supervised learning process to elicit fraud cases in online auction sites," *13th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 168–174, 2011.

[22] L. Zhang, J. Yang, and B. Tseng, "Online modeling of proactive moderation system for auction fraud detection," *Proceedings of the 21st International Conference on World Wide Web*, pp. 669–678.

[23] S. Pandit, D. H. Chau, S. Wang, and C. Faloutsos, “Netprobe: a fast and scalable system for fraud detection in online auction networks,” *Proceedings of the 16th International Conference on World Wide Web*, pp. 201–210, 2007.

[24] W. Chang and J. Chang, “A novel two-stage phased modeling framework for early fraud detection in online auctions,” *Expert Systems with Applications*, vol. 38, no. 9, pp. 11244–11260, 2011.

[25] R. J. Kauffman and C. A. Wood, “Running up the bid: detecting, predicting, and preventing reserve price shilling in online auctions,” *Proceedings of the 5th International Conference on Electronic Commerce*, pp. 259–265, 2003.

[26] W. Wang, Z. Hidvegi, and A. B. Whinston, “Shill-proof fee (spf) schedule: The sunscreen against seller self-collusion in online English auctions,” *Goizueta Paper Series, Emory University*, 2004.

[27] B. Bhargava, M. Jenamani, and Y. Zhong, “Counteracting shill bidding in online English auction,” *International Journal of Cooperative Information Systems*, pp. 245–263.

[28] X. Li, L. Ling, and M. Ahamad, “Countering feedback sparsity and manipulation in reputation systems,” *International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2007. CollaborateCom 2007*, pp. 203–212.

[29] H. Dia, “An agent-based approach to modelling driver route choice behaviour under the influence of real-time information,” *Transportation Research Part C: Emerging Technologies*, vol. 10, no. 56, pp. 331–349, 2002.

[30] V. Grimm, E. Revilla, U. Berger, F. Jeltsch, W. M. Mooij, S. F. Railsback, H.-H. Thulke, J. Weiner, T. Wiegand, and D. L. DeAngelis, “Pattern-oriented modeling of agent-based complex systems: Lessons from ecology,” *Science*, vol. 310, no. 5750, pp. 987–991, 2005.

[31] D. W. Bunn and F. S. Oliveira, “Agent-based simulation-an application to the new electricity trading arrangements of England and Wales,” *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 5, pp. 493–503, 2001.

[32] L. Tesfatsion, *Chapter 16 Agent-Based Computational Economics: A Constructive Approach to Economic Theory*, vol. 2, pp. 831–880. Elsevier, 2006.

[33] J. M. Epstein, “Modeling civil violence: An agent-based computational approach,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99, no. Suppl 3, pp. 7243–7250, 2002.

[34] C. M. Macal and M. J. North, “Agent-based modeling and simulation,” *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pp. 86–98, 2009.

[35] E. Bonabeau, “Agent-based modeling: methods and techniques for simulating human systems,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 99 Suppl 3, pp. 7280–7287, 2002.

[36] R. Bapna, P. Goes, and A. Gupta, “Simulating online yankee auctions to optimize sellers revenue,” *Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 2001.*, p. 10 pp., 2001.

[37] C. Yue, S. Mabu, Y. Wang, and K. Hirasawa, “Multiple-round English auction agent based on genetic network programming,” *IEEJ Transactions on Electrical and Electronic Engineering*, vol. 5, no. 4, pp. 450–458, 2010.

[38] A. C. Davison, *Bootstrap methods and their application*, vol. 1. Cambridge university press, 1997.

[39] T. Lange, V. Roth, M. L. Braun, and J. M. Buhmann, “Stability-based validation of clustering solutions,” *Neural Comput.*, vol. 16, no. 6, pp. 1299–1323, 2004.

[40] D. Pelleg and A. Moore, “X-means: Extending k-means with efficient estimation of the number of clusters,” pp. 727–734, 2000.

[41] B. Student, "The probable error of a mean," *Biometrika*, vol. 6, no. 1, pp. 1–25, 1908.

[42] F. J. Massey Jr, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American Statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

[43] P. R. Rosenbaum, "An exact distribution-free test comparing two multivariate distributions based on adjacency," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 67, no. 4, pp. 515–530, 2005.

[44] J. L. Myers, A. D. Well, and R. F. Lorch, *The Pearson Product-Moment Correlation Coefficent*, ch. 18.5 Introducting correlation and regression using z scores, pp. 443–447. Routledge, 3 ed., 2010.

[45] B. McCune, J. Grace, and D. Urban, *Analysis of Ecological Communities*, ch. 24 MRPP (Multi-response Permutation Procedures), pp. 188–197. MjM Software Design, 2002.

[46] H. Fang and Y. Saad, "Farthest centroids divisive clustering," *Seventh International Conference on Machine Learning and Applications, 2008. ICMLA '08.*, pp. 232–238, 2008.

[47] W. Chang and J. Chang, "Using clustering techniques to analyze fraudulent behavior changes in online auctions," pp. 34–38, 2010.

[48] C. Drummond and R. C. Holte, "C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling," *Workshop on Learning from Imbalanced Datasets II*, vol. 11, 2003.

[49] L. Akoglu, M. McGlohon, and C. Faloutsos, "Oddball: Spotting anomalies in weighted graphs," *Advances in Knowledge Discovery and Data Mining*, pp. 410–421, 2010.

[50] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," *SIGMOD Rec.*, vol. 29, no. 2, pp. 93–104, 2000.

[51] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 1–58, 2009.

[52] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 2004.

[53] S. B. Seidman, "Network structure and minimum degree," *Social networks*, vol. 5, no. 3, pp. 269–287, 1983.